Journal of Systems Architecture 60 (2014) 783–795

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

MoNoC: A monitored network on chip with path adaptation mechanism

Edson Moreno, Thais Webber, César Marcon, Fernando Moraes [*], Ney Calazans

PUCRS University, Computer Science Department, Porto Alegre 90619-900, Brazil

ARTICLE INFO

Article history:
Received 5 November 2013
Received in revised form 6 August 2014
Accepted 6 October 2014
Available online 12 October 2014

Keywords:
NoC
Networks on chip
Monitoring
Routing methods
Adaptive control

ABSTRACT

Complex systems on chip containing dozens of processing resources with critical communication requirements usually rely on the use of networks on chip (NoCs) as communication infrastructure. NoCs provide significant advantages over simpler infrastructures such as shared busses or point to point communication, including higher scalability, more efficient energy management, higher bandwidth and lower average latency. Applications running on NoCs with more than 10% of bandwidth usage attest that the most significant portion of message latencies refers to buffered packets waiting to enter the NoC, whereas the latency portion that depends on the packet traversing the NoC is sometimes negligible. This work presents an adaptive routing architecture, named *Monitored NoC* (MoNoC), which is based on a traffic monitoring mechanism and the exchange of high priority control packets. This method enables to adapt paths by choosing less congested routes. Practical experiments show that the proposed path adaptation is a fast process, enabling to transmit packets with smaller latencies, up to 9 times smaller, by using non-congested NoC regions.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

New technologies for manufacturing silicon chips enable to integrate a huge amount of transistors, which allows the inclusion of numerous modules with possibly diverse structures in the same die, including electronic digital and analog components, optical and even magnetic components. These features enable the construction of entire systems on a chip, the so-called SoCs [1].

SoCs are widely used to meet the demand of applications with special requirements such as low power consumption and/or high reliability. In addition, the parallelism enabled by SoC architectures often fulfill the increasing need for computing power required in applications where tasks mapped into different Processing Elements (PEs) imply large amounts of communication. To meet such communication demands, SoCs employ NoC topologies with high bandwidth, low latency, high scalability and high-energy efficiency [2].

Communication resources like links and buffers in overloaded NoCs display very high utilization rates, which increases average message latencies and affects applications' execution time. This performance issue motivates the proposition of adaptive approaches. This paper presents the *Monitored NoC* (MoNoC), a network on chip architecture with virtual channels and mechanisms for runtime path adaptation based on resource monitoring and workload distribution policies to minimize the overall delay of packets. Distributed monitors provide NoC status information using high-priority packets that flow in specialized virtual channels. The MoNoC runtime adaptive mechanism reduces the average latency in congested scenarios, when compared to a similar NoC without monitoring. The designer cannot neglect this gain since the congestion inside the NoC can dramatically affect the execution time of applications. An example of congestion arrives in hotspot scenarios, for example, when a set of Processing Elements tries to have access to a shared memory or a PE that interfaces with the external world.

The rest of this article is organized as follows. Section 2 presents related works on techniques for NoC monitoring. Section 3 explains the MoNoC, highlighting the router architecture, the design of virtual channels, and the monitoring mechanism. Section 4 presents the runtime routing exploration, i.e. it describes the runtime path adaptation with methods for both adaptive and non-adaptive protocols and path decision policies. Section 5 presents the obtained experimental results, with the corresponding discussion. Finally, Section 6 proposes a set of conclusions of this work.

2. Related work and proposed approach

This section presents an encompassing discussion of recent works related to NoC monitoring. Almost all of these works adopt

* Corresponding author.
E-mail addresses: cesar.marcon@pucrs.br (C. Marcon), fernando.moraes@pucrs.br (F. Moraes), ney.calazans@pucrs.br (N. Calazans).

http://dx.doi.org/10.1016/j.sysarc.2014.10.002
1383-7621/© 2014 Elsevier B.V. All rights reserved.

2D-mesh topology as target communication architecture, due to its scalability, regularity and easiness of mapping in traditional planar CMOS technologies. However, some of these works employ distinct topologies (e.g. DMesh [3], 3D Mesh–Bus Hybrid [4] and 2D Torus [5]) while other works claim that their monitoring method is generic enough to be employed in other tile-based topologies (e.g. [6,7]). Although the experiments presented in this paper are based on mesh topology, MoNoC's monitoring mechanism is independent of the NoC topology. The proposed mechanism requires only a set of control packets containing communication costs according to chosen paths between source and target PEs. Therefore, the MoNoC monitoring mechanism can be employed to any topology.

Both centralized and distributed monitoring mechanisms are available in the literature, but most of them adopt a distributed approach, to fulfill scalability needs. Usually, the global view provided by centralized monitoring (e.g. [7–10]) allows better results for small NoCs. Nevertheless, as the NoC/system size increases, traffic load and monitoring packets latency penalize the centralized approach. It indeed tends to generate a hotspot region around the PE responsible for treating the monitored packets, overloading this PE. Aiming to minimize this problem, some works (e.g. [11]) employ a cluster based technique, which centralizes the monitoring in particular clusters. Nevertheless, some works (e.g. [12,13]) blend distributed and centralized monitoring aiming to join the scalability of distributed approaches with the global view of centralized approaches. Additionally, some research proposals (e.g. [14,15]) employ the hierarchical monitoring concept over the distributed approach dealing with levels of granularity of the system (e.g. application and cell), which enables to take selective decisions concerning the use of the systems resources. MoNoC, as the majority of works presented in this section (e.g. [16–30]), employs distributed monitoring based on communicating pairs of PEs, being scalable, supporting the requirements of the large amount of modules of recent and future systems.

Routing algorithms that take decisions based on the local status (e.g. querying neighbor routers) may be inefficient due to their partial view of the traffic load. In addition, the adoption of a global manager to deal with the monitored information is hardly scalable. Our work adopts an original approach, which is easily scalable. For any pair of communicating PEs, the target PE monitors the received packets, and the source PE selects a new path from a set of pre-computed paths containing the costs of each segment that composes the path. MoNoC routing is deterministic and partially adaptive at the same time. During packet transmission, the source–target path does not change, characterizing deterministic routing. When a control packet signaling congestion is received by the source PE, a new path is adopted. As the path taken between different packets may change, the routing is also partially adaptive, minimizing the average message latency.

NoC designers currently implement monitoring mechanisms with one of three approaches: (i) *using a small-dedicated network that operates in parallel with the data network* (e.g. [4,13,22]). It is a non-intrusive method, and without considering implementation aspects, it seems the most efficient method. However, the extra monitoring network requires additional circuits and wires, consuming area and energy, which may compromise the overall system performance; (ii) *using control packets with high priority over the same data network* (e.g. [7,8,16]). This approach reduces temporarily the data transmission rate, implying some design attention when the approach is applied to real-time systems. On the other hand, this method has low implementation cost, and it is efficient when the monitoring packets are only a small part of the overall communication; (iii) *using distributed agents that detect local traffic behavior to infer and/or predict the global system behavior* (e.g. [3,5,10]). This approach does not require a communication architecture to transmit monitoring packets, avoiding the problems stated in the previous approaches. The drawback of this approach is that it is only efficient for router traffic observation, or at most its connected neighbors, since local traffic status is never broadcasted to the distributed agents. As a result, some local decisions taken at a given agent can neglect the overall system performance. MoNoC adopts the second approach, i.e. employs control packets with high priority, since it presents a good compromise between NoC area overhead and global performance.

Finally, the evaluated proposals employ monitoring mechanisms mainly to fulfill requirements such as latency minimization (e.g. [3,4,18,20]), throughput increase (e.g. [3,4,19,25]) and energy consumption or power dissipation reduction (e.g. [5,12,28,30]). Some of these proposals focus on providing Quality-of-Service (QoS) in terms of granting minimum bandwidth (e.g. [17,27]), and others focus on fault detection and tolerance (e.g. [6,14,15,29]), and efficient thermal distribution (e.g. [4,7]). Our monitoring approach uses the traffic information to balance the communication, which allows to minimize packets congestion, and consequently reducing the average message latency.

Table 1 shows a summary of recent NoC monitoring works, emphasizing the employed topology, the chosen monitoring type and method, and the specific goals to achieve.

3. The MoNoC architecture

This Section details the MoNoC architecture. MoNoC relies on hardware implementations for most of its features, requiring no software to control its basic components. Conceptually, MoNoC is a generic communication infrastructure (i.e. topology agnostic) that aggregates mechanisms to monitor data traffic and policies for efficient use of NoC resources. The main goal is to achieve fair workload distribution. Without loss of generality, this work assumes the use of a 2D mesh topology, given the adopted routing algorithms employed in the implementation. Fig. 1 shows a MoNoC implemented as a 2D mesh topology, highlighting its four basic components: (i) Routers and Links; (ii) Intra Monitors; (iii) Inter Monitors; (iv) Network Interfaces (NIs).

What makes MoNoC topology agnostic is its adoption of source routing. The packet structure is illustrated in Fig. 2. The packet header contains the path (i.e. sequence of router output ports) to the target router and the payload size. The path to the target (Path) field may contain any number of flits starting at 1. Each direction is encoded in four bits as follows: 0x0$\rightarrow$ East, 0x1$\rightarrow$ West, 0x2$\rightarrow$ North, 0x3$\rightarrow$ South, and 0xF$\rightarrow$ Invalid. In the example of Fig. 2 the first flit defines that the packet must travel two hops in the East direction and then 2 hops in the North direction. When an invalid code is found, this implies the packet reached its destination. The path field necessarily ends with a *terminator* flit (with value 0xFFFF). This method ensures scalability, since it can be used with any NoC size.

All experiments presented here adopt as baseline the Hermes-SR NoC [33], which employs an arbitration scheme distributed over the router's output ports. If two or more input ports simultaneously require the same output port, one input port will have its requisition granted, while the other one is queued to be served later, according to the arbitration policy. The present work adopts a *first come, first served* (FCFS) arbitration policy.

At design time, applications' tasks are mapped to PEs. For each communicating pair, the source PE statically computes a set of possible paths to the target PE. Paths are computed using deadlock-free adaptive algorithms as the ones described in [31] or [32]. The proposed path adaptation mechanism uses this set of paths. The number of paths for each communicating pair is kept small (typically four), a trade-off between path diversity and scalability. Path diversity means that each path uses different NoCs regions,

Table 1
Summary of recent works on NoC monitoring.

Work	NoC topology	Monitoring		Goals
		Type	Method	
[3]	Special mesh	Distributed	Monitoring of network congestion, by inspecting the status of router input ports	Throughput increase and latency reduction
[4]	Mesh–Bus Hybrid	Distributed	Monitoring platform placed on a 3D NoC that utilizes bus arbiters to exchange monitoring information directly with other arbiters without using NoC channels	Fault tolerance and thermal management
[5]	Torus	Distributed	Monitoring using counters and timers in each router channel. According to the communication pattern, a runtime mechanism adapts the buffer size	Throughput increase, power and latency reduction
[6]	Any	Distributed	Monitoring by computing the minimum expected delivery time of a packet from its current location to its final destination	Fault tolerance
[7]	Several	Centralized	Monitoring of several types of services (e.g. DVFS – Dynamic Voltage Frequency Scaling) through special data packets with priority levels	Power reduction and thermal distribution
[8]	Mesh	Centralized	Monitoring using special packets containing global traffic information	Congestion avoidance and traffic balancing
[9]	Mesh	Centralized	Monitoring performed by two modules: (i) feedback that monitors the traffic load across the links; and (ii) control that adjusts the routing according to the most traffic load, in order to maximize load balancing and to reduce congestion	Load balancing and congestion reducing
[10]	Mesh	centralized	Non-intrusive real-time monitoring through programmable hardware units, and dynamically configured monitoring agents integrated on network interfaces	Efficient resource management
[11]	Mesh	Centralized cluster based	Traffic monitoring managed by a cluster agent, which captures the links load, the source–target paths inside the cluster, and the overall data loads of the IP cores	Congestion avoidance in NoC regions
[12]	Mesh	Centralized/ distributed	Monitoring using local and global traffic pattern (table-driven) predictors to forecast end-to-end traffic behavior	Thermal distribution and power reduction
[13]	mesh	Centralized/ distributed	Runtime management architecture that integrates fine and coarse grain monitoring (through a dedicated NoC), plus decision-making and reconfiguration	Performance increase and energy minimization
[14] [15]	Mesh	Hierarchical distributed	NoC's architecture based on a hierarchy of agents (4 monitoring levels: application, cluster, cell and platform) providing different granularities for system optimization	Latency reduction, and fault tolerance
[16]	Mesh	Distributed	Employs a hardware/software infrastructure for runtime observability performed with packets sent through the regular data network in a hardly intrusive way	Bandwidth guarantee (QoS requirement)
[17]	Mesh	Distributed	Employs 2 monitoring ways: (i) the router status determining the NoC traffic level based on switching activity; and (ii) the FIFO buffers status focusing on links usage	Latency reduction and throughput increase
[18]	Mesh	Distributed	Employs a neural network for learning network state and adapt routing decisions. Monitoring is performed with packets sent through the regular data network	Latency reduction
[19]	Mesh	Distributed	Monitoring of local buffers with global fault information to determine best routing algorithm	Throughput increase, power and latency reduction
[20]	Mesh	Distributed	Monitoring of every node with per-destination congestion state (average delay) to all other nodes through candidate output ports	Latency reduction
[21]	Mesh	Distributed	Monitoring using data packets to send traffic information (which is stored in tables located in each router) with QoS	Latency reduction
[22]	Mesh	Distributed	Monitoring using extra wires added to routers that indicate congested and/or faulty links	Latency reduction and faulty links avoidance
[23]	Mesh	Distributed	Monitoring of network congestion, analyzing the time a flit stays on the same buffer position	Latency and traffic saturation
[24]	Mesh	Distributed	Monitoring of NoC status, beyond neighboring nodes, applying a congestion propagation network solution	Latency and power dissipation reduction
[25]	Mesh	Distributed	Monitoring of packet switching through a dynamic programming (DP) network, which provides on-the-fly optimal path planning	Throughput increase and faulty links avoidance
[26]	Mesh	Distributed	Monitoring of application's traffic to reconfigure the network by using a dynamic bypass circuit	Latency reduction
[27]	Mesh	Distributed	Monitoring through NoC status packets used to provide information for adaptive routing, coupled with adaptive buffer assignment	Bandwidth guarantee (QoS requirement)
[28]	Mesh	Distributed	Monitoring technique based on memory access time that employs a simple and effective control algorithm to DVFS operation	Power saving
[29]	Mesh	Distributed	A distributed monitoring mechanism that investigates how much the routing algorithms impact message overhead for faulty links detection	Fault tolerance
[30]	Mesh	Distributed	Monitoring of information provided by mechanisms placed on each router to implement a routing algorithm that enables to deflect non-critical packets	Reduce latency and energy and combat aging
This work	Mesh	Distributed	Distributed monitoring using data packets to transport traffic rate and channels' occupation status	Congestion avoidance and traffic balancing to reduce message latency

enabling to find some of the non-congested NoC regions. A smaller number of packets simultaneously seeking a new path ensure scalability. It is important to mention that in real MPSoCs tasks are mapped closer to each other. Therefore, searches for new paths occur in different NoC regions, in this way reducing NoC resource sharing. It is also possible to replace this static approach by a dynamic computation of a new path when adaptation is required. This work does not implement this last option to avoid the computational cost involved in calculating new paths at runtime.

3.1. Router architecture and virtual channels

Fig. 3 illustrates a 2D mesh MoNoC router with five input ports and five output ports, interconnected through a crossbar switch.

The input port is responsible for receiving incoming packets, obtaining the next direction the packet is to take, and for requesting its transmission to the selected output port. Remember that due to the use of source routing no decision is needed at the router, at least concerning routing. Since arbitration is distributed, there is

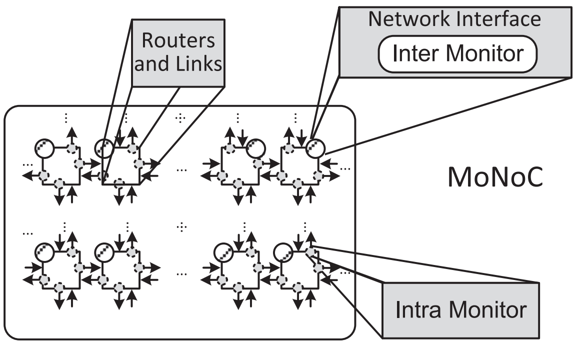


Fig. 1. Main components of MoNoC over a mesh topology: (i) Routers and Links; (ii) Intra Monitors; (iii) Inter Monitors; (iv) Network Interfaces.

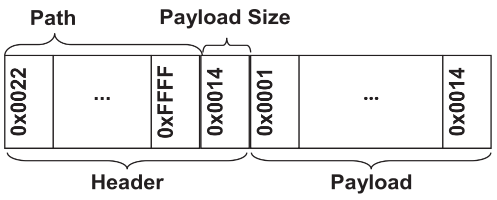


Fig. 2. Packet structure adopted in the present work.

one arbiter at each output port. This arbiter selects the incoming port to be treated, using a FIFO that supports preemption by higher priority packets. Preemption enables to interrupt a low priority flow when a control packet requests a busy port.

Each MoNoC channel employs two virtual channels: (*i*) a ***data lane***, responsible for transmitting data packets; (*ii*) a ***control lane***, responsible for transporting control packets. By definition, the control lane has priority higher than the data lane. However, the priority of virtual channels also depends on the *Granter* negotiation with the *Regulator* of the connected router, refer to Fig. 3. If the resources used on the neighbor input port are busy, the virtual channel priority is temporarily changed. Additionally, the control lane serializes the requests of monitoring services, with the goal of ensuring their arrival order.

Other techniques may replace virtual channels, such as multiple physical channels or channel slicing, which consists in dividing a wide data channel into multiple, narrower data channels. The goal here is to provide the communication infrastructure with a mechanism to implement differentiated services as that required by MoNoC, which consists in monitoring and adaptation.

3.2. Intra Monitor (IAM)

The IAM is a nonintrusive circuit, responsible for monitoring the use of each output port, and to transmit this information when requested by a control packet. The IAM module, detailed in Fig. 4 is a piece of hardware connected to the output port.

At each clock cycle, the IAM reads the *observation interface* signals that correspond to the signals responsible for flow control between routers. According to these, the IAM updates its internal state. Fig. 5 depicts the finite state machine (FSM) used internally in the IAM.

The FSM used in the IAM starts in the ***Free*** state and remains in it as long as there is no packet traversing the output port. The ***Transmitting*** state is reached when a flit is being transferred to a neighbor router (tx ↑ and credit_in ↑). The ***Stalled*** state is reached when there is at least one flit to be transmitted, but there is no buffer position available to receive this flit at the neighbor router (tx ↑ and credit_in ↓).

The structures used to compute the transmitted rate are:

i. OTS, or *Observation Time Slice*, a parameterizable monitoring window.
ii. OVS, or *Observed Value Structure*, which stores the number of clock cycles that the interface remains at each state.
iii. CVS, or *Current Value Structure* - at the end of each OTS, the OVS values are transferred to the CVS.
iv. AVS, or *Average Value Structure*, that corresponds to a weighted average of the CVS values.

Fig. 6 shows a synthetic example of transmission state transitions using these structures. The value of OTS is fixed at 1000 clock cycles. We arbitrarily attributed Free, Transmitting, and Stalled states during each one of the three example OTSs. OVS contains counters, which are transferred to CVS and initialized at the end of the OTS period. As illustrated in Fig. 6, the AVS is a weighted average, since the most recent values have a higher weight compared to older ones.

The second IAM interface, *operation interface*, receives and transmits control packets. The reception of control packets may (re)define IAM parameters, such as OTS, for example. The

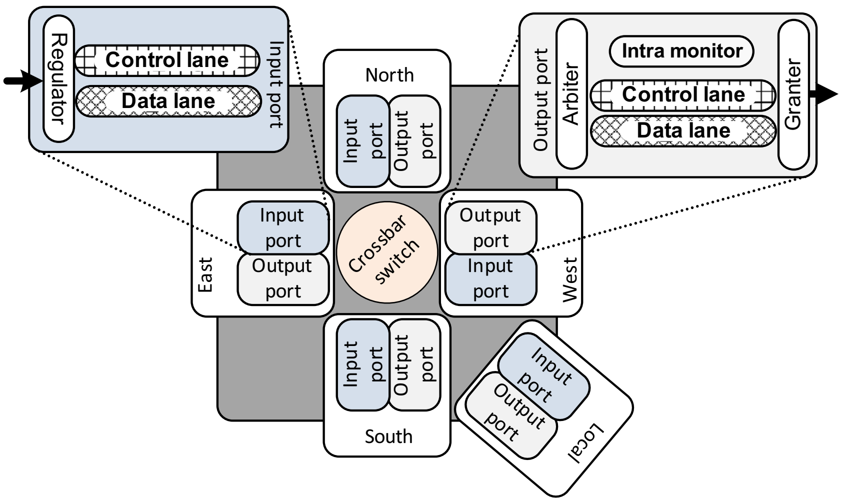


Fig. 3. Example of a 2D mesh router that uses the MoNoC architecture on each input/output port.

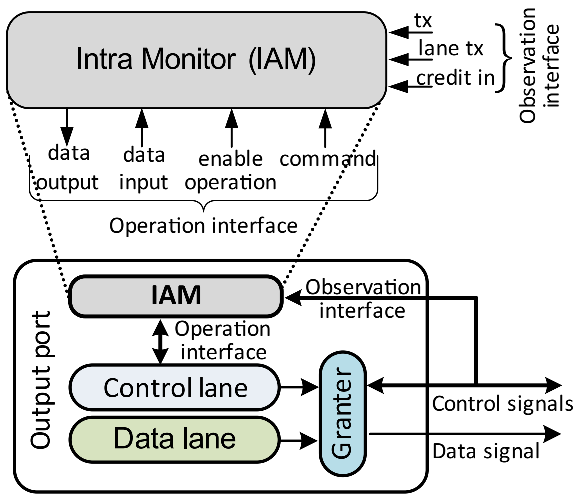


Fig. 4. Block diagram and signals of an Intra Monitor of MoNoC, located at each router's output port.

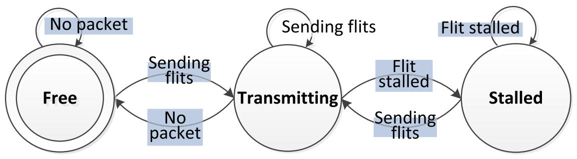


Fig. 5. FSM used by the IAM.

transmission of control packets injects into the NoC values stored in the CVS and AVS structures.

3.3. Network Interface (NI)

The NI is responsible for the communication between the NoC and a PE, which is usually a processor or a memory bank. The NI provides an abstraction for the communication protocol used by a communicating pair of PEs.

Fig. 7 presents the internal NI modules:

i. ***NISender*** and ***NIReceiver*** are modules in charge of transmitting packets from the PE to the NoC and vice versa.
ii. End-to-end monitoring modules: ***mstMonitor***, ***mstProbe***, ***slvMonitor*** and ***slvProbe***. These modules correspond to the parts of the Inter Monitor and their interconnection appears in Fig. 1.

The IRM (*Inter Monitor*) is also a nonintrusive circuit, implemented inside the NI. The IRM executes end-to-end transmission rate monitoring between communicating pairs. This rate is computed based on a parameterizable *Monitored Time Slice* (MTS), which can differ from OTS. This module is employed only when a monitored communication is required. Otherwise, the IRM remains idle. When a monitored communication is established, the master is defined as the source PE, and the slave is the target PE. The master side employs modules ***mstMonitor*** and ***mstProbe***, while the slave side utilizes modules ***slvMonitor*** and ***slvProbe.***

The main difference between IRM and IAM is that the later can dispatch notifications about undesired situations rather than just observe the traffic rate.

The ***mstMonitor*** receives from the master PE the requested transmission rate, selects the path to the slave PE, and reserves the slave monitor (*slvMonitor*) at the slave PE. The master PE configures the ***mstMonitor*** through a *configuration packet* with parameters such as MTS and the amount of data in *flits* to be transmitted during a given MTS. These parameters define a transmission rate contract, detailed in Section 4. Once a contract is defined, the PE can start injecting data packets in the NoC. The ***mstProbe*** monitors the transmitted data with a dedicated communication link to the ***mstMonitor***.

The ***slvMonitor*** is responsible for monitoring the contract, notifying the ***mstMonitor*** when the contract agreement is violated. The ***mstMonitor*** transmit packets to configure the ***slvMonitor***. The ***slvProbe*** is a module used as support by the ***slvMonitor*** for contract monitoring. It observes the received data. Both, ***mstProbe*** and ***slvProbe*** modules contain structures similar to those of IAMs, using the MTS as temporal reference for the monitoring window.

It is assumed in the present work that a given PE may execute only one monitored task, which corresponds to a task with guaranteed services. All other tasks executing in the same PE are Best Effort (BE) tasks. If more than one task per PE should be monitored, additional monitors should be implemented in the NI. This restriction avoids inter-task interferences in a system with QoS constraints [34].

4. Runtime path adaptation

Distinct communicating pairs may use multiple shared paths, overloading communication resources and producing congestion. To overcome such scenarios, designers normally employ two strategies: (i) consider the traffic load of the entire network, to compute an optimal distribution for every path, an action frequently carried out at design time, since it is time-consuming; (ii) evaluate the current network status at runtime to take routing decisions.

Runtime traffic evaluation implies the adoption of adaptive routing algorithms and some level of knowledge about network usage. Most partially adaptive routing algorithms [35] use the congestion status of one or two hops around the current packet position. This partial reach of network usage can be inefficient, since a

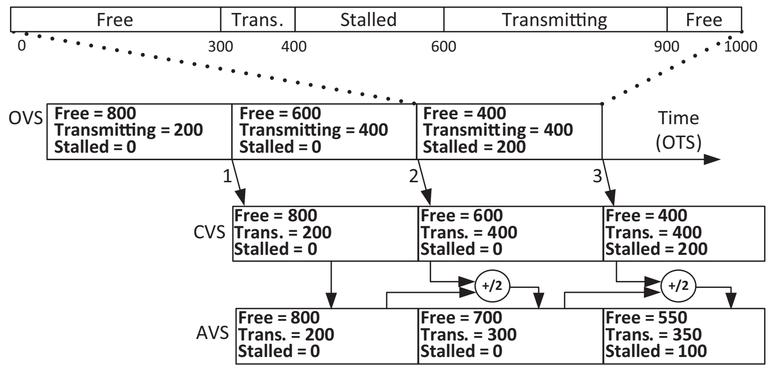


Fig. 6. Example of the monitoring structure computation operation, used to define transmission rates along an arbitrary set of state transitions.

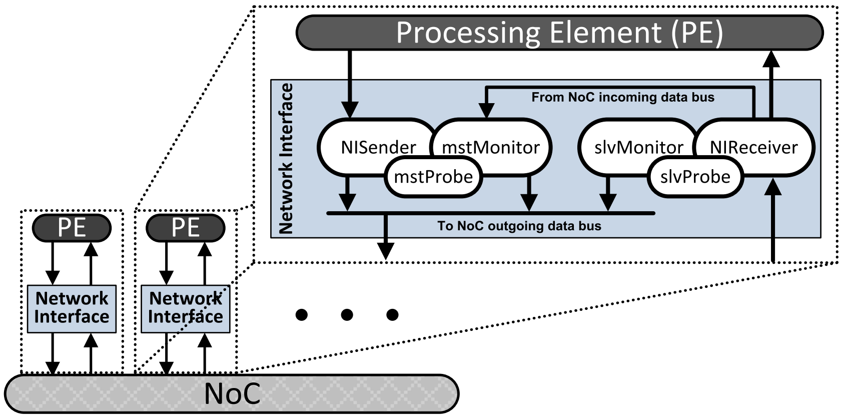


Fig. 7. Network interface internal architecture with emphasis on the Inter Monitor structures.

given packet may take a locally uncongested path just to fall in a congested region some hops ahead. In addition, adaptive routing algorithms may induce out-of-order packet reception, which may require special mechanisms at the PE level. MoNoC merges both design and runtime strategies. As Section III mentioned, for each communicating pair, the source PE computes at design time a set of possible paths to reach the target PE. At runtime, whenever the requested transmission rate is violated, the NoC tries to adapt the entire path of this pair, selecting the less congested one among the pre-computed ones.

In MoNoC, a *contract* refers to a transmission rate required by a communicating pair. To keep a contract, MoNoC provides mechanisms to permanently observe the communication data flow, as well as the traffic load of each flow. It offers adaptive and non-adaptive communication approaches. When a non-adaptive communication is requested, no contract is agreed (BE behavior). On the other hand, when an adaptive communication is requested, a contract must be established for the communicating pair, and the transmission rate is monitored to provide adaptability, if needed. Fig. 8 illustrates a message chart diagram containing the set of messages applied during the session opening stage of an adaptive communication.

To fulfill the agreed service, the source PE requests a session opening to the ***NISender***. Then, the source PE transmits the following data: (i) target PE address; (ii) MTS; (iii) rate to transmit data during a given MTS, i. e. the *Agreed Contract* (AC) rate; (iv) set of alternative paths. The ***NISender*** retransmits these data to the ***mstMonitor***. The ***mstMonitor*** executes three actions:

- It sends a notification to the ***NISender*** about the session opening, and the ***NISender*** then asks the PE to block access by the requesting task to the NoC.
- Dispatches a setup packet to the ***NIReceiver*** containing the parameters necessary to path adaptation.
- Enables the monitoring on the source side, releasing the source PE.

The ***NIReceiver*** requests the monitoring of the transmission rate by the ***slvMonitor***. After the reception of the setup packet by the ***slvMonitor***, the session is opened. Note that session opening actions at Sender and Receiver sides are thus asynchronous. Then, every packet exchanged between source–target PEs is monitored at both sides, to maintain the requested rate. By default, the first path used in the communication corresponds to an XY path. The *contract* can only be released by the source PE. When required, it transmits a setup packet, releasing IRM in the target NI.

4.1. Path adaptation protocol

Adaptation of a communication path takes place if a contract violation occurs. The ***slvMonitor*** checks the transmission rate by counting the number of received flits after each MTS. The target NI notifies the source NI, in case of non-compliance. The source NI is responsible for firing the path adaptation procedure. Fig. 9 presents the path adaptation protocol. Note that this protocol is transparent to PEs. As will be presented in Section IV.B, PEs may not access the NoC during some clock cycles, which can slightly affect PE performance.

The values used in the path adaptation protocol are the following:

i. AC, the Agreed Contract rate.
ii. CRR, the Current Reception Rate, obtained from the CVS structure of the ***slvProbe***.
iii. AIR, the Average Injection Rate, obtained from the AVS structure of the ***mstProbe***.

Once a contract is established, the target NI starts the monitoring processes. At the end of an MTS period, the current CRR is verified against the AC. If a violation is detected ($CRR < AC$) the ***slvMonitor*** stops the monitoring process and notifies the violation to the ***mstMonitor*** (noted as Event 1 in Fig. 9). When the source NI receives the notification, the AIR is verified against AC. If the average injection rate (AIR) is smaller than the required rate (AC), this means that the source PE is injecting packets at lower rates (marked as Event 2 in Fig. 9). In this case, no adaptation is required, and the source NI notifies the target NI that the observed violation does not represent a real violation, and that the monitoring processes at the target PE can continue. On the other hand, if the average rate at the source PE is being respected, but at the target PE a violation is detected, this means that the path is congested (marked as Event 3 in Fig. 9). This situation fires the path adaptation process, the source NI stops the ***mstMonitor*** and transmits a set of control packets to the target PE, using the set of available paths (marked as Event 4 in Fig. 9). The IAMs (Intra Monitors) are effectively used at this step. They compute the resource utilization of each pre-computed path. Control packets read the average link use (AVS structure), since they abstract the history of the link usage.

Fig. 10 presents an example of the *path selection* for the adaptation purposes. Is this example two paths are available between the source and target PEs. Each control packet reads the AVS structure of each IAM in the path, computing the average and maximum

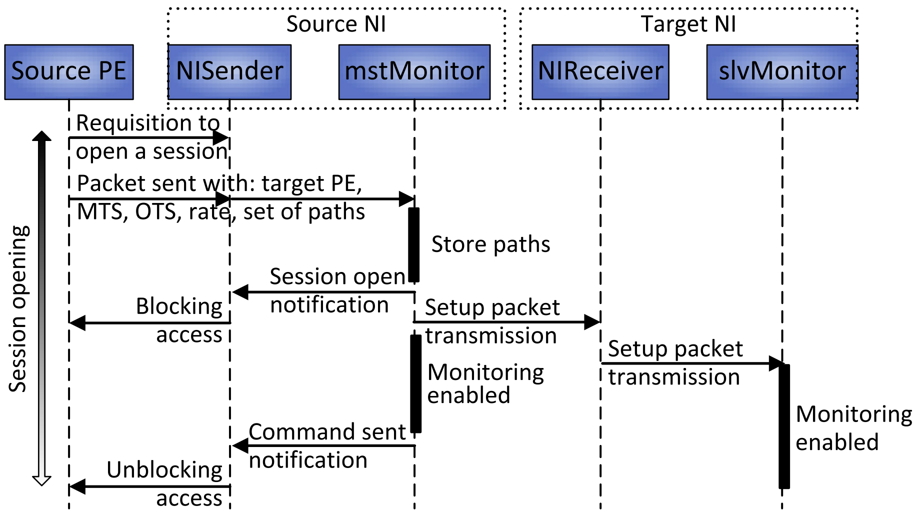

Source NI
Target NI
Source PE
NISender
mstMonitor
NIReceiver
slvMonitor
Session opening
Requisition to open a session
Packet sent with: target PE, MTS, OTS, rate, set of paths
Store paths
Session open notification
Blocking access
Setup packet transmission
Setup packet transmission
Monitoring enabled
Monitoring enabled
Command sent notification
Unblocking access

Fig. 8. Message chart diagram for session opening in MoNoC.

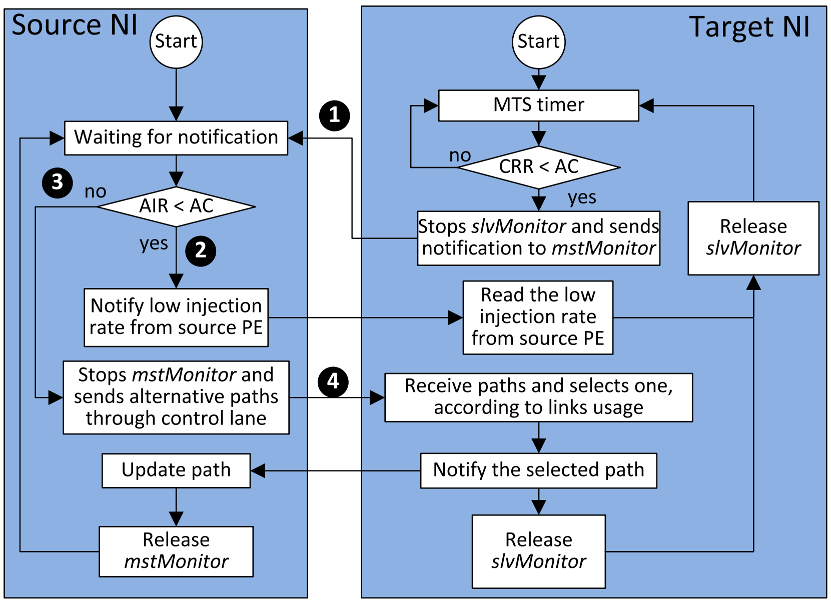

Source NI
Start
Waiting for notification
3
no
AIR < AC
yes
2
Notify low injection rate from source PE
Stops mstMonitor and sends alternative paths through control lane
Update path
Release mstMonitor
1
4
Target NI
Start
MTS timer
no
CRR < AC
yes
Stops slvMonitor and sends notification to mstMonitor
Release slvMonitor
Read the low injection rate from source PE
Receive paths and selects one, according to links usage
Notify the selected path
Release slvMonitor

Fig. 9. The path adaptation protocol as seen from both source and target NIs.

path rate. After receiving all proposed paths, the ***slvMonitor*** selects the one with the lowest average rate. Remember the number of paths to be received is defined during the setup process, as discussed when presenting Fig. 8. In case more than one path presents the same average rate, the one with the lowest peak is selected. In this example, the selected path is Route A. After the path selection, the ***slvMonitor*** notifies the source PE the new path, and monitoring is re-enabled at both sides. In addition, after path updating, all subsequent data packets are transmitted using this new path.

The control traffic has been designed to provide very low communication overhead. Control packets are sent only when adaptation is required, and typically contain just 5 flits. As previously mentioned, the number of paths to explore is small, ensuring few added traffic due to the path selection information.

4.2. Path adaptation cost

Fig. 11 illustrates an example of the path adaptation process and its cost, in clock cycles. Values were obtained during a MoNoC RTL simulation running a synthetic traffic scenario. The simulation evaluates the three main steps of the adaptation process: (i) session opening; (ii) path adaptation and (iii) session closing.

Session opening takes 78 clock cycles, from the moment when the source PE requests to open a session, until the moment when monitoring is enabled at the target NI (details are available in Fig. 8 and the accompanying discussion).

The path adaptation phase starts when the target NI detects a reception rate smaller than the agreed rate, i.e. CCR < AC. (which occurs at time: 2100 cycles). The source NI receives the notification (at time 2546 cycles), and verifies that AIR < AC, characterizing a congestion in the path (marked as Event 3 in Fig. 9). In this example, four paths are available for path adaptation. The source NI sends 4 control packets, using the four different paths. The target NI receives the fourth control packets at time 2674 cycles, and selects the third path as the one reaching the target NI with smaller contention. The source NI receives the notification of the selected path at time 2758 cycles. From this moment on, all data packets are transmitted using this new path. Finally, at time 7000 cycles

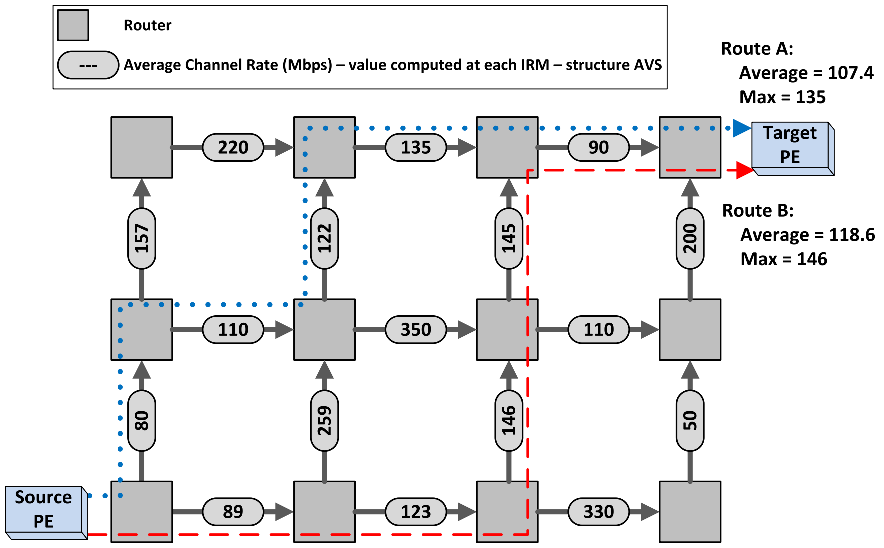
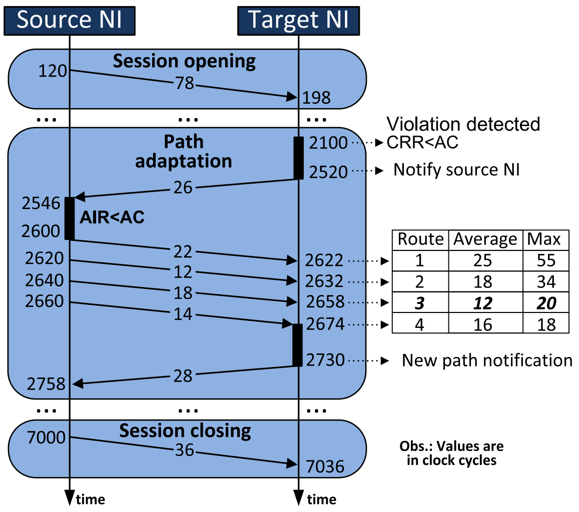


Fig. 10. Path search example. Path (A) presents the lowest peak and average rate of channels occupancy.



Route	Average	Max
1	25	55
2	18	34
3	***12***	***20***
4	16	18

Fig. 11. An example of path adaptation and the computation behind it, measured in clock cycles. The numbers along the vertical lines represent the number of cycles taken from the start of the simulation run.

the source NI sends a notification message to close the session. The process to close the session takes 36 clock cycles.

The source PE cannot communicate with the target PE from the reception of the path adaption request (at time 2546 cycles) up to the path update moment (at time 2758 cycles). This represents only 212 clock cycles. This is a remarkable result, and is a consequence of using a hardware-only implementation of the path adaptation process. From the point of view of the PE, the impact on the performance of a given application is insignificant, since a few hundred of clock cycles represents just the equivalent of a small amount of assembly instructions executed on a processor that often is the core of the PE.

5. Results

This Section evaluates the area cost of MoNoC and the performance of the path adaptation process, using as metric the latency of applications. Section 5.1 evaluates the area consumption overhead of the MoNoC monitoring and adaptation mechanisms, comparing it with the Hermes SR NoC, the baseline NoC used for this MoNoC implementation. The Hermes SR NoC is identical to the MoNoC, except for the monitoring and adaptation mechanisms. Section 5.2 presents the experimental setup that serves to evaluate the latencies associated with the MoNoC operation. Section 5.3 defines the relevant latency metrics, while Sections 5.4 and 5.5 respectively evaluate the application and network latencies for both static and dynamic traffic scenarios.

5.1. MoNoC area consumption

With the goal of evaluating the area and timing overhead of the monitoring and adaptation mechanisms of MoNoC, a 5-port router was synthesized targeting a Virtex 5 device, with 16-bit flit size and buffer depth equal to 4 flits. This follows the Hermes-SR implementation [33]. The main differences of both implementations are the virtual channels (VCs), the Intra Monitors (IAMs), and the algorithms to implement monitoring and adaptation mechanisms. Table 2 presents the area consumption results in terms of Look-Up Tables (LUTs) and Flip-Flops (FFs).

The area of MoNoC's router is large, mostly due to the virtual channels, which take near 70% of the total MoNoC's router area. In contrast, the IAM and the mechanisms for monitoring and adaptation take only 30% of the total area of the MoNoC's router. Compared to the Hermes-SR router, the area overhead is large (around 3.5 times larger). However, this is justified by the hardware implementation of monitoring and adaptation mechanisms that enable very fast detection of contract agreement violation and very fast path adaptation. Area consumption optimization is an ongoing work not explored in this article, since the goal here is to demonstrate a new method to avoid congestion and its trade-offs.

5.2. Setup of experimental results

Fig. 12 shows the experimental setup used to run 216 distinct simulations scenarios. Each scenario is a combination of NoC adopted (5×5) and planned traffic scenarios. The latter arise from the combination of a communicating pair and a disturbing traffic. For the disturbing traffic, two axes were varied: (i) the injection rate of the disturbing sources, and (ii) the physical usage of the communication channels. For the injection rate, from very low to high injection rates were employed, to evaluate the adaptation capability of the architecture. Additionally, definition of the traffic

Table 2
Area consumption and overhead of a MoNoC router versus the Hermes-SR router, both with 5 input ports, 16-bit flits and 4-flit buffer depth.

Hardware	Hermes-SR	MoNoC					
		Total	IAM		VCs and input port buffer		Algorithm and additional logic
			Per output port	All output ports	Per port	All ports	
LUTs	1080	3810	135	675	272	2720	415
FFs	700	1310	66	330	94	940	40

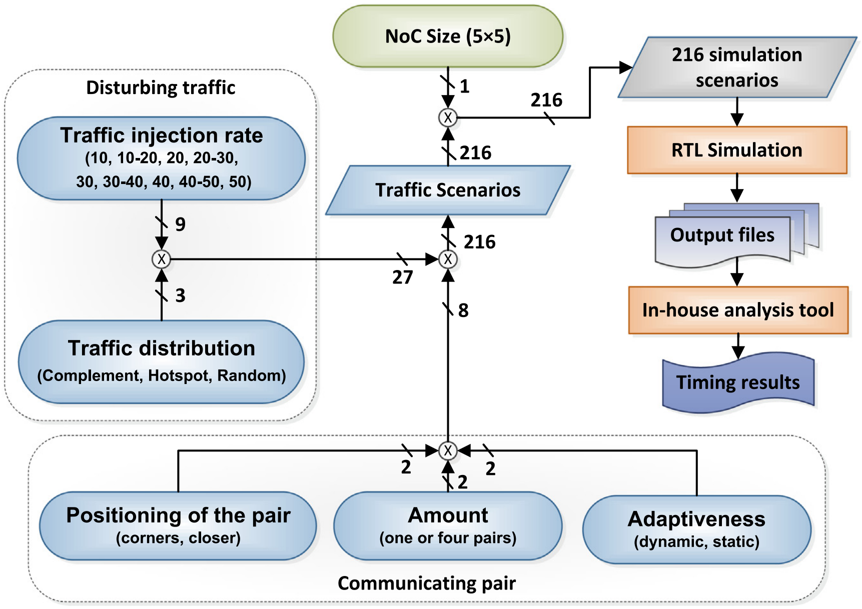

NoC Size (5×5)
1
216
216 simulation scenarios
RTL Simulation
Output files
In-house analysis tool
Timing results
Disturbing traffic
Traffic injection rate (10, 10-20, 20, 20-30, 30, 30-40, 40, 40-50, 50)
9
3
Traffic distribution (Complement, Hotspot, Random)
216
Traffic Scenarios
216
27
8
2
2
2
Positioning of the pair (corners, closer)
Amount (one or four pairs)
Adaptiveness (dynamic, static)
Communicating pair

Fig. 12. Setup of experimental results.

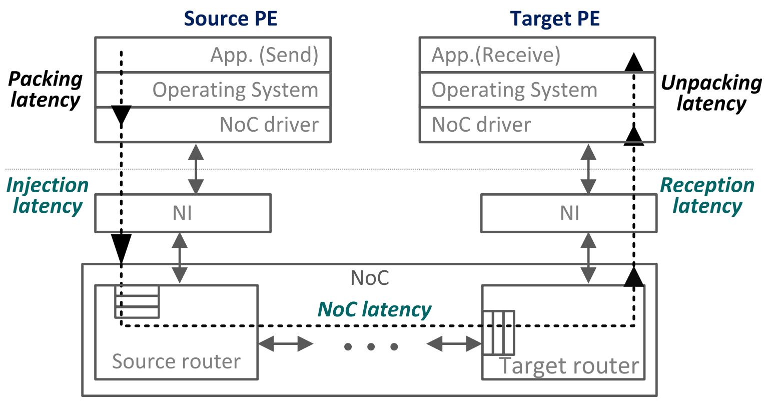


Fig. 13. The five different components that define the latency of a packet in MoNoC.

distribution occurs as a way to guarantee the occupation of a specific channel (i.e. complement and hotspot) or a dynamic occupation of the channels. Finally, the communicating pair depicts the source and sink configurations elaborated to guarantee a better comparison. In this axis, proposed variations were: (i) the position of the source and the sink in MoNoC, (ii) the amount of source and sink pairs that coexist at the NoC, and (iii) if or not the observed values were for a pair that could adapt the communication path. While MoNoC was completely described in VHDL, the evaluation resources employ SystemC code. All scenarios were simulated with a clock cycle accurate simulator, Modelsim. This produced a set of results stored into output files during simulation. Next, an in-house tool allows parsing the corresponding files and extracting timing results. The next sections present and discuss the most relevant results.

5.3. Latency analysis

Fig. 13 shows the five distinct parts that compose the packet's latency in MoNoC:

i. *Packing latency*, the time required to generate a packet, by concatenating control information (e.g. header and/or tail) and to transmit the packet through software levels.
ii. *Injection latency*, the time required to insert the packet onto the source router through the network interface.

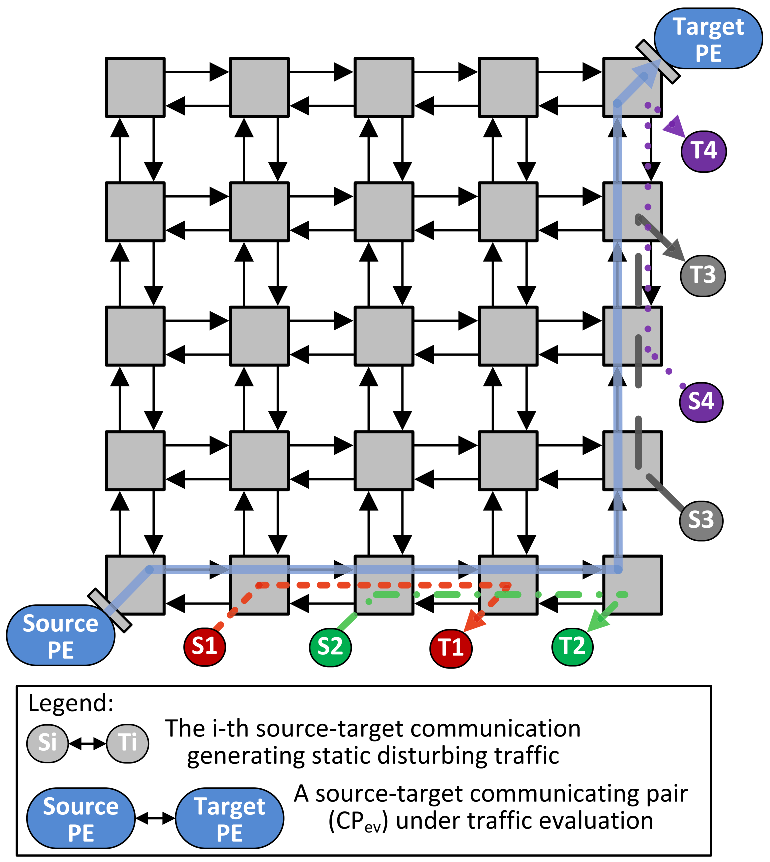

Target PE
T4
T3
S4
S3
Source PE
S1
S2
T1
T2
Legend:
Si
Ti
The i-th source-target communication generating static disturbing traffic
Source PE
Target PE
A source-target communicating pair (CP$_{ev}$) under traffic evaluation

Fig. 14. Experiment with four disturbing flows concurring for the same path of a CP$_{ev}$.

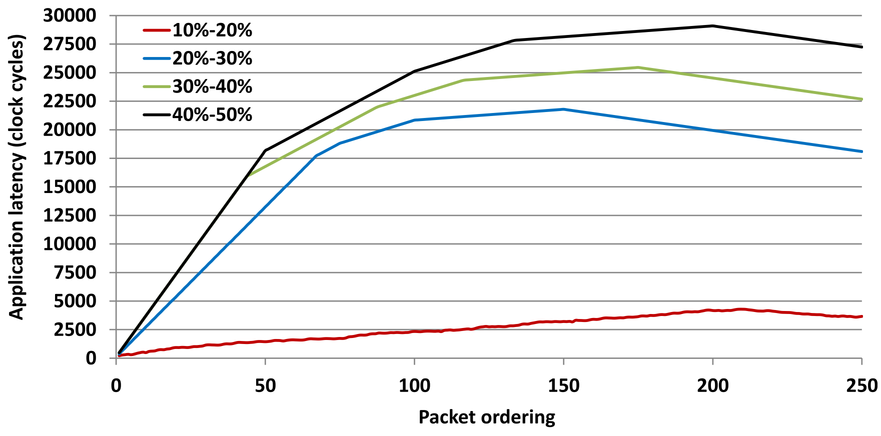


Fig. 15. Application latencies obtained with a static traffic scenario and without path adaptation.

iii. *NoC latency*, time spent by the packet to traverse the NoC until reaching the output port of the target router. Packets competing for NoC resources (e.g. links, buffers and arbitration) affect this value directly.
iv. *Reception latency*, time required for the target PE to receive the packet from the NI.
v. *Unpacking latency*, time required to extract the payload and transfer it to the application.

The *packing* and *unpacking* latencies depend on the behavior of the operating system, the memory hierarchy, and the type and speed of the processor. Since this work focuses on the ability to modify the path at runtime, thus reducing the packet latency, only *injection*, *NoC* and *reception* latencies are considered for latency evaluation. The sum of these three latencies is referred here as the *application latency*.

5.4. Results for static traffic scenarios

Our static traffic scenario contains a set of flows that send packets during the entire simulation. A *disturbing* traffic is a flow that interferes in the path of the communicating pair under evaluation (CP$_{ev}$). Fig. 14 illustrates the experiment discussed in this section with four disturbing flows applied along the path of the CP$_{ev}$.

The first experiment contains no path adaptation, and all of its flows use the XY routing algorithm. Fig. 15 presents the *application*

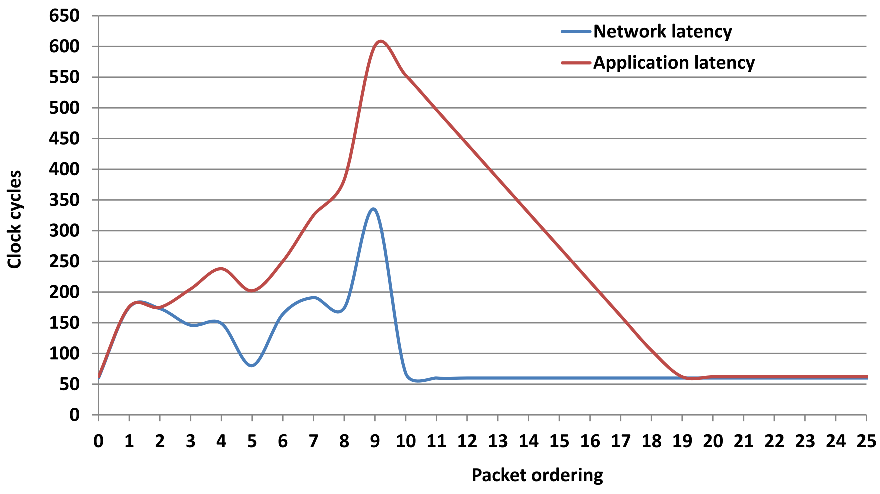


Fig. 16. Network and application latencies obtained with a static traffic scenario and path adaptation. The packet ordering axis shows the behavior observed for the first 25 packets of the application ordered according to their transmission start time.

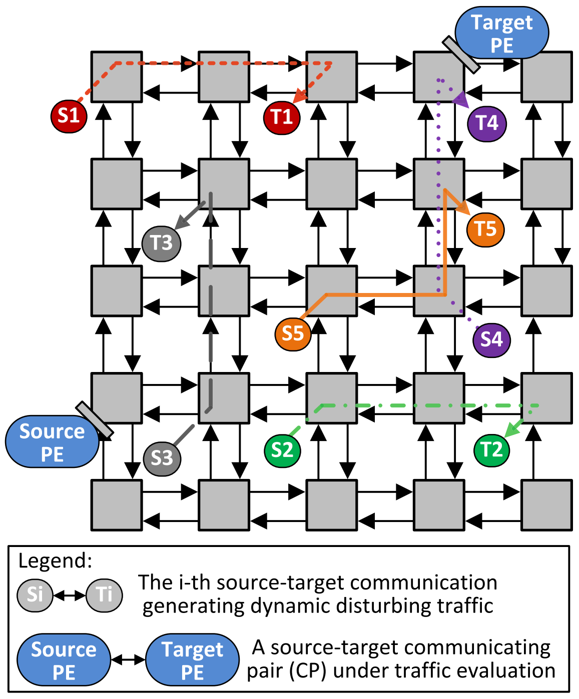

Target PE
Source PE
S1
T1
T4
T3
T5
S5
S4
S3
S2
T2
Legend:
Si
Ti
The i-th source-target communication generating dynamic disturbing traffic
Source PE
Target PE
A source-target communicating pair (CP) under traffic evaluation

Fig. 17. Experiment with dynamic interference traffic concurring with the communication path of the CP_{ev}.

latency of the CP_{ev} for a sequence of packets. In all, the simulation shows the transmission of 250 packets over the NoC. Packets are numbered in ascending order of their injection start time. This accounts for the *X*-axis in the plot of Fig. 15, which reproduces the latency seen by the Slave PE as it receives packets from the Master PE of the CP_{ev}. Each curve corresponds to one simulation with a given injection rate of disturbing flows, according to the channel transmission capacity (i.e. from 10% to 20%; from 20% to 30%; from 30% to 40%; and from 40% to 50%). Even when the disturbing traffic acts with a small injection rate (from 10% to 20%), congestion leads to application latencies up to 4000 clock cycles. For a disturbing traffic with higher injection rates, latencies can end up in more than 20,000 clock cycles.

The second experiment shows the effect of using path adaptation in the CP_{ev}. Fig. 16 details the network and application latencies achieved for a disturbing traffic with an injection rate from 10% to 20%. Other curves are omitted, because these resulted in very similar behavior. The figure shows the first 25 packets of the synthetic application, which are again ordered according to their transmission start time. The disturbing traffic is active during the whole simulation.

The effect in the latency was expected, since any packet sent by the CP_{ev} that does not use the initial XY routing finds non-congested links. Relevant data to analyze include:

- *The Peak Latency* – Defines the biggest latency a packet suffers inside the network, specifically due to the adaptation process. The adaptation process sends packets with higher priority in the network, causing this peak, and induces a momentary increase in contention for some data packets. According to the discussion of Section IV.B, the network latency increases less than 300 clock cycles. The impact on the application latency is naturally higher, due to the packet buffering process at the NI.
- *The Reaction Time* – defines how long it takes to restore the latency to its minimum or close to minimum values, when the cause of the congestion disappears. In this simple experiment we associate the notion of time to the number of transmitted packets. Here, once congestion is detected, after less than 10 packets network latencies restore its reference values, and after 19 packets the network latency achieves near-minimum values.

This first experiment demonstrates the effectiveness of the path adaptation approach. It changes the path in a short time, restoring latencies to their minimal values.

5.5. Results for dynamic traffic scenarios

These experiments apply the MoNoC path adaptation when traffic flow changes at runtime. Fig. 17 shows the scenario used in these experiments, which employs a CP_{ev} and several disturbing flows, varying: (i) the transmission rate of the disturbing traffic, (ii) the amount of transmitted packets.

Fig. 18 presents the application latency with and without adaptation, using a disturbing traffic with an injection rate from 20% to 30% of the channel transmission bandwidth. Note that this is a typical injection rate that makes 2D mesh networks saturate [36]. The disturbing flows start after the 100th packet transmitted

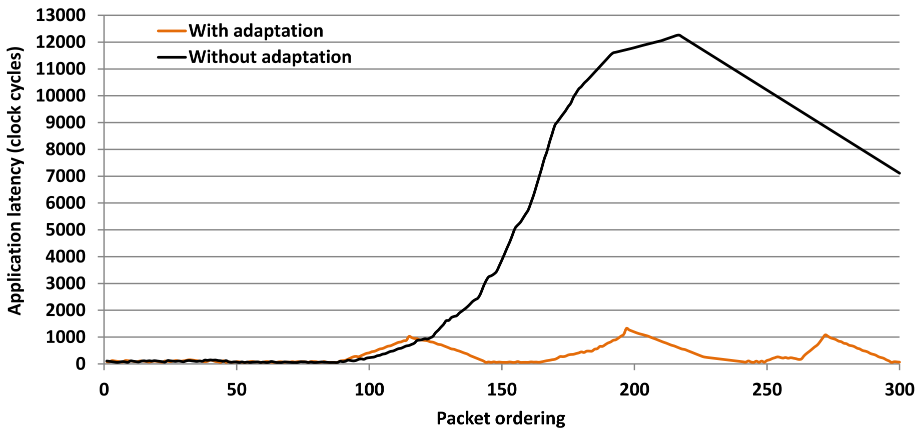


Fig. 18. Application and NoC latencies results for a dynamic traffic scenario. The disturbing traffic changes every time the Source PE sends one hundred packets to the Target PE. The disturbing traffic has an injection rate that varies from 20% to 30% of the channel transmission capacity. Packets are ordered according to their transmission start time.

by the source PE. At the target NI, the MTS was defined to send notification packets after receiving 50 packets.

Without adaptation, the application latency grows to up to 12,000 clock cycles. When applying path adaptation, we can observe three peaks in the application latency, corresponding to the moments when path adaptation is executed. The highest application latency observed was 1300 clock cycles.

This second scenario demonstrates the benefits of the hardware implementation, which leads to very small reaction times. This enables to restore the small latencies even with disturbing flows appear very close to each other in time.

It was observed that the use of the proposed path adaptation mechanism leads to better results for higher injection rates in the disturbing traffic. There is another experiment not depicted on the plots discussed in this section. It demonstrated that if the MTS parameter is below 30 received packets, a large number of path adaptations occur, reducing the overall performance of the approach. This last experiment displays a tradeoff that NoC designers can explore, and which depends on the application, on the disturbing traffic characteristics as well as on the NoC topology.

6. Conclusions

This paper presented a mechanism for changing paths a NoC uses at runtime to reduce congestion. The mechanism relies on traffic monitoring schemes and protocols. The runtime adaptive routing mechanism was implemented inside MoNoC – a NoC aggregating distributed traffic monitoring and deemed to obtain QoS without recourse to inflexible schemes like circuit switching. MoNoC uses virtual channels to distribute the communication network status, with high priority control packets, but with low communication volume and only in specific moments, that do not compromise bandwidth or NoC latency.

At the application level, a developer can define a required rate, and a set of possible paths between each communicating pair. All adaptation executed at lower levels, at the network interfaces and at the NoC. Therefore, there is neither a performance impact at the operating system level nor at the application level. This approach is an alternative to methods requiring software monitoring, and results in a short reaction times.

Experiments explored static and dynamic traffic scenarios. In both scenario types, MoNoC could ensure a change to optimal routes under the point of view of minimum latency, as soon as it detected a violation in the agreed/contracted rate. Typically, after few packets the requested minimum latency is restored.

This approach can be extended to include considerations related to fault tolerance and dynamic path computation. The target PE can receive a corrupted or incomplete packet. In this case, a request for new paths may be sent to the source PE. The present implementation targets static mapping scenarios. However, in a dynamic workload scenario, when path adaptation is requested, the operating system can compute new paths at runtime.

Acknowledgment

The authors acknowledge the support of research agencies grants as follows. CNPq under grants 472126/2013-0, 302625/2012-7, 310864/2011-9, 552699/2011-0, 141247/2005-3; CAPES under grant 708/11; FAPERGS under grants 11/1445-0, 12/1777-4 and Docfix SPI 2843-25.51/12-3. This article is an extension of a paper presented at the ISCAS 2014 Conference.

References

[1] J. Henkel, Closing the SoC design gap, Computer 36 (9 (September)) (2003) 119–121.
[2] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, Computer 35 (1 (January)) (2002) 70–78.
[3] C. Wang, W.-H. Hu, N. Bagherzadeh, Congestion-aware network-on-chip router architecture, in: Proceeding of the International Symposium on Computer Architecture and Digital Systems (CADS), pp. 137–144, 2010.
[4] A.-M. Rahmani, K. Vaddina, K. Latif, P. Liljeberg, J. Plosila, H. Tenhunen, High-performance and fault–tolerant 3D NoC-bus hybrid architecture using ARB-NET based adaptive monitoring platform, IEEE Trans. Comput. 63 (3 (March)) (2014) 734–747.
[5] D. Matos, C. Concatto, A. Kologeski, L. Carro, F. Kastensmidt, A. Susin, M. Kreutz, Monitor–adapter coupling for NoC performance tuning, in: International Conference on Embedded Computer Systems (SAMOS), pp. 193–199, 2010.
[6] R. Radetzki, Fault–tolerant differential Q routing in arbitrary NoC topologies, in: IFIP International Conference on Embedded and Ubiquitous Computing (EUC), pp. 33–40, 2011.
[7] J. Zhao, S. Madduri, R. Vadlamani, W. Burleson, R. Tessier, A dedicated monitoring infrastructure for multicore processors, IEEE Trans. VLSI Syst. 19 (6 (June)) (2011) 1011–1022.
[8] M. Yazdi, M. Modarressi, H. Sarbazi-Azad, a load-balanced routing scheme for NoC-based systems-on-chip, in: Workshop on Hardware and Software Implementation and Control of Distributed MEMS, pp. 72–77, 2010.
[9] R. Manevich, I. Cidon, A. Kolodny, I. Walter, S. Wimer, A cost effective centralized adaptive routing for networks-on-chip, in: Euromicro Conference on Digital System Design (DSD), pp. 39–46, 2011.
[10] G. Kornaros, D. Pnevmatikatos, Real-time monitoring of multicore SoCs through specialized hardware agents on NoC network interfaces, in: International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 248–255, 2012.
[11] P. Gorski, D. Timmermann, Centralized traffic monitoring for online-resizable clusters in networks-on-chip, in: International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–8, 2013.
[12] Y. Huang, K. Chou, C.-T. King, S.-Y. Tseng, NTPT: On the end-to-end traffic prediction in the on-chip networks, in: Design Automation Conference (DAC), pp. 449–452, 2010.
[13] L. Guang, E. Nigussie, J. Plosila, J. Isoaho, H. Tenhunen, Coarse and fine-grained monitoring and reconfiguration for energy-efficient NoCs, in: International Symposium on System on Chip (SoC), pp. 1–7, 2012.

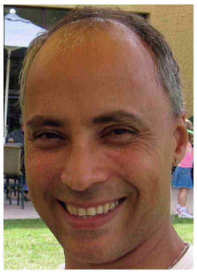
[14] A. Yin, L. Guang, P. Liljeberg, P. Rantala, E. Nigussie, J. Isoaho, H. Tenhunen, Hierarchical agent based NoC with dynamic online services, in: IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 434–439, 2009.
[15] L. Guang, E. Nigussie, P. Rantala, J. Isoaho, H. Tenhunen, Hierarchical agent monitoring design approach towards self-aware parallel systems-on-chip. ACM Transactions on Embedded Computing Systems (TECS), vol. 9, no. 3, pp. 25:1–25:24, February, 2010.
[16] M. Al Faruque, T. Ebi, J. Henkel, ROAdNoC: runtime observability for an adaptive network on chip architecture, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 543–548, 2008.
[17] V. Rantala, T. Lehtonen, P. Liljeberg, J. Plosila, Distributed traffic monitoring methods for adaptive network-on-chip, in: NORCHIP, pp. 233–236, 2008.
[18] T. Ebi, M. Al Faruque, J. Henkel, NeuroNoC: neural network inspired runtime adaptation for an on-chip communication architecture, in: IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 223–230, 2010.
[19] A. Mehranzadeh, A. Khademzadeh, A. Mehran, FADyAD – fault and congestion aware routing algorithm based on DyAD algorithm, in: International Symposium on Telecommunications (IST), pp. 274–279, 2010.
[20] R. Ramanujam, B. Lin, Destination-based adaptive routing on 2D mesh networks, in: Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 1–12, 2010.
[21] L. Tedesco, T. Rosa, F. Moraes, A message-level monitoring protocol for QoS flows in NoCs, in: International Symposium on System on Chip (SoC), pp. 84–88, 2010.
[22] P. Lotfi-Kamran, A. Rahmani, M. Daneshtalab, A. Afzali-Kusha, Z. Navabi, EDXY – a low cost congestion-aware routing algorithm for network-on-chips, J. Syst. Architect. 56 (7 (July)) (2010) 256–264.
[23] J. Jose, J. Shankar, K. Mahathi, D. Kumar, M. Mutyam, BOFAR: buffer occupancy factor based adaptive router for mesh NoCs, in: International Workshop on Network on Chip Architectures (NoCArc), pp. 23–28, 2011.
[24] S. Ma, N. Jerger, Z. Wang, DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip, in: International Symposium on Computer Architecture (ISCA), pp. 413–424, 2011.
[25] T. Mak, K. Lam, P. Cheung, W. Luk, Adaptive routing in network-on-chips using a dynamic programming network, IEEE Trans. Ind. Electr. 58 (8 (August)) (2011) 3701–3716.
[26] L.-W. Wu, W.-X. Tang, Y. Hsu, A novel architecture and routing algorithm for dynamic reconfigurable network-on-chip, in: International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 177–182, 2011.
[27] M. AlFaruque, T. Ebi, J. Henkel, AdNoC: runtime adaptive network-on-chip architecture, IEEE Trans. VLSI Syst 20 (2 (February)) (2012) 257–269.
[28] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, U. Ogras, In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches, in: International Symposium on Networks on Chip (NoCS), pp. 43–50, 2012.
[29] A. Garbade, S. Weis, S. Schlingmann, B. Fechner, Impact of message based fault detectors on applications messages in a network on chip, in: Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 470–477, 2013.
[30] D. Ancajas, K. Chakraborty, S. Roy, Proactive aging management in heterogeneous NoCs through a criticality-driven routing approach, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1032–1037, 2013.
[31] C. Glass, L. Ni, The turn model for adaptive routing, J. ACM 41 (5 (September)) (1994) 874–902.
[32] S. Hu, S. Yat-sen X. Lin, A symmetric odd-even routing model in network-on-chip, in: IEEE/ACIS International Conference on Computer and Information Science (ICIS), pp. 457–462, 2012.
[33] E. Moreno, C. Marcon, N. Calazans, F. Moraes, Arbitration and routing impact on NoC design, in: IEEE International Symposium on Rapid System Prototyping (RSP), pp. 193–198, 2011.
[34] E. Carara, G. Almeida, G. Sassatelli, F. Moraes, Achieving composability in NoC-based MPSoCs through QoS management at software level, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1–6, 2011.
[35] G. Ascia, V. Catania, M. Palesi, D. Patti, Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip, IEEE Trans. Comput. 57 (6 (June)) (2008) 809–820.
[36] W. Dally, B. Towles, Principles and practice of Interconnection Networks, Elsevier, 2004. 550p.

Edson Ifarraguirre Moreno received the M.Sc. degree (2004) and Ph.D. degree (2010) in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS). He is currently an associate professor at the same University, and IC designer at CEITEC. From 2006 to 2007, he made a PHD internship at TIMA (Grenoble, France). His main research interests include Multiprocessor Systems on Chip (MPSoC), electronic system level design (ESL), and networks on chip networks (NoCs).

Thais Christina Webber dos Santos received the M.Sc. degree (2003) and Ph.D. degree (2009) in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS). She is currently member of embedded systems research group working on hw/sw design projects, performance evaluation through modeling and simulation techniques. Since 2011 she has been working on the analysis of parallel and distributed systems, modeling applications in the context of several areas such as software engineering, hardware design and wireless sensor networks.

César Augusto Missio Marcon Ph.D. is an Associate Professor at the School of Computer Science, Catholic University (PUCRS), Brazil, since 1995. He received his Ph.D. in Computer Science from Federal University of Rio Grande do Sul, Brazil, in 2005. He is the author and co-author of papers published covering a broad range of scientific topics within the disciplines of Computer Architecture and Digital Systems. His research interests are in the areas of embedded systems on the telecom domain, computer architecture, intra-chip communication architectures, partitioning and mapping application tasks, software & hardware testing, fault tolerance, parallel processing, real-time operating systems. At PUCRS, he is works at Embedded Systems Group (GSE) and at Hardware Design Assistance Group (GAPH). Since 2003 Prof. Marcon coordinated 7 research projects. Currently, he participates in 5 research projects and he works as advisor for MsC. and Ph.D. graduate students.

Fernando Gehm Moraes received the Electrical Engineering and M.Sc. degrees from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively. In 1994 he received the Ph.D. degree from the Laboratoire d´Informatique, Robotique et Microélectronique de Montpellier (LIRMM), France. He is currently at PUCRS, where he has been an Associate Professor from 1996 to 2002, and Full Professor since 2002. From 1998 to 2000 he joined the LIRMM as an Invited Professor for 3 months each year. He has authored and co-authored 22 peer refereed journal articles in the field of VLSI design, comprising the development of networks on chip and telecommunication circuits. One of these articles, "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip", is cited by more than 350 other papers. He has also authored and co-authored more than 180 conference papers on these topics. He has co-advised 3 MSc, advised 22 MsC, advised 4 PhD and co-advised 3 PhD works. His primary research interests include Microelectronics, FPGAs, reconfigurable architectures, NoCs (networks on chip) and MPSoCs (multiprocessor system on chip). SBC, SBMICRO and IEEE Senior Member.


Ney Laert Vilar Calazans holds a PhD degree in Applied Sciences, Electricity Group, from the Université Catholique de Louvain, Belgium (1993). He also holds a Bachelor degree in Electrical Engineering from the Universidade Federal do Rio Grande do Sul (1985) and a Master of Sciences degree in Computer Science from the same University (1988). He is a Full Professor at the Pontifícia Universidade Católica do Rio Grande do Sul, having worked at the PUCRS since 1986, and holds a Research Productivity scholarship (Level 1D) from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Professor Calazans published more than 150 papers in journals and conferences. He also published 6 book chapters and 1 book. He already advised 3 Ph.D. thesis and 21 MSc dissertations, as well as more than 60 undergraduate students in research and end of term works. His main research interests include non-synchronous circuits and systems, networks on chip, multiprocessor systems on chip, embedded systems and telecommunication applications.