PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
GRADUATE PROGRAM IN COMPUTER SCIENCE

IMPROVING QOS BY EMPLOYING MULTIPLE PHYSICAL NOCS ON MPSOCS

DOUGLAS ROBERTO GUARANI DA SILVA

Submitted in partial fulfillment of the requirements for the degree of Master of Computer Science at Pontifical Catholic University of Rio Grande do Sul.

ADVISOR: PROFESSOR FERNANDO GEHM MORAES, PH.D.

PORTO ALEGRE
2016

Dados Internacionais de Catalogação na Publicação (CIP)

S586i Silva, Douglas Roberto Guarani da

Improving QoS by employing multiple physical NoCs on MPSoCs / Douglas Roberto Guarani da Silva. – 2016.
87 p.

Diss. (Mestrado) – Faculdade de Informática, PUCRS.
Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Multiprocessadores. 2. Arquitetura de Computador. 3. Informática. I. Moraes, Fernando Gehm. II. Título.

CDD 23 ed. 004.35

Ramon Ely CRB 10/2165
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Improving QoS by Employing Multiple Physical NoCs on MPSoCs" apresentada por Douglas Roberto Guarani da Silva como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 03 de março de 2016 pela Comissão Examinadora:

Prof. Dr. Fernando Gehm Moraes – Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Missio Marcon – — PPGCC/PUCRS

Prof. Dr. Everton Alceu Carara - — UFSM

Prof. Dr. César Albenes Zeferino - — MCA/UNIVALI

Homologada em 22/03/2016, conforme Ata No. 005 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes — Coordenador.

ATESTADO

Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

ACKNOWLEDGEMENTS

I would like to express my appreciation to those who have made this dissertation possible.

To my advisor Professor Fernando Gehm Moraes for accepting me as his student and for all his guidance, advice, support, suggestions, comments and revisions of this work. I thank him for his trust in me and in my work.

To Professor César Marcon for all his suggestions and revisions since the research proposal until this final volume. To Professor Cesar Zeferino and Professor Everton Carara for the comments and evaluation of this dissertation.

To Marcelo Ruaro for the technical discussions and for his help in debugging the HeMPS platform.

To my colleagues and friends at GAPH and PUCRS for the friendship throughout all these years.

To my family, especially to my parents Claudio and Magali, for always encouraging and supporting me.

To the Professors at PPGCC for all transmitted knowledge and teachings.

To CNPq and Dell for financing this work.

APRIMORANDO QOS UTILIZANDO MÚLTIPLAS NOCS FÍSICAS EM MPSOCS

RESUMO

Sistemas embarcados adotam MPSoCs baseados em NoCs visto que um número grande de elementos de processamento (PEs) permitem a execução simultânea de várias aplicações, onde algumas dessas aplicações necessitam de restrições de tempo real (RT). PEs comunicam-se utilizando troca de mensagens em MPSoCs com memória distribuída. Essas mensagens podem ser classificadas como mensagens de aplicação, sendo os dados gerados pelas aplicações, e mensagens de gerência, utilizadas para garantir a operação correta da plataforma. Visto que a comunicação possui um forte impacto no desempenho da aplicação, uma preocupação importante no projeto de MPSoCs é de melhorar o desempenho da comunicação das aplicações, especialmente para aplicações RT. Dois métodos possíveis para otimizar o desempenho de comunicação incluem: (i) priorizar as mensagens das aplicações de RT sobre as mensagens geradas por aplicações de melhor esforço (do inglês, *best effort,* BE); (ii) isolar as mensagens de aplicações das mensagens de gerência, considerando que MPSoCs complexos necessitam de um grande número de serviços de gerência para satisfazer os requisitos de desempenho. Na literatura sobre NoCs há vários trabalhos que diferenciam classes de tráfego, propondo o isolamento dessas classes de tráfego pela utilização de múltiplas NoCs físicas (do inglês, multiple physical NoCs, MP NoCs), reduzindo interferências entre fluxos pertencentes a classes diferentes. O principal objetivo deste trabalho é propor e avaliar MP NoCs, onde uma rede é dedicada para mensagens de aplicação e uma segunda rede é utilizada para mensagens de gerência (M-NoC). Baseado na avaliação do impacto do tráfego de gerência na comunicação da NoC, duas versões da M-NoC são implementadas e avaliadas. Outra consideração importante para aplicações RT é garantir que os *deadlines* dessas aplicações sejam satisfeitos. A execução dessas aplicações deve ser priorizada sobre as aplicações BE através do fornecimento de mais recursos de processamento utilizando um escalonador RT especializado. Esse trabalho apresenta e avalia uma plataforma MPSoC capaz de suportar QoS de comunicação e de computação, sendo extensível para um número grande de serviços de gerência pelo uso de MP NoCs. Resultados mostram que as M-NoCs podem ser personalizadas para terem um pequeno impacto de área. A utilização de M-NoCs melhora o desempenho de comunicação, latência e jitter, mesmo considerando que a plataforma já possui mecanismos de QoS (como fluxos prioritários e chaveamento de circuitos), pelo isolamento do tráfego de gerência do tráfego de aplicação.

Palavras-chaves: MPSoC, Múltiplas NoCs Físicas, Qualidade de Serviço, Tempo Real.

IMPROVING QOS BY EMPLOYING MULTIPLE PHYSICAL NOCS ON MPSOCS

ABSTRACT

Embedded systems adopt NoC-based MPSoCs since a large number of processing elements (PEs) enables the simultaneous execution of several applications, where some of these applications require real-time (RT) constraints. PEs communicate using messages in distributed memory MPSoCs. These messages can be classified as application messages, being the data generated by the applications, and management messages, used to ensure the correct operation of the platform. As the communication has a large impact on the application performance, an important concern in the design of MPSoCs is to improve the performance of the applications' communication, particularly for RT applications. Two possible methods to optimize the communication performance includes: (*i*) prioritize the RT application messages over the messages generated by best-effort (BE) applications; (*ii*) isolate the application messages from the management messages, considering that complex MPSoCs require a large number of management services to meet the performance constraints. The NoC literature contains several works that differentiate traffic classes, proposing the isolation of these traffic classes by the use of multiple physical (MP) NoCs, reducing interferences among the flows belonging to different classes. The main goal of this work is to propose and to evaluate MP NoCs, with one network dedicated to the application messages and a second network for the management messages (M-NoC). Based on the evaluation of the impact of the management traffic in the overall NoC communication, two different versions of M-NoCs are implemented and evaluated. Another important consideration for RT applications is to ensure that these applications meet their deadlines. The execution of these applications must have higher priority over the BE applications by dedicating more processing resources using a specialized RT scheduler. This work presents and evaluates an MPSoC platform capable of supporting both communication and computation QoS, being extensible for a large number of management services by to the use of MP NoCs. Results show that M-NoCs may be customized to have a small area overhead. The adoption of M-NoCs improves the communication performance, latency and jitter, even when the network used in the platform has QoS mechanisms (e.g. priority flows and circuit switching), by isolating the management traffic from the application traffic.

Keywords: MPSoC, Multiple Physical NoCs, QoS, RT.

LIST OF FIGURES

Figure 2.1 – Network topologies. Source [BAL08]. 20
Figure 2.2 – Efficiency of the evaluated topologies. Source [BAL08]. 21
Figure 2.3 – Efficiency of the CMesh topology. Source [BAL08]. 21
Figure 2.4 – MECS topology. Source [GRO09]. 22
Figure 2.5 – Block diagram of the TRIPS chip, showing networks interconnections. Source [GRA07]. 28
Figure 3.1 - HeMPS-QoS MPSoC organization [CAS13]. 32
Figure 3.2 – PE Internal Architecture. Adapted from [CAR09a]. 33
Figure 3.3 – Example of an application task graph. Task A is an initial task. 36
Figure 3.4 – Real-time constraint model. Source [RUA15]. 39
Figure 3.5 – MPEG real-time constraints configuration. Source [RUA15]. 40
Figure 3.6 – Computation latency (a) MPEG (b) DTW during profiling. 42
Figure 4.1 – Task dependency graph for the MPEG application. 46
Figure 4.2 – Scheduling graph of the MPEG application. 46
Figure 4.3 – Task dependency graph for the DTW application. 47
Figure 4.4 – Scheduling graph of the DTW application. 47
Figure 4.5 - Task dependency graph for the synthetic application. 48
Figure 4.6 – Task dependency graph for the Producer/Consumer application. 48
Figure 5.1 – Full mesh management NoC topology. 53
Figure 5.2 – NI receive control logic FSM. 54
Figure 5.3 – NI send control logic FSM. 55
Figure 5.4 – Source code responsible for selecting the network and adapting the packet header. 56
Figure 5.5 – Mesh between the masters management topology. 56
Figure 5.6 – NI receive control logic FSM supporting network change. 58
Figure 5.7 – NI send control logic FSM supporting network change. 58
Figure 5.8 – Algorithm used to calculate management packet routes. 59
Figure 5.9 – Serializer interface. 60
Figure 5.10 – Serializer FSM. 60
Figure 5.11 – Deserializer Interface. 61
Figure 5.12 – Deserializer FSM. 61

Figure 5.13 – Mesh topology network interconnecting the cluster PEs and another mesh topology interconnecting the clusters. 67
Figure 5.14 – Ring topology network interconnecting the cluster PEs, and a mesh topology interconnecting the clusters. 67
Figure 5.15 – Clos topology network, where a single router interconnects multiple PEs in the cluster, and the higher levels interconnect the clusters. 68
Figure 6.1 – Scenario 1 task mapping. Arrows represent the flows according to the Hamiltonean routing. 71
Figure 6.2 – MPEG computation latency for Scenario 1 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*. 72
Figure 6.3 – Scenario 2 task mapping. 74
Figure 6.4 – MPEG iteration latency for Scenario 2 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*... 74
Figure 6.5 – Scenario 3 task mapping. 76
Figure 6.6 – DTW computation latency for Scenario 3 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*. 77
Figure 6.7 – Scenario 4 task mapping. 78
Figure 6.8 – MPEG iteration latency for Scenario 4 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*... 79
Figure 6.9 – DTW iteration latency for Scenario 4 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*..... 79

LIST OF TABLES

Table 2.1 – NoC monitoring strategies comparison. Source [CIO06]. 26
Table 2.2 – Works exploring the use of multiple parallel physical NoCs. 29
Table 3.1 – Services supported by the reference platform. 43
Table 4.1 – Number of flits transmitted by service. 49
Table 4.2 – Percentage of flits transmitted by specific management services. 50
Table 4.3 – Spatial distribution of the management traffic. 50
Table 5.1 – Router area comparison for different network types (Lib. CORE65GPSVT, 1.0V, 25º C) 52
Table 5.2 – Router area comparison for different flit widths (Library CORE65GPSVT, 1.0V, 25º C).60
Table 5.3 – Average Application Network Latency and Jitter (Clock Cycles) for different M-NoC configurations 63
Table 5.4 – Average Management Network Latency and Jitter (Clock Cycles) for different M-NoC configurations 64
Table 5.5 – Execution Time (milliseconds) with different M-NoC configurations 66
Table 6.1 – Standard deviation of the MPEG computation latency for Scenario 1. 72
Table 6.2 – Average network latency and jitter (in clock cycles) for the Scenario 1. 73
Table 6.3 – Management communication statistics for the Scenario 1. 73
Table 6.4 - Standard deviation of the MPEG computation latency for Scenario 2. 75
Table 6.5 – Average network latency and jitter for Scenario 2. 75
Table 6.6 – Management communication statistics for Scenario 2 75
Table 6.7 – Standard deviation of the DTW computation latency for Scenario 3. 77
Table 6.8 – Average network latency and jitter for Scenario 3. 77
Table 6.9 – Management communication statistics for the Scenario 3 78
Table 6.10 – Standard deviation of the computation latency for Scenario 4. 80
Table 6.11 – Average network latency and jitter for the Scenario 4. 80
Table 6.12 - Management communication statistics for the Scenario 4. 81

LIST OF ACRONYMS

API	Application Programming Interface
BE	Best-Effort
CS	Circuit Switching
DMA	Direct Memory Address
DVFS	Dynamic Voltage and Frequency Scaling
DTW	Dynamic Time Warping
FPGA	Field Programmable Gate Array
HeMPS	Hermes Multiprocessor System
FSM	Finite State Machine
IO	Input/Output
M-NoC	Management Network-on-Chip
MIPS	Microprocessor without interlocked pipeline stages
MP	Multiple Physical
MPI	Message Passing Interface
MPSoC	Multiprocessor System on Chip
NI	Network Interface
NoC	Network on Chip
PE	Processing Element
QoS	Quality of Service
RT	Real-Time
RTL	Register-transfer level
SM	Shared Memory
TCB	Task Control Block
VC	Virtual Channel

CONTENTS

1. Introduction 16
1.1. Goals 18
1.2. Contributions 18
1.3. Document Organization 18

2. State of the Art on Multiple Physical NoCs 19
2.1. Proposal and evaluation of network characteristics and topologies 19
2.2. Energy Efficiency 23
2.3. Specific Applications 26
2.4. Platforms employing multiple physical networks 27
2.5. Final Remarks 29

3. Reference Platform 32
3.1. Hardware 33
3.2. Management 34
3.3. Communication QoS 37
3.4. Computation QoS 39
3.5. Supported Service Summary 43
3.6. Final Remarks 44

4. Evaluation of The Traffic Behavior 46
4.1. Evaluated Applications 46
4.2. Traffic Evaluation 48

5. Management Network Design 51
5.1. Full Mesh M-NoC 52
5.2. Mesh between the managers 56
5.3. Serialization/Deserialization 59
5.4. Management Network Evaluation 61
5.5. Qualitative Evaluation of Other M-NoC Topologies 66
5.6. Final Remarks 69

6. Platform Evaluation 70
6.1. Scenario 1 71
6.2. Scenario 2 73

6.3. Scenario 3 76
6.4. Scenario 4 78
6.5. Final Remarks 81

7. Conclusions and Future Works 83
7.1. Future Works 83
7.2. Publications 84

References 85

1. INTRODUCTION

Multi-Processor System-on-Chip (MPSoCs) supporting a large number of application classes are becoming prevalent in embedded systems [WOL08]. An important design consideration in MPSoCs is its capacity of delivering high-performance computation under tight area and energy budgets. Numerous processors working in parallel can be used to attain this objective, relying on distributing the computation between the available processing elements (PE). MPSoCs with a large number of elements require a communication infrastructure capable of supporting the communication load, being a Network-on-Chip (NoC) an interesting option due to its inherent scalability [KUM02][BJE06].

The communication between the resources in an MPSoC is related to the memory architecture, mainly classified as [ALM09]:

(i) *shared memory* [BEN12]*:* PEs share the same memory resources, requiring a cache coherence mechanism to ensure that modifications done by a processor to a memory location become visible to the remaining PEs in the system. The communication is defined in software through the use of shared memory locations and a locking mechanism;

(ii) *distributed memory* [JOV08][ALM09][CAR09a][FUW14]*:* requires a private memory, and the PEs in the platform communicate through the exchange of messages. The communication exists only between the PEs exchanging data. The communication is defined in software through the use of *send()/receive()* primitives.

Shared memory has a higher communication overhead due to its cache coherence mechanism and does not scale well in large systems [AGA07]. Distributed memory reduces the communication overhead and allows a more efficient use of the communication infrastructure, in respect of area and energy consumption. It is possible to employ both architectures simultaneously, partially reducing the shared memory communication load and increasing the versatility of the platform [KRA93].

Each application executed in an MPSoC has different performance requirements. Real-Time (RT) applications have specific time requirements for its computation results while Best-Effort (BE) does not have time requirements [PAS08]. To ensure that RT applications meet its requirements, the system must be capable of providing Quality-of-Service (QoS) – the capability

to provide sufficient resources for the application – for both the *communication* and the *computation* aspects of the application.

Despite the memory architecture and the type of the application, BE or RT, the traffic flows can be categorized into:

(i) *data traffic:* data generated by the applications;

(ii) *management traffic*: not directly related to the applications, however with an important role in the control of the platform.

The *data traffic* is subject to several interferences present in the platform, such as traffic generated by other applications and the platform management traffic. The communication of RT applications can be prioritized applying a different treatment for this type of traffic, such as the use of a differentiated routing algorithm, exclusive physical path, prioritized arbitration, among others. Communication prioritization allows reducing the traffic interference, improving the performance of the application.

The *management traffic* comprises all the traffic related to several services available on the platform, such as performance monitoring, actuation messages for QoS, control of the MPSoC resources, debugging, task mapping, fault tolerance, power management, temperature monitoring, security. As the complexity of the MPSoC platform increases and more elements are included in the platform, the requirements of the management traffic also increase [KOR13]. At the same time, the management traffic competes with the application traffic for the communication resources, negatively impacting the application performance. Therefore, it is vital to provide the *isolation* of the traffic according to its function in the platform, separating the application traffic from the management traffic. This isolation can be achieved by providing a communication infrastructure dedicated for the management traffic.

RT applications are subject to deadlines – a bounded time for the application to deliver its computation results [LIU00]. The platform must guarantee that the application has executed enough time to generate its result before reaching a deadline, ensuring that sufficient computation resources are granted to the application. This action presents a challenge since in parallel to the application subject to RT constraints, several other applications are sharing the same computation resources. The platform must be capable of prioritizing the application processing using a specialized scheduler, aware of the platform resources and application requirements.

1.1. Goals

The strategic goal of this work is to explore a scalable MPSoC architecture supporting a vast number of management services and capable of delivering both computation and communication QoS by using multiple physical NoCs. This strategic goal requires the following specific objectives:

- Integration of the features available in two distinct MPSoC platforms into a single platform. The platforms are HeMPS-QoS, supporting *communication* QoS, and HeMPS-RT, supporting *computation* QoS;
- Evaluate the traffic characteristics in the platform, concerning the temporal and the spatial distribution of the packets;
- Isolation of the management traffic from the application traffic, through the use of multiple physical (MP) NoCs;
- Exploration of the aspects of the network used for the management traffic.

1.2. Contributions

The main contributions of this work include:

- An MPSoC platform supporting both *communication* and *computation* QoS;
- Evaluation of the traffic behavior in an MPSoC platform;
- Development of a communication infrastructure based on multiple physical NoCs, targeting the different traffic categories in MPSoCs.

1.3. Document Organization

This document is organized as follows. **Chapter 2** presents the state of the art in the area of multiple physical NoCs. **Chapter 3** presents the reference platform implemented in this work. **Chapter 4** presents the available applications and conducts an evaluation of the traffic in the platform. **Chapter 5** describes the design of the network used for the management traffic and presents an evaluation of different topologies for this network. **Chapter 6** presents an evaluation of the platform considering all implemented resources for QoS. **Chapter 7** presents conclusions and directions for future works.

2. STATE OF THE ART ON MULTIPLE PHYSICAL NOCS

Several proposals explore the use of multiple physical (MP) NoCs [AGA07][BAL08][ABO12], where each subnetwork is specialized for different traffic classes. This Chapter presents a summary of these works, highlighting some of their characteristics that are also explored in this work. At the end of this Chapter, a comparative table between the described works is presented, situating this work with the state of the art.

These works can be classified according to their respective goals, such as: *(i)* proposal and evaluation of different network characteristics and topologies; *(ii)* energy efficiency; *(iii)* specific NoC applications; *(iv)* platforms employing multiple physical networks.

2.1. Proposal and evaluation of network characteristics and topologies

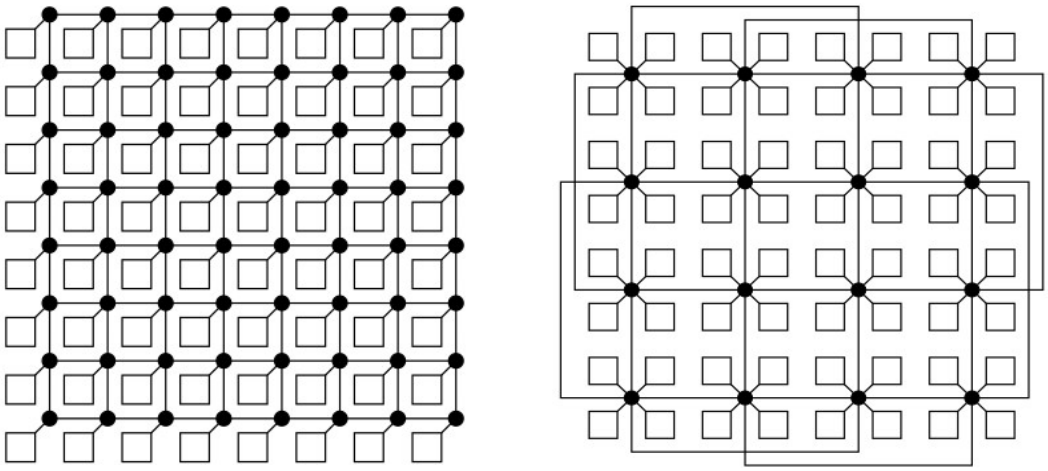
These works focus on the performance, area and energy efficiency of different network characteristics and topologies, including in their evaluation the use of MP networks. A common characteristic of these works is the adoption of the same traffic model, cache-coherence communication protocol, which can be divided into multiple subclasses, such as Read Requests/Replies, Write Requests/Replies, among others.

2.1.1. Design Tradeoffs for Tiled CMP On-Chip Networks [BAL08]

This work proposes an area and energy model and investigates how different network aspects such as topology, channel width, routing strategy and buffer size affect performance, area and energy efficiency, evaluating the tradeoffs between these characteristics.

Several network topologies are evaluated in this work, including a mesh topology with two parallel subnetworks (MeshX2) and a concentrated mesh with two subnetworks, where each router services four processors (CMeshX2). Both are shown in Figure 2.1(a) and Figure 2.1(b).

Two strategies are proposed to distribute the traffic over the subnetworks: *(i)* one subnetwork is used to transport short packets, such as read requests and write replies, and the other is used to transport long packets, such as read replies and write requests. This allows a heterogeneous architecture, optimizing each subnetwork for each packet type; *(ii)* one subnetwork transport packets associated with read transactions while the other one transport packets associated with write transactions. This architecture is homogeneous in respect to its subnetworks.



(a) Mesh (b) CMesh

Figure 2.1 – Network topologies. Source [BAL08].

The networks are evaluated using a cycle-accurate interconnection network simulator using various synthetic traffic patterns. Two aspects are analyzed: *(i) Area-delay:* measured as the product of the workload completion time and the chip area; *(ii) Energy-delay:* measured as the product of the completion time and energy expended in the network.

The CMeshX2 is the preferred topology by the Authors since it presents better results when compared to the other evaluated topologies, as shown in Figure 2.2. This topology is further analyzed, as shown in Figure 2.3, evaluating different topologies (single and duplicated network), different flit widths and different traffic distribution strategies (homogeneous and heterogeneous). The work assumes a constant channel width, e.g., the flit width of the duplicated network is the half of the single network. The duplicated networks present better efficiency when compared to a single network since narrower flit widths requires fewer repeaters to drive the wires interconnecting the routers. Also, the homogeneous strategy to distribute the traffic also achieved better efficiency results, since it promotes a better balancing of loads between the subnetworks.

The Authors conclude that the addition of a second network significantly improves both performance and energy efficiency while having a negligible impact on the chip area because the additional routers are positioned in areas initially allocated for channels in the first network. In overall, the CMeshX2 presents a 24% improvement in area efficiency and a 48% improvement in energy efficiency over other networks evaluated in the study.

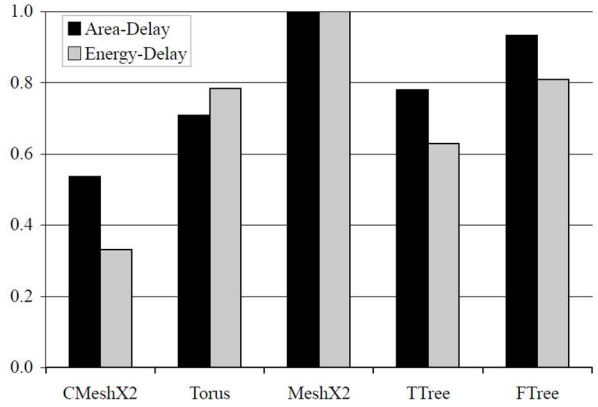


Figure 2.2 – Efficiency of the evaluated topologies. Source [BAL08].

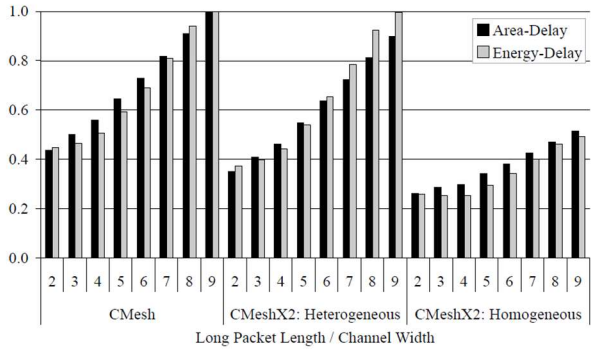


Figure 2.3 – Efficiency of the CMesh topology. Source [BAL08].

2.1.2. Express Cube Topologies for On-Chip Interconnects [GRO09]

This work proposes the Multidrop Express Channels (MECS) topology for NoCs, where multiple cores are connected to a single router and the connections between the routers follow a one-to-many configuration. Figure 2.4 presents this topology. Every router output port injects packets into a bus that is connected to all routers input ports in its direction (repeaters are inserted to improve the channel energy and delay). The input ports are connected to a single bus. Therefore, each router has multiple input ports in each direction, equal to the number of routers remaining in that direction. This topology allows to reduce the number of hops required for a packet to reach its destination.

The Authors also propose replicating the network, comparing this implementation to a single network implementation, where the sum of all multi-network channels bandwidth is equal to the single network channel bandwidth. The remaining network parameters, such as buffer depth is equal for both cases.

The networks are evaluated using synthetic workloads (such as bit complement, uniform random and transpose traffic) and application traces generated using the PARSEC benchmark. In

average, the single network implementation has a latency 10% smaller when compared to a two networks implementation. However, the energy-delay product (product of the completion time and energy expended in the network) of the two networks implementation is around 10% smaller, showing that replicating networks are effective at minimizing the network energy.

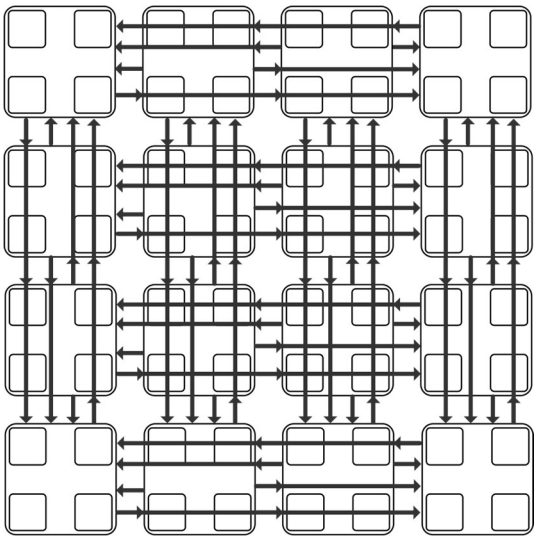


Figure 2.4 – MECS topology. Source [GRO09].

2.1.3. Virtual Channel and Multiple Physical Networks: Two Alternatives to Improve NoC Performance [YOO13]

This work presents a comparison between NoCs with VCs and MP NoCs, considering that their channel bandwidth is equal (the VC router has a flit width equal to the sum of all MP flit widths), and the total input port buffer storage is also equal (the total buffer capacity in bits for each input port is equal for both cases). The networks are evaluated considering multiple parameters, such as channel width, buffer storage, the number of physical channels, the number of virtual channels and frequency. Several aspects of both networks are analyzed, such as:

- *Maximum operating frequency*: MP has a simpler logic and can operate faster in the architecture evaluated by the Authors;
- *Area consumption*: MP routers are smaller than VC routers when using shallow queues, and MP router area increases more linearly when varying the frequency;
- *Power dissipation*: The difference is negligible when using 65 nm and 90 nm technologies. However, this difference becomes more relevant with smaller technology nodes due to leakage power. With buffers depth equal or smaller than 8 flits and channel width of 128 bits, MP routers dissipate equal or less than the equivalent VC routers in more recent technologies;

- *Minimum size*: The routers are synthesized targeting minimum area instead of operating frequency and uses the shallowest possible buffers. MP routers can operate faster (clock period of 1 ns) than VC routers (clock period of 1.6 ns), and MP routers have an area overhead of 38% when compared to a single network with a single channel implementation. VC routers have an area overhead of 56% to 121% when compared to this same single network. These values consider two physical channels or two VCs;
- *Synthesis for FPGAs*: MP router with four networks occupy less area and is around 18% to 35% faster than a VC router (using 4 networks/VCs);
- *Performance*: Under the same loads, with the traffic well-distributed, VC gives better maximum throughput and average latency. However, with traffic patterns that generate hotspots in the NoC channels, MP provides better maximum sustained throughput.

In summary, MP routers tend to be more power and area efficient when compared to VC routers when using shallow buffers, and can have additional benefits when considering technology scaling. The performance depends on the traffic patterns.

The Authors also observe that MP may provide a robust network infrastructure when controlled by a fault-tolerance policy, and allows a heterogeneous partitioning, where one subnetwork is dedicated to efficient data transfers while others are used to control several aspects of the platform.

2.2. Energy Efficiency

These works take advantage of MP networks to optimize energy efficiency without degrading the performance, distributing the traffic subclasses into the available networks. Each network is specialized for the characteristics or performance requirements of each traffic subclass.

2.2.1. Déjà Vu Switching for Multiplane NoCs [ABO12]

This work suggests that cache coherence messages can be classified in critical and non-critical messages. All data messages (e.g. cache lines) are considered non-critical messages while control messages (e.g. data requests, invalidations, acknowledgments) are considered critical.

A dedicated subnetwork is used for each message class (both networks adopt the mesh topology), and the networks differ with respect to the flit width, being used a 10-byte width for the data network and a 6-byte width for the control network. The control subnetwork uses a

regular packet switching based on the XY routing algorithm while the data subnetwork uses the Déjà Vu switching, which pre-establishes a circuit to the target router using a special reservation packet transmitted on the control network. After the path establishment, the data packet can traverse its subnetwork without additional delays due to the routing process.

The use of dedicated networks for the control messages and data messages allows a more efficient use of buffers and bandwidth resources, saving power. The data plane voltage and frequency can also be reduced to avoid that the data packet must wait in the network for its path to be established, which allows the data network to operate slower without negatively impacting the performance.

The network is evaluated using synthetic traces and benchmarks executed on a full system simulator. The network is compared to a baseline implementation without MP networks. The data network has its operating frequency set to 2.66 GHz while the control network and the baseline implementation operates at 4 GHz. The switching method proposed by the Authors allows to reduce the average energy consumption by 50%, without negatively impacting the performance.

2.2.2. CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache-Coherent Servers [VOL12]

The CCNoC is a dual-network architecture specialized for directory-based cache-coherence traffic. The *request* network is mainly used for request messages, primarily consisting of block fetch request and clean replacement notifications. This network is optimized for short messages, having a smaller flit width (64-bits), and uses two VCs to avoid deadlocks related to the cache coherence protocol. The *response* network primarily transmits messages carrying a cache block and supports wider flits (112-bits) than the *request* network, and does not require VCs. Both networks use the wormhole switching and adopt the mesh topology.

This network is evaluated in a full system simulator using the TPC-C benchmark suite and has its results compared to two single-network implementation, one with a flit width of 176 bits, and the other with a flit width of 128 bits. The CCNoC has a power consumption 28% smaller in average compared to the 176-bits flit-size implementation without performance loss and reduces the network area by 55%, since it requires fewer VCs. Compared to the 128-bits flit width network, it presents a reduction in power consumption of 18%, while being around 5% faster and has an area 10% smaller.

2.2.3. Catnap: Energy Proportional Multiple Network-on-Chip [DAS13]

This work proposes the use of MP NoCs, where initially, a single subnetwork is active and used to transmit packets while the remaining subnetworks are power gated. As the subnetwork starts to congest, an additional subnetwork is enabled. Compared to a bandwidth equivalent single-network implementation, this increases the time and the number of components power gated at runtime, reducing the overall power consumption.

Every subnetwork has a specific priority. Packets are initially injected in the highest priority subnetwork, and, as this subnetwork starts getting close to congestion, a lower priority subnetwork is activated, and the load is distributed between the active networks. The congestions are detected according to the buffer occupancy. When a subnetwork detects that its immediate higher priority subnetwork is not congested, it starts its process to deactivate the subnetwork.

This network is evaluated under several synthetic traffic patterns and application traces. In overall, this strategy allows to reduce the network power to 44% of an equivalent single-network implementation, with a performance overhead of 5%.

2.2.4. Data Criticality in Network-On-Chip Design [MIG15]

This work aims to improve the NoC energy efficiency by delaying fetching of cache data blocks until they are really required. Some of these fetches are considered *critical*, meaning that they needed right away, while other fetches considered *non-critical*, which can have its transmission delayed through the use of low-power techniques applied to the NoC. When an instruction is currently waiting in the pipeline for a data word, this fetch is considered *critical*, while memory blocks fetched as a consequence of bulk-fetching are considered *non-critical*.

To support this message differentiation, this work proposes the NoCNoC, where the NoC is divided into multiple subnetworks, each one operating at different frequencies and voltages. The non-critical network employs dynamic voltage-frequency scaling (DFVS) to slow down its operation. The frequency is dynamically controlled according to the proportion of traffic injected into the *non-critical* subnetwork compared to the *critical* subnetwork, aiming to balance the utilization preventing high congestion on either network. Therefore, the network can adapt to different application requirements.

The NoCNoC is compared to a baseline implementation with a single network. The baseline implementation uses a 128-bit channel while NoCNoC uses an 88-bit channel for the

critical subnetwork and a 40-bit channel for the *non-critical* subnetwork. The *non-critical* subnetwork operates from 500 MHz to 2 GHz, in steps of 250 MHz. The *critical* subnetwork and the baseline implementation operates at 2 GHz. On average, the NoCNoC achieves in average 27.3% energy consumption reduction (up to 60.5%) compared to the baseline implementation, while increasing the runtime by 3.6%.

2.3. Specific Applications

An additional network can be used to transfer communication not directly related to the application data. These works propose a parallel network for specific purposes in the platform.

2.3.1. NoC Monitoring: Impact on the Design Flow [CIO06]

This work explores different NoCs monitoring strategies to increase its observability. This work observes the challenge in interconnecting the monitoring probes, evaluating scalability, non-intrusiveness, run-time usage, reconfigurability and area cost. As NoCs are a scalable interconnect, the Authors suggest to include in the NoC design flow additional communication infrastructures for the monitoring infrastructure.

Three strategies are explored in this work for the monitoring infrastructure: *(i)* MP NoCs, one used for the original NoC application and a simpler one used for monitoring; *(ii)* router reuse, adding additional physical channels and ports; *(iii)* sharing of the existing network infrastructure, monitoring packets use the same networks as the application packets.

The strategies are evaluated with respect to its impact on the design flow, non-intrusiveness (the impact that the monitoring has on the user traffic), area cost and reuse capability after debugging. The results are summarized in Table 2.1, with respect to its positive and negative aspects.

Table 2.1 – NoC monitoring strategies comparison. Source [CIO06].

	i	**ii**	**Iii**
Design Flow	++	+	-
Non-intrusiviness	+	+	+/-
Area Cost	-	-	+
Reconfigurability	-	+	+

2.3.2. Reconfigurable Security Architecture for Disrupted Protection Zones in NoC-Based MPSoCs [SEP15]

This work proposes the creation of secure zones in MPSoCs, to protect sensitive information exchanged through the NoC. The platform suggested in this work adopts two NoCs, a data NoC, used for the application data, and a service NoC used to exchange the security control packets of the MPSoC.

To achieve security, the information exchanged among IPs in a security zone is encrypted. The cryptograph keys are exchanged using the Diffie-Hellman algorithm, using control packet transmitted by the service NoC. The data packets are encrypted by a Secure Interface, which communicates with both networks and controls the establishment of the security zones.

2.4. Platforms employing multiple physical networks

Some platforms that employ several traffic classes count on the use of MP networks, specializing each network for a specific traffic class, providing isolation and prioritization of the traffic.

2.4.1. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs [AGA02]

The Raw microprocessor has 16 computing units interconnect by four networks, where each network adopts the mesh topology and have a flit width of 32 bits. Two networks are considered static, and the remaining ones are considered dynamic. Both networks are accessible by the software through mapped registers.

The static networks are configured before a packet is sent, and allows to establish a circuit between two computing tiles. Thus, flits sent to these networks have a low latency, since it enables a single cycle per-hop latency, and are not susceptible to interferences.

The dynamic networks use the wormhole routing and the XY routing algorithm. These networks are subdivided into the memory network and the general network. The memory network is restricted to trusted clients (OS, data cache, interrupts, hardware devices, DMA and I/O) while the general network is available to the user applications.

2.4.2. On-chip Interconnection Networks of the TRIPS Chip [GRA07]

The TRIPS chip has two networks, the OCN network, which replaces a traditional memory bus, and the OPN network, which replaces a traditional operand bypass (transfer of a single word between the execution units) and L1 cache buses. Figure 2.5 presents the block diagram of this processors, showing the network interconnections.

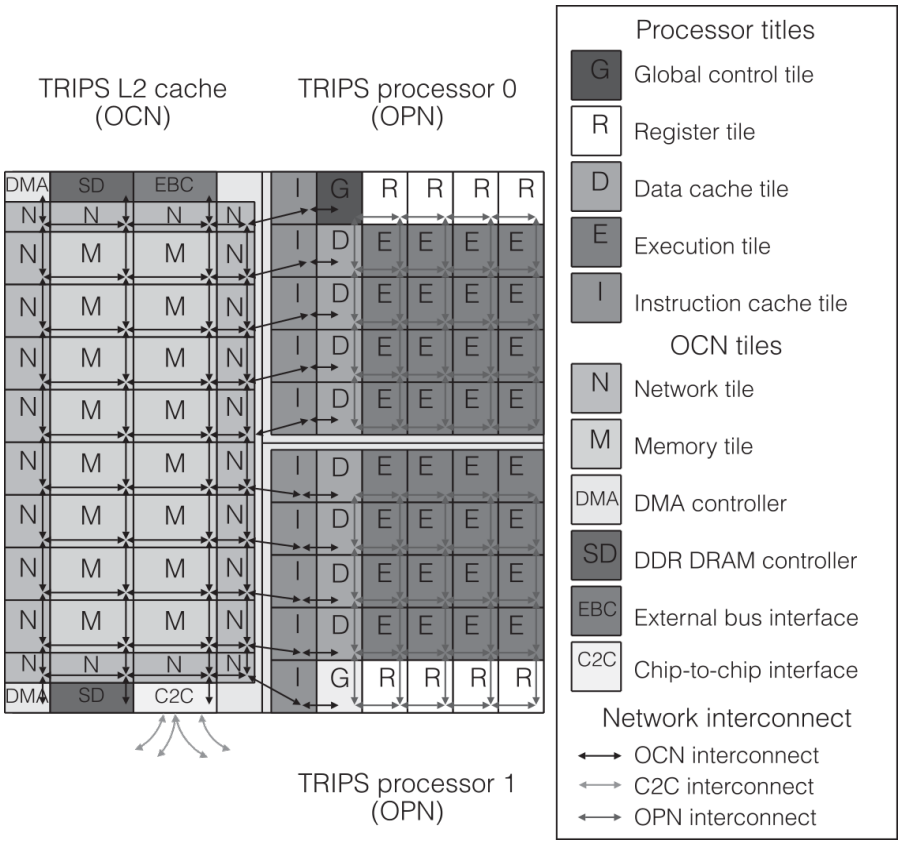


Figure 2.5 – Block diagram of the TRIPS chip, showing networks interconnections. Source [GRA07].

In Figure 2.5, the OCN network connects the two processor cores to the L2 cache and I/O units. Within the processor cores, the OPN connects the tiles implementing the register files, data caches and execution units. The networks also differ with respect to its characteristics, such as flit width (OCN uses 138-bits while OPN uses 142 bits), number of VCs (OCN uses 4, OPN does not support VC) and flow control (OCN uses wormhole, OPN uses single-flit packets).

2.4.3. On-Chip Interconnection Architecture of the Tile Processor [AGA07]

The Tile processor is a commercial product where its first implementation, the Tile64, contains 64 cores and several I/O devices, all interconnected by five 2D mesh NoCs. Each subnetwork is used for different traffic classes, and are described as follows: *(i)* UDN, used to communicate user threads or processes executing in different cores by the means of operands

(transfer of a single word between the processors), socket-like channels or message passing, ; *(ii)* IDN, used for I/O and system level traffic, isolating it from the user applications; *(iii)* MDN, used to communicate with the off-chip DRAM; *(iv)* TDN, which works together with the MDN as a portion of the memory system, used to send cache request to other cores; *(v)* STN, also used by user threads or process, which can be used to establish a circuit between two cores, ensuring low-latency and high-bandwidth. Networks *(i)* to *(iv)* uses the XY routing algorithm and wormhole switching, while the STN network uses circuit switching and its path is pre-established before sending the packet.

This platform has a simultaneous support for both shared memory and distributed memory architectures, leaving the decision of which architecture to be used for communication to the application programmer. The shared memory approach in this platform has a higher communication overhead and is less scalable when compared to the distributed memory approach, however, according to the Authors, it is simpler to program. The use of MP networks allows to support both architectures without interferences between them and also provides a clear separation of the system traffic from the application traffic. The STN network also provides a way to ensure low latency communication between two cores.

The Authors comment on the choice of using multiple physical networks, instead of using multiple virtual channels, justifying that there is a large availability of wires in modern fabrication processes, and most of the area cost is spent on buffers, which is also required when using virtual channels.

2.5. Final Remarks

Table 2.2 compares the characteristics of the previously reviewed works, and the last table line presents the characteristics of the proposed work.

A common characteristic of most analyzed works is the differentiation of traffic classes in the platform, and the proposal of isolating these traffic classes by the use of MP NoCs, reducing interferences among the flows belonging to different classes. Each subnetwork has its parameters (flit width, buffer width, and others) adjusted according to the traffic characteristics, improving the network efficiency.

Table 2.2 – Works exploring the use of multiple parallel physical NoCs.

Year	Title	#NoC	Asymmetrical	Evaluated Workload	Traffic Classes	Network Selection Policy
2002	The RAW Microprocessor [AGA02]	4	**Yes (2 types)** Different flit width and router architecture	Real applications (available as an IC)	Messages/IO/ Interruptions/ Memory/ Operands	Two networks (*static*) are used for operands, other two networks (*dynamic*) are used for remaining traffic.
2006	NoC Monitoring: Impact on the Design Flow [CIO06]	2	Not specified	N/A	Monitoring/ Data	Monitoring data uses a dedicated network, while the remaining traffic uses the other network.
2007	On-Chip Interconnection Networks of the TRIPS Chip [GRA07]	2	**Yes** Different flit widths and router architecture	Real applications (available as an IC)	Operands/ Memory	One network is dedicated for operands and the other is dedicated for the memory.
2007	On-Chip Interconnection Architecture of the Tile Processor [AGA07]	5	**Yes (2 types)** One NoC (*STN)* supports CS, other NoCs doesn't	Real applications (available as an IC)	Messages/IO/ Memory/ Operands	*UDN* and *STN* networks are used for operands and messages, *IDN* network is used for IO and system traffic, *MDN* and *TDN* networks are used for the memory and cache coherence.
2008	Design Tradeoffs for Tiled CMP OnChip [BAL08]	2	**Yes** Different flit widths	Synthetic Traffic Patterns	Cache Coherence	Two suggested strategies: *(i)* Read requests and write replies in one network, read replies and write requests in the other; *(ii)* One network transmit read transactions, while other transmits write transaction.
2009	Express Cube Topologies for On-Chip Interconnects [GRO09]	2	**No**	Synthetic Traffic Patterns/ Application Traces	Cache Coherence	N/A
2012	Déjà Vu Switching for Multiplane NoCs [ABO12]	2	**Yes** Different voltage and frequency	Synthetic traces/ Benchmarks executed on a full system simulator	Cache Coherence	Critical and non-critical messages.
2012	CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache-Coherent Servers [VOL12]	2	**Yes** Different flit widths and router architecture	Benchmarks executed on a full system simulator	Cache Coherence	Requests messages in one network, response messages in the other.
2013	Catnap: Energy Proportional Multiple Network-on-Chip [DAS13]	1-8	**No**	Synthetic traces/ Application traces	Cache Coherence	Initially uses a single network, other networks are powered off. As the network becomes congested, enables a new network and starts balancing the traffic between the active networks.
2013	Virtual Channels and Multiple Physical Networks: Two Alternatives to Improve NoC Performance [YOO13]	4	**Yes** Different flit widths	Benchmarks executed on a full system simulator	Cache Coherence	Each network is used for a specific class of cache coherence messages
2015	Data Criticality in Network-On-Chip Design [MIG15]	2	**Yes** Different flit widths and frequency	Benchmarks executed on a full system simulator	Cache Coherence	Critical and non-critical messages.
2015	Reconfigurable Security Architecture for Disrupted Protection Zone in NoC-Based MPSoCs [SEP15]	2	**Yes** Different flit widths and buffer depth	Benchmarks executed on a cycle accurate simulator	Security Ctrl/ Data	One network (Data) is used to transmit the data among the IPs of the MPSoC, the other (Service) is used for security services
-	**This Work**	2	**Yes** Different router architecture	Applications executed on a RTL MPSoC description	Application Msg/ Management Msg	Application data uses one network, management traffic uses the other network.

Most works on Table 2.2 focus on modeling the traffic as a cache coherence protocol because most contemporary systems adopt *shared memory* architecture. This work target *distributed memory* architectures since it is scalability compared to shared memory (SM) architectures because traffic hotspots near to the SM are avoided and cache coherence mechanism has a higher communication overhead than message exchange [AGA07].

Some systems may require the use of messages not related to the application to manage several platform aspects, such as performance monitoring, power management, mapping requests, actuation messages for QoS, security, among others.

The use of MP networks allows improvements in area and energy efficiency when compared to an equivalent performance single network implementation. It also presents an opportunity to employ several low power techniques, such as DVFS.

Most works explore the NoC architectural parameters, using platform simulators for evaluation. This work implements the MP architecture on a complete MPSoC platform, analyzing the impact of this strategy on a real system, with a clock-cycle validation.

3. REFERENCE PLATFORM

The reference platform adopted in this work is based on the HeMPS-QoS [CAR11] platform, an MPSoC architecture composed of several Processing Elements (PEs) interconnected by a Network on Chip (NoC), represented in Figure 3.1. All packets in the NoC encapsulate messages that are exchanged among the PEs for communication, accessible at the software level by an MPI-like API. Each message in this platform has a specific service identification according to its function in the platform. The applications executed in the MPSoC are divided into tasks, which run in parallel in several PEs. Multiple tasks can execute in the same PE, and each task has a dedicated memory page. This MPSoC adopts a distributed memory architecture. Each task executes in its private memory, and no memory is shared between the PEs.

The platform is distributed with a platform generator, which generates the platform, configuring the hardware and the software according to parameters specified in a configuration file.

Next sections detail the platform. Section 3.1 presents the main hardware features. Section 3.2 details how the platform is managed. Section 3.3 explain the communication QoS capabilities of the platform. Section 3.4 details the computation QoS system employed in the platform.

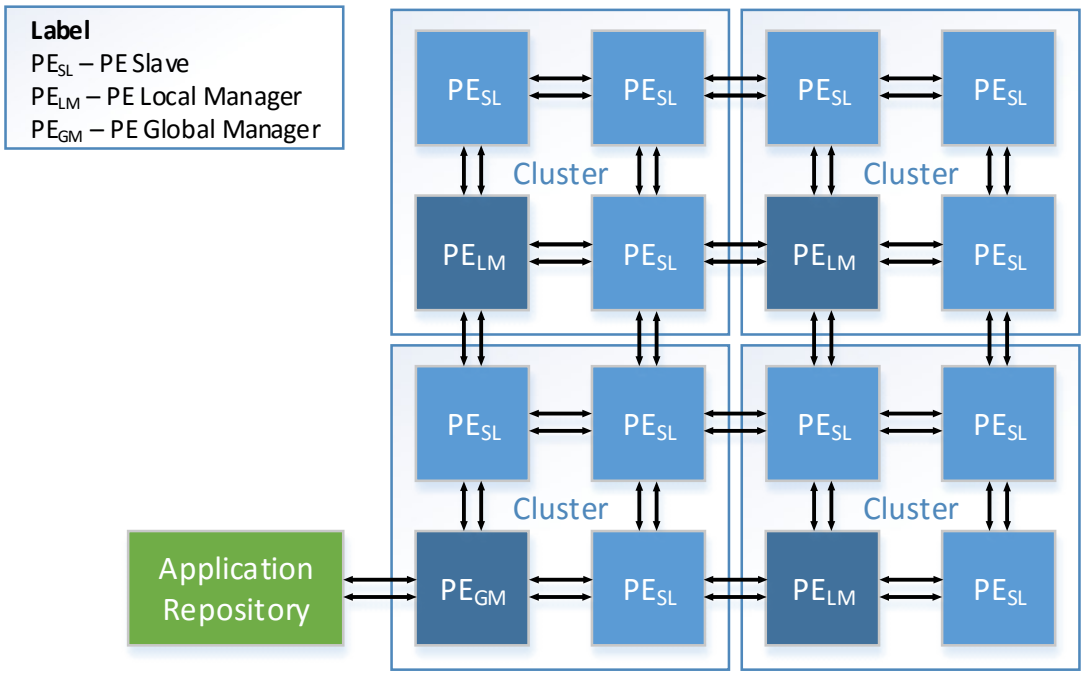


Figure 3.1 - HeMPS-QoS MPSoC organization [CAS13].

3.1. Hardware

The platform is available both in RTL-VHDL and in a cycle-accurate SystemC representation, where the latter simulates faster [PET12]. Many hardware aspects of the platform are parameterizable, such as the MPSoC size, cluster size and the number of pages at each PE (the platform adopts a paged memory organization using the scratchpad memory).

This platform is homogeneous in respect to the PE architecture, represented in Figure 3.2. Each PE contains the following hardware components:

(i) a Plasma processor [RHO10] implementing a subset of the MIPS architecture;

(ii) a private scratchpad memory;

(iii) a Direct Memory Access (DMA) module, allowing to transmit data from/to the memory directly to the NoC without the processor interference;

(iv) a Network Interface (NI) module, responsible for handling the communication between the NoC and the DMA.

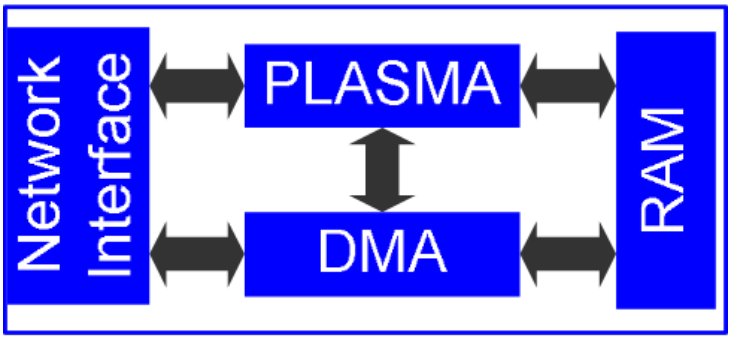


Figure 3.2 – PE Internal Architecture. Adapted from [CAR09a].

The NoC used to interconnect the PEs is the Hermes-QoS [CAR09b], which has some of its characteristics detailed below:

(i) mesh topology, where each central router has two input ports and two output ports in each direction (North, South, East, West, Local). Border routers have unused ports removed;

(ii) packets have multiple flits, consisting of header and payload. The header contains both the target address and configurations specific for each packet. An additional bit transmitted together with the flits indicates the end of packet;

(iii) support for two flows priorities through the usage of two physical channels (high priority channel and low priority channel). High priority flows use both channels to

avoid congestions while low priority flows are constrained to the low priority channel. Packets are configured individually by setting special flags in the header;

(iv) simultaneous support for wormhole switching, used for Best Effort traffic, and circuit switching (CS), used for Guaranteed Throughput traffic. When a CS is established, the high-priority channels in the path are reserved;

(v) simultaneous support for deterministic and partially adaptive Hamiltonian routing algorithm. Also configured individually for each packet by special flags in the header;

(vi) no virtual channels;

(vii) each input port has a credit based input buffer;

(viii) centralized round-robin arbitration for the output ports.

Both the input buffer depth and flit width are configurable at design time. The HeMPS-QoS platform adopts a fixed buffer depth of 8 flits, and each flit has 16 bits.

This platform has an external memory to the MPSoC, where the object code of the applications is stored.

3.2. Management

This MPSoC uses a hierarchical management architecture [CAS13]. PEs adopt one of the following three roles in the platform: *(i)* the slaves (PE_{SL}) are dedicated to the task execution; *(ii)* the local manager (PE_{LM}) controls the cluster resources; *(iii)* the global manager (PE_{GM}) provides a high-level management of the MPSoC resources and communicates with the application repository. All PEs run a microkernel, dedicated to its attributed role.

In summary, the management in this MPSoC consists in task mapping, resource sharing between clusters, control of communication QoS between the application tasks, control of the computation resources and task migration between the PE_{SL}.

The MPSoC is divided into clusters, defined at design time. Each cluster has multiple PE_{SL} and a single manager. One of these managers is selected to manage all MPSoC resources (PE_{GM}), besides managing its cluster resources.

All management messages use only the low priority channel and use the deterministic version of the Hamiltonian routing algorithm.

When the MPSoC starts, the PE_{GM} sends a message with the service *INITIALIZE CLUSTER* to all PE_{LM}, indicating the cluster position, size, and the PE_{GM} address. Then, all managers send an *INITIALIZE SLAVE* message to the PE_{SL} under their control, indicating their manager address.

3.2.1. Task Communication

The application developer can specify the communication between tasks through system calls (*syscalls)*. Producer tasks can send data messages using a non-blocking *Send()* syscall [BAG08], which accepts as parameters the target task identification and a pointer to a structure containing both the message size and its contents. Consumer tasks can receive data messages using a blocking *Receive()* syscall that accepts as parameters the producer task identification and a pointer to where the data is going to be stored after its reception.

Data messages are only injected into the network when they are requested. When the consumer task calls the *Receive()* syscall, a control message with the service *MESSAGE REQUEST* is sent to the producer PE. This request is stored in a memory table in the producer until the message is ready to be delivered. Then, the producer PE sends a *MESSAGE DELIVERY* to the consumer PE. If the data message is generated before the producer PE receives a *MESSAGE REQUEST*, the data is stored in a software pipe present in the producer PE microkernel, allowing the task to continue its execution. After the reception of the *MESSAGE REQUEST*, the message is removed from the pipe and sent to the consumer PE.

Each PE_{SL} has a task location table in its microkernel that stores the physical location of each task that it has previously communicated with. If the PE does not know where a given task is located, i.e. the first message that a producer sends to the consumer, the PE_{SL} generates *a TASK REQUEST* message to its manager requesting the location of the other task. If the task is still not allocated, the manager first maps the task to an available processor and then informs the producer PE the location of the consumer PE using a *TASK ALLOCATED* message (and also informs the consumer PE the location of the producer PE). Later, the consumer generates a *MESSAGE REQUEST* to the producer, and the message can be delivered.

3.2.2. Mapping

Applications are modeled as a task graph, as shown in Figure 3.3. In this graph, each vertex represents a task, and the edges represent the tasks' dependencies. Tasks are allocated at runtime, having its object code loaded from the application repository. Tasks that do not have any

reception dependence are allocated first. The remaining tasks are dynamically allocated when the already allocated task tries to communicate with a non-mapped task.

Applications have a configurable start time. The application repository notifies the PE_{GM} when a new application must start, and then, the PE_{GM} selects a cluster where the application will execute according to the available cluster resources. If the selected cluster does not correspond to the PE_{GM} cluster, the PE_{GM} sends a *NEW APP* message to the cluster PE_{LM}, containing the task graph information. The manager of the selected cluster executes a mapping heuristic [MAN15] that takes into account the CPU utilization of each PE_{SL} in the cluster, and selects a PE_{SL} to execute the task. The task can now be loaded from the application repository. If an application mapped to a cluster that does not belong to the PE_{GM}, the PE_{LM} sends a *NEW TASK* message to the PE_{GM}, indicating the PE_{SL} address where the task must be allocated. Then, the PE_{GM} sends a *TASK ALLOCATION* message to the PE_{SL}, containing the task object code.

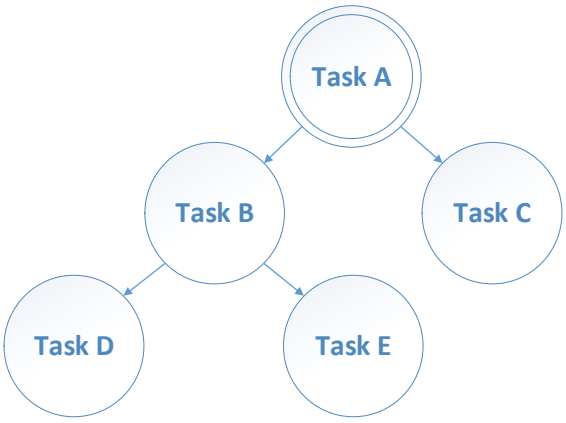


Figure 3.3 – Example of an application task graph. Task A is an initial task.

When the task finishes its execution, the PE_{SL} sends a *TASK TERMINA*TED message to its manager, signalizing that a cluster resource is now available. After all tasks of an application have finished its execution, the PE_{LM} sends an *APP TERMINATED* message to the PE_{GM}, releasing the cluster resources.

3.2.3. Reclustering

Resources can be shared between clusters at runtime. A resource consists of a memory page in a PE_{SL} used to execute a task. When there is no available resource in a cluster, and the manager must allocate a new task, the original manager sends a *LOAN PROCESSOR REQUEST* to the other managers in the MPSoC, asking for a resource [CAS13]. The other managers reply to this request using a *LOAN PROCESSOR DELIVERY* message, indicating if there is a resource available in its cluster, and the address of the PE_{SL} if a resource can be lent. The original master receives the

LOAN PROCESSOR DELIVERY message and selects the PE_{SL} that would require the minimum amount of hops to communicate to the other tasks in the application. The managers that are not selected receive a *LOAN PROCESSOR RELEASE* message, canceling this request and releasing the shared resource. If during this process, the original manager receives a *TASK TERMINATED* message, the reclustering process is canceled and the original cluster uses the resource that was just released. Otherwise, the original manager can now allocate a new task to the shared resource. During the task execution, all management traffic related to the application is sent to the original manager. After the task has finished its execution, a *TASK TERMINATED OTHER CLUSTER* message is sent to the other manager, returning this resource to the other manager.

3.2.4. Scheduling

Considering a PE_{SL} executing only BE applications, tasks are scheduled using a round-robin scheduler, which is called by a processor interrupt at the end of a time slice counter. Each task has its private memory page and a data structure in the microkernel named Task Control Block (TCB), which contain the schedule status for this task [SIL08]. The possible status for a BE task are: *(i) READY*, if this task is available for execution; *(ii) RUNNING*, if this task is currently executing; *(iii) WAITING,* if this task cannot be scheduled because it is currently waiting for a message. At each scheduler run, a task that has the *READY* or *RUNNING* state is selected to execute. If there is no task to be executed, the PE_{SL} is put in a hold state, disabling its execution until a new external interruption is received (e.g., from the NoC).

3.3. Communication QoS

The communication performance between application' tasks is critical for applications subject to RT constraints. High packet latency delays the application processing, affecting its requirements. The HeMPS-QoS platform supports the simultaneous execution of RT and BE applications through the use of a QoS policy that prioritizes the RT applications traffic over the BE applications traffic.

Consumer tasks can specify the communication QoS requirements using the *SetQoSProducer()* syscall. This syscall accepts three parameters: the producer task identification, the maximum acceptable message latency (in clock cycles) and the minimum acceptable throughput (received bytes in a specified time window). The time window defaults to 1 ms and can be changed using the *SetQoSMonitoringWindow*() syscall, which accepts as parameter the new time window value. The producer task must respectively specify their consumer tasks which have

QoS requirements using the *SetQoSConsumer()* syscall, which accepts as parameter the consumer task identification. The *SetQosProducer()* syscall generates a *QOS REQUEST SERVICE* to its manager, which now can automatically take actions to meet the application communication requirements. [RUA13]

3.3.1. Monitoring

When a task requires QoS constraints, all *MESSAGE DELIVERY* packets have in its header a flag indicating that the consumer PE must generate a *MONITORING PACKET* message when the *MESSAGE DELIVERY* is received. This packet is generated in hardware by the NI and is sent to the manager. The monitoring packet contains the task identification, message latency, and message size. The manager processor analyzes all monitoring packets, verifying if the communication constraints are met, and, after a certain number of violations, it may act to restore the application QoS constraints.

3.3.2. Communication Priority

To improve the application latency parameters, the communication flow can change to three possible states: *Low Priority*, *High Priority*, and *Guaranteed Throughput*. All data messages are initially sent as *Low Priority*, using just a single NoC channel and uses the deterministic version of the Hamiltonian routing algorithm. If the manager detects that the latency constraints are violated, a *CHANGE FLOW QOS* message is sent from the manager to the producer PE changing the flow priority to *High*, allowing the data message to use both NoC channels and use the adaptive version of the routing algorithm. If the latency constraints are still violated, the manager changes the flow priority to *Guaranteed Throughput* also using a *CHANGE FLOW QOS* message, establishing a CS channel between the two communicating PEs.

In the case of throughput violations, the priority is raised to *Guaranteed Throughput*, even if the previous priority was *Low*, using the *CHANGE FLOW QOS* message.

The message priorities are automatically downgraded after some time, according to the current priority (5 ms if the priority is set to *HIGH,* 10 ms if the priority is set to *Guaranteed Throughput*).

3.3.3. Circuit Switching

The CS guarantees the maximum throughput and minimum latency between two PE_{SL}. The CS exclusively uses the NoC high priority channel, and blocks this flow to any other message that does not belong to these two PEs [DUA03]. The CS is established using a specialized network packet with the service *OPEN CONNECTION SERVICE*, sent from the producer PE to the consumer PE. When the CS is established, the producer PE signalizes its manager using the *UPDATE CS CONTROL* message. Later, the producer PE can close the connection using a *CLOSE CONNECTION SERVICE* message, and also signalizes to its manager that the CS was closed using the *UPDATE CS CONTROL* message. The manager computes all CS paths to ensure that they only exist inside a cluster and to avoid that two CS tries to use the same path.

3.4. Computation QoS

Besides guaranteeing the communication parameters, another important requirement for RT applications is that its tasks must execute for enough time to complete its processing. The reference platform guarantees this characteristic through the use of a scheduler that ensures this requirement, called *HQoS* [RUA15], allowing the execution of soft real-time applications. This scheduler follows a hierarchical organization, meaning that it is executed both in the cluster manager (global scheduler) and in the PE_{SL} (local scheduler).

3.4.1. Local Scheduler

The local scheduler uses four constraints (shown in Figure 3.4) to determine which RT task must be executed: *(i) period*, since the scheduler follows a periodic behavior; *(ii) deadline*, indicating the maximum acceptable time in a period that the task must finish its execution; *(iii) execution time*, indicating the time that a task must execute at each period; *(iv) utilization*, which is the percentage that a task uses the processor, and is calculated using the following equation: $utilization = (execution\ time * 100)/period$ [LIU00].

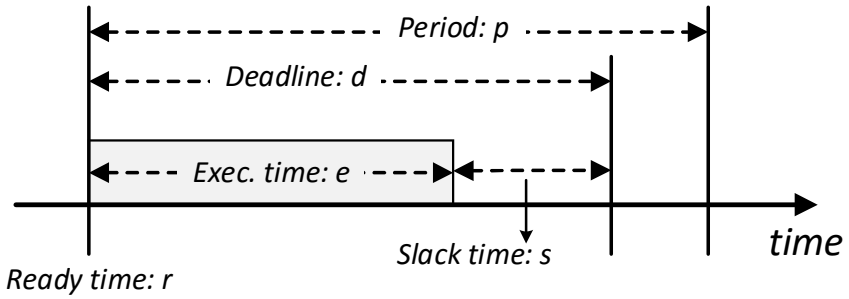


Figure 3.4 – Real-time constraint model. Source [RUA15].

The local scheduler selects the RT task according to the one that currently has the least slack time. This task can also be preempted to execute another, if this new task has entered a new period and has a slack time lower than the current one. After scheduling all RT tasks, the scheduler uses the Round-Robing algorithm to select a BE task, as described in Section 3.2.4. To support the RT scheduling, the scheduler supports the additional *SLEEPING* status, when the task has executed the required time in its period. When the task has this status, it is not scheduled until its next period.

The application developer can use the *RealTime()* syscall to specify the tasks RT constraints. This syscall can be called multiple times during the execution to update the tasks requirements according to the workload characteristics. A *REAL TIME CHANGE* message is also sent to the manager when this syscall is called to inform the manager these new requirements.

When considering the application, this scheduler assumes that the application iteration fits in a hyper period (i.e., all tasks of the application have the same period), meaning that the configured period must cover all tasks constraints as shown in Figure 3.5. Assuming that there are enough communication and computation resources available on the platform, there is a minimum time variation between each application iteration, since no time is spent waiting for messages and availability of the CPU.

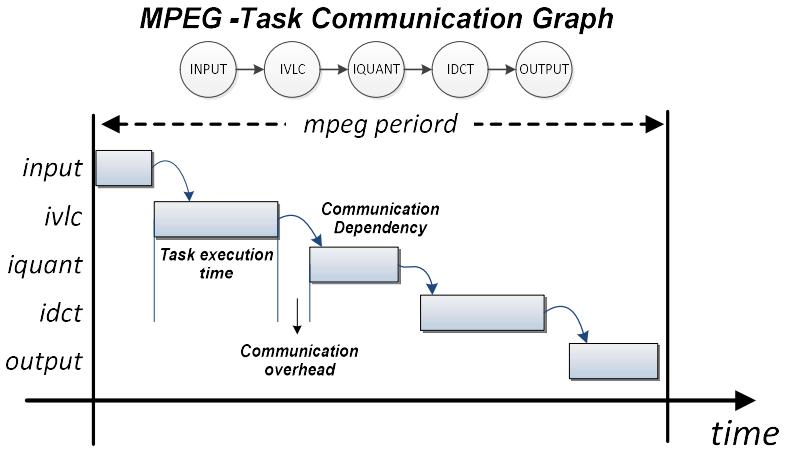


Figure 3.5 – MPEG real-time constraints configuration. Source [RUA15].

3.4.2. Global Scheduler

The global scheduler has two main function: *(i)* selects the PE_{SL} to execute the task according to its utilization (in fact, it is responsible for the mapping); *(ii)* handle monitoring messages containing the PE_{SL} utilization and deadlines misses from the PE_{SL}. The global scheduler

can migrate tasks from one PE_{SL} to another to ensure that the tasks can execute for the required time.

The manager uses a heuristic to select which PE_{SL} a task must be allocated, which aims to find the PE_{SL} that has the most computation resources available. The heuristic is described below, in sequence:

- The manager generates a set containing all the PE_{SL} in the cluster;
- The set is filtered to contain only the slave PEs with the highest average slack time, other PE_{SL} are removed from the set;
- The set is filtered to contain the slave PEs with the smallest number of allocated RT tasks;
- The set is filtered to contain only the slave PEs with the highest absolute slack time (total idle time of the PE_{SL});
- The set is filtered to contain only the slave PEs with the smallest number of BE tasks;
- At the end, the first slave PE in the set is selected to receive the task.

3.4.3. Monitoring

The global scheduler is dependent on two types of monitoring messages, both generated by the PE_{SL}: *(i) DEADLINE MISS REPORT,* sent every time a task misses its deadline; *(ii) SLACK TIME REPORT*, sent every 10 ms, and reports the PE_{SL} absolute slack time, meaning the total time that the PE_{SL} was idle in the last 10 ms.

3.4.4. Task Migration

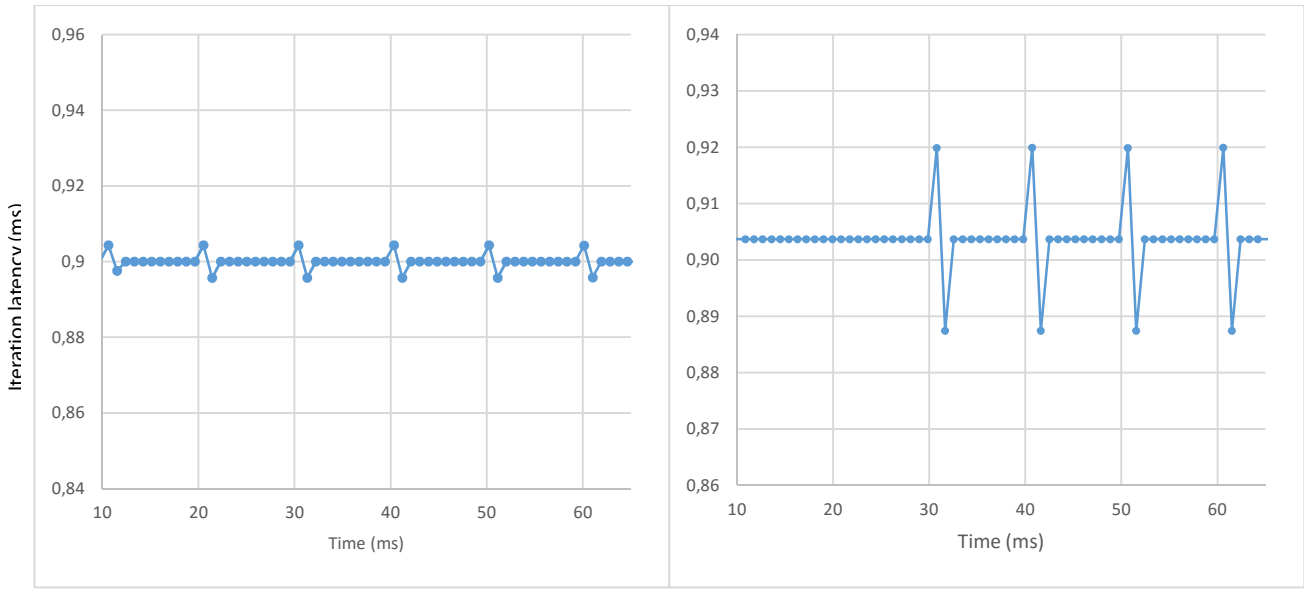
Deadline misses can initiate a task migration if the manager determines that the PE_{SL} has insufficient resources to execute a task. A heuristic similar to the one used to allocate a task is used to migrate tasks. The difference is that an additional criterion (executed first after the set is generated) is used to select the PE_{SL}: *utilization*. Only the slave PEs that have a remaining utilization higher or equal than the task utilization are selected. This heuristic, however, is not ideal, and can be improved in future works. For example, consider a cluster with two PE_{SL} available to execute an RT task. Each one of these PE_{SL} is executing an RT task with a utilization of 40%. Then, a new RT task, with a utilization of 80% must be allocated to one of these PE_{SL}. Using the current heuristic, this task would not be able to fulfill the requirements. The heuristic to fulfill the requirements can

be improved by grouping RT tasks by task migration, and mapping the new task to the PE_{SL} that became available.

The manager PE is responsible for starting the migration. The manager sends a *TASK MIGRATION* message to the PE_{SL}, containing the new location where the task must be migrated. After receiving this message, the PE_{SL} can start its migration process. First, the object code is migrated to the new address. During this process, the task can keep executing. Then, the task is interrupted, and the remaining data is migrated in the following order: TCB, task location table; message requests table; stack memory; BSS memory. After the task is migrated, the original slave sends a *TASK MIGRATED* message to its manager. The original PE may still receive a *MESSAGE REQUEST* after the task was migrated. In this case, this message is redirected to the new PE and a *UPDATE TASK LOCATION* is sent to the consumer PE, indicating the new location of the task [MOR12].

3.4.5. Application profile

To define the QoS constraints, applications must execute alone on the platform in such a way to verify if it is possible to meet the applications requirements. This initial execution corresponds to the application profiling. Figure 3.6 presents the application profile for two applications (MPEG and DTW), i.e., the application is executed without any other application in the MPSoC, and only one task is executed at each PE. There is a warm-up period of 10 ms, not shown in the Figures.



(a) (b)

Figure 3.6 – Computation latency (a) MPEG (b) DTW during profiling.

It is possible to observe that each iteration processing time is kept constant, except the variation observed at every 10 ms for both the MPEG and DTW applications. This variation happens because every PE_{SL} reports its computation stats to its manager every 10 ms using the *SLACK TIME REPORT* message, affecting the application processing.

The iteration latency may present temporary variations if there are other applications in the MPSoC, presenting an interference to the communication and computation resources available to this application. Some other services may also have a temporary impact in the iteration processing time, such as the CS establishment, which stops the application communication until the CS is established. These variations in the iteration time are compensated in the next iterations, meaning that the time required to complete its next iteration is lower than the average, and the following iterations are computed in the average time.

3.5. Supported Service Summary

Table 3.1 lists all services supported by the reference platform. Besides the application data type, all remaining services are considered management services. This table highlights the important number of management services in this platform.

Table 3.1 – Services supported by the reference platform.

Type	Service	Description	Direction
Application Data	*MESSAGE REQUEST*	Request for a *MESSAGE DELIVERY*.	Consumer PE_{SL} → Producer PE_{SL}
	MESSAGE DELIVERY	Message containing data exchanged between the tasks.	Producer PE_{SL} → Consumer PE_{SL} *OR* Original PE_{SL} → New PE_{SL}
PE Initialization	*INITIALIZE CLUSTER*	Initializes the cluster.	PE_{GM} → PE_{LM}
	INITIALIZE SLAVE	Initializes the slave PEs in a cluster.	PE_{GM} → PE_{LM}
Mapping	*TASK ALLOCATION*	Task object code.	PE_{GM} → PE_{SL}
	TASK ALLOCATED	The physical location of the task.	Manager → PE_{SL}
	TASK REQUEST	Requests the task location. If the task is not mapped, also requests its allocation.	PE_{SL} → Manager PE
	TASK TERMINATED	Indicates that a task has finished its execution.	PE_{SL} → Manager PE
	NEW APP	Signalize the start of a new application in a cluster.	PE_{GM} → PE_{LM}
	APP TERMINATED	All application tasks have finished its execution.	PE_{LM} → PE_{GM}
	NEW TASK	Requests the allocation of a task to a PE_{SL}	PE_{LM} → PE_{GM}

Table 3.1 – cont.

Type	Service	Description	Direction
Reclustering	*LOAN PROCESSOR REQUEST*	Requests a resource from another cluster.	Requesting Mng → Other Mng
	LOAN PROCESSOR DELIVERY	Confirm the resource loaning, and informs the address of the PE_{SL} containing the resource	Other Mng → Requesting Mng
	LOAN PROCESSOR RELEASE	Returns the loaned resource.	Requesting Mng → Other Mng
	TASK TERMINATED OTHER CLUSTER	Task executing in the loaned resource has finished its execution.	PE_{SL} → Lender Mng
Communication QoS	*CHANGE FLOW QOS*	Communication flow must change its priority.	Manager → Producer PE_{SL}
	QOS REQUEST SERVICE	Task requires QoS management.	Consumer PE_{SL} → Manager
	MONITORING PACKET	*MESSAGE DELIVERY* statistics.	Consumer PE_{SL} → Manager
	UPDATE CS CONTROL	CS successfully established.	Producer PE_{SL} → Manager
	OPEN CONNECTION SERVICE	Open a CS connection.	Producer PE_{SL} → Consumer PE_{SL}
	CLOSE CONNECTION SERVICE	Closes a CS connection.	Producer PE_{SL} → Consumer PE_{SL}
Computation QoS	*REAL TIME CHANGE*	Task RT constraints	PE_{SL} → Manager
	SLACK TIME REPORT	Reports the PE_{SL} absolute slack time	PE_{SL} → Manager
	DEADLINE MISS REPORT	Task has missed its deadline	PE_{SL} → Manager
Migration	*TASK MIGRATION*	Indicates that a task must migrate to another PE.	Manager → PE_{SL}
	MIGRATION CODE	Task object code.	Original PE_{SL} → New PE_{SL}
	MIGRATION TCB	Task TCB.	Original PE_{SL} → New PE_{SL}
	MIGRATION TASK LOCATION	Task location table.	Original PE_{SL} → New PE_{SL}
	MIGRATION MSG REQUEST	Pending message requests.	Original PE_{SL} → New PE_{SL}
	MIGRATION STACK	Task stack.	Original PE_{SL} → New PE_{SL}
	MIGRATION DATA BSS	Task BSS memory.	Original PE_{SL} → New PE_{SL}
	UPDATE TASK LOCATION	Indicates that future *MESSAGE REQUEST* for the task must be sent to another processor.	Original PE_{SL} → Consumer PE_{SL}
	TASK MIGRATED	Migration procedure complete.	Original PE_{SL} → Manager PE

3.6. Final Remarks

This chapter presented the *first contribution* of this work: the integration of features present in two existing platforms, HeMPS-QoS [CAR11] and HeMPS-RT [RUA15]. This integration required an effort to stabilize both the hardware and the software of the new platform, given the large number of protocols involved (as presented in Table 3.1).

This new HeMPS-QoS platform is the most comprehensive version of the HeMPS platform, providing support for both communication and computation QoS simultaneously.

4. EVALUATION OF THE TRAFFIC BEHAVIOR

This chapter presents the applications used to evaluate the platform and presents the behavior of the platform traffic. Such evaluation is used to guide the design of the MP NoCs.

4.1. Evaluated Applications

4.1.1. MPEG

The MPEG application contains 5 tasks and has its task dependency graph shown in Figure 4.1, characterizing a pipeline communication flow. This application can be configured to support both computation and communication QoS.

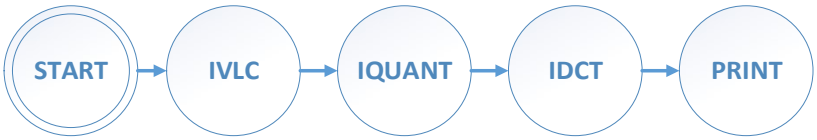


Figure 4.1 – Task dependency graph for the MPEG application.

Figure 4.2 presents the scheduling graph for the MPEG application when using the HQoS scheduler. At each scheduling period, a single MPEG frame is processed. The time required to process a single MPEG frame is around 0.75 ms when there is no other application interfering with the MPEG processing. This application usually only has one task active at a time, because the processing is not balanced between all the available tasks.

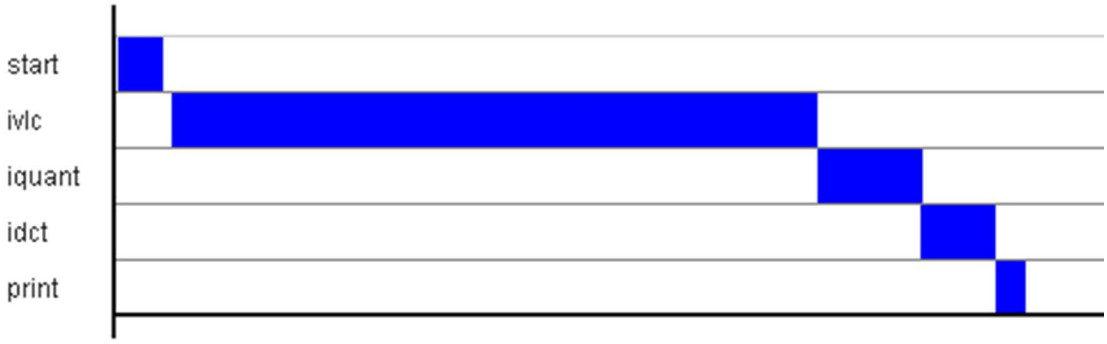


Figure 4.2 – Scheduling graph of the MPEG application.

The MPEG application is evaluated using a scheduling period of 0.9 ms, leaving around 0.15 ms of slack time. The communication performance between all the tasks is also monitored to detect latency and throughput violations.

4.1.2. DTW

The DTW application contains 6 tasks and has its task dependency graph shown in Figure 4.3. This application can be configured to support both computation and communication QoS.

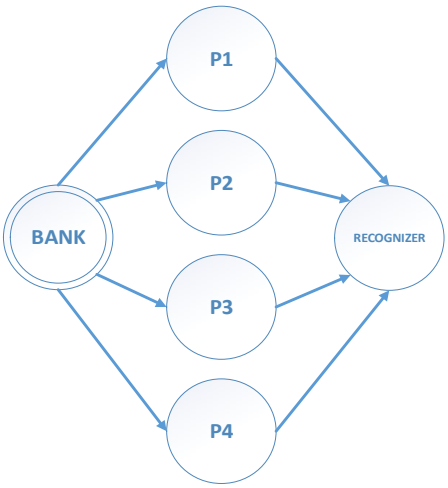


Figure 4.3 – Task dependency graph for the DTW application.

Figure 4.4 presents the scheduling graph for the DTW application when using the HQoS scheduler. This application differs from the MPEG application because multiple tasks are usually executed in parallel. This application can process *N* patterns in parallel, where *N* is equal to the number of worker tasks (four worker tasks are used in the evaluation, P1 to P4). A worker task requires around 0.65 ms to process a pattern.

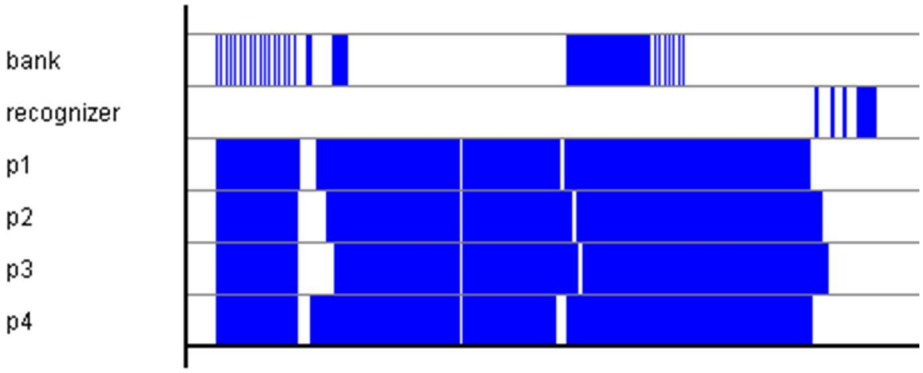


Figure 4.4 – Scheduling graph of the DTW application.

The DTW application is evaluated using a scheduling period of 0.45 ms, and every worker task requires two scheduling periods process a pattern, leaving around 0.125 ms of slack time for each scheduling period. The communication performance between the worker tasks and the bank is also monitored to detect latency and throughput violations.

4.1.3. Synthetic

The Synthetic application contains 6 tasks and has its task dependency graph shown in Fig. 7. Packets are injected at a similar rate to the MPEG application. This application does not have QoS constraints.

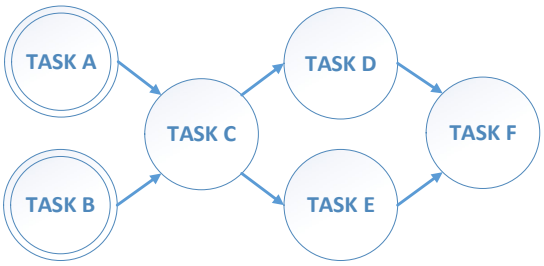


Figure 4.5 - Task dependency graph for the synthetic application.

4.1.4. Producer/Consumer

This application has two tasks and has its task dependency graph shown in Figure 4.6. This application is used in the evaluation mainly to disturb the communication of the other applications. The producer tasks generate messages at the maximum rate supported the PE, and each message has a payload size of 2048 bytes. The average utilization of a communication channel between the producer and consumer tasks reaches an average of 20% of the channel bandwidth when using this application. This rate is far from the maximum bandwidth supported by the channel because the operating frequency of the NoC is equal to the PE, and there is additional delays due to the task and kernel processing. This application does not have communication and computation QoS constraints.

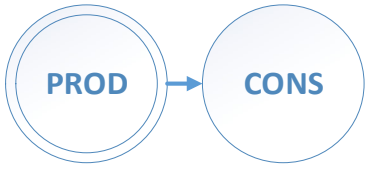


Figure 4.6 – Task dependency graph for the Producer/Consumer application.

4.2. Traffic Evaluation

This section presents the second contribution of this work: evaluation of the volume and spatiality of the management traffic, comparing it to the application traffic. The results listed in this section were extracted from selected test cases that illustrate some common workload in the reference platform. The test cases evaluated here uses three different applications: *(i)* MPEG; *(ii)*

DTW; *(iii)* Synthetic. Both the MPEG and DTW applications have their communication and computation QoS requirements enabled.

All test cases are simulated for 100 ms (10^7 clock cycles), each PE_{SL} can execute simultaneously 2 tasks, and all applications are mapped at the beginning of the simulation. The test cases are configured as follows:

- **Test case 1:** 8 applications (4 MPEGs and 4 DTWs) executed in a 6x6 MPSoC with four 3x3 clusters;
- **Test case 2:** 9 applications (3 MPEGs, 3 DTWs and 3 synthetics) executed in a 6x6 MPSoC with four 3x3 clusters;
- **Test case 3:** 16 applications (8 MPEGs and 8 DTWs) executed in an 8x8 MPSoC with four 4x4 clusters;
- **Test case 4:** 15 applications (5 MPEGs, 5 DTWs and 5 Synthetics) executed in an 8x8 MPSoC with four 4x4 clusters;

Test case 5: 16 applications (4 MPEGs, 4 DTWs and 8 Synthetics) executed in an 8x8 MPSoC with four 4x4 clusters.

Table 4.1 presents a summary of the number of transmitted flits for each TC, divided by traffic categories. The last row indicates the proportion of the management traffic compared to the total traffic. In average, this proportion is between 10% to 15% of the total traffic in the reference platform.

Table 4.1 – Number of flits transmitted by service.

Category	TC 1	TC 2	TC 3	TC 4	TC 5
Application	792,700	752,088	1,453,534	1,270,876	1,336,360
Management	150,158	134,884	293,672	220,410	200,362
% Mng	15.93	15.21	16.81	14.77	13.04

Table 4.2 indicates the services with the most significant participation in the management traffic, presenting the percentage of the flits contributed by this service compared to the total management traffic. Most of the traffic is composed of monitoring messages, which are small, but very frequent in the platform, and task allocation and migration messages, which are less frequent, however have a large packet size.

Table 4.2 – Percentage of flits transmitted by specific management services.

Service	TC 1	TC 2	TC 3	TC 4	TC 5
Communication Monitoring	43.06	35.44	39.89	34.53	25.42
RT Monitoring	6.41	6.14	6.85	8.20	6.55
Task Allocation	39.37	43.56	40.26	44.43	46.11
Migration	4.28	6.57	5.02	4.19	12.09
Others	6.88	8.29	7.98	8.66	9.83

Table 4.3 presents the spatial distribution of the management traffic, in percentage to the total management traffic. The results show that the traffic has a high locality inside the cluster. The traffic between the PE_{GM} and the PE_{SL} from other clusters is also significant, being composed mainly of *TASK ALLOCATION* messages. The traffic between the Intra-cluster traffic and between PE_{SL} is entirely composed of migration messages.

Table 4.3 – Spatial distribution of the management traffic.

Traffic	TC 1	TC 2	TC 3	TC 4	TC 5
Intra-cluster traffic: Manager PE ↔ PE_{SL}	58.94	56.28	54.77	51.60	41.75
Inter-cluster traffic: PE_{GM} ↔ PE_{SL}	34.62	34.71	37.84	41.20	42.56
Traffic between managers: PE_{GM} ↔ PE_{LM}	2.19	2.49	2.38	3.05	3.68
Intra-cluster traffic between PE_{SL}: PE_{SL} ↔ PE_{SL}	4.25	6.52	5.02	4.15	12.01

The evaluation highlighted the behavior of the management traffic: *(i)* important number of injected flits, especially if the application has QoS requirements; *(ii)* monitoring and allocation traffic has an important role in the management traffic; *(iii)* spatial locality between the PE_{SL} and their manager. Such evaluation suggests the usage of an additional network for the management traffic to avoid interferences between the application and management traffic.

5. MANAGEMENT NETWORK DESIGN

This Chapter describes the *main contribution* of this work, the implementation of a *management NoC (M-NoC)*, parallel to the original Hermes-QoS NoC in the HeMPS-QoS platform. Two topologies are explored: *(i)* full mesh interconnecting all the PEs; *(ii)* mesh interconnecting all cluster managers. These M-NoCs topologies are integrated within the platform generation tool, and can be selected or disabled in the platform configuration file. The flits used by the M-NoC can be serialized/deserialized using specific components, allowing a further reduction of the network area. Other topologies are qualitatively evaluated, based on the results presented by the implemented topologies.

The *M-NoC* is used to offload the management traffic from the Hermes-QoS NoC (referred as *Data Network* from now on) while the original network is mainly used to transmit the application data traffic. This enables the *isolation* of the different traffic categories in the platform, reducing interferences between management traffic and application traffic.

The NoC used as the reference for the *M-NoC* in both topologies is the Hermes NoC [MOR04]. Some of its characteristics are listed below:

- *(i)* mesh topology, where each central router has one input port and one output port in each direction (North, South, East, West, Local). Border routers have unused ports removed;
- *(ii)* packets have multiple flits, with header and payload. The header contains the target address and unused bits which can be used to signalize information specific to the packet (the network does use those bits for any purpose). An additional bit transmitted together with the flits indicates the end of packet – this characteristic is modified from the original implementation;
- *(iii)* no QoS support;
- *(iv)* wormhole switching;
- *(v)* deterministic XY routing algorithm;
- *(vi)* no virtual channels;
- *(vii)* each input port has a credit based input buffer;
- *(viii)* centralized round-robin arbitration for the output ports.

Both the input buffer depth and flit width are configurable. The management network implemented in this work adopts a fixed buffer depth of two flits and the flit-width is configurable. The adoption of a 2-flit depth buffer aims the minimization of the area overhead of the M-NoC.

The M-NoC is simpler and considerably smaller than the data NoC employed in the platform mainly due to having fewer and smaller buffers. The data router has a buffer depth of 8 flits, while the management router has a buffer depth of 2 flits. The data router also has 10 buffers in total, since it has duplicated physical channels, while the management router has 5 buffers. The flit width of both routers in this evaluation is of 16 flits. Table 5.1 lists the central router area results for both networks using the buffer and flit configuration described previously. The central router has input ports in all directions, being the largest possible version of this router. The routers were synthesized using a 65 nm technology targeting the operating frequency of 100 MHz. The area of management router is 30.7% of the data router area. In both cases, most of the area is spent on the buffers, accounting for 60.7% of the data router area and 63.5% of the management router area.

Table 5.1 – Router area comparison for different network types (Lib. CORE65GPSVT, 1.0V, 25º C)

	Area (μm²)	
Component	**Data Router**	**Management Router**
Buffer (Avg.)	3,290	1,689
Crossbar	5,007	1,426
Control Logic	3,497	1,794
Router	43,361	13,294

MP networks is chosen over the use of VCs due to the possibility of adjusting the network parameters, such as buffer depth and flit size, individually for each network. In addition, most of the router area is spent in the buffers and crossbar, and this characteristic is not improved by the use of VC, since each input port requires a number of input buffers equal to the number of VC used by the network. Furthermore, when using shallow buffers, MP networks are more area and energy efficient when compared to networks using VCs [YOO13].

5.1. Full Mesh M-NoC

In this topology, represented in Figure 5.1, the *M-NoC* interconnects all the PEs in the network alongside the data network. All management traffic traverses through the management

NoC. The data NoC is dedicated to the application data services (as specified in Table 3.1), and to the *OPEN CONNECTION SERVICE* and *CLOSE CONNECTION SERVICE* packets, since they are used to establish/close the CS, which is only supported by the data NoC.

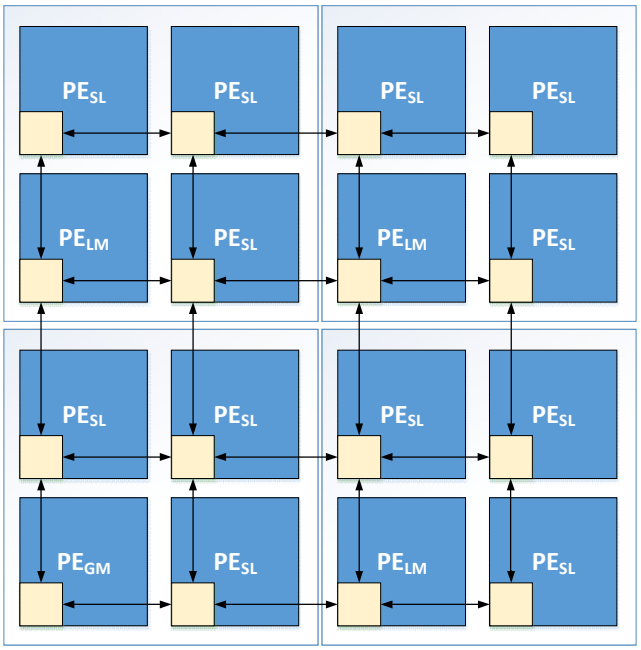


Figure 5.1 – Full mesh management NoC topology.

The communication to the network is controlled by the NI, which supports three physical channels (two for the data network, and one for the *M-NoC*). The NI is divided in receive and send blocks, which allows the NI to simultaneously send and receive packets. The NI is also responsible for serializing/de-serializing the flits, since the processor word width is 32 bits while the interface used by both networks is 16 bits. Smaller flit widths for the M-NoC are serialized outside the NI in dedicated modules.

The receive block of the NI contains control logic and an input buffer, which has the same depth of the buffers used in the data network. Both the NI control logic and the input buffer are shared between all physical channels. Thus, it is only possible to receive from a single channel at a time. Packets in other channels must wait until the first packet is fully received. When a flit is stored in the input buffer, the processor is interrupted so that the microkernel can configure the DMA to receive the rest of the packet.

Figure 5.2 presents the receive control logic FSM. Initially, the control logic is in the *Network* state, waiting the arrival of a packet from one of the networks. In this state, the NI signalizes to all NoCs that there is no credit available in the input buffer, even if the input buffer is not full. When one of the NoCs signalize the availability of packet through the *rx* signal, and there is space

available in the NI input buffer, indicated by the *slot_available* signal, the control logic goes to the *Header* state to consume the header. Then, it alternates between the *Receive Low* and *Receive High* states to de-serialize the packet, until the *eop_i* signal is active, indicating the end of the packet.

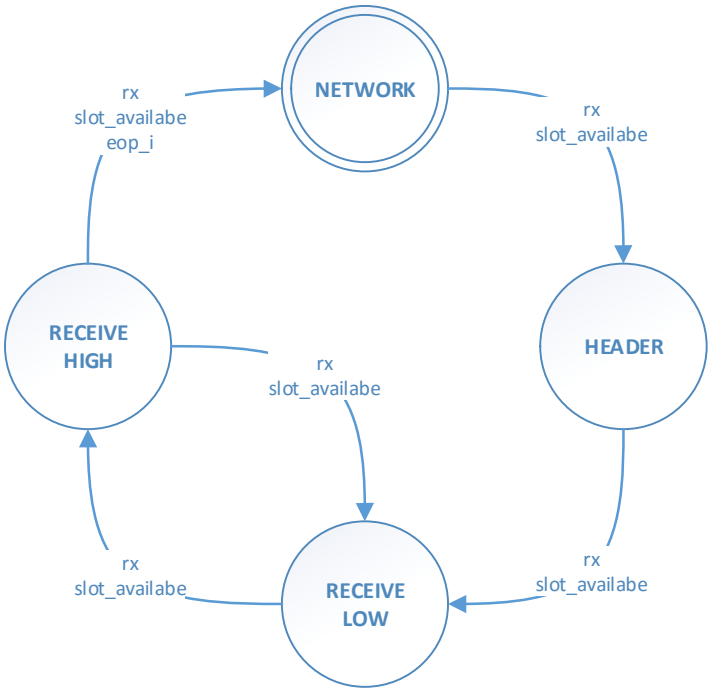


Figure 5.2 – NI receive control logic FSM.

The NI also generates monitoring packets after receiving a *MESSAGE DELIVERY* packet that has a monitoring flag set in its header. When a packet of this type is received, the NI captures specific positions of the packet that contains the packet size (used for the throughput control), timestamp (used for the latency control), consumer task identification and producer task identification. This information is used to generate the monitoring packet in the send block.

The send control logic is also shared for all output NoCs. Thus, the network can only send packets to one channel at a time. The NI selects the network that a packet is going to be injected according to an additional control word written to the NI before the packet contents. Packets using the wormhole switching are injected into the low priority channel, regardless of the packet priority, while packets using the CS are injected into the high priority channel. This allows simplifying one of the router input buffers since high priority packets using wormhole switching can change to the high priority channel inside the network, and does not represent a bottleneck since the send control logic can just send a single packet at a time

Figure 5.3 presents the send control logic FSM. Initially, the control logic is in the *Network* state, waiting the DMA to start transmitting the packet contents from the memory. The DMA uses the *send_data* signal to indicate that a word is available to be sent to the NoC. The first word is

consumed by the NI and used to select the network that the packet is going to be sent. Then, the header is sent to the network in the *SEND HEADER* state. The *CHANGE_NETWORK* is a special flag set in the header that indicates that a packet can traverse multiple networks, thus the packet has multiple headers. It is not used in the full mesh management network topology, however, it is used in the management topology described later. The circuit state can advance if there is a new word available from the memory and there is credit available in the NoC input buffer, indicated by the signal *credit_i*. The payload size must follow the header, and is used by the NI to control the number of remaining flits in the packet. After sending the packet size, the flits are serialized and sent to the NoC in the *SEND HIGH* and *SEND LOW* states.

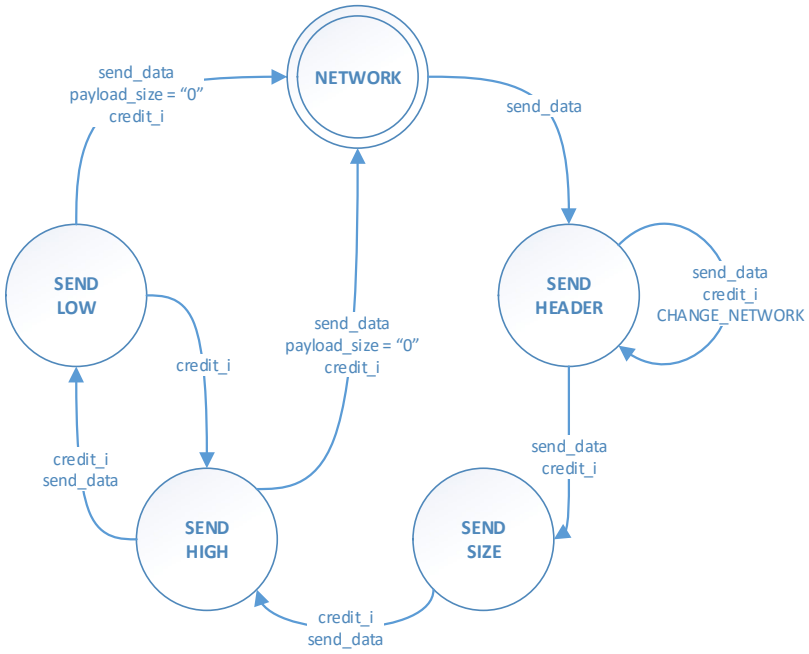


Figure 5.3 – NI send control logic FSM.

The monitoring packet is sent to the manager PE when the send control logic is in the *Network* state, and no other packet needs to be sent. In this situation, the send logic is temporarily disabled, and a packet with a fixed structure is generated according to the information captured from the *MESSAGE DELIVERY* packet.

At the software layer, in the microkernel, all packets are sent by the *send_packet()* procedure. This procedure verifies the packet service, configures the packet to be sent to the correct network, and prepares the packet header to be compatible to the selected network. In the case of the data network, the header must include its QoS flags and the address must be converted to the Hamiltonian address, since all address are represented as XY in the microkernel. Figure 5.4 presents a snippet of the code responsible for these actions.

```
if (p->service == MESSAGE_DELIVERY ||
    p->service == MESSAGE_REQUEST  ||
    p->service == OPEN_CONNECTION_SERVICE ||
    p->service == CLOSE_CONNECTION_SERVICE) {
      p->header1 = DATA_NETWORK;
      p->header2 = get_qos_flags(p->net_priority, p->qos_flags) |
                   xy_to_ham_addr(p->target_PE);
} else {
      p->header1 = MNG_NETWORK;
      p->header2 = p->target_PE;
}
```

Figure 5.4 – Source code responsible for selecting the network and adapting the packet header.

5.2. Mesh between the managers

In this topology, represent in Figure 5.5, the management network interconnects only the manager PEs. The wire size between each router is kept small through the inclusion of repeaters.

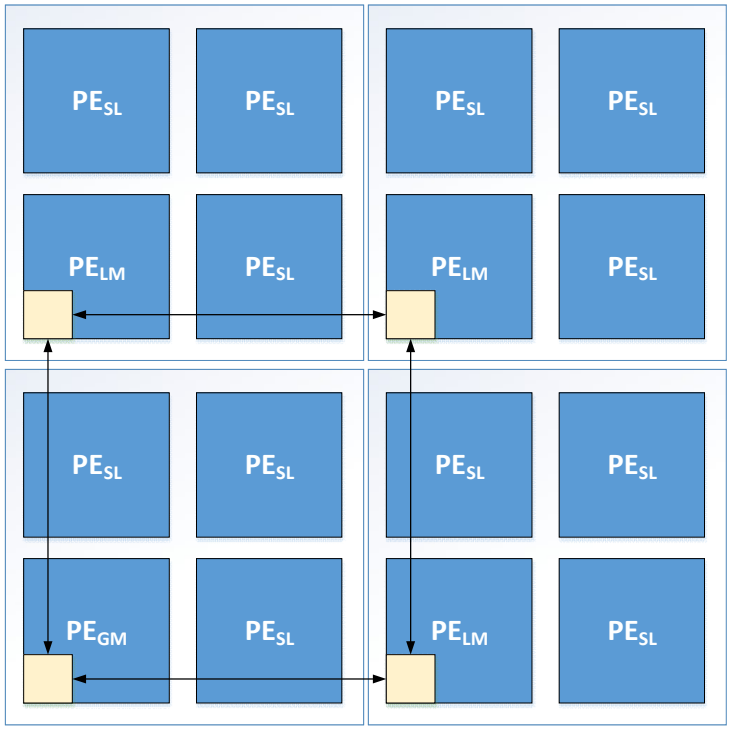


Figure 5.5 – Mesh between the masters management topology.

This network is used for the management traffic between different clusters. The application and CS packets are constrained to the data network. However, in this topology, the data network also transmits intra-cluster management packets.

The main motivation behind this topology is to avoid the interference between traffics from different clusters, using a low-cost strategy. In addition, the manager inter-cluster communication requires fewer hops since it is not necessary to cross multiple routers to reach its destination, wasting fewer clock cycles due to router switching.

To reach its destination, a packet can take a path traversing both the management and data networks. To support this path, a packet can have multiple headers, equal to the number of networks that this packet will traverse. Each header contains the network target address and has a special flag that indicates if the packet must change its network when it reaches its target.

Packets can only change the network in the manager PEs. Packets that must change its network are delivered to manager NI and then, redirected to the new network. Every time that this procedure is executed, the NI consumes a header flit.

All managers in this topology uses a modified version of the NI described in the Section 5.1. The PE_{SL} uses the same NI described previously, however, the signals which were previously connected to the management network are grounded, and is not necessary to generate an additional control word to select the packet network. All PE_{SL} packets are automatically sent to the data network.

Figure 5.6 presents the receive control logic FSM for the manager NI. When a packet reaches a manager NI and has the *CHANGE_NETWORK* flag in its header, the control logic advances to the *CONSUME_HEADER* state, removing the first header and then advances to the *CHANGE_NETWORK* state in the next cycle. In this state, the control logic verifies if the packet was sent in its entirety to the other network through the *eop_i* signal, incoming from the NoC.

Figure 5.7 presents the send control logic FSM for the manager NI. When the receive logic is in the *CHANGE_NETWORK* state, the send logic advances to the *WAIT CHANGE NETWORK* state. This state connects the signals from the NoC that are used to receive packets (*data_i, credit_o, eop_i, rx*), to the NoC signals that are used to send packets (*data_o, credit_i, eop_o, tx*). If the packet arrives from one the channels from the data network, the packet is redirected to the management network. Conversely, if the packet arrives from the management network, it is redirected to data network low priority channel.

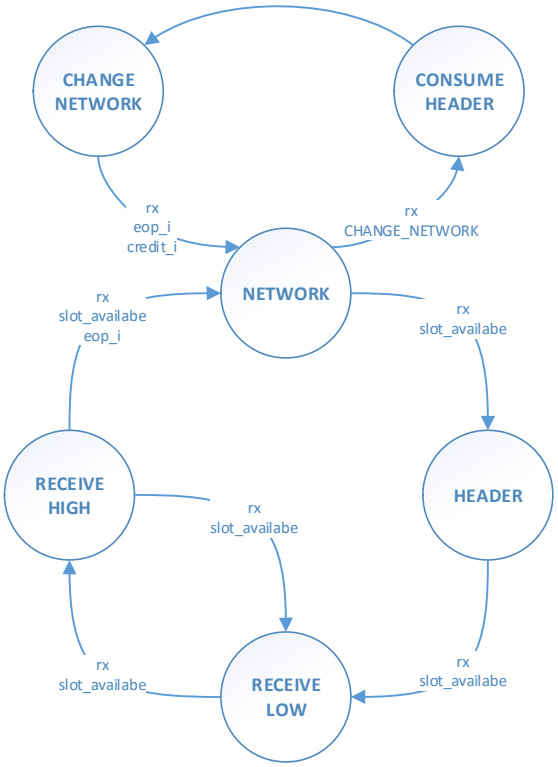


Figure 5.6 – NI receive control logic FSM supporting network change.

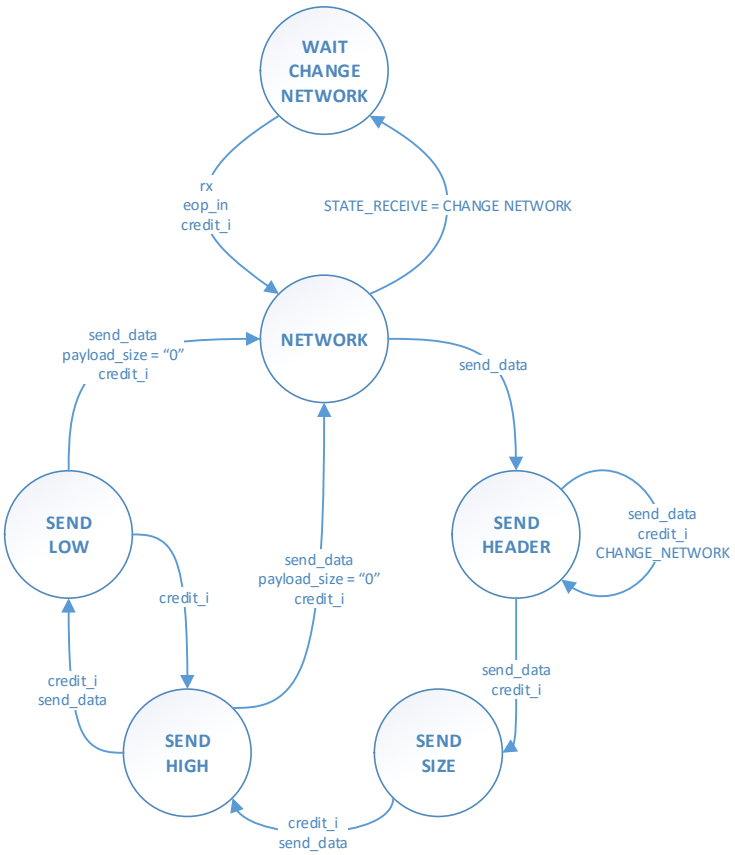


Figure 5.7 – NI send control logic FSM supporting network change.

The change network operation uses the receive and send portions of the NI, thus, the manager cannot receive nor send packets until the operation is completed.

In the software layer, the *send_packet()* procedure computes the route used by the management packets using the algorithm shown in Figure 5.8 (*CHANGE_NETWORK* is abbreviated to CN). **Lines 2 to 17** are responsible for computing the route when the target PE is in a different cluster than the sender PE. **Lines 3 to 9** generates the headers required when the target is a PE_{SL} in another cluster, thus requiring a network change. **Lines 11 to 16** are used when the target is a manager in a different cluster. **Lines 19 to 22** are used when the target is in the same cluster.

```
 1  IF target is in a different cluster THEN
 2      IF target is not a manager THEN
 3          IF sender is a manager THEN
 4              add control word to use the management network
 5          ELSIF sender is not a manager THEN
 6              add header targeting the sender manager with CN flag
 7          END IF
 8          add header targeting the cluster address with CN flag
 9          add header targeting the final destination
10      ELSIF target is a manager THEN
11          IF sender is a manager THEN
12             add control word to use the management network
13          ELSIF sender is not a manager THEN
14             add header targeting the sender manager with CN flag
15          END IF
16          add header targeting the final destination
17      END IF
18  ELSIF target is in the same cluster THEN
19      IF sender is a manager THEN
20         add control word to use the data network
21      END IF
22      add header targeting the final destination
23  END IF
```

Figure 5.8 – Algorithm used to calculate management packet routes.

5.3. Serialization/Deserialization

To support smaller flit widths in the M-NoC, two modules, a serializer and a deserializer modules interfaces the NI with the M-NoC. These modules are automatically inserted when defining smaller flit widths for the M-NoC. The goal to use such modules is to reduce the area overhead due to the additional network. Table 5.2 presents an area consumption comparison for two different flit widths (16 bits and 8 bits). Reducing the flit width allows to reduce the router area by 32%, due to a reduction in the area of buffer and crossbar.

Table 5.2 – Router area comparison for different flit widths (Library CORE65GPSVT, 1.0V, 25º C).

	Area (µm²)	
Component	**Flit Width 16**	**Flit Width 8**
Buffer (Avg.)	1,689	1,081
Crossbar	1,426	877
Control Logic	1,794	1,688
Router	13,294	9,030

5.3.1. Serializer

The serializer module is inserted between the NI data output signals and the M-NoC local input port signals. Figure 5.9 presents the serializer module interface.

Figure 5.9 – Serializer interface.

The serializer module stores the flit received from the NI in a register, and follows the FSM shown in Figure 5.10 to serialize the flit to the M-NoC. The *HEADER* state formats the packet header for the M-NoC, due to the flit width difference. The *PAYLOAD* state extracts parts (from the lowest to the highest part) of the flit stored in the register and send it to the network. The *END* state assures that the last flit and its EOP signal is correctly received by the M-NoC.

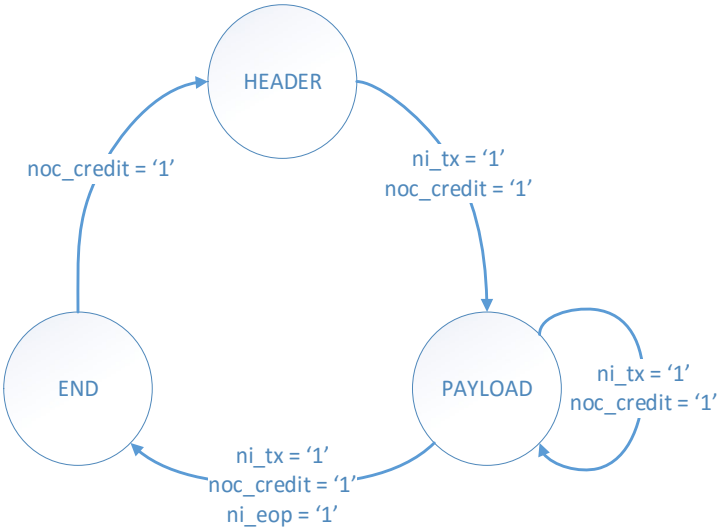


Figure 5.10 – Serializer FSM.

5.3.2. Deserializer

The deserializer module is inserted between the M-NoC output port signals and the NI data input signals. Figure 5.11 presents the deserializer module interface.

Figure 5.11 – Deserializer Interface.

The deserializer module reads the flit received from the M-NoC, deserializes it, storing the NI flit in a register during this process and signalizes the NI when a flit compatible with the NI flit width is ready. Figure 5.12 presents the FSM used to deserialize the flit from the M-NoC to the NI. The *HEADER* state adapts the packet header for the format used by the NI. The *PAYLOAD* state reads the flit from the M-NoC and stores it in a part of the register, until it has read enough flits to complete a NI word. The END state ensures that the last flit and its EOP signal are correctly received by the NI.

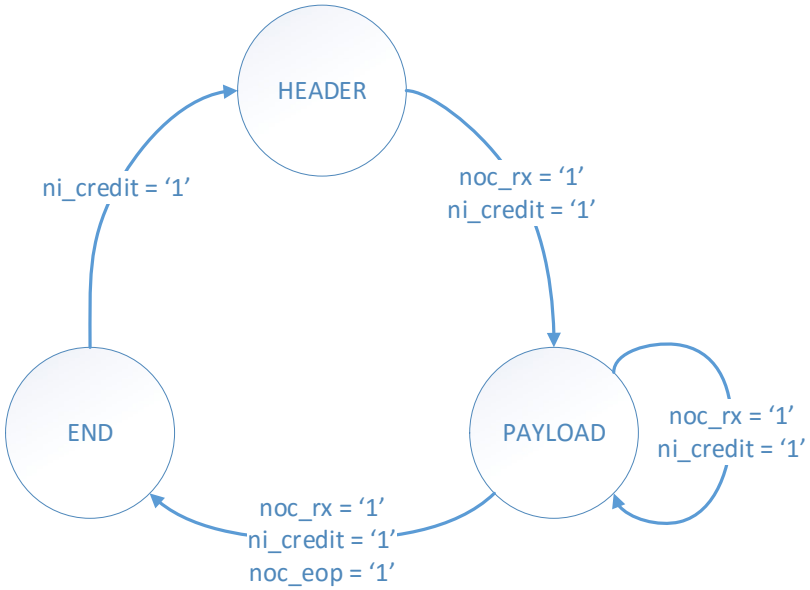


Figure 5.12 – Deserializer FSM.

5.4. Management Network Evaluation

This section evaluates the implemented M-NoC network topologies, and compares it to the original implementation with a single network.

5.4.1. Experimental Setup

To compare the different M-NoC topologies, six different scenarios are simulated in the platform. Scenario 1 to 3 are executed in a 6x6 MPSoC with four 3x3 clusters. Scenario 4 to 6 are executed in an 8x8 MPSoC with four 4x4 clusters. All scenarios are simulated for 100 ms, and each PE_{SL} has two memory pages. Each scenario executes multiple instances of the application, occupying all PE_{SL} pages in the system, and the application is reallocated when one instance of it finishes. The applications are listed below:

- **Scenario 1:** DTW;
- **Scenario 2:** MPEG;
- **Scenario 3:** DTW, MPEG, Synthetic;
- **Scenario 4:** DTW;
- **Scenario 5:** MPEG;
- **Scenario 6:** DTW, MPEG, Synthetic.

The DTW and MPEG application only have their communication QoS enabled. The computation QoS has a large impact on the execution time, since it ensures that the task only executes for the defined execution time in a single period, therefore, the computation QoS is disabled for this evaluation. The results are extracted from packet and application logs and analyzed using scripts written in Python, developed for the evaluation of this work.

5.4.2. Application Messages Network Latency and Jitter Evaluation

Table 5.3 presents the average network latency and jitter for the *application packets*, considering different management NoC topologies and flit widths. The results without the M-NoC are also included, and in this case, the network latency and jitter are equal for both M-NoC flit width columns. The latency is the number of clock cycles between the injection of the header flit into the network until it reaches its target. Since only the first flit of the packet is considered in the evaluation, and the NoC uses the wormhole switching mechanism, the results are independent of the packet size, which has a large variation in this platform. The jitter is the standard deviation of the network latency.

The first evaluation concerns the 16 and 8-bit "*Full-mesh M-NoC*". The non serialized "*Full-mesh M-NoC*" presents better latency and jitter values than the implementation without the M-NoC, reducing the average network latency in 30% to 70% (46% in average between the evaluated

scenarios) and the jitter in 40% to 80% (60% in average). The serialized version of the M-NoC still brings improvements over the version without the M-NoC, improving the network latency in 20% to 60% (40% in average) and the jitter is improved up to 65% (37% in average).

Table 5.3 – Average Application Network Latency and Jitter (Clock Cycles) for different M-NoC configurations

		M-NoC 16-bit Flit Width Non serialized		**M-NoC 8-bit Flit Width Serialized**	
Scenario	**M-NoC**	**Avg. Lat.**	**Jitter**	**Avg. Lat.**	**Jitter**
1	None	41.3	257.4	-	-
	Full-Mesh	23.4	122.5	25.6	109.8
	Mesh Mng	31.0	236.1	31.2	270.4
2	None	40.1	311.4	-	-
	Full-Mesh	26.3	179.2	32.5	378.8
	Mesh Mng	39.9	283.9	49.5	499.5
3	None	33.3	230	-	-
	Full-Mesh	23.9	72.5	25.4	103.7
	Mesh Mng	26.7	192.1	51.1	504.9
4	None	113.7	792.5	-	-
	Full-Mesh	35.9	157.5	46.6	271.0
	Mesh Mng	106.7	841.4	119.2	1266.1
5	None	95.3	562.2	-	-
	Full-Mesh	34.3	139.6	37.5	390.3
	Mesh Mng	85.6	574.6	150.8	1160.0
6	None	64.8	463.8	-	-
	Full-Mesh	37.1	267.9	40.2	302.8
	Mesh Mng	61.2	433.4	78.6	663.9

The second evaluation concerns the 16 and 8-bit "*Mesh between the managers M-NoC*" (Mesh Mng. rows). The "*Mesh between the managers M-NoC*" also presents performance improvements when compared to the implementation without the M-NoC. The average network latency is reduced up to 25%, being improved by 11.1% in average between the evaluated scenarios. The jitter improves up to 17% (5% in average), while having a small degradation up to 2% in some specific cases. The serialized version of the M-NoC presents a network latency degradation up to 53% (22% higher latency in average) and the jitter is degraded in 5% to 120% (66% in average), when compared to the version without the M-NoC. The use of serialization when

the packet is transmitted between networks with different flit width present a degradation because it causes congestions in the network which has the larger flit width.

5.4.3. Management Messages Network Latency and Jitter Evaluation

Table 5.4 presents the average network latency and jitter for the *management packets*, considering different management NoC topologies and flit widths. The results without the M-NoC are also included, and in this case, the network latency and jitter are equal for both M-NoC flit width columns. The latency and jitter for the management packets are defined in the same manner as the application packets.

Table 5.4 – Average Management Network Latency and Jitter (Clock Cycles) for different M-NoC configurations

		M-NoC Flit Width 16		**M-NoC Flit Width 8**	
Scenario	**M-NoC**	**Avg. Lat.**	**Jitter**	**Avg. Lat.**	**Jitter**
1	None	329.8	1256.8	-	-
	Full-Mesh	331.6	1246.2	366.5	1088.3
	Mesh Mng	580.2	1721.8	755.1	2336.2
2	None	126.3	593.3	-	-
	Full-Mesh	154.6	662.3	395.1	1168.5
	Mesh Mng	296.2	1450.7	679.3	2343.8
3	None	357.0	1087.2	-	-
	Full-Mesh	315.5	1248	362.7	1276.7
	Mesh Mng	429.4	1328.3	788.6	2785.7
4	None	758.0	3019.4	-	-
	Full-Mesh	648.3	2593.5	1326.7	3867.4
	Mesh Mng	1330.8	4796	2878.8	8514.2
5	None	603.6	1907.7	-	-
	Full-Mesh	618.4	1918	1016.8	2457.5
	Mesh Mng	976.8	2897.2	2004.6	5435.1
6	None	710.6	2631.8	-	-
	Full-Mesh	599.8	2106.8	795.1	2033.1
	Mesh Mng	891.4	2982.2	1557.5	4077.3

For the management traffic, the improvements when using the M-NoC are less noticeable when observing the average network latency. It is possible to observe that the average latency is significantly higher for management packets than application packets. This emphasizes the

behavior of the reference platform: when the PE is receiving consecutive packets with a small interval, the first packet is treated immediately, however subsequent packets have to wait in the network until past packets are processed. Since multiple PE_{SL} generates traffic addressed to the same manager PE, this situation is more evident for management packets. The use of a management network reduces congestions for the application, since the management packets are constrained to a different network.

The average latency for management packets when using the "Full-mesh M-NoC" ranges between 20% higher in some scenarios, and 15% lower in others. The average latency between all scenarios presents a small improvement of 2% when compared to the version without the M-NoC. The jitter values range between 15% higher and 20% lower, being the average similar when compared to the version without M-NoC.

For the "Mesh between the managers M-NoC", the average latency is increased in 20% to 135% when compared to the version without the M-NoC (65% higher in average). The jitter increases in 13% to 144% when compared to the version without the M-NoC (55% higher in average).

The serialization had a large performance degradation for the management traffic. For the "Full-mesh M-NoC" the average network latency increased by 63% and the jitter increased by 22% when compared to the version without the M-NoC. The "Mesh between the managers M-NoC", the average latency increased by 220% and the jitter increased by 160%.

5.4.4. Execution Time

Table 5.5 lists the average application execution time, in milliseconds, for different M-NoC configurations. When using the M-NoC flit width of 16 bits, the full-mesh presented an average speed up of 3.1% compared to the single network version, while the mesh between the manager did not present a significant difference in the execution time.

When using serialization, the results presented a small degradation. For the serialized full-mesh network, the average execution time increased by 1.5% when compared to the not serialized full-mesh network, while still being 1.7% faster than the version without M-NoC. The serialized version of the mesh between the managers increases the average execution time by 2.1% compared to the non-serialized version, and by 2.3% compared to the version without M-

NoC. The increase in the execution time for the serialized version is mainly due to the increased time required to allocate the intermediate application tasks.

In overall, the impact of the M-NoC in the execution time is small (the 16-bit Full-Mesh reduced in average 3.1% the execution time). This is part explained when observing that the average network latency of the application packets is not very high, being around 100 cycles even for the worse cases. Applications with larger network latencies are more significantly impacted.

Table 5.5 – Execution Time (milliseconds) with different M-NoC configurations

			M-NoC Flit Width 16		**M-NoC Flit Width 8**	
Scenario	**Application**	**No M-NoC**	**Full-Mesh**	**Mesh Mng**	**Full-Mesh**	**Mesh Mng**
1	DTW	34.9	33.2	35.9	33.8	36.1
2	MPEG	31.3	31.1	30.7	31.5	30.9
3	DTW	30.1	28.8	30.0	29.5	31.3
	MPEG	35.6	34.6	35.1	34.6	35.3
	Synthetic	45.3	42.7	43.4	43.7	44.6
4	DTW	32.6	31.8	33.5	32.2	34.2
5	MPEG	30.6	30.5	31.8	31.1	32.2
6	DTW	27.5	28.3	27.9	28.4	29.0
	MPEG	34.7	32.2	35.8	32.9	35.9
	Synthetic	43.8	41.4	42.2	41.8	44.0
Average execution time compared to a system without M-NoC			-3.1%	+0.3%	-1.7%	+2.3%

5.5. Qualitative Evaluation of Other M-NoC Topologies

Other management network topologies are qualitatively analyzed in this Section based on the results obtained from the two M-NoC topologies, traffic behavior of the platform and the evaluations conducted on the implemented M-NoC topologies. The other possible topologies for the M-NoC are described as below.

- *Hierarchical mesh:* the M-NoC topology shown in Figure 5.13 follows a two-level hierarchy. The mesh topology is adopted for both levels. The first level of this hierarchy interconnects the PEs in a cluster and the second level interconnect the clusters. A specific router is connected to both hierarchy levels, allowing the communication

between different clusters. This topology is a hybrid between the “Full-mesh network” and the “Mesh between the managers network”.

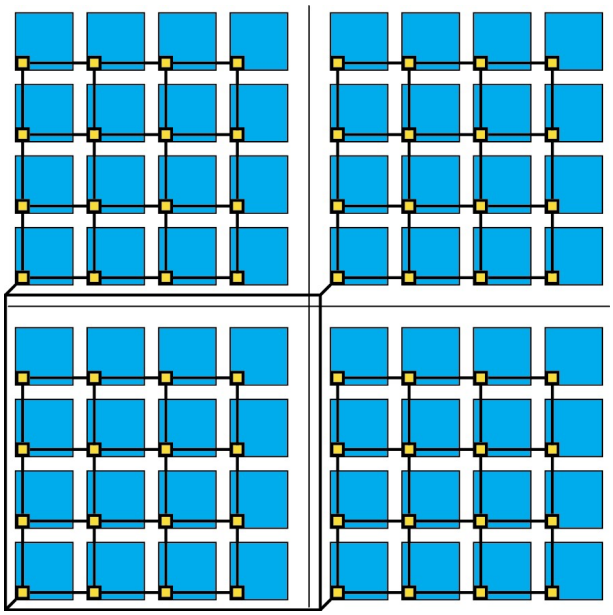


Figure 5.13 – Mesh topology network interconnecting the cluster PEs and another mesh topology interconnecting the clusters.

- *Cluster ring and mesh:* The M-NoC topology proposed in Figure 5.14 also follows a two-level hierarch, and adopts a similar organization, where the first level interconnects the PEs in a cluster and the second level interconnect the clusters. However, instead of using the mesh topology for the cluster network, the PEs are connected using the ring topology.

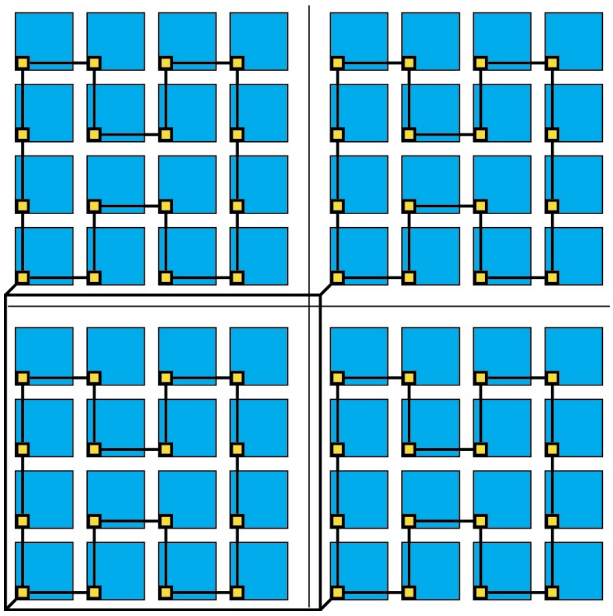


Figure 5.14 – Ring topology network interconnecting the cluster PEs, and a mesh topology interconnecting the clusters.

- *Clos:* The M-NoC topology presented in Figure 5.15 is based on the clos network [PAS08]. Up to four PEs are connected to a first stage router. The middle stages connect the routers in the same cluster and to other middle stage routers.

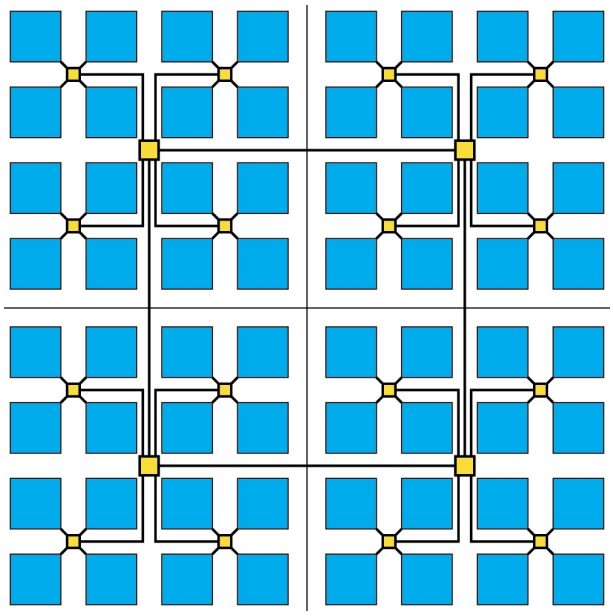


Figure 5.15 – Clos topology network, where a single router interconnects multiple PEs in the cluster, and the higher levels interconnect the clusters.

The motivation behind the proposed topologies is to reduce the number of overall input buffers in the network (which are responsible for the largest area consumption). Considering a central 4x4 cluster (with other clusters in its North, South, East and West), the *hierarchical mesh* allows to reduce the number of input ports by 15%, from 80 to 68 (64 ports in the cluster plus 4 second level ports). The *cluster ring and mesh* allows to reduce the number of input ports by 55%, from 80 to 36 (32 cluster ports plus 4 second level ports). The *clos* reduces the number of input ports by 65%, from 80 to 28 (5 ports for each first stage router and 8 ports for the middle stage).

The *hierarchical mesh* increases the number of hops required for the PE_{SL} when communication between different clusters is required. This type of communication is infrequent in the platform, but is required in some reclustering and migration cases. The *cluster ring and mesh* topology increases in overall the number of hops for the management packets. The *clos* network reduces the number of required hops, since there are less routers in the platform.

These proposed networks tend to concentrate the traffic in a single point. The *hierarchical mesh* and the *cluster ring and mesh* topologies increases the number of congestions near the router capable of redirecting the packet to other network level. Furthermore, the lower number

of possible paths for the *cluster ring and mesh* and *clos* topologies may increase the number of congestions in the network if the remaining network parameters are maintained constant.

The *hierarchical mesh* and *cluster ring and mesh* topologies require additional processing in software to account the multiple network levels, similar to the *mesh between the managers* M-NoC.

The implementation of the *cluster ring and mesh* for variable cluster sizes is not trivial, since it requires a Hamiltonian cycle inside the cluster, which may not be possible for certain cluster sizes.

Both the *hierarchical mesh* and *cluster ring and mesh* present a single point of failure for an entire cluster, since only one router connects an entire router.

5.6. Final Remarks

This chapter presented the main contribution of this work, the implementation of MP NoCs in the reference platform isolating the management traffic from the application traffic. The evaluation showed that this strategy brings benefits to the application communication parameters.

Among the proposed topologies, the Full-mesh topology simplifies the routing and implementation of MP networks, being the chosen management topology for the reference platform. The use of serialization is a promising path, since it presents a reduction in the area consumption with a small performance impact when compared to the non-serialized version.

6. PLATFORM EVALUATION

In this chapter, several scenarios are evaluated aiming to demonstrate the influence of the different QoS strategies. Three different platform configurations are used for each scenario:

(i) *compQoS* - with computation QoS mechanisms (scheduler and task migration), without communication QoS and not using the M-NoC;

(ii) *fullQoS* - with computation and communication QoS mechanisms, not using the M-NoC;

(iii) *fullQoS_MNoC* - with computation and communication QoS mechanisms, using the M-NoC.

The results without communication QoS and using the M-NoC are not included in this evaluation since the amount of management traffic is significantly lower because there are no communication QoS monitoring packets, on RT monitoring packets. The RT monitoring traffics is smaller than the communication monitoring traffic, according to Table 4.1.

At each scenario, a specific application being affected by several communication interferences is thoroughly analyzed according to its computation and communication latency. The management traffic latency of each scenario is also evaluated.

The *computation latency* is defined as the time (in clock cycles) required to execute an iteration of the application subject to RT, which is the sum of the execution time of all tasks in the period, with the the computation interferences and the communication waiting time. For example, for the MPEG application, the computation latency is the time from when the encoded frame starts being read by the *START* task until it is completely decoded and finally presented by the *PRINT* task. Another example, the DTW application, the computation latency is the time from when the pattern is initially prepared to be sent by the *BANK* to the first worker task until the reception of the last result received by the *RECOGNIZER* task. Therefore, the computation latency is subject to several interferences, as resource sharing in the same processor, treatment of interruptions generated from incoming packets, and traffic congestion in the NoC.

The *network latency* is defined as the time (in clock cycles) from when the message header flit is initially injected into the network by the NI until its consumption at reception in the target NI. Analyzing the latency considering just the header allows the latency value to be independent of packet size, which has a large variation in this platform. The jitter is the standard deviation of

the network latency. The results evaluates the network latency and jitter for each application communication flow and for the management traffic.

All scenarios in this Chapter are simulated for 75 ms, and consider a warm up time of 10 ms. The scenarios use a static mapping, aiming to demonstrate the impact of the disturbing traffic over specific flows. For the results using the M-NoC, the adopted topology in all scenarios is the *Full-Mesh*, since it presented real improvements, as shown in the management network evaluation. The results are extracted from packet and application logs and analyzed using scripts written in Python, developed for the evaluation of this work.

6.1. Scenario 1

This scenario consists in an MPEG application with computation and communication QoS and 5 disturbing tasks. Figure 6.1 presents the task mapping (MPEG application is shown in green, disturbing applications are shown in orange) and communication paths (shown in red) of the disturbing applications for this scenario. The mapping utilized in this scenario aims to create a case where the communication flow of the disturbing tasks mainly affects the communication between the final tasks of the MPEG application, IDCT and PRINT. The utilization of the low priority channel in the path used for the communication between those tasks is consistently kept around 100% due to the disturbing applications. The remaining flows of the MPEG application are not heavily affected.

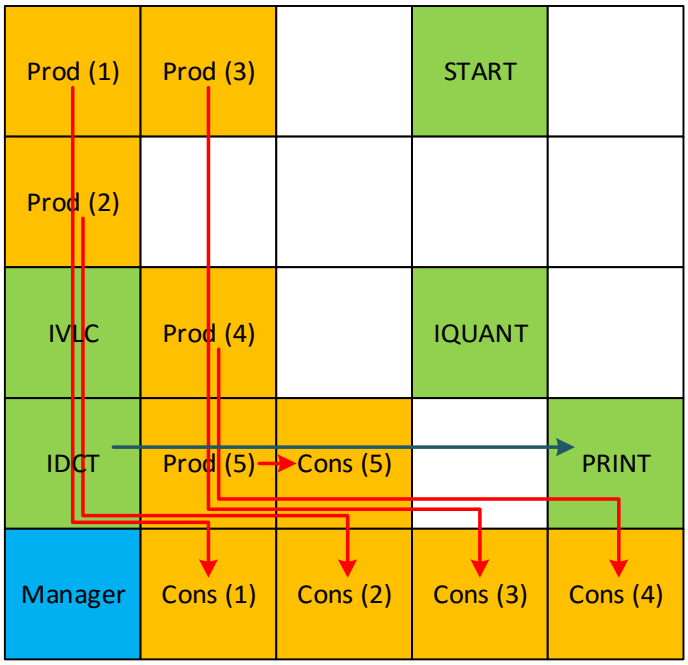


Figure 6.1 – Scenario 1 task mapping. Arrows represent the flows according to the Hamiltonean routing.

Figure 6.2 and Table 6.1 presents the computation latency graph and the latency standard deviation for the MPEG application, respectively. The *fullQoS* platform allows to reduce the standard deviation of the computation latency by 80% compared to *compQoS,* since it allows the MPEG application traffic to use the high priority communication channel, avoiding the disturbing traffic. An important aspect of this scenario is that no communication flow changes its priority to CS, the use of high priority flows is sufficient to guarantee a performance similar to the scenario without interferences. Even when the priority is downgraded, the overall performance is not affected since just one flow degrades its priority at a time, and its priority is quickly restored after this. The impact of the network latency during the period when the priority is downgraded is in part absorbed by the tasks' slack time. The *fullQoS_MNoC* does not present a significant advantage in this scenario since it has a low management traffic load because all applications are allocated at the beginning of the execution and only the MPEG application generates monitoring traffic.

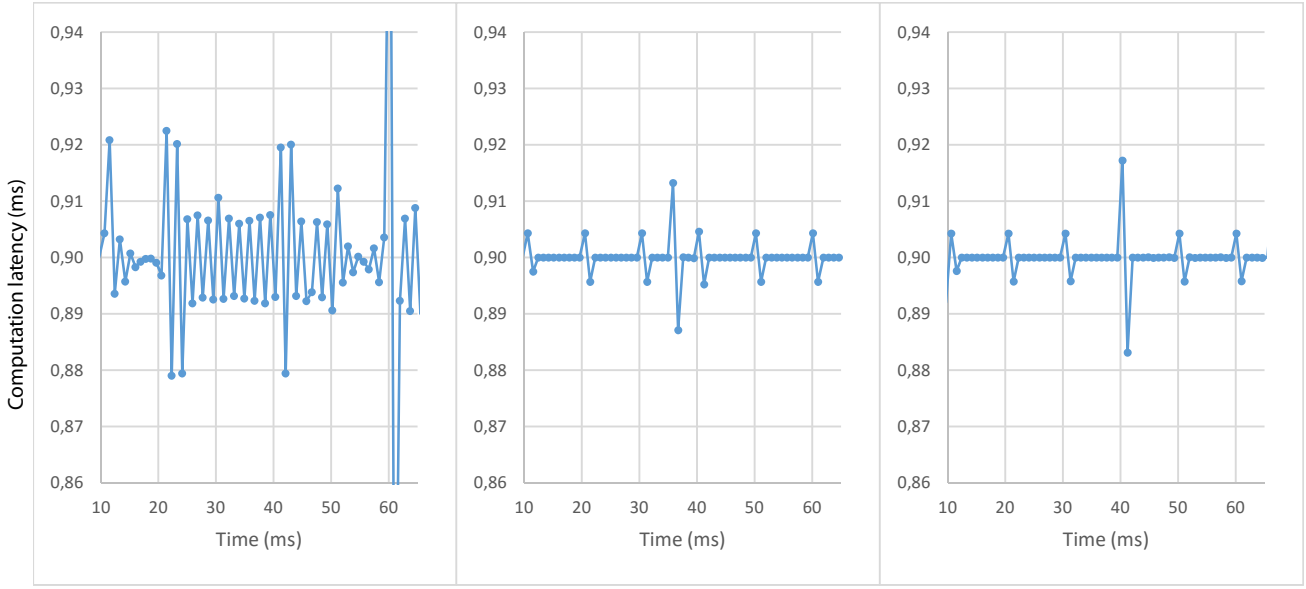


Figure 6.2 – MPEG computation latency for Scenario 1 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*.

Table 6.1 – Standard deviation of the MPEG computation latency for Scenario 1.

	compQoS	***fullQoS***	***fullQoS_MNoC***
Standard Deviation (µs)	18.90	3.81	3.82

Table 6.2 presents the average network latency and jitter (in clock cycles) for different communication flows of the MPEG application. The disturbing applications only affects the communication between the last three tasks of the MPEG application: IQUANT, IDCT and PRINT. The use of *fullQoS* allows to reduce the average network latency and jitter when compared to

compQoS, improving the average network latency by 71.8% and the jitter by 26.5% in the IDCT → PRINT flow. The *fullQoS_MNoC* further improves the latency result for this flow by 14.4% when compared to the *fullQoS*.

Table 6.2 – Average network latency and jitter (in clock cycles) for the Scenario 1.

	compQoS		***fullQoS***		***fullQoS_MNoC***	
Communication Flow	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**
START → IVLC	30.0	0.0	30.0	0.0	31.0	0.0
IVLC → IQUANT	20.0	0.0	20.0	0.0	21.0	0.0
IQUANT → IDCT	**270.5**	374.4	**114.1**	396.6	**83.3**	222.9
IDCT → PRINT	**809.4**	705.2	**227.6**	517.7	**194.9**	585.7

Table 6.3 presents statistics for the management traffic in this scenario. As the *fullQoS_MNoC* provides an exclusive path for the management traffic, the network latency improves by 13.5% and the jitter by 13.1% when compared to the *fullQoS*. The *compQoS* has a lower amount of management packets since only RT monitoring packets are generated.

Table 6.3 – Management communication statistics for the Scenario 1.

	compQoS	***fullQoS***	***fullQoS_MNoC***
Number of packets	253	567	514
Avg. Lat. (clock cycles)	1189.7	825.2	713.8
Jitter (clock cycles)	1607.6	1830.1	1590.0

6.2. Scenario 2

This scenario consists in an MPEG application and 4 disturbing applications. Figure 6.3 presents the task mapping and communication paths of the disturbing applications for this scenario. In this scenario, the used mapping creates a situation where the disturbing traffic interferes with most of MPEG tasks, instead of a specific flow. There is also an intra-application disturbing since some of the flows overlap other tasks, i.e., the IDCT task is between the path of the START and IVLC tasks. The average utilization of the links in the path used by the MPEG application is around 40% to 80%.

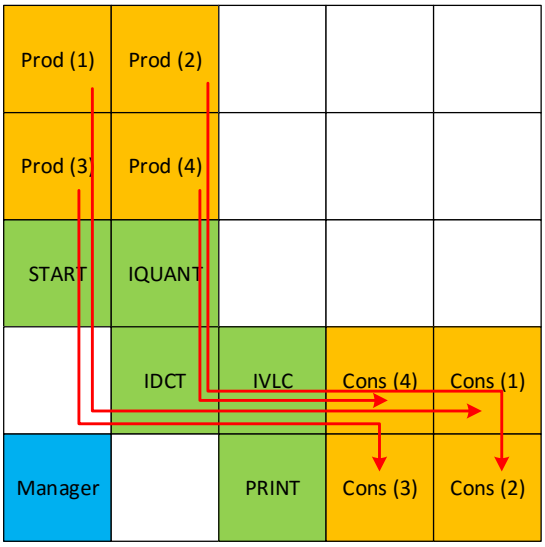


Figure 6.3 – Scenario 2 task mapping.

Figure 6.4 and Table 6.4 presents the computation latency graph and the latency standard deviation for the MPEG application in this scenario. The *fullQoS* reduces the standard deviation of the computation latency by 63.4% compared to *compQoS*. The latency only stabilize when multiple tasks have their communication priority changed to high, which can be observed in the graphs for cases (b) and (c), around 20 ms to 30 ms and 40 ms to 50 ms. Outside these periods, the communication priority of some flows downgrades, affecting the overall computation performance. Thus, this scenario highlights how the behavior of the communication flows impacts the computation QoS constraints. The *fullQoS_MNoC* did not present a significant advantage in the computation performance since the management traffic is similar to the Scenario 1.

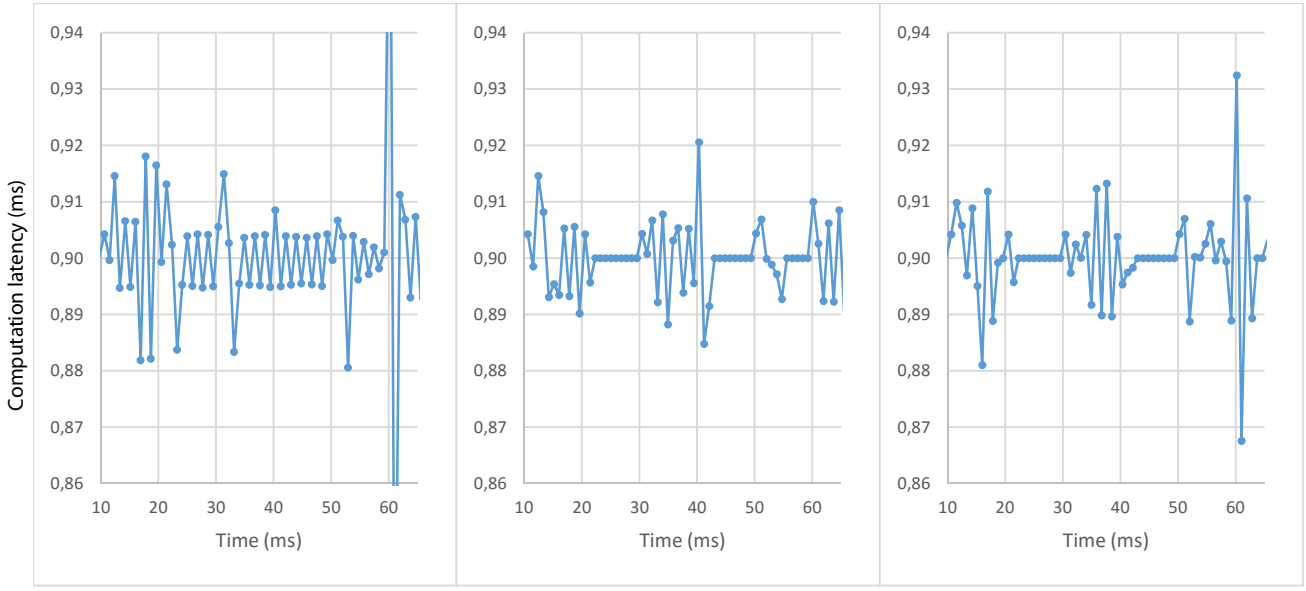


Figure 6.4 – MPEG iteration latency for Scenario 2 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*.

Table 6.4 - Standard deviation of the MPEG computation latency for Scenario 2.

	compQoS	*fullQoS*	*fullQoS_MNoC*
Standard Deviation (µs)	17.03	6.26	8.36

Table 6.5 presents the average network latency and jitter for different communication flows of the MPEG application. The *fullQoS* platform present an improvement in the network latency (66% in average) and jitter (51% in average) when compared to the *compQoS*. The *fullQoS*_MNoC further improves the network latency (14% in average) and jitter (11.2% in average), when compared to the *fullQoS*.

Table 6.5 – Average network latency and jitter for Scenario 2.

	compQoS		*fullQoS*		*fullQoS_MNoC*	
Communication Flow	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**
START → IVLC	498.0	570.2	151.7	327.4	109.5	262.4
IVLC → IQUANT	15.1	0.4	15.0	0.3	16.1	1.1
IQUANT → IDCT	237.5	909.1	110.0	274.9	87.3	235.5
IDCT → PRINT	497.2	473.7	155.9	352.7	162.4	350.2

Table 6.6 presents the average network latency and jitter for the management traffic. The average latency of the management packet is high, and there is no significant advantage using the *fullQoS*_MNoC in this case. This situation emphasizes a behavior of the reference platform: in a situation where a PE is receiving consecutive packets with a small interval between these packets, the first packet is treated by the PE at the moment the message reaches the NI. However, the subsequent packets must wait in the network until the past packets are processed, increasing significantly the network latency observed in the results. This behavior is further amplified in the manager PEs, since this situation is more common because all PE_{SL} generate packets addressed to the manager.

Table 6.6 – Management communication statistics for Scenario 2

	compQoS	*fullQoS*	*fullQoS_MNoC*
Number of packets	247	565	513
Avg. Lat. (clock cycles)	1460.0	899.7	889.2
Jitter (clock cycles)	2086.5	1975.8	1848.4

6.3. Scenario 3

This Scenario is similar to the Scenario 2 concerning the task mapping, however, instead of using the MPEG application, the DTW application is used. Figure 6.5 presents the task mapping of this scenario, showing that the disturbing tasks affect the communication between the Bank and the worker tasks. Differently from the MPEG application, the DTW application has multiple tasks executed in parallel, and a single task (BANK) communicates with multiple other tasks.

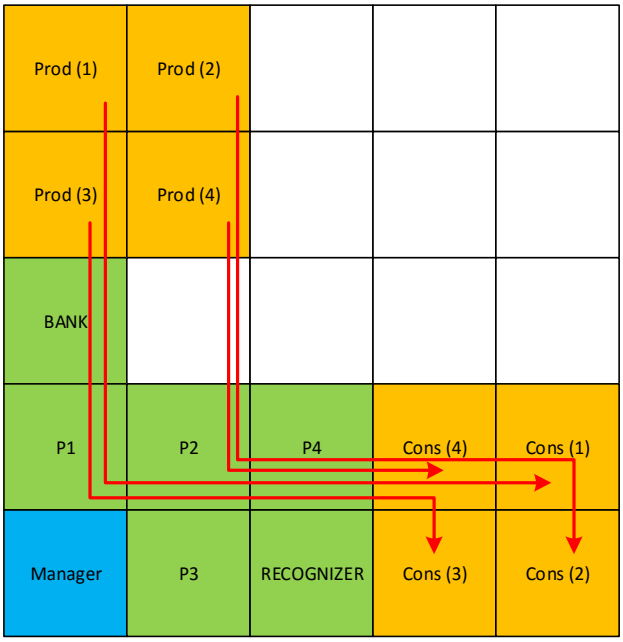


Figure 6.5 – Scenario 3 task mapping.

Figure 6.6 and Table 6.7 presents the computation latency graph and the latency standard deviation for the DTW application. In this scenario, the *fullQoS* presented a degradation in the performance when compared to *compQoS*. This is mainly due to the CS, demonstrating a limitation of this strategy. During the execution, a CS is established between the BANK and the worker tasks. While the CS is enabled between two tasks, the high priority channel is dominated by a single flow, negatively affecting the others, which are constrained to the low priority channel. For example, a CS established between the BANK and the worker task P4 affects the communication between the BANK and the tasks P1, P2 and P3. Another factor that affects the computation performance, and is highlighted in this scenario is interruptions caused by packet reception. When a packet is received, the processor is promptly interrupted to treat the received packet, regardless the packet type and if the processor is currently executing a RT task, which impacts negatively on the performance. This behavior is further emphasized by the DTW application since this application is more susceptible to interruptions when compared to the MPEG application because multiple tasks are executed in parallel. The *fullQoS_MNoC* presented

an advantage in this scenario since it had an overall lower network packet latency and a smaller number of circuits (CS) established during the execution of the application.

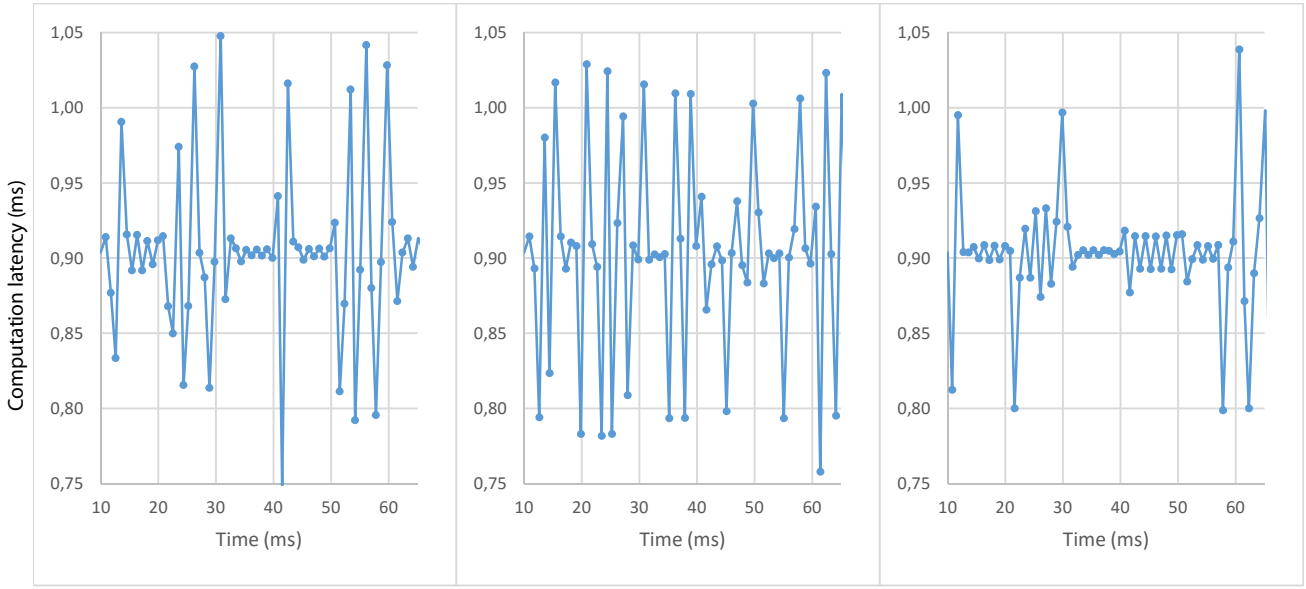


Figure 6.6 – DTW computation latency for Scenario 3 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*.

Table 6.7 – Standard deviation of the DTW computation latency for Scenario 3.

	compQoS	***fullQoS***	***fullQoS_MNoC***
Standard Deviation (µs)	60.00	71.94	41.02

Table 6.8 presents the average network latency and jitter for different communication flows of the DTW application. The use of *fullQoS* presents and improvement in the network latency (around 56%) and a small improvement in the jitter (around 15.1%) compared to *compQoS*. The *fullQoS_MNoC* presents also noticeable advantages for the tasks P1, P2 and P3, which are around the manager, reducing the average network latency by 25.3% when compared to the *compQoS*.

Table 6.8 – Average network latency and jitter for Scenario 3.

	compQoS		***fullQoS***		***fullQoS_MNoC***	
Communication Flow	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**
BANK → P1	225.0	351.5	132.7	328.4	89.6	242.7
BANK → P2	238.6	455.4	123.5	306.2	101.4	244.0
BANK → P3	391.4	427.7	232.5	926.1	129.7	286.0
BANK → P4	721.2	1201.2	206.0	505.2	197.9	423.1

Table 6.9 presents statistics for the management traffic in this scenario. Overall, the *fullQoS_MNoC* reduces the management traffic latency by 10% compared to *fullQoS*. The

fullQoS_MNoC presented an advantage because the management packets are constrained to the M-NoC instead of sharing the same path with application packets, reducing the number of packets in the Data NoC around the manager. Thus, avoiding a situation where multiple management packets are occupying the network, delaying the reception of the application packets by a PE_{SL} near the manager, as shown in the flows communicating with the tasks P1, P2 and P3.

Table 6.9 – Management communication statistics for the Scenario 3

	compQoS	***fullQoS***	***fullQoS_MNoC***
Number of packets	266	586	539
Avg. Lat. (clock cycles)	1065.2	775.4	701.1
Jitter (clock cycles)	1729.3	1828.9	1660.9

6.4. Scenario 4

This scenario consists in a DTW and a MPEG application with computation constraints, and multiple disturbing applications. The mapping for this scenario is shown in Figure 6.7. In this mapping, disturbing tasks are allocated between the communication paths of the evaluated applications. The RT applications in this scenario are executed during all the evaluation, while the disturbing application executes for a smaller period, around 10 ms. However, when a disturbing application finishes its execution, a new one is reallocated, and its tasks are mapped to the same position. This leads that a new disturbing task is allocated every 500 us during the evaluation, and allows simulating a larger amount of management traffic.

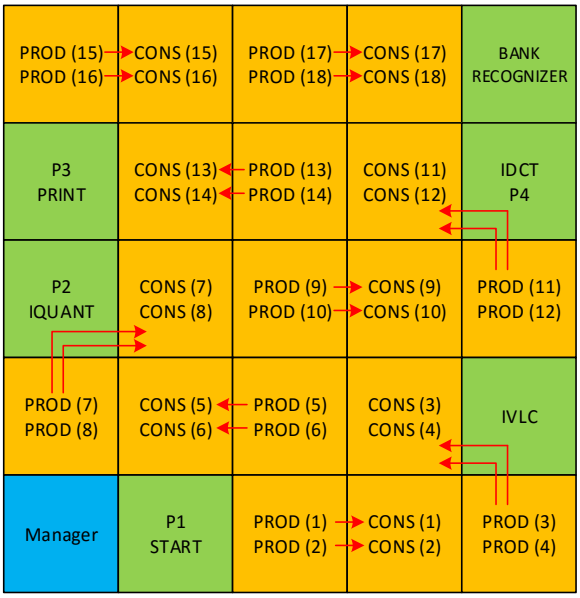
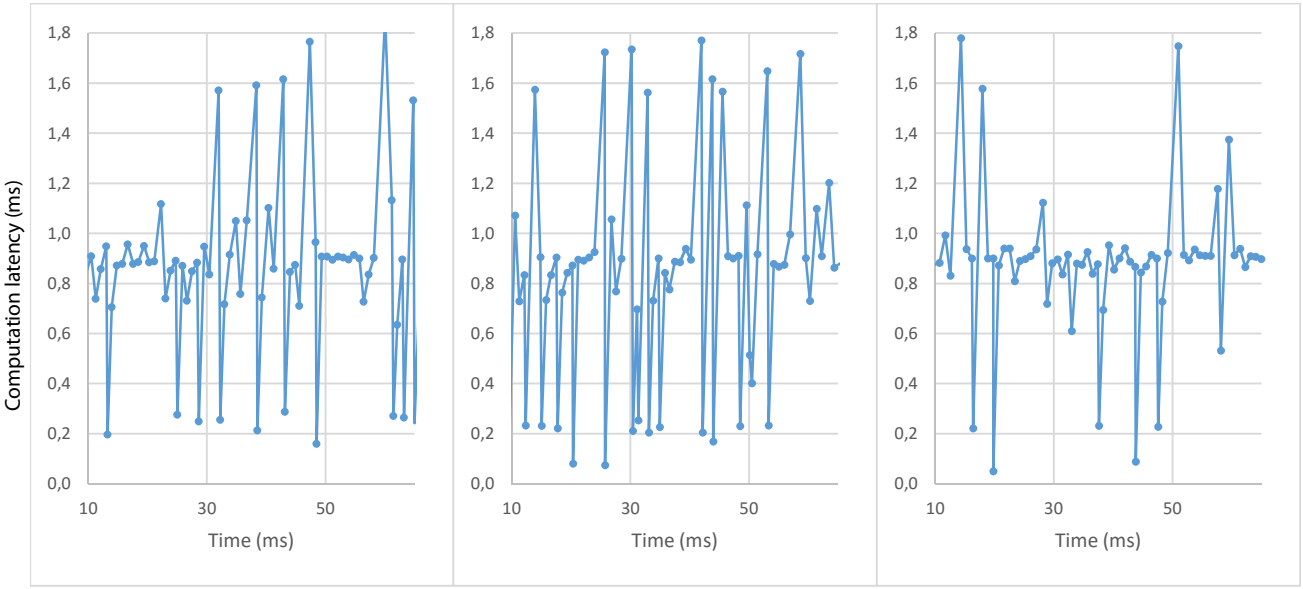


Figure 6.7 – Scenario 4 task mapping.

In this scenario, there are multiple applications executing in the same PE. The computation resources are shared between the RT tasks, since there are multiple RT tasks executing in the same PE. The PEs dedicated to the execution of the disturbing applications also executes multiple disturbing tasks. The link use between the producer/consumer tasks is around 20%.

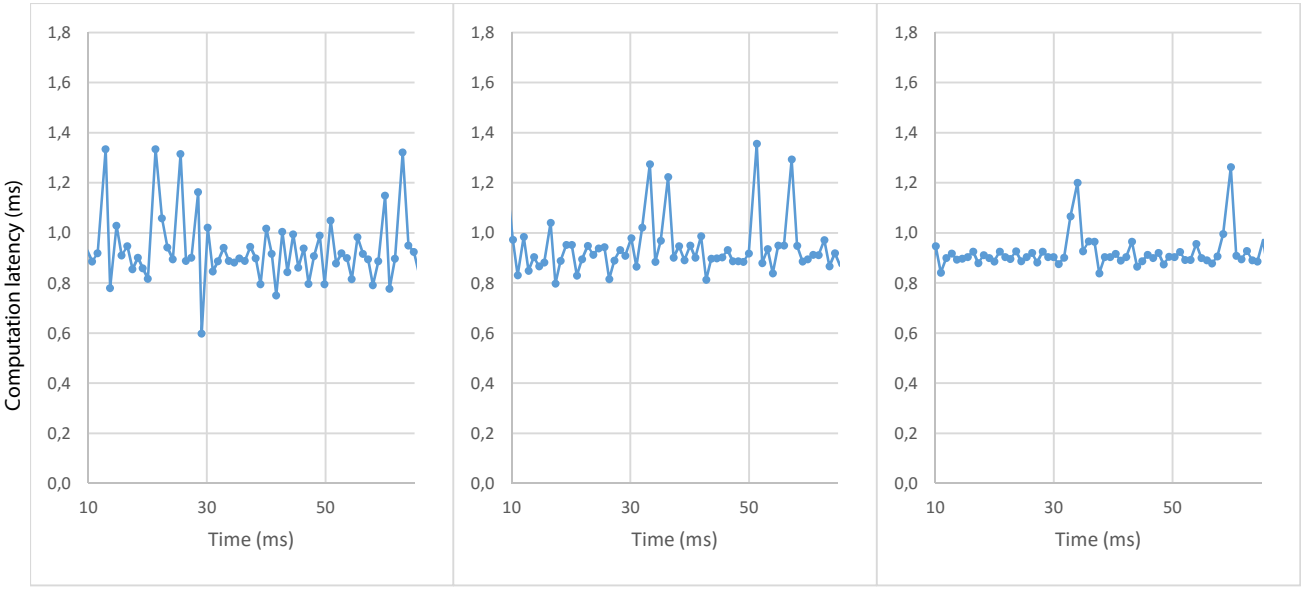
Figure 6.8, Figure 6.9 and

Table 6.10 presents the computation latency and the latency standard deviation for the MPEG and DTW applications, respectively.



(a) (b) (c)

Figure 6.8 – MPEG iteration latency for Scenario 4 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*.



(a) (b) (c)

Figure 6.9 – DTW iteration latency for Scenario 4 (a) *compQoS*; (b) *fullQoS*; (c) *fullQoS_MNoC*..

Table 6.10 – Standard deviation of the computation latency for Scenario 4.

	compQoS	***fullQoS***	***fullQoS_MNoC***
MPEG Std. Dev. (µs)	352.4	426.3	281.6
DTW Std. Dev. (µs)	137.7	104.3	68.6

In this scenario, the number of management packets is considerably larger than the scenarios described previously, and it is possible to notice an advantage using the *fullQoS_MNoC* for both the DTW and MPEG applications, presenting an improvement of 35% in the standard variation of the computation latency for both applications when compared to the case using *compQoS*.

Table 6.11 presents the average network latency and jitter for different communication flows of the MPEG and DTW applications in this scenario. Some of the flows have its average latency greatly reduced due to CS, such as the flow BANK → P1 in the case using only *fullQoS*. In overall, the network latency is greatly reduce by around 70% when using *fullQoS*, and is further improved by 35% when using the *fullQoS_MNoC*. The jitter also presents a reduction of 45% when using *fullQoS*, and is further improved by 35% when using *fullQoS_MNoC*.

Table 6.11 – Average network latency and jitter for the Scenario 4.

		compQoS		***fullQoS***		***fullQoS_MNoC***	
	Communication Flow	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**	**Avg Lat**	**Jitter**
MPEG	START → IVLC	692.3	1430.0	299.0	746.8	143.3	305.2
	IVLC → IQUANT	883.0	1212.7	50.8	148.4	80.8	236.0
	IQUANT → IDCT	757.2	909.6	420.5	1245.9	128.1	297.6
	IDCT → PRINT	567.6	1177.3	67.9	221.4	177.0	679.5
DTW	BANK → P1	3707.6	2402.5	16.0	0.0	36.0	42.0
	BANK → P2	4386.9	3131.0	2070.5	2342.0	1603.9	1574.2
	BANK → P3	3061.4	2165.9	1307.4	2165.2	558.9	1286.5
	BANK → P4	10.1	0.5	13.1	24.2	11.1	0.4

Table 6.12 presents the average latency and jitter for the management traffic in this scenario. The use of *fullQoS_MNoC* allows significantly reducing in the average latency (77.4%) and jitter (63.4%) when compared to *fullQoS*.

Table 6.12 - Management communication statistics for the Scenario 4.

	compQoS	***fullQoS***	***fullQoS_MNoC***
Number of packets	811	1450	1438
Avg. Lat. (clock cycles)	891.6	518.6	117.0
Jitter (clock cycles)	1700.8	1268.4	463.5

6.5. Final Remarks

This Chapter presented an evaluation of the platform, considering the implemented QoS architecture. The evaluation highlighted several important aspects related to QoS in MPSoCs, such as:

- The application communication performance has a significant impact in the computation performance, especially when the application is under tight time constraints;
- The use of communication QoS allows a large reduction of the application traffic network latency and jitter, reducing it by more than 70% in some cases (as shown in Scenario 1);
- Even when the NoC links have a relatively low utilization, around 20% for example, the network latency and jitter suffers a significant degradation;
- The network structure must be able to support multiple high priority flows in the same path. The current implementation of the CS dominates the use of the high priority channel by a single flow, which may result in an overall performance degradation;
- The M-NoC brings improvements for the application communication performance, benefitting both RT and BE applications. Even when the network used in the platform supports QoS, isolating the management traffic and application traffic through the use of MP NoCs benefits the average network latency and jitter for the application;
- The M-NoC allows to reduce the number of QoS actions in the platform;
- The M-NoC improves the management traffic network performance, allowing the platform to act faster according to the platform requirements;
- The number of management services in the platform can largely increase when using the M-NoC, without impairing the application traffic performance;

- The management traffic concentrates around the PE responsible for controlling the management services in the platform and interferes with the PEs executing applications near the manager PE. The M-NoC allows to mitigate this problem;
- Even an average load of management traffic impacts the application traffic. Larger management loads increase the advantage of using the M-NoC;
- Interruption has a large impact on the application computation latency. Delaying the treatment of interruption, and executing the treatment latter in non critical parts of the software can bring large improvement for the computation latency. At the same time, delaying the interruption treatment brings more congestion to the network. The treatment of interruptions in the platform requires a further study.

7. CONCLUSIONS AND FUTURE WORKS

This work presented an MPSoC architecture supporting a large number of management services, including *communication* QoS and *computation* QoS. This platform is the result of several improvements to the HeMPS-QoS platform, including the restructuring of software source code, correction of several bugs in the software, hardware and tools for this platform, and the integration of features developed in other HeMPS platforms, such as support for computation QoS. The platform has reached a stable level, and can be used as a starting point for the research of further aspects related to the project of MPSoCs.

A main contribution of this work is the provision of an isolation between the different traffic classes in MPSoCs by the use of MP networks. The decision of using MP in MPSoCs is based on the platform traffic behavior and in the state of the art where multiple works explore the aspects of MP networks. The distinct characteristic of the main traffic classes in the reference platform – application data and management data – motivated the implementation of two different topologies for the NoC targeting the management traffic, including a complete mesh topology, similar to the network used for the application data, and a mesh interconnecting only the managers, exploring the clusterization aspect of the platform. The use of MP NoCs with an equal topology, adjusting the network parameters for each network, such as smaller flit widths is a promising path for MPSoCs.

The evaluation of the management network highlighted several important aspects related to QoS in MPSoCs. The separation of the traffic allows significant improvement on both the computation and communication aspects of the applications executing in the platform, even when considering that the single network implementation already has QoS resources.

7.1. Future Works

Specific future works targeting improvements in the QoS aspects of the platform, and the use of MP networks include:

- Exploration of the physical aspects (energy and area) of MP networks;
- Implementation and evaluation of more applications for the platform;
- Improvements in the application network targeting communication QoS, such as the use of multiple simple disjoint networks, allowing the support for multiple CS flows in the same path;

- Improvements in the migration algorithm, allowing a better resource allocation in the platform;
- Improvements in the interruption treatment, avoiding a computation overhead impacting RT applications;
- Inclusion of more management services in the platform, such as security and fault tolerance

7.2. Publications

The set of publications published during the development of the work included:

1. Martins, A.; **Silva, D.**; Castilhos, G.; Monteiro, T.; Moraes, F. *"A Method for NoC-based MPSoC Energy Consumption"*.In: ICECS, 2014, pp. 427-430.
    - Describes a method to characterize the power and energy of the NoC and the processor of the PE. Despite the fact that power is not the performance figure evaluated in this work, the method to generate and characterize the NoC enabled to master the CAD tools to obtain the results of this work.

2. **Silva, D.**; Oliveira, B.; Moraes, F. *"Effects of the NoC Architecture in the Performance of NoC-Based MPSoCs"*. In: ICECS, 2014, pp. 431-434.
    - Evaluates the main architectural NoC parameters in such a way to determine the influence of the buffers, the crossbar, and the control logic in the performance and area of NoCs. The results of this paper guided the customization of the M-NoC, as shallow buffers and serialized flits.

3. **Silva, D.**; Moraes, F. *"Differentiation of MPSoCs Message Classes Using Multiple NoCs"*. In: ICECS, 2015, pp. 312-315.
    - This publication contains the results presented in Chapter 5 related to the two M-NoCs developed. The effect of serialization is not presented in this paper.

REFERENCES

[ABO12] Abousamra, A.; Melhem, R.; Jones, A. "Déjà Vu Switching for Multiplane NoCs". In: NoCS, 2012, pp. 11-18.

[AGA02] Agarwal, A.; et. al. "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs". IEEE Micro, vol. 22, pp 25-35, 2002.

[AGA07] Agarwal, A.; et. al. "On-Chip Interconnection Architecture of the Tile Processor". IEEE Micro, vol. 27, pp 15-31, 2007.

[ALM09] Almeida, G.; Sassatelli, G.; Benoit, P.; Saint-Jean, N.; Varyani, S.; Torres, L.; Robert, M. "An Adaptive Message Passing MPSoC Framework". International Journal of Reconfigurable Computing, vol. 2009, 20p, 2009.

[BAL08] Balfour, J.; Dally, W. "Design tradeoffs for tiled CMP on-chip networks". In: ICS, 2006, pp. 187-198.

[BAG08] Bagherzadeh, N.; Matsuura, M. "Performance Impact of Task-to-Task Communication Protocol in Network-on-Chip". In: ITNG, 2008, pp. 1101- 1106.

[BEN12] Benini, L.; Flamand, E.; Fuin, D.; Melpignano, D. "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator". In: DATE, 2012, pp. 983-987.

[BJE06] Bjerregaard, T.; Mahadevan, S. "A survey of research and practices of Network-on-chip". ACM Computing Surveys, vol. 38, 51p, 2006.

[CAR09a] E. Carara; R. Oliveira; N. Calazans; F. Moraes. "HeMPS - a framework for NoC-based MPSoC generation". In: ISCAS, 2009, pp 1345-1348.

[CAR09b] Carara, E.; Calazans, N.; Moraes, F. "Managing QoS flows at task level in NoC-based MPSoCs". In: VLSI-SoC, 2009, pp. 133-138.

[CAR11] Carara, E. "Serviços de Comunicação Diferenciados em Sistemas Multiprocessados em Chip Baseados em Redes Intra-Chip". Ph.D. thesis, PUCRS, Porto Alegre, Brazil, 2011.

[CAS13] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. "Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes". In: ISVLSI, 2013, pp. 153-158.

[CIO06] Ciordas, C.; Goossens, K.; Basten, T. "NoC Monitoring: Impact on the Design Flow". In: ISCAS, 2006, pp. 1981-1984.

[DAS13] Das, R.; Narayanasamy, S.; Satpathy, S.; Dreslinski, R. "Catnap: Energy Proportional Multiple Network-on-Chip". In: ISCA, 2013, pp. 320-331.

[DUA03] Duato, J.; Yalamanchili, S.; Ni, L. “Interconnection Networks: An Engineering Approach”. Morgan Kaufmann. 2003. 600p

[FUW14] Fu, W.; Chen, T.; Wang, C.; Liu, L. “Optimizing memory access traffic via runtime thread migration for on-chip distributed memory systems”. The Journal of Supercomputing, vol 36, pp. 1491-1516, 2014.

[GRA07] Gratz, P.; Kim, C.; Sankaralingam, K.; Hanson, H.; Shivakumar, P.; Keckler, S.; Burger, D. “On-Chip Interconnection Networks of the TRIPS Chip”. IEEE Micro, vol. 27, pp 41-50, 2007.

[GRO09] Grot, B.; Hestness, J.; Keckler, S.; Mutlu, O. “Express Cube Topologies for on-Chip Interconnects”. In: HPCA, 2009, pp. 163-174.

[JOV08] Joven, J.; Font-Bach, O.; Castells-Rufas, D.; Martinez, R.; Teres, L.; Carrabina, J. “xENoC - An eXperimental Network-On-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures”. In: PDP 2008, pp. 141-148.

[KOR13] Kornaros, G.; Pnevmatikatos, D. “A survey and taxonomy of on-chip monitoring of multicore systems-on-chip”. ACM Transactions on Design Automation of Electronic Systems, vol. 18, 38p, 2013.

[KRA93] Kranz, D.; Johnson, K.; Agarwal, A.; Kubiatowicz, J.; Lim, B. “Integrating message-passing and shared-memory: early experience”. In: PPoPP, 1993, pp. 54-63.

[KUM02] Kumar, S.; Jantsch, A.; Soininen, J.; Forsell, M.; Millberg, M.; Öberg, J.; Tiensyrjä, K.; Hemani, A. “A Network on Chip Architecture and Design Methodology”. In: ISVLSI 2002, 8p.

[LIU00] Liu, J.W.S. “Real-Time System”. Printice Hall, New Jersey, 2000, 592p.

[MAN15] M. Mandelli; L. Ost; G. Sassatelli; F. Moraes. “Trading-off System Load and Communication in Mapping Heuristics for Improving NoC-Based MPSoCs Reliability”. In: ISQED, 2015, pp. 392-396.

[MIG15] Miguel, J.; Jerger, N. “Data Criticality in Network-On-Chip Design”. In. NoCS, 2015, 8p.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. “HERMES: an infrastructure for low area overhead packet-switching networks on chip”. In: Integration, the VLSI Journal, vol. 38, pp. 69-93, 2004.

[MOR12] Moraes, F.; Madalozzo, G.; Castilhos, G.; Carara, E. “Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs”. In: ISCAS, 2012, pp. 644-647.

[PAS08] Pasricha, S.; Dutt, N. “On-Chip Communication Architectures: System on Chip Interconnect”. Morgan Kaufmann. 2008. 522p.

[PET12] Petry, C.; Wachter, E.; Moraes, F.; Calazans, N.; Castilhos, G. “A Spectrum of MPSoC Models for Design and Verification Spaces Exploration”. In: RSP, 2012, pp. 30-35.

[RHO10] Rhoads, S. “Plasma CPU”. Available at : http://plasmacpu.no-ip.org, 2010.

[RUA13] Ruaro, M.; Carara, E.; Moraes, F. “Adaptive QoS Techniques for NoC-Based MPSoCs”. In: SoC, 2013, 6p.

[RUA15] Ruaro, M.; Madalozzo, G.; Moraes F. “A Hierarchical LST-Based Task Scheduler for NoC-Based MPSoCs with Slack-Time Monitoring Support”. In: ICECS, 2015, pp. 308-311.

[SEP15] Sepulveda, M.; Florez, D.; Das, S.; Gogniat, G., “Reconfigurable Security Architecture for disrupted protection zones in NoC−Based MPSoCs”. In: ReCoSoC, 2015, 8p.

[SIL08] Silberschatz, A.; Galvin, P.; Gagne. G. “Operating System Concepts”. Wiley & Sons, New Jersey, 2008, 992p, 8th edition.

[VOL12] Volos, S.; Seieculescu, C.; Grot, B.; Pour, N.; Falsafi, B.; Micheli, G. “CCNoC: Specializing On-Chip Interconnects for Energy Effiency in Cache-Coherent Servers”. In: NoCS, 2012, pp. 67-74.

[WOL08] Wolf, W.; Jerraya, A.; Martin, G. “Multiprocessor System-on-Chip (MPSoC) Technology”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and System, vol. 27, pp. 1701-1713, 2008.