Emulation platform synthesis and NoC evaluation for embedded systems : towards next generation networks

Otavio Junior Alcantara de Lima

UNIVERSITÉ JEAN MONNET
SAINT-ÉTIENNE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JEAN MONNET

Spécialité : **Microélectronique**

Présentée par

Otávio Alcântara de Lima Júnior

Thèse dirigée par **Frédéric Rousseau**
et codirigée par **Virginie Fresse**

préparée au sein du **Laboratoire Hubert Currien UMR CNRS 5516**
et de l'École Doctorale **Sciences, Ingénierie, Santé**
En collaboration avec le laboratoire TIMA, Grenoble

Synthèse de plateformes d'émulation et évaluation de NoCs pour les systèmes embarqués : vers les réseaux du futur

Thèse soutenue publiquement le **9 September, 2015**
devant le jury composé de :

Madame Dominique Borrione
Professeur, Université Grenoble Alpes, France, Présidente
Monsieur Ian O'Connor
Professeur, Université de Lyon, France, Rapporteur
Monsieur Sébastien Pillement
Professeur, Université de Nantes, France, Rapporteur
Monsieur Hamed Sheibanyrad
Chargé de Recherche CNRS, Laboratoire LIP 6, France, Examinateur
Monsieur Helano de Sousa Castro
Professeur, Université Fédéral du Ceara, Brésil, Examinateur
Monsieur Frédéric Rousseau
Professeur, Université Grenoble Alpes, France, Directeur de thèse
Madame Virginie Fresse
Maître de conférences, Université Jean Monnet, France, Co-Directrice de thèse

UNIVERSITÉ JEAN MONNET
SAINT-ÉTIENNE

THESIS

To obtain the title of

DOCTOR OF JEAN MONNET UNIVERSITY

Speciality: **Microelectronics**

Presented by

Otávio Alcântara de Lima Júnior

Thesis advisor **Frédéric Rousseau**
and co-advised by **Virginie Fresse**

elaborated at **Hubert Currien laboratory UMR CNRS 5516**
Doctoral School of **Sciences, Engineering, Health**
In colaboration with TIMA laboratory, Grenoble

Emulation platform synthesis and NoC evaluation for embedded systems: towards next generation networks

Thesis public presented in **September 9, 2015**
for the jury composed of:

Mrs. Dominique Borrione
Professor, Grenoble Alpes University, France, President
Mr. Ian O'Connor
Professor, University of Lyon, France, Rapporteur
Mr. Sébastien Pillement
Professor, Université de Nantes, France, Rapporteur
Mr. Hamed Sheibanyrad
Researcher CNRS, LIP 6 Laboratory, France, Examiner
Mr. Helano de Sousa Castro
Professor, Federal University of Ceara, Brazil, Examiner
Mr. Frédéric Rousseau
Professor, Grenoble Alpes , France, Advisor
Mrs. Virginie Fresse
Assistant Professor, Jean Monnet University, France, Co-advisor

Université Jean Monnet
SAINT·ETIENNE

I dedicate this thesis to my family. A special gratitude to my loving parents, Otávio and Socorro whose moral strength and wisdom inspire me. Heartfelt thanks to my sister Karol. I give special thanks to my wife Waneska and our daughter Fernanda for their outstanding support and love.

Abstract

The ever-increasing complexity of many-core embedded system applications demands a flexible communication structure capable of supporting different traffics requirements at run-time. The Networks-on-Chip (NoCs) emerge as the most promising communication technology for the modern many-cores SoC (System-on-Chip), whereby they have greater scalability than other solutions such as buses and point to point connections.

As NoCs become de facto standard for on-chip systems, NoC performance evaluation tools become critical for SoCs design. The FPGA-based emulation platforms accelerate NoC benchmarking as well as design space exploration. Those platforms have high accuracy and low execution time in relation to NoC simulators. An FPGA-based emulation platform is composed by tens or hundreds of distributed components. These components should be timely managed in order to execute an evaluation scenario. There is a lack of standard protocols to drive FPGA-based NoC emulators. Such protocols could ease the integration of emulation components developed by different designers, as well as they could enable the configuration of the emulation nodes without FPGA re-synthesis and the extraction of emulation results.

The NoC hardware emulation is quite challenging. It is important to validate new NoC architectures with realistic workloads, because they provide much more accurate results. The generation of applications traffic patterns is a key concern for NoC emulation. The dependency-aware traces are an appealing solution for the generation of realistic traffic workloads. They are more accurate than ordinary traces for a broad range of NoC architectures because they contain packets dependencies information. However, they tend to be bigger than the original ones what demands more FPGA resources.

This thesis aims the synthesis of FPGA-based NoC emulation platforms for the future multi-core embedded systems. We are interested in investigating strategies to generate realistic traffic patterns for NoCs emulated on FPGAs, as well as the management of the emulation platform using standard protocols inspired by the computer networks protocols.

One contribution of this thesis is a trace analysis framework which addresses the packets dependencies extraction problem. The proposed framework analyzes traces from a message passing application in order to build a Model of Computation (MoC). This MoC reproduces the communicative behavior of an application node. A

dependency-aware Traffic Generator (TG) is created from the proposed MoC. This TG generates the application traffic pattern during an FPGA-based NoC emulation. Another contribution is a light version of SNMP (Simple Network Management Protocol) to manage an FPGA-based NoC emulation platform. An FPGA-based emulation platform architecture is proposed based on the principles of SNMP protocol. This platform has a high-level interface to the emulation components provided by that protocol, which also eases the integration of emulation components created by different designers. The emulation platform and the protocol capacities are evaluated during a task mapping and mesh topology design space exploration.

A prospective analysis of future NoCs architectures is also a contribution of this thesis. In this analysis, a conceptual architecture of a future multi-core embedded system is used as model to extract these networks requirements. From this analysis, it is proposed some networking mechanisms. The first mechanism is a congestion-aware routing algorithm, which is an adaptive routing algorithm that selects the output path for a given packet based on a simple prioritized scheme of sets of rules. It is also proposed a congestion-control mechanisms for the vertical links interconnecting the layers of a 3D NoC. This mechanism is based upon the diffusion of congestion information by a piggyback protocol. Another proposed mechanism is a fault tolerant NoC based upon external bypass links, which circumnavigate the faulty routers. The last contribution is an analysis of FPGA-based emulation tools requirements for future NoCs.

Résumé

La complexité croissante des systèmes embarqués multi-cœur exige des structures de communication flexibles et capables de supporter de nombreuses requêtes de trafics au moment de l'exécution. Les Réseaux sur Puce (NoC) émergent comme la technologie de communication la plus prometteuse pour les SoCs (Systèmes sur Puce), du fait de leur plus grande flexibilité par rapport aux autres solutions comme les bus et les connexions points à points.

Les NoCs sont devenus le standard comme support de communication pour les SoC, mais les outils d'évaluation de performances deviennent critiques pour ces systèmes. Les outils d'émulation sur FPGA accélèrent l'analyse comparative de NoC ainsi que l'exploration de l'espace de conception. Ces outils ont une grande précision et un faible temps d'exécution par rapport aux simulateurs de NoC. Un outil d'émulation basé sur FPGA est composé de dizaines ou de centaines de composants distribués. Ces composant doivent être correctement gérés afin d'exécuter différents scénarii d'évaluation de trafic. Pour cela, il faut être à même de re-programmer les composants, en utilisant un protocole standard qui permet alors de piloter l'émulateurs de NoC sur FPGA. Ces protocoles facilitent l'intégration des composants d'émulation développés par différents concepteurs et simplifient la configuration des nœuds d'émulation sans resynthèse ainsi que l'extraction des résultats d'émulation.

Bien que l'émulation matérielle de NoC soit assez difficile, il est important de valider de nouvelles architectures de NoC avec des trafics basés sur les applications réelles pour permettre d'obtenir des résultats plus précis. La génération de modèles de trafic basés sur des applications est une préoccupation majeure pour l'émulation de NoC. Les traces intégrant des informations de dépendances sont plus précises que les traces ordinaires, ceci pour un large éventail d'architectures de NoC. Cependant, elles ont tendance à être plus grosses que les traces originales et exigent plus de ressources FPGA.

L'objectif de cette thèse est la synthèse de plateformes d'émulation de NoC sur FPGA pour les futurs systèmes embarqués multi-nœuds. Une recherche approfondie s'est portée sur les stratégies éventuelles pour la génération des modèles réalistes de trafic pour le NoC émulé sur FPGA, et pour la gestion des plateformes d'émulation en utilisant des protocoles standard inspirés des protocoles de réseaux informatiques. Une première contribution de cette thèse est une structure (« framework ») d'analyse de traces capable

d'extraire les dépendances de paquets. La plateforme proposée analyse un ensemble de traces extraites d'une application embarquée basée sur l'échange de messages afin de construire un modèle de calcul (MoC). Un générateur de trafic (TG) intégrant cette dépendance est créé à partir du MoC proposé. Ce TG reproduit le motif de trafic d'une application pour une plateforme d'émulation sur FPGA. Une seconde contribution est une version allégée du protocole SNMP (Simple Network Management Protocol) pour la gestion d'une plateforme d'émulation de NoC sur FPGA. L'architecture de la plateforme d'émulation proposée est basée sur les concepts du protocole SNMP. Elle offre une interface standard de haut niveau pour les composants d'émulation fournis par le protocole SNMP. Ce protocole facilite également l'intégration de composants d'émulation créés par différents concepteurs.

Une analyse prospective des futures architectures de NoC constitue également une contribution dans cette thèse. Dans cette analyse, une architecture conceptuelle d'un système embarqué multi-nœuds du futur constitue un modèle pour extraire les contraintes de ces réseaux. A partir de cette analyse, certains mécanismes de réseau sont proposés. Le premier mécanisme est un algorithme de routage basé sur la congestion, algorithme de routage adaptatif qui sélectionne le chemin de sortie pour un paquet donné sur la base de priorités issues simple d'un ensemble de règles. Un mécanisme de contrôle de congestion pour les liens verticaux reliant les couches d'un NoC 3D est également proposé. Ce mécanisme repose sur la diffusion des informations de congestion par un protocole de type « piggyback ». Un autre mécanisme présenté est un NoC tolérant aux pannes, basé sur l'utilisation de liens de contournement. Enfin, la dernière contribution repose sur une analyse de base des besoins des futurs NoC pour les outils d'émulation sur FPGA.

Acknowledgments

I would like to offer my sincerest gratitude to my supervisors Prof. Frédéric Rousseau and Asst. Prof. Virginie Fresse for their excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. It has been an honor to be their Ph.D. student.

Equally, I want to thank all the members of the jury who evaluated this work specially this thesis examiners, Prof. Ian O'Connor and Prof. Sébastien Pillement. I wish to thank all the members of the Hubert Curien laboratory and TIMA laboratory who made my stay a memorable period.

I also want to thank the direction of Federal Institute of Technology of Ceara (IFCE) which generously released me from my teaching duties in order to carry out this thesis. I wish to thank all my colleagues from the department of computer science of IFCE for their support throughout the doctorate program.

I am extremely grateful to la Région Rhône Alpes and to the Brazilian National Council for Scientific and Technological Development (CNPq) for their financial support of this thesis.

List of Figures

2.1 Architecture of 2x2 Mesh NoC. 6
2.2 Architecture of a five ports router. 7
2.3 NoC layered model. 8
2.4 SAF example. H, head flit; B, body flit and T, tail flit. 9
2.5 VCT example. H, head flit; B, body flit and T, tail flit. 9
2.6 Wormhole example. H, head flit; B, body flit and T, tail flit. 10
2.7 Topologies examples. (a) Ring. (b) 2D Mesh. (c) 3D Mesh. 10
2.8 General organization of an FPGA-based NoC Performance Evaluation Platform. 14
2.9 Direct-mapping and direct-mapping over multi-FPGA. 16
2.10 Clock schema of Virtual NoC. 17
2.11 Virtualized Router. 17
2.12 Platform configuration. 20

3.1 Hardware implementation of TGs and TRs. 29
3.2 A simple example of emulation platform configuration. 30
3.3 SNMP architecture and operations. 32
3.4 SNMP model adapted to NoC emulation. 33
3.5 Architecture overview. 34
3.6 Packet format. 36
3.7 Examples of the protocol execution 37
3.8 Traffic model example. 38
3.9 Example of application task graph. 39
3.10 MIB organization. 40
3.11 From a traffic scenarios to SET operations and SET packets. 41
3.12 Differential MIB update. 42
3.13 Execution flow. 42
3.14 Packets exchange during the execution flow. 43
3.15 MIB update evaluation. 46
3.16 FPGA occupation for synthetic traffic implementation 47

3.17 LUTs (a) and registers (b) occupation share for one emulation node of the synthetic traffic implementation . . . 48
3.18 FPGA occupation for task graph traffic implementation . . . 49
3.19 LUTs (a) and registers (b) occupation share for one emulation node of the task graph traffic implementation . . . 50
3.20 Task mapping exploration flow based on FPGA-based NoC platform . . . 51

4.1 Examples of trace-based execution. . . . 56
4.2 Example of packets dependency extraction problem. . . . 57
4.3 Overview of the trace analysis components and proposed MoC. . . . 60
4.4 Example of traces from a 2x2 mesh NoC. . . . 61
4.5 Example of packets tagging process of T_{RX}^{01}. . . . 62
4.6 Dependency Table (DT) organization . . . 64
4.7 DT execution example . . . 65
4.8 Superposition example. . . . 66
4.9 Base set example. . . . 67
4.10 Overview of the traces analysis framework. . . . 68
4.11 Initial traces analysis . . . 69
4.12 One iteration of the Base Set extraction algorithm . . . 70
4.13 Random walk strategy . . . 71
4.14 Base set evaluation . . . 72
4.15 Dependency-aware traffic generator . . . 73
4.16 Odd-Even routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits. . . . 76
4.17 West-First routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits. . . . 77
4.18 DyXY routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits. . . . 78
4.19 Negative-First routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits. . . . 79
4.20 Applications errors analysis . . . 79
4.21 3D Mesh NoC, XYZ routing scenarios. (a) 32-flits Model A. (b) 16-flits Model A. (c) 4-flits Model A. (d) 32-flits Model B. (e) 16-flits Model B. (f) 4-flits Model B. . . . 81
4.22 Random task mapping evaluation . . . 82

5.1 3D Mesh NoC. . . . 87
5.2 Vertically partially connected 3D-NoC. . . . 87
5.3 Conceptual NGNoC-based 3D-IC architecture. . . . 89
5.4 Layer 1 architecture. . . . 90
5.5 Layer 2 architecture. . . . 92
5.6 NGNoC packet frame. . . . 94
5.7 Adjacent routers interconnect to share congestion information. . . . 96

5.8 Example of inefficient TSVs utilisation. 97
5.9 Organization of the sets of rules. 99
5.10 Output paths sequence updates. 99
5.11 Set of rules evaluation. 100
5.12 Evaluation under transpose traffic. 101
5.13 Average total latency for uniform transpose (30 packets, 50% injected charge). 101
5.14 FPGA resources analysis for a Virtex 6. (a) number of registers for implementing a NoC. (b) number of LUTs for implementing a NoC. 102
5.15 Fault-tolerant version of FlexOE. 104
5.16 a) Connection details between routers in no-fault scenario. b) Connection details between routers in a faulty scenario. 105
5.17 The proposed topology. 106
5.18 Routing of a packet sent by router 2 to router 6 when routers 0 and 5 are faulty. 108
5.19 Average latency for (a) 5% of faulty routers, (b) 12% of faulty routers and (c) 19% of faulty routers. 109
5.20 Emulation node with LMCs connections. 110
5.21 Flit organization. 111
5.22 Multi-local port router organization. 112
5.23 NoC topology to connect the emulation nodes. 112

1 Architecture d'un NoC 2x2 maillé. 125
2 Modèle de couche d'un NoC. 126
3 L'organisation d'une plateforme d'évaluation de performances de NoC sur FPGA. 127
4 Aperçu de l'architecture d'interface SNMP entre le PC hôte et le NoC à émuler. 132
5 Exemples d'exécutions basées sur traces. 136
6 Exemple du problème d'extraction de dépendances de paquets 137
7 Vue de l'ensemble de l'outil d'analyse de traces. 140
8 Architecture conceptual de NGNoC sur un circuit 3D. 142
9 Exemple de l'utilisation de TSVs. 143
10 Version tolérante aux pannes du FlexOE. 145
11 Un nœud d'émulation avec des liens LMCs. 146

List of Tables

2.1 Traffic models description. 18
2.2 FPGA-based NoC Evaluation Platforms 24

3.1 A summary of messages categories for a protocol to manage a FPGA-based emulation platform 30
3.2 SNMP manager library 44
3.3 SNMP operations execution time 46
3.4 Task mapping and mesh topology exploration results according to different performance metrics 51

4.1 Trace example 55
4.2 Possible packets dependencies. 58
4.3 Data storage requirements 74
4.4 FPGA occupation 74

5.1 TSVs interconnect between layer 1 and layer 2 91
5.2 Layer 1 traffic flows description 91
5.3 TSVs interconnect between layer 2 and layer 3 92
5.4 Layer 2 traffic flows description 93
5.5 Comparative of congestion-control mechanisms for NoCs 95

1 Plateformes d'évaluation de performance de NoC sur FPGA 129
2 Exemple de trace 135
3 Les dépendances possibles. 137

Contents

Abstract iii

Résumé v

Acknowledgments vii

List of Figures ix

List of Tables xiii

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Main contributions . . . 3
- 1.3 Thesis organization . . . 3

2 State of the art for the FPGA-based NoC evaluation platforms 5
- 2.1 Networks-on-Chip: an overview . . . 6
  - 2.1.1 NoC layer model . . . 7
    - 2.1.1.1 Links layer . . . 7
    - 2.1.1.2 Network layer . . . 8
    - 2.1.1.3 Network adapter layer . . . 11
  - 2.1.2 NoC research areas . . . 11
    - 2.1.2.1 Congestion control . . . 11
    - 2.1.2.2 Fault-tolerance . . . 12
    - 2.1.2.3 Task mapping . . . 13
    - 2.1.2.4 Performance evaluation . . . 13
- 2.2 FPGA-based NoC performance evaluation platforms . . . 14
  - 2.2.1 Network model . . . 15
    - 2.2.1.1 Direct-mapping and direct-mapping over multi-FPGA . 15
    - 2.2.1.2 Simulation and virtualization . . . 16
  - 2.2.2 Traffic generation model . . . 17
    - 2.2.2.1 Synthetic traffic . . . 18

2.2.2.2 Application-based traffic 19
2.2.3 System management model 20
2.2.4 Limitations of FPGA-based evaluation platforms 21
2.2.4.1 Design partition 21
2.2.4.2 Reusability 21
2.2.4.3 Debugging 21
2.2.4.4 Synthesis Time 21
2.2.5 Platforms examples 22
2.2.6 Targeted research challenges 23
2.2.6.1 The inter-operability problem 23
2.2.6.2 The realistic workload problem 23
2.3 Conclusions 25

3 A SNMP-like protocol to manage an FPGA-based NoC emulation platform 27
3.1 Overview 28
3.2 Analysis of standard computer protocols to manage a NoC emulation platform 31
3.3 Adapting SNMP to NoC emulation management 32
3.4 A SNMP-like architecture of an FPGA-based NoC emulation platform . . 34
3.4.1 Architecture overview 34
3.4.2 Protocol implementation 35
3.4.3 Traffic models 36
3.4.3.1 Generating traffic scenarios using TGFF 38
3.4.4 MIB organization 39
3.4.5 Execution flow 41
3.4.6 Implementation details 43
3.4.6.1 SNMP hardware components 43
3.4.6.2 SNMP manager library 44
3.5 Experiments 44
3.5.1 SNMP operations evaluation 45
3.5.2 MIB update evaluation 45
3.5.3 Resources analysis 46
3.5.4 Case study: task mapping and mesh topology exploration 49
3.5.5 Results discussion 51
3.6 Conclusions 52

4 The synthesis of FPGA-Based NoC emulation platforms from application traces 53
4.1 Overview 54
4.2 Fundamental definitions 59

4.2.1 Traces and packets 60
4.2.2 Dependency pattern 62
4.2.2.1 DPs properties 62
4.2.2.2 DPs merging 63
4.2.2.3 Alternative notation of a DP 63
4.2.3 Dependency table 63
4.2.3.1 DTs properties 63
4.2.3.2 Generating traffic with a Dependency Table 64
4.2.3.3 Patterns superposition 65
4.2.4 Base Set 65
4.2.5 Message generation table 66
4.3 Problem definiton 66
4.4 Framework workflow 67
4.4.1 Patterns extraction 68
4.4.2 Base set extraction 69
4.4.3 Random walk strategy 71
4.4.4 Base set evaluation 71
4.5 Dependency-aware traffic generator 72
4.6 Experiments 73
4.6.1 Data storage requirements 73
4.6.2 FPGA occupation 74
4.6.3 Evaluation of the MoC 74
4.6.3.1 2D Mesh evaluation 75
4.6.3.2 3D Mesh evaluation 80
4.6.3.3 Random task mapping evaluation 80
4.6.4 Results analysis and discussions 80
4.7 Conclusions 83

5 Towards the next-generation NoCs: A prospective analysis 85
5.1 Overview 86
5.2 A conceptual architecture of a NGNoC 88
5.2.1 The layers description 90
5.2.2 Internetworking on NGNoCs 93
5.3 Congestion control on NGNoCs 94
5.3.1 Analysis of congestion control requirements of the conceptual architecture 96
5.3.2 Perspectives on congestion control 97
5.3.3 FlexOE: congestion-aware routing protocol for NoCs 98
5.4 Fault tolerance on NGNoCs 102

5.4.1 Analysis of fault tolerance requirements of the conceptual architecture 104
5.4.2 A fault tolerant NoC architecture 105
5.5 FPGA-based emulation on NGNoCs 108
5.6 Conclusions 113

6 Conclusions and perspectives **115**
6.1 Conclusions 115
6.2 Future work 117

Glossary **119**

List of Publications **121**

Resumé en Français **123**

References **149**

Chapter 1: Introduction

The beginning is always today.

Mary Shelley

Contents

1.1 Overview 1
1.2 Main contributions 3
1.3 Thesis organization 3

1.1 Overview

The accelerated development of the VLSI microelectronics technology enables the integration of tens or hundreds of complex logic cores in a System-On-Chip (SoC). Those SoCs consist of a set of processors, memories, mixed signals components and specialized logic blocks that share a communication infrastructure. The growing communication requirements of those systems demand scalable, flexible and parallel communication architectures. The Networks-on-Chips emerge as the most promising communication technology for the modern many-cores SoCs, whereby they have greater scalability than other solutions such as buses and point to point connections. The NoCs have a large design space concerning different aspects like topology, routing, flow control, Quality of Service (QoS), among others.

A SoC designer must make several architectural choices and performance evaluations to find out the best Network-On-Chip (NoC) configuration for a given project. Furthermore, all those system aspects must be tested and validated. An important part of a SoC design's costs and time is spent in Design Space Exploration (DSE) and validation of the communication architecture. NoC specialized design tools are used to reduce the design costs. High level simulations [CCG+04, HAAN+07] are used in the early stages of a development process for a faster, however less accurate, design space exploration. Usually those simulations are written in high level languages like C, C++ and Java. The Cycle Accurate Bit Accurate (CABA) simulations [Pra10] have higher accuracy in detriment of very high execution times. They are used to validate a Hardware Description Language (HDL) model.

The Field Programmable Gate Array (FPGA) based emulation tools [GADM+05, TFR11, KCdlTR08, FGTR12] are a promising technique for validating a novel NoC architecture. These platforms drastically reduce a NoC evaluation time and they also have high accuracy. The FPGA resources limitation is a constraint which determines the size of the emulated NoC. As full-system emulation is not always possible, an emulator uses specific components to inject and to extract packets in and from the network. A Traffic Generator (TG) is responsible for generating traffic based on real applications or synthetic models. A Traffic Receptor (TR) is responsible for retrieving the packets and computing performance metrics.

The future NoCs will deploy advanced networking features in order to respond to the growing requirements of embedded systems applications. The complexity of FPGA-based emulation platforms increases according to the features implemented on the NoCs, hence the need of an efficient management of those platforms components. There is a lack of standard protocols targeted at providing an interface to manage the components of NoC emulators. Furthermore, the proposals of FPGA-based emulation platform do not offer emulation components inter-operability. The adoption of a standard protocol can ease the integration of emulation components created by different designers. Moreover, this protocol shall provide an interface to the emulation components from the perspective of the management software running at the host PC in charge of the emulation.

An FPGA-based NoC emulation heavily depends on the traffic generation models, which specify the space-time distribution of a traffic workload. As NoCs become de facto standard for on-chip systems, traffic generation models become critical for embedded systems design. Traditional traffic generation models based on synthetic workloads do not accurately represent the characteristics of real applications [HGK10]. It is important to validate new NoC architectures with realistic workloads, because they provide much more accurate results. The synthesis of TGs based on applications traffic is a key concern for NoC emulation.

Executing realistic workloads in a emulated FPGA-based NoC is quite challenging. The FPGA resources are scarce and they must be used wisely. In some cases the application nodes are not suitable for emulation and only communication traces are available. A trace can easily comprise millions of packets. The traces size is an important issue for NoC emulation because of the limited internal RAM available on FPGAs. The traces are often allocated on the external RAM, which becomes a bottleneck for generating real-time traffic. Furthermore, trace-based execution distorts the injection rate and effects of congestion due to the lack of packets dependencies information. Some works [NMFA11, HCT+12, TAA+11] propose the utilization of dependency-aware traces. These traces contain packets dependencies information, which make them more accurate to be run in a broad range of NoCs configurations. However, they demand more data storage space than regular traces.

The future multi-core embedded systems will demand several complex networking mechanisms such as fault tolerance, network management, congestion control, QoS, among others. The next-generation of NoCs (NGNoCs) will have to be implemented with these advanced networking mechanisms. These networks will be deployed on 3D-ICs. Therefore, these networks will be divided in heterogeneous layers connected by reliable transversal links. The layers and the inter-layers connections may have specific physical constraints. Each layer implements an independent NoC with its own topology, number

of nodes, QoS, among other parameters. Many implementation questions arise from this new technology trend, we can cite a few: the congestion control intra and inter NoC layers and fault tolerant mechanisms. It is also important to analyze in prospective the performance evaluation of these advanced mechanisms by FPGA-based NoC emulation tools.

1.2 Main contributions

This thesis main contribution is the synthesis of FPGA-based NoC emulation platforms for the future multi-core embedded systems. We investigate the strategies to generate applications traffic for NoCs emulated on FPGAs, as well as the management of an emulation platform using standard protocols inspired by the computer networks' protocols. Moreover, a prospective analysis of the NGNoCs architectures is also carried out.

The first contribution of this thesis is the proposition of a System Network Management Protocol (SNMP) like protocol to manage an FPGA-based NoC emulation platform. The SNMP protocol and its related components are adapted to a hardware implementation. This protocol ensures a standard high-level interface to the emulation components, while it eases the integration of emulation components created by different designers. This protocol capacities are evaluated during a task mapping and mesh topology design space exploration.

Another specific contribution of this work is the proposition of a trace analysis framework which addresses the packet dependencies extraction problem. The proposed framework analyzes traces from a message passing application in order to build a packet dependency-aware Model of Computation (MoC), which can be synthesized as a traffic generator for NoC emulation. This framework does not depend on the original network architecture and it reduces the data storage requirements to execute real applications traffic on FPGAs while keeping a high accuracy.

A prospective analysis of NGNoCs architecture is also a contribution of this thesis. In this analysis a conceptual architecture of a future multi-core embedded system is used as model to extract the networking requirements of the NGNoCs. From this analysis, some NGNoC mechanisms are proposed. The first mechanism is a congestion-aware routing algorithm, which is an adaptive routing algorithm that selects the output path for a given packet based on a simple prioritized scheme of sets of rules. We also propose a congestion-control mechanism for the vertical links of a NGNoC which is based upon the diffusion of congestion information by a piggyback protocol. Another proposed mechanism is a fault tolerant NoC based upon external bypass links which circumnavigates faulty routers. The last contribution is a perspective analysis of FPGA-based emulation tools architecture for NGNoCs.

1.3 Thesis organization

The remainder of the thesis is organized as follows:

- **Chapter** 2 reviews the state of art of FPGA-based NoC performance evaluation platforms. It starts with an overview of the NoCs. The NoCs layers model is presented in which we highlight their main characteristics concerning routing, switching, flow control, among others. We also highlight four key NoCs research

areas: congestion control, fault tolerance, task mapping and performance evaluation. Those subjects are treated in this thesis work. Afterwards, the FPGA-based NoC performance evaluation platforms are also presented using a layered model, in which these platforms components are divided in three categories: networking, traffic generation and system management. The limitations of FPGA-based evaluation of NoCs are presented. At last, We present several examples of evaluation platforms categorized according to the layered model previously presented.

- **Chapter 3** presents our first contribution a light version of SNMP to manage an FPGA-based NoC emulation platform. It starts with an overview of the NoC emulation platforms. Afterwards, the SNMP is presented as a generic network management protocol, which is capable of integrating networked devices from different designers. We evaluate the features of SNMP to manage an FPGA-based NoC platform. Although other protocols are also analyzed to this task, the SNMP is chosen because of its simplicity and its flexibility to deal with devices from different manufacturers. Then we present the architecture of an FPGA-based NoC emulation platform based on the SNMP. The emulation platform is composed by synthesizable hardware components and by software components. We describe the protocol implementation as well as the traffic models used. An execution flow is proposed to carry out the emulation tasks. This flow is used on a design space exploration of a NoC based on task mapping and mesh topology parameters. At last, We present a set of experiments to evaluate the proposed protocol and the emulation platform.

- **Chapter 4** presents our second contribution a framework which analyzes a single application trace in order to retrieve the packets dependencies information. These packets dependencies are used to build an executable MoC which accurately replaces the application nodes onto a NoC emulation. The packets dependency extraction problem is formulated and the framework workflow is described. Afterwards, the architecture of a dependency-aware traffic generator is presented. At last, a set of experimental evaluations are carried out on a broad range of NoC architectures in order to evaluate the performance of the proposed framework.

- **Chapter 5** presents a prospective analysis of NGNoCs architecture. It starts with a description of the 3D-IC features. Afterwards a conceptual architecture of a NGNoC based on 3D-IC is presented. This conceptual architecture serves as a study model to explore the high-level communication requirements of a NGNoC. From this conceptual architecture of a NGNoC-based embedded system, We analyse the requirements of a NGNoC in three key aspects: congestion control, fault tolerance, and FPGA-based emulation. A prospective discussion on these subjects is carried out as well as we present some NGNoC propositions that fit the requirements of this conceptual architecture.

At last, **Chapter 6** concludes this thesis and presents possible future directions.

Chapter 2: State of the art for the FPGA-based NoC evaluation platforms

The aim of art is to represent not the outward appearance of things, but their inward significance.

Aristotle

Contents

2.1 Networks-on-Chip: an overview 6
2.1.1 NoC layer model 7
2.1.2 NoC research areas 11
2.2 FPGA-based NoC performance evaluation platforms 14
2.2.1 Network model 15
2.2.2 Traffic generation model 17
2.2.3 System management model 20
2.2.4 Limitations of FPGA-based evaluation platforms 21
2.2.5 Platforms examples 22
2.2.6 Targeted research challenges 23
2.3 Conclusions 25

Networks-on-Chip (NoCs) are currently the most appropriate interconnection architecture for the modern many-core embedded systems. A NoC offers a scalable, flexible and parallel communication technology to integrate complex logic cores in SoC. As NoCs become de facto on-chip communication standard, performance evaluations methodologies emerge as critical components for the validation of new NoCs architectures as well as their DSE. One of the most promising tools for NoC evaluation is the FPGA, because it accelerates the execution of evaluation scenarios while keeping their accuracy. This chapter presents a state of art of FPGA-based NoC performance evaluation platforms. The chapter is organized as follows. Section 2.1 presents a brief description of the NoCs architecture, as well as a highlight of some key research areas. Each one of these areas is subject of our research efforts during this thesis development. Afterwards, in section 6.2, the main features of those FPGA-based platforms are analyzed as well as their limitations. From this analysis, two research problems are selected, which are further analyzed in the next chapters. The chapter conclusions are presented in 2.3.

2.1 Networks-on-Chip: an overview

A SoC design has four main components types: computation, memory, I/O and communication. The communication components have attracted much more attention because of the advances on microchip technology, which they make for large scale SoCs comprised of tens or hundreds of complex IP blocks. These IP blocks must be interconnected by a flexible and scalable on-chip communication infrastructure. Therefore, the chip design migrates from a processing-centric paradigm to a communication-centric one.

NoCs emerge as the most promising on-chip communication technology, since they have greater scalability, flexibility and parallelism than other solutions, as well as buses and point-to-point connections. A NoC replaces design-specific wires with a scalable, general-purpose and multi-hop interconnection network. In fact, many industrial products use the NoCs technology, such as: Intel SCC [VHR+08], Polaris [HDH+10] and Tilera64 [BEA+08].

Despite its clear advantages over other on-chip communication technologies, a NoC has a simple four components architecture, which are: applications cores, Network Interfaces (NIs), routers and links. Figure 2.1 depicts a NoC architecture implemented on a 2x2 mesh. The applications cores use the NoC's communication services. They use the network to carry out their communications flows. These cores are manifold, but usually they are general purposes processors, DSPs, memory blocks or specialized IP blocks. The NIs are responsible for the interface between the cores and the network. They are also responsible for the end-to-end communication flows as well as for providing different communication services to the core such as: connection-oriented, connectionless and QoS enabled services. The routers are responsible for routing the packets through the network as well as to provide flow control mechanisms. They are connected to the NIs through the local port. At last, the links are the physical wires interconnecting the routers ports in order to match the network topology.

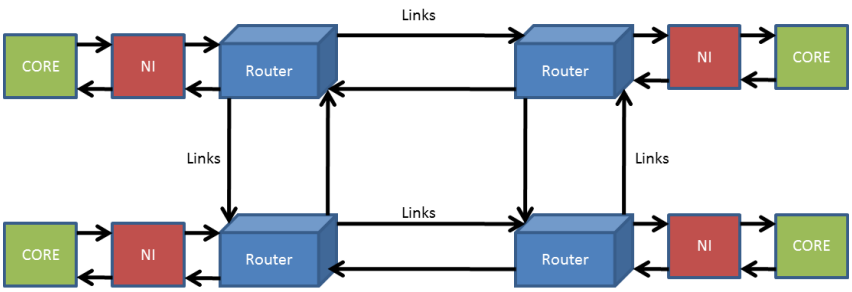


Figure 2.1: Architecture of 2x2 Mesh NoC.

The router is the main component of a NoC. Figure 2.2 depicts the architecture of a five ports router. This router has a local port connected to the NI and other four ports, which are used to connect it with its neighbors routers. Each input port has a FIFO buffer to hold the flits (flow control digits) during a packet transfer. The flit is the basic unit of storage allocation in a NoC. Whenever a new packet arrives, the control unit verifies its header in order to define the output routing path. The same unit is responsible for configuring the crossbar to connect the input port with the selected output port. When a packet transmission is finished the control unit updates the crossbar configuration

and it frees the other allocated resources. This is a simple router architecture for mesh networks, but other mesh routers architectures were proposed with different number of ports such as: diagonal ports routers [WHLB10] or double physical ports routers [CCM09].

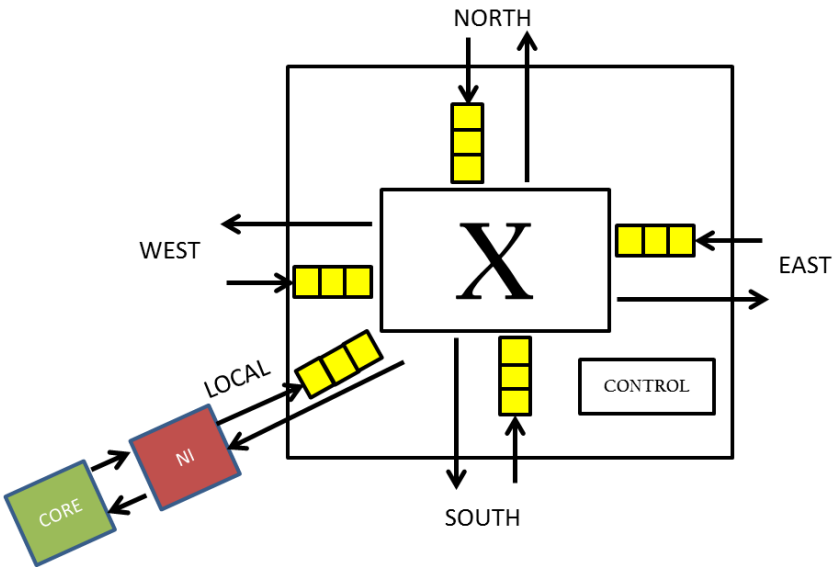


Figure 2.2: Architecture of a five ports router.

2.1.1 NoC layer model

Herein the NoC model is compared in relation to the interconnection network reference model Open Systems Interconnect (OSI). The NoC model has four layers which are mapped onto the seven layers of OSI model. Figure 2.3 depicts the relationship between the NoC layers and the reference model OSI. The system layer comprises the services of the two top OSI layers. This layer represents the application cores requirements. The network adapter layer is responsible for controlling the end-to-end communication services. This layer models the services of the session and transport layers of OSI. The network layer, in this turn, provides the same services of the network layer in OSI model. This layer services are implemented by the hardware router. At last, the link layer comprises the logical and physical links which interconnect the NoC components. The next subsections provide more details about the network adapter, network and links layers. The system layer is not further described here because we focus on the interconnection of the application cores rather than on their description.

2.1.1.1 Links layer

The links layer is responsible for the physical wires and the data-link control. The physical wires design focus on the design of signal drivers/receivers for the implementation of unreliable transmission lines. Eventual communication errors must be detected and corrected at the data-link control by using Error Correcting Code (ECC).

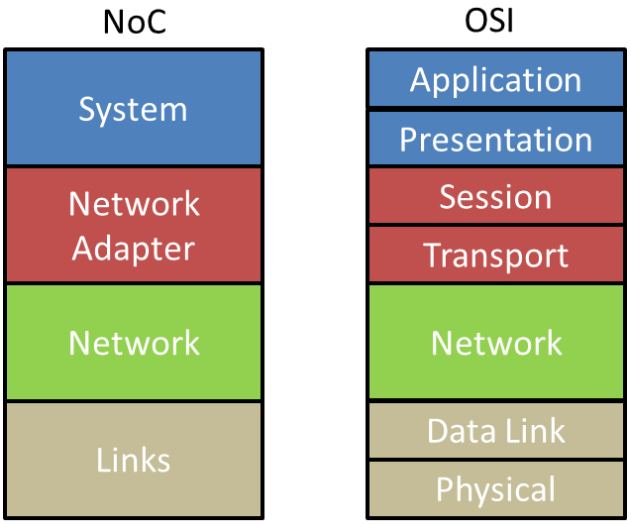


Figure 2.3: NoC layered model.

Adaptive error control methods may be applied to improve the links reliability in noise environments as describe in [YA08] and [YA12]. This layer also provides low level flow control mechanisms as credit-based or On/Off flow control.

2.1.1.2 Network layer

The network layer defines the protocols used for switching, flow control and routing. The switching is the transport of data. In order to transport data through the network, it is necessary to allocate channels and buffers. The flow control mechanisms allocate channel and buffer resources to transport data through a path, which is determined by the routing.

The two primary switching techniques are circuit-switching and packet switching. In circuit-switching, a physical path is established between the source and destination. This path is established before the transmission of the data. At the end of the transmission, the path is released. This technique adds substantial overhead due to the time spent on connection management, which increases the communication latency. In packet switching, the data is formatted in packets, which are sent without path pre-reservation. There are three primary packet switching methods: Store-and-Forward (SAF), Virtual Cut-Through (VCT), and wormhole. Each one of these switching techniques demands flow control to allocate the network resources, such as the channel bandwidth, buffer capacity and control state, to traversing packets.

In SAF, only after receiving the entire packet, the flow control allocates the buffers and channels of the next hop. It adds extra packet delay at every router stage. Furthermore, it also demands more buffer space. Figure 2.4 depicts an SAF example, a five-flit packet is routed through a four hops path. At each router, the packet is forwarded after the reception of all the flits. The transmission takes 20 cycles.

In VCT, a packet is forwarded to the next router as soon as there is enough space to store the packet. It overcomes the additional latency of SAF. However it still requires

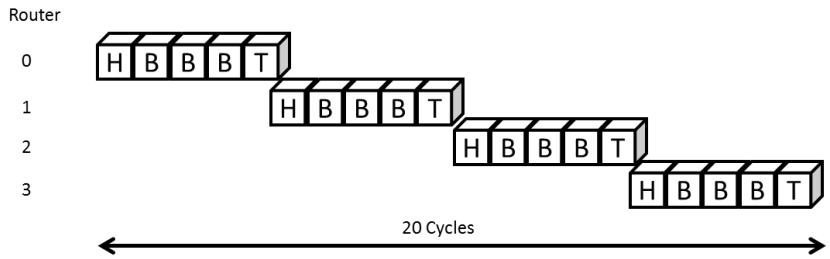


Figure 2.4: SAF example. H, head flit; B, body flit and T, tail flit.

the same amount of buffer resources, because when the next stage is blocked, the entire packet still needs to be stored in the buffers. Figure 2.5 depicts a VCT example, a five-flit packet is routed through a four hops path. In the third cycle, router 2 has only three buffer slots available and this adds a contention, which is depicted by the red square. When router 2 has all the buffer slots available, the packet is forwarded. Even with the contention of router 2, the packet transmission takes only 10 cycles.

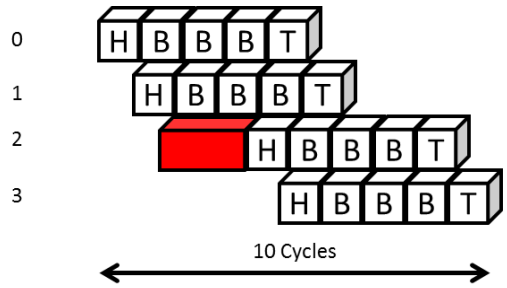


Figure 2.5: VCT example. H, head flit; B, body flit and T, tail flit.

In SAF and VCT, message flow control is performed at packet's level. In wormhole, the flow control is performed at flit level. In this way, wormhole requires only one empty downstream slot to forward a flit. In Figure 2.6, router 2 sends the header flit at cycle 3, even if there were not space to fit an entire packet. The packet transmission takes only 8 cycles. This method is faster than the previous two.

These three techniques allocate the channels at the packet granularity. Only one packet uses the channel during its transmission. In Virtual Channels (VCs) techniques, two or more FIFO buffers share one physical channel. The VCs improve the physical channel utilization.

Routing algorithms determine the traversal path of a packet for a given source-destination pair. In order to describe the routing algorithms, it is necessary to describe a fundamental aspect of NoCs: the topology. The topology determines the physical layout and links between nodes, routers and channels. It also influences the latency and power consumption. Therefore, NoCs usually are implemented in regular and simple topologies. Figure 2.7 depicts three typical NoC topologies. The ring topology is widely used for a small number of nodes. It has a very simple structure, which eases the implementation of routing and flow control. The 2D mesh topology offers

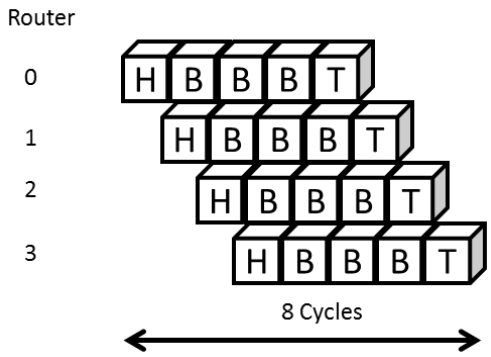


Figure 2.6: Wormhole example. H, head flit; B, body flit and T, tail flit.

more scalability than that of the ring one. It is widely used for scientific and industrial applications. The 3D mesh NoC consists of multiple tiers of 2D mesh NoCs implanted as a 3D IC. It achieves significant reductions in packet latency and power consumptions and it also allows the deployment of a large number of IP cores. The green rectangles in the figure depicts the transversal links which connect the different layers.

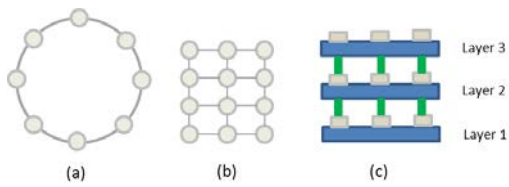


Figure 2.7: Topologies examples. (a) Ring. (b) 2D Mesh. (c) 3D Mesh.

For a given topology, the routing algorithm calculates the traverse path for a packet. We coarsely divide the routing algorithm in two categories: deterministic and adaptive. In deterministic routing, the path between each pair of source-destination is always the same. The path depends only on the coordinates of the communicating nodes. In opposite, the adaptive algorithms uses information about the network state to optimize the path during a packet transmission. Both routing strategies have potential problems like deadlock, livelock and starvation. A deadlock scenario is characterized when two or more packets are waiting to be routed because each packet waits for resources allocated to other packet creating a cyclic resources dependency. A livelock scenario happens when a packet keeps spinning without ever arriving in its destination. This is a problem that occur with adaptive non-minimal routing algorithms. A starvation scenario happens when low priority packets are always depreciated in favor of higher priority packets. The consequence is that the low priority packets never arrive at their destinations. The routing algorithms must implement mechanisms to avoid these problems.

Another important service of the network layer is the QoS mechanisms. They differentiate communication flows in order to ensure some performance requirements. Basically, there are two types of traffic flows. The first one has some kind of quality requirements, which are described by some desired communication performances like

maximum latency or minimum bandwidth. This kind of traffic is called Guarantee Services (GS). In its turn, the second one has only deliverance requirement and it is called Best Effort (BE). Usually, GS traffic receives network special attention, in such a way that resources allocation and routing decisions are biased by this traffic. The QoS mechanisms have deep influence in the design of NoC features such as: topology [WB12, CCM09, L+10b], arbitration [SMAG12, HKB12] and flow control [W+09, FXH11].

2.1.1.3 Network adapter layer

The network adapter layer decouples the cores from the network. It handles the end-to-end flow control and encapsulates the applications messages for its bottom layer. These messages are broken into packets or into connection-oriented streams. This layer offers a standard interface to the core applications to access the network services. It also manages the services provided by the network such as: connection-oriented, connectionless and QoS enabled services.

Error control mechanisms are incorporated in this layer in order to verify that no packet is lost in the lower layers. Another important service is the network congestion control, which can be implemented using admission control mechanisms, which regulate the network input load at the price of throughput.

Furthermore, this layer ensures independence of the network implementation details and it also translates the low-level communication protocols of the network the upper layers.

2.1.2 NoC research areas

In the last years the research community dedicated a lot of efforts to improve the NoC technology. In this subsection, we present an overview of some key research areas of NoCs. We have selected four key areas to show a brief description of the latest developments. In each one of these areas, we have contributed with relevant work during this thesis development. The main thesis contributions are on the NoC performance evaluation domain, the other contributions are secondary in the scope of this thesis.

2.1.2.1 Congestion control

The network congestion is an issue as it affects the overall performance of a NoC. Basically, a congested NoC can be understood as a NoC which suffers from degraded performance caused by the presence of too many packets in a determined network area. A possible solution to this problem is the adoption of a congestion-aware routing algorithm. Such algorithm uses congestion information in the routing process. Less congested paths are chosen to route packets. Those algorithms share some characteristics such as adaptive routing scheme and congestion detection scheme.

Some adaptive versions of the deterministic XY routing algorithm are used due to its simplicity [VMPL10, WHB10, WB12]. The Odd-Even model is also applied by [HH09] and [MKM10] because it has more adaptability than other partially adaptive algorithms [Chi00]. The DyXY [LZJ06] is a congestion-aware routing algorithm, which sends the packets along a shortest path between the source and the destination. If there is more than one shortest path, the least congested path is taken. This algorithm requires two virtual channels to avoid deadlocks.

We have proposed the FlexOE [dLJFR13], which is a congestion-aware routing algorithm which selects a packet route based on a simple and flexible scheme of prioritized sets of rules. These sets of rules are based on the Odd-Even turn model, minimal paths checking, congestion information from adjacent routers and availability of output path. The FlexOE features are presented in the chapter 5.

2.1.2.2 Fault-tolerance

A NoC is subject to diverse faults arising from different sources. For example, a fault in only one transistor of a NoC may cause serious failure in one chip [DM04]. The faults in transistors can be caused by diverse factors connected to the manufacture process and fatigue of its own material, and amongst these factors, we can highlight: oxide breakdown [Sta01] and negative bias temperature instability [Ala03]. Additionally, the electrical conductors utilized in chips are subject to electro-migration [Gha82], which can cause failures.

The proposals for faults tolerance mechanisms for NoCs can be divided into two groups: fault tolerance algorithms and architectural solutions. The proposals for fault tolerance algorithms for NoCs, explore the multiple paths between the source and destination nodes, in order to augment resilience. We can divide these algorithms into three categories: stochastic, adaptive and partially adaptive. The stochastic is based upon probabilistic sending of multiple copies of the same packet by different routes. Examples of these algorithms are found in [ZPG07] and [PLB+04], such as: directed flooding, N random walk and gossip flooding. The major problems concerning these algorithms are related to the high consumption of energy and low performance. The adaptive algorithms use routing tables in each router that are periodically utilized to reflect the actual state of the network. However, these algorithms contain various disadvantages, such as: update time of the route tables and failure scalability, since the route tables increase with the size of the network. [PP12] proposes an adaptive and fault-tolerant routing algorithm based on gradient lines, which divide the destination nodes into eight zones. This algorithm does not use routing tables and it finds the shortest path to reach the destination node. In its turn, the partially adaptive algorithms limit the degree of adaptability by imposing diverse rules in the routing with the intention of avoiding deadlocks. Some examples of these algorithms are the proposals based upon turn models [GN92], such as West-first; North-last and Negative-first. Nevertheless, owing to the restrictions concerning the possible turns in order to avoid deadlocks, it is not always possible to define a route for each packet. On the other hand, the architectural solutions deal with fault tolerance through hardware redundancy and employment of ECCs. Examples of these approaches are provided by Vicis [DFB+12] and DPB [KMAMP08].

We have proposed [CdLJ13] a NoC architecture for a fault-tolerant multiprocessor chip based upon the use of backup paths which are external to the router, which allows maintaining communication between non-faulty network's routers. In the event of a permanent failure in a router, the control mechanism of backup paths is triggered in such a way that the faulty router is by-passed and the data is routed in a transparent way. This NoC architecture is presented in the chapter 5.

2.1.2.3 Task mapping

Task mapping is a main concern in the NoCs research. It maps the tasks to the application cores of a NoC. The tasks placement affects the overall performance and the energy consumption of a system. In recent years, several researches on mapping application tasks on NoCs have been carried out. The task mapping algorithms use either dynamic or static strategies. Dynamic mapping enables tasks placement during an application execution. The run-time approach of incremental mapping [COM08] and the congestion-aware algorithms [CM08] are well-known dynamic mapping techniques. Some algorithms create tasks partitions before executing the dynamic task mapping as proposed by [ASA+12]. These algorithms regulate the tasks placement online according to the systemic feedback. Static mapping is implemented before the application execution, i. e., it is an offline execution. The static mapping solutions are generally recommended because the computational overhead of dynamic mapping algorithms increases the overall delay and the energy consumption of a system [SC13]. Two-step Genetic Algorithm (GA) [LK03],Branch-and-Bound Algorithm (BB) [HM05] and Template-based Efficient Mapping Algorithm (TEM)[WYJL09] are typical static mapping techniques.

We have explored the task mapping in conjunction with mesh topology exploration using FPGA-based NoC emulation [PFYDLJ15]. This research is presented as a case study in chapter 3.

2.1.2.4 Performance evaluation

The NoCs have a large design space concerning different aspects like topology, routing, flow control, QoS, among others. A SoC designer must make several architectural choices and performance evaluations to find the best NoC configuration for a given project. Furthermore, all those system aspects must be tested and validated. The design and verification time for complex on-chip communication systems continue to grow exponentially. NoC specialized design tools are used to reduce the design costs.

Analytical modeling provides quick evaluation models to quantify some NoC metrics as power, area, packets delay [Qia13, KLJ13, FTHJ09]. However, as the embedded systems complexity grows modeling their behavior without a detailed simulation is challenging. System simulation is an essential tool to evaluate new designs. High level simulation [CCG+04, HAAN+07, BDM+05] are used in the early stages of development process for a faster, however less accurate performance evaluation. Usually those simulations are written in high level languages like C, C++, SystemC, and Java. Cycle Accurate and Byte Accurate (CABA) simulations [Pra10] have higher accuracy in detriment of very high execution times. They are used to validate an HDL model. Full-system simulation is a very accurate evaluation tool due to its detailed model of the application cores as well as the NoC. However, its execution time may be on the order of many weeks or months [HGK10].

The FPGA-based emulation platforms [GADM+05, LPG11] are a promising technique for NoC design space exploration and benchmarking. Those platforms drastically reduce the design time and they also have high accuracy. We have evaluated the adoption of a management protocol to run an FPGA-based NoC emulation [dLJFR14]. This protocol is presented in the chapter 3. We also investigate the synthesis of a FPGA-based

NoC emulation platform from the analysis of simulation traces. This trace analysis framework is presented in the chapter 4.

2.2 FPGA-based NoC performance evaluation platforms

The FPGA-based NoC performance evaluation platforms are important tools for NoC design. Indeed, the FPGA potential to run fast and accurate NoCs evaluations enables the design of several architectures of evaluation platforms. We coarsely divide these platforms in two categories according to their objective: prototyping and simulation. The prototyping platforms are used to verify a NoC design. The FPGA-based prototyping platforms are complex because they accurate model all the components of a NoC. In its turn, FPGA-based simulation platforms accelerate the evaluation of a NoC using a less accurate network model but it is used as a faster early design simulation for a well-known NoC architecture. Despite the differences between these two approaches, we depict a general organization of those platforms in Figure 2.8.

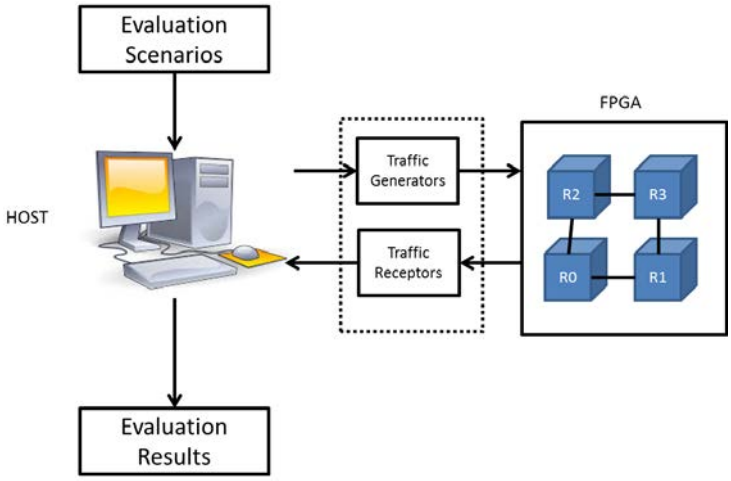


Figure 2.8: General organization of an FPGA-based NoC Performance Evaluation Platform.

The main components of a performance evaluation platform are depicted in Figure 2.8. The host is a computer connected to an FPGA board through a data link. A software running on the host is responsible for the setup of the platform according to pre-defined evaluation scenarios. These scenarios may be generated by another software tool in order to evaluate a NoC. The host is also responsible for extracting the evaluation results from the platform. These results may be also target of further analysis by specific software analysis tools. We identify the host tasks as system management tasks, because it manages the platform configuration and it must offer the platform services to other tools on a NoC design workflow.

A TG is responsible for generating traffic based on real or synthetic models. A TR is responsible for retrieving the packets and computing performance metrics. In a full-system evaluation, these components are the real application cores. In other evaluation scenarios, specific components are used as TGs and TRs. These components are not

placed on the FPGA in Figure 2.8, because there are different architectural strategies for them. They can be implemented as software in soft or hard processor core, as well as they can be implemented as specific IP blocks on the FPGA. A NoC performance evaluation depends on traffic generation models, which specify the spatial-temporal distribution of a traffic workload. It is important to validate new NoC architectures with realistic workloads, because they provide much more accurate results. Traditional traffic generation models based on synthetic workloads do not accurately represent the characteristics of real applications [GK10]. Executing real applications in a FPGA-based NoC emulation is quite challenging. The TG and TR are understood as the components of the traffic generation model.

The last component is the NoC, which is the evaluated component. The NoC model used on a platform depends on the objective of the evaluation. Prototyping platforms aim at the validation of a complete network and it entails high implementation and verification efforts. On the other side, simulation platforms use simpler models to accelerate the evaluation. These simulations can be used for DSE in the early stages of development. Moreover, simulation platforms can apply Time Division Multiplexing (TDM) techniques, which enable the use of one router model to simulate a whole network. This increases the size of a network implemented on a FPGA. These architectural implementations are understood as the network model.

The analysis of a generic FPGA-based NoC performance evaluation platform lead to an architectural model divided in three sub-models: system management, traffic generation and network. These models are explained in the next subsections.

2.2.1 Network model

An FPGA-based NoC performance evaluation platform aims at reproducing an accurate network model. A NoC is a complex system and its verification entails great efforts and it also may demand a lot of FPGA's resources. The choice of the network model to be implemented on such an evaluation platform is a trade-off between accuracy and resources utilization. The prototyping of a fully functional NoC is necessary for the validation of a new NoC design. However, a less complex model may be used to DSE. Herein, we present four implementation techniques of the network model: direct-mapping, multi-FPGA, simulation and virtualization.

2.2.1.1 Direct-mapping and direct-mapping over multi-FPGA

The natural approach to evaluate a NoC on FPGA is to synthesize the NoC in its entireness. This approach is useful for validating a new NoC design. However, the FPGA's available resources may restricts the network size. For instance, in [LPG11] a 5x5 mesh NoC is implemented using direct-mapping approach. Other platforms propositions use this approach but they also have a small number of routers. In [SZ12] and [HGM+12] a 3x3 mesh is implemented and in [TFR11] a 4x4 mesh is implemented.

The utilization of several FPGAs mitigates the lack of resources, but it adds up new challenges. The inter-FPGA communications are slower than the internal data links. This throughput mismatch may reduce the execution accuracy. On the other hand the number of implemented routers is higher than the number of implemented routers in direct-mapping platforms. For instance, a 48-nodes NoC is implemented on four FPGAs

in [LH08]. The abundance of resources of a multi-FPGA platform enables the emulation of a NoC and its applications cores. In [L+10a] four FPGAs are used to evaluate a 4x4 many-core system comprising the NoC and the processors cores.

Figure 2.9 depicts the organization of both approaches for the implementation of a 9x5 Mesh NoC. This network is implemented in four FPGAs in a not symmetrical nodes distribution or in just one resourceful FPGA.

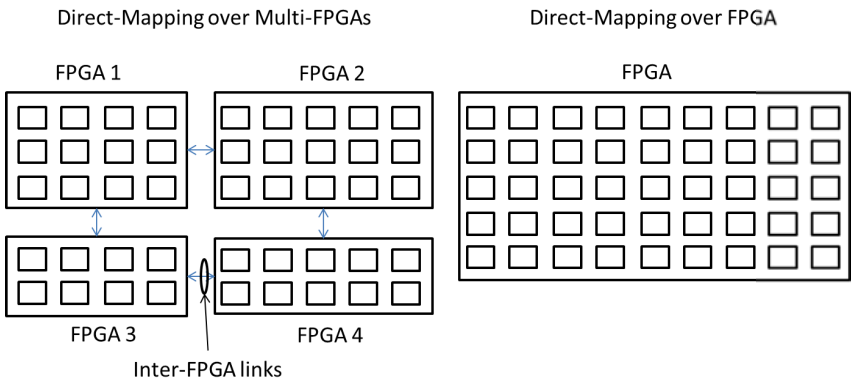


Figure 2.9: Direct-mapping and direct-mapping over multi-FPGA.

2.2.1.2 Simulation and virtualization

FPGA-based NoC simulation accelerates the evaluation time in relation to software simulation. However, it can not replace prototyping to validate a new NoC design, because it uses a simpler and lighter network model. A natural approach for NoC simulation is to design a lighter router to replace the actual router hardware, which is much more complex. In [PHM11] the NoC is modeled as a set of load-delay curves. These curves are obtained through training for a given network configuration and traffic pattern. In [WLV+14] a flexible NoC simulation engine is presented. This simulator uses a fully-connected collection of fixed-functions components to model a NoC without re-synthesizing it.

Those approaches may be also very resource demanding. Designers have proposed to use TDM to further optimize resources utilization in detriment of execution time. In TDM one single physical node emulates the behavior of the entire network. This virtual network takes more time to execute because the routers are executed in sequence. Figure 2.10 depicts the temporal organization of the simulation of a virtual NoC. In this example, a logical node execution takes two FPGA clock cycles. One emulator cycle takes $2 * N * FPGA_{clock}$, where N is the number of virtual nodes. On the other side, this approach leads to the simulation of an important number of routers. For instance, a 256-nodes TDM simulation is presented in [Pap11]. Moreover, in [CSK14] a 32x32 mesh NoC is evaluated using TDM. This platform is capable of simulating more than one thousand nodes.

TDM requires special care to keep the network events order and careful state management to ensure a consistent simulation. The routers' state is kept on memory buffers, which must be updated after the execution of neighbors logical nodes. The number

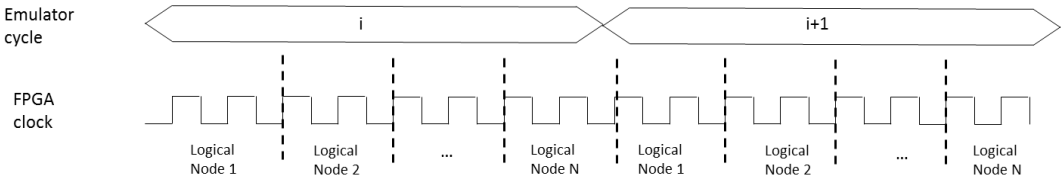


Figure 2.10: Clock schema of Virtual NoC.

of simulated nodes depends on the resources allocated to hold their state. Figure 2.11 represents the organization of a virtualized router. The router's state holds the buffers and links allocation's information, as well as other important state's information. The router logic circuit is connected to a TR and a TG. These components must be specifically designed to deal with the clock schema of a virtual NoC.

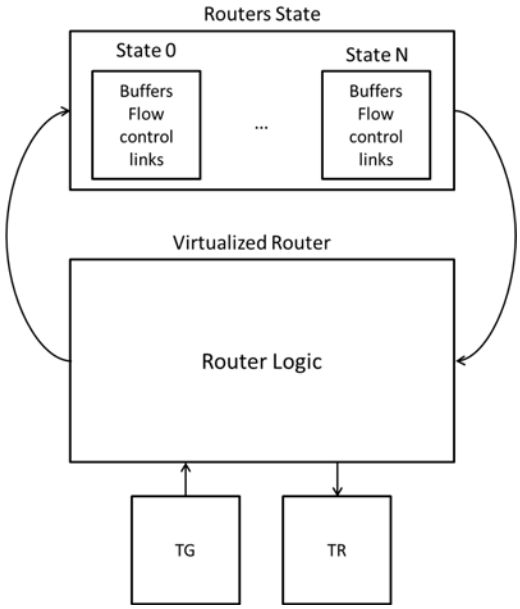


Figure 2.11: Virtualized Router.

2.2.2 Traffic generation model

A traffic generation model describes the spatial-temporal distribution of the traffic workload used to evaluate a NoC. It deals three important questions: when to send packets, where the packets should be sent to and how big these packets are. This is a main concern on NoC performance evaluation. A well balanced NoC design accommodates the application traffic without over-provisioning the network resources. A not consistent traffic pattern leads to a not optimized NoC architecture.

There are two approaches for generating traffic on NoCs: synthetic traffic and application-based traffic. Synthetic traffic is used to stress the network in order to gain understanding of its performance. This traffic does not represent the characteristics

of real applications, but it can be used to evaluate the overall performance of a NoC. Application-based traffic tries to emulate the traffic generated by real application nodes. This traffic is more closer to the real one and it can be used to evaluate the performance of a NoC for a specific application. It is up to the system designer to select the appropriate traffic model to evaluate a NoC according to the application requirements as well as the limitations of the FPGA-based platform. Table 2.1 briefly presents the main traffic models described in this section.

Traffic Model	Type	Description
Bit permutation	Synthetic	The destination address is a function of the source address
Uniform random	Synthetic	Each node sends uniform distributed traffic to the other nodes
Trace-based	Application	It uses traces from full-system simulation
Dependency-aware Traces	Application	It uses traces enriched with packet dependencies information
Stochastic	Application	Stochastic models generated from application traces
Application Cores	Application	It is based on the execution of the real application cores or their execution models

Table 2.1: Traffic models description.

2.2.2.1 Synthetic traffic

Synthetic workloads provide insight into a network's performance bottlenecks. However, it fails to predict a network's performance under a realistic traffic workload. Their main advantage is the simplicity to design TGs in software or in hardware. They are used in many FPGA-based platforms [WHS07, TFR11, WLV+14].

In synthetic workloads, a stochastic random process is used to inject packets into the network. The spatial distribution is chosen in order to model the characteristics of realistic workloads. The synthetic patterns used in NoC evaluations include the following: uniform random and bit permutation patterns.

The uniform random traffic distributes equally the traffic over the network nodes. Each node sends packets randomly to other nodes with same probability. Some variations of this pattern change the transmission probability depending on the nodes's distance (Nearest Neighbor Traffic) or to create artificial congestion (Hot Spot Traffic). Additionally, the latency between packets and the packet length are chosen using uniform distributions.

The bit permutation patterns use an one-to-one communication pattern. Each node sends all its packets to only one destination node. The destination address is chosen according to the source address. As this traffic concentrates the load on individual source-destination pairs, it tends to stress the load balance of a NoC. Bit complement is

an example of bit permutation traffic. In order to calculate the destination address, a bit-wise inversion of the source address is performed and it provides a balanced load. Other examples of bit permutation patterns capable of generating balanced load are: bit complement, shuffle, bit rotation and N complement. The transpose pattern chooses the destination node via the transpose of the source coordinates. It generates a highly imbalanced load.

2.2.2.2 Application-based traffic

Realistic traffic patterns are based on the behaviors of real applications. It is important to evaluate NoC architectures under real traffic because it provides more accurate results. We divide the realistic traffic pattern propositions in three categories: trace-based, stochastic and application cores execution. Trace-based traffic is a well-know technique for generating realistic workloads. A full-system simulation is carried out in order to gather the communication traces. Executing trace-based traffic in an FPGA-based NoC evaluation platform is quite challenging. Some evaluation platforms enable trace-based traffic like [WLV+14, GADMB07, KSPK10]. A trace can easily comprise millions of packets. The traces size is an important issue for NoC emulation because of the limited internal RAM available on FPGAs. Furthermore, trace-based execution distorts the injection rate and effects of the congestion due to the lack of packets dependencies information. Some works [NMFA11, TAA+11] propose the utilization of dependency-aware traces, which are traces enriched with the packets dependencies information. These traces tend to be bigger than the original ones. However, they are more accurate than ordinary traces for a broad range of network architectures.

A dependency-aware trace keeps tracks of each packet dependencies, thus it ensures proper interleaving of network messages, which increases the fidelity of the simulation by eliminating the artificial network contention. In [HGK10], dependency-aware traces are used to model the traffic in a CMP (Chip Multiprocessor) simulation. A similar approach is used in [TAA+11]. In both cases, the network traffic consisted in cache memory references and cache coherence protocol messages. In [NMFA11] multiple simulations traces from different network configurations are used to create a dependency-aware trace. A methodology to determine the number of input traces is not proposed. This proposition is also evaluated only for shared memory systems. In [HCT+12] the packets dependencies are asserted by observing the packets received within a determined time window. However, there is no clear method to find out this time window parameter. A table structure to periodically generate traffic is also proposed, but a technique to find out the traffic generation period is not proposed. The previous dependency-aware traces propositions are targeted to software simulations. Some of them also depend on the network architecture used to generate the analyzed traces.

The stochastic traffic is a technique that generates synthetic traffic having statistical properties similar to real applications. It models the traffic by using trace analysis. In [VM04], a synthetic generator is proposed based on the statistical properties obtained from real video applications. Another empirically-derived stochastic traffic model for NoCs is presented in [SWP06], which uses three statistical parameters to model the spatial-temporal distribution of the traffic. A similar approach is proposed in [BB09]. All these works generates synthetic traces based on real applications. These traces are used on FPGA-based NoC evaluations. However, they do not model the packets

dependencies information as the traces gathered from full-system simulations do.

The last technique to generate realistic traffic is to execute the application cores. A detailed implementation of the application cores can be used on the evaluation, but it usually demands a lot of FPGA resources. In [LH08], four FPGAs are used to emulate 48-processors cores and one NoC. HAsim [PAK+11] is a FPGA-based multicore simulator that uses TDM techniques to simulate 16 processors cores and a NoC. The application core's behavior can be emulated by a discrete-event system like FSMs, Petri Nets or Kahn Process. These models can be used to build simpler traffic generators.

2.2.3 System management model

The system management model can be understood as an abstraction layer which offers a simple and efficient interface to the services provided by an FPGA-based NoC evaluation platform. The platform components must be accessed through a standard interface, which eases the platform setup task as well as the extraction of the evaluation results. Standard network management protocols may be adapted to carry out those tasks. This may ease the sharing of IP blocks targeted at NoC evaluation on FPGAs. This layer is omnipresent in the FPGA-based evaluation platforms. The majority of the proposed platforms do not highlight its properties, they aim at the network and traffic models. However, this layer is essential to the execution of a NoC evaluation and to the integration of a platform in a NoC design flow.

A high level description of a traffic workload is one of the inputs to start a platform configuration task. The system management component analyses a traffic description and it translates it into one in a set of low-level commands to program the platform's components like TGs and TRs. This task must be executed independently if those components are implemented as hardware or as software in an embedded processor core. These commands are sent through the data link between the host and the FPGA. A network configuration may also be needed when executing simulations. This configuration may change the NoC topology, buffers size, flow control techniques, among other parameters. This procedure is depicted in Figure 2.12.

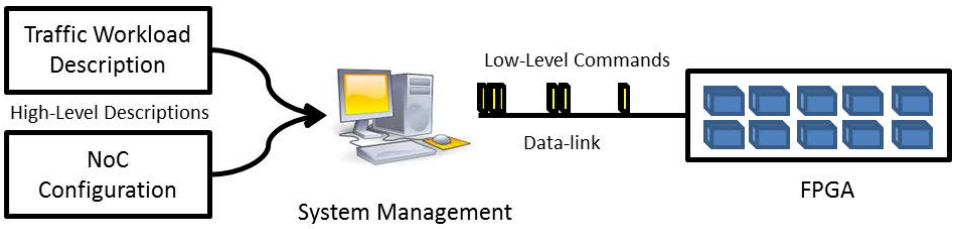


Figure 2.12: Platform configuration.

After the setup of the platform, the system management component must monitor the execution of the evaluation scenario. Afterwards, it must gather the evaluation results allocated on the TRs. This also demands the transmission of a series of low-level commands between the host and the platform. The results must be further processed in order to calculate the performance metrics.

It is important to highlight the role of the system management for an FPGA-based NoC evaluation platform. The design of a new NoC architecture or its DSE are complex

tasks which follow a specific design flow. This design flow uses several development tools in order to model, synthesize, simulate, validate a design. An FPGA-based evaluation platform is one more tool in this design flow. Thus, it is necessary to provide standard protocols to interface an FPGA-based evaluation platform to third parties tools. Moreover, these standard protocols enables the utilization of emulation components created by different designer with minimum development effort.

2.2.4 Limitations of FPGA-based evaluation platforms

Despite the advantages of FPGA-based NoC evaluation platforms, there are some significant challenges to overcome. Herein we describe four limitations of those platforms. Prototyping platforms are more affected by these limitations, but they may also affect simulation platforms. These limitations are intrinsic of FPGA technology, thus they must be carefully considered in order to carry out a FPGA-based NoC evaluation.

2.2.4.1 Design partition

Unfortunately not all designs fit in one FPGA. Multi-FPGA platforms are the only alternative to many NoCs and full-systems emulations. The partition of a network and the emulation components across several devices is not straightforward. Indeed, this task is error prone and it requires a lot of interconnection pins, which quickly overcome the FPGA availability. Moreover, in order to improve the system performance, a Printed Circuit Board (PCB) design is required to accommodate all the FPGAs. This entails a high development effort, which could enlarge a project schedule. A solution for this problem is to connect standard FPGA boards in order to create a multi-FPGA platform deployed over several boards. This solution has as main drawback the low performance of the multi-boards connections.

2.2.4.2 Reusability

The reuse of design components saves a lot of design time and lower a project risk. Specific emulation/simulation FPGA boards may not fulfill a new project's requirements. As NoCs grows in complexity, they may no longer fit in older FPGAs. This is even more problematic for Multi-FPGAs platform, because it may require the design of a new board.

2.2.4.3 Debugging

The debugging capabilities of modern FPGAs are limited. Probing signal inside an FPGA in real-time is a challenge. The alternative is recompiling the design to move the probes, but it usually takes too long. The FPGA's internal logic analyzers address the signals visibility issues. However they have several limitations, including memory size and long reconfiguration time to change probes.

2.2.4.4 Synthesis Time

The FPGA's synthesis time of a complex design can easily take several hours. A minor change in the design of a NoC or in any platform component may start a slow synthesis

process. The simulation platforms are less affected by this limitation because they are more flexible to change several NoC's parameters without re-synthesis. The dynamic reconfiguration features of some FPGAs may mitigate this problem in some cases.

2.2.5 Platforms examples

Table 2.2 presents some of the main propositions of FPGA-based NoC evaluation platforms. From this table it is possible to gather interesting information about the different approaches to evaluate NoCs on FPGAs. At first, the number of evaluated nodes depends on the network model. Platforms using virtualization techniques enable the evaluation of bigger networks. However, direct-mapping usually has a small number of nodes mainly when it is combined with the emulation of the application cores. Another important point is the execution frequency. Direct-mapping platforms exhibit high execution frequencies whereas simulation and virtualization platforms have smaller execution frequencies. There are three exceptions: [CSK14], [PHM11] and [Pap11], but the displayed frequencies are from the simulation engine circuit. They do not correspond to the virtual router execution frequency, which is much lower due to the sequential execution of the virtual routers.

Some proposals [LPG11, WHS07] go for software implementations of the TGs and TRs in a dedicated processor. This approach enables easy configuration of the emulation platform. However the communication between the dedicated processor and the NoC is considered as a bottleneck. A single processor cannot emulate the traffic generated by all application cores for large NoCs at high offered loads, neither it emulates the parallel nature of the application cores.

AcENoCs [LPG11] is a flexible and cycle-accurate FPGA emulation platform for validating synchronous and GALS-based NoC architectures. The platform is divided into two frameworks. The hardware framework comprises the interconnection network to be emulated. The software framework is implemented in a soft core processor and it consists of logical implementation of traffic generators, source queues and traffic receptors. The connection between the two frameworks represents a performance bottleneck.

DART [WJS11] is an FPGA-based NoC emulation platform that enables different NoC architectures to be simulated without recompiling/resynthesizing the DART engine. The routers are organized in partitions and the partitions are connected through a crossbar switch. DART simulates a NoC by mapping the simulated NoC to DART partitions and by reconfiguring the crossbar. DRNoC [KCdlTR08] is a NoC emulation platform. This platform is based on FPGA partial reconfiguration capabilities. Therefore, it enables the dynamic insertion or removal of routers and cores in a mesh NoC.

It is important to highlight some missing features in the listed platforms. There is no platform which uses a discrete-event system to model the application cores. The propositions, which use application cores traffic, implement a detailed processor core circuit to generate the application traffic. Discrete-event systems may be an interesting approach to model the application cores with lighter implementation costs. Another handicap is the inexistence, it is the system management model features. The presented platforms do not cover its integration in a generic NoC design workflow. They neither provide standard protocols to manage the platform components nor provide interoperability to integrate emulation components from different designers. The adoption of a standard

protocol eases the integration of emulation components created by different designers.

2.2.6 Targeted research challenges

From the analysis of the main characteristics of the FPGA-based NoC evaluation platforms, it is possible to highlight two key research challenges: interoperability and realistic workload generation. These two research challenges are not sufficiently explored in the recent scientific reviews. Nevertheless, both lead to important improvements on the nowadays and the next-generation of NoCs.

2.2.6.1 The inter-operability problem

The FPGA-based NoC emulation platforms uses TGs and TRs which are implemented as hardware or software components. In order to execute an evaluation scenario, all the evaluation components must be configured and then they must be accessed to extract the emulation results. There is no standard interface to manage the emulation components from the host PC perspective. The previous proposals of FPGA-based evaluation platform do not address the issues related to the inter-operability of evaluation platforms components. The adoption of a standard protocol eases the integration of emulation components created by different designers.

The management of the TGs and TRs imposes challenges. The adoption of already tested management concepts enables taming the complexity of the NoC emulation platform. Moreover, a management protocol creates a common interface to manage emulation components from different manufacturers from a host PC. Management protocols such as SMBus [smb], NC-SI [nc-], WMI[wmi], RDP [rdp], NetFlow [net] and SNMP could be solutions for integrating TGs and TRs. It is necessary to investigate the features of these protocol to evaluate their fitness to the NoC emulation context. The inter-operability of evaluation platforms components rises as a key research challenge on NoCs' design. Providing standard protocols is essential to ease the sharing of IP blocks targeted at NoC evaluation on FPGAs.

2.2.6.2 The realistic workload problem

Improving the accuracy of the workload used to carry out NoC evaluations is essential to the development of the next-generation of NoCs. However, the traditional traffic generation models based on synthetic workloads do not accurately represent the characteristics of real applications [GK10]. The most accurate way to evaluate a NoC is running real applications. However, these full-system simulations require a lot of resources of the FPGA. To overcome this problem, designers use traces from full-system simulations, which are faster to execute. However, the temporal information found on traces is not valid for a broad range of architectures. Dependency-aware traces are more accurate, but they demand more storage resources. The traces generated by statistical traffic modelling have the same problems of traces gathered from full-system simulations. It is necessary to investigate new approaches for the synthesis of lightweight and accurate TG architectures based on applications traces.

Platform	Network Model	Traffic Generation	Number of Routers	FPGA	Frequency	Year
[CSK14]	Virtual	Synthetic	1024	Virtex 7	42 MHz	2014
[FGTR12]	Multi-FPGA	Synthetic	18	2 Virtex 5	-	2012
[HGM+12]	Direct-Mapping	Application Cores	9	Virtex 5	-	2012
[TFR11]	Direct-Mapping	Synthetic	36	Virtex 5	-	2012
[PAK+11]	Virtual	Application Cores	16	Virtex 5	3.15 MHz	2011
[KC11]	Virtual	Application Cores	64	Virtex 2	-	2011
[PHM11]	Simulation	Synthetic	576	Virtex 6	300 MHz	2011
[Pap11]	Virtual	Traces	256	Virtex 6	152 MHz	2011
[LPG11]	Direct-Mapping	Traces and Synthetic	25	Virtex 5	-	2011
[WHLB10]	Simulation	Synthetic	49	Virtex 6	50 MHz	2010
[KSPK10]	Direct-Mapping	Traces	64	Virtex 2	45 MHz	2010
[L+10a]	Multi-FPGA	Application Cores	16	5 Virtex 5	-	2010
[LK09]	Direct-Mapping	Application Cores	9	Virtex 2	50 MHz	2009
[LH08]	Multi-FPGA	Application Cores	48	4 Virtex 2	-	2008
[KMSP08]	Multi-FPGA	Synthetic	16	2 Virtex 2	-	2008
[KCdlTR08]	Direct-Mapping	Synthetic	16	Virtex 2	-	2008
[WHS07]	Virtual	Synthetic	64	Virtex 2	91.6kHz	2007
[GADMB07]	Direct-Mapping	Synthetic	6	Virtex 2	50MHz	2007

Table 2.2: FPGA-based NoC Evaluation Platforms

2.3 Conclusions

In this chapter, we present an overview of NoCs, which are the most appropriate on-chip communication technology for the modern embedded many-core systems. The organization of a NoC services is modeled as a protocol stack inspired on the OSI model. Furthermore, a short overview of four key NoC research areas is described. In each one of these areas, we have contributed with relevant work which is presented in the next chapters.

We also present the main features of FPGA-based NoC evaluation platforms. These tools are essential for accelerating the evaluation of new NoCs architectures as well as their DSE. A three-component model is presented to describe the organizations of those platforms. These models enables a clear understanding of the different architectures available for evaluating NoCs on FPGAs.

From the features analysis of previous FPGA-based platforms, we select two key research problems. These research problems are the main subject of this thesis. They are treated in depth in the next chapters.

Chapter 3: A SNMP-like protocol to manage an FPGA-based NoC emulation platform

We do not imitate, but are a model to others.

Pericles

Contents

3.1 Overview 28
3.2 Analysis of standard computer protocols to manage a NoC emulation platform 31
3.3 Adapting SNMP to NoC emulation management 32
3.4 A SNMP-like architecture of an FPGA-based NoC emulation platform 34
3.4.1 Architecture overview 34
3.4.2 Protocol implementation 35
3.4.3 Traffic models 36
3.4.4 MIB organization 39
3.4.5 Execution flow 41
3.4.6 Implementation details 43
3.5 Experiments 44
3.5.1 SNMP operations evaluation 45
3.5.2 MIB update evaluation 45
3.5.3 Resources analysis 46
3.5.4 Case study: task mapping and mesh topology exploration . . . 49
3.5.5 Results discussion 51
3.6 Conclusions 52

An FPGA-based emulation platform can drastically reduce the time needed to evaluate a NoC, even if this network is composed by tens or hundreds of distributed components. Moreover, additional components are implemented specifically to run the emulation of a NoC. These emulation components are responsible for generating data traffic, measuring network performance, communicating with a host PC, among

other functions. Using software running on a host PC, a designer manages an emulation platform and its components. The communication between the host PC and the emulation components is vital to the evaluation of a NoC. Nevertheless, there is a lack of standard protocols to drive FPGA-based NoC emulations. These protocols could enable the configuration of an emulation platform without FPGA re-synthesis, the extraction of evaluation results, as well as they could ease the integration of emulation components developed by different designers. In this chapter, we evaluate a light version of SNMP to manage an FPGA-based NoC emulation platform. The SNMP protocol and its related components are adapted to a hardware implementation. Some experiments highlight that the selected protocol is quite simple to implement and very efficient for a light resources overhead. The proposed emulation platform is also evaluated on a case study in which a NoC design space exploration is carried out. The rest of this chapter is organized as follows. Section 3.1 presents the context of FPGA-based emulation platforms. Section 6.2 evaluates the adoption of different management protocols in the context of NoC emulation. In section 3.3 is described the proposed modifications on the SNMP to manage a NoC emulation platform. Afterward, the emulation platform is presented in section 3.4. The evaluation experiments are presented in section 3.5. At last, section 3.6 presents the chapter conclusions.

3.1 Overview

The FPGA-based emulation platforms accelerate NoC benchmarking as well as design space exploration. Those platforms have high accuracy and low execution time in relation to NoC simulators. Although, the NoC hardware emulation is quite challenging. Indeed, a platform shall also provide ways to execute different traffic scenarios without FPGA re-synthesis. An FPGA-based emulator uses specific components to inject and to extract packets in and from the network: the Traffic Generators (TGs) and the Traffic Receptors (TRs). A TG injects traffic according to a traffic workload description. A TR extracts the data packets and calculates performance metrics. Those components are implemented in hardware or software.

Some proposals [LPG11, WHS07] execute software implementations of the TGs and TRs in a processor core, usually implemented as a node or a component of the emulation platform. This approach enables easy configuration of the emulation platform. However the communication between the dedicated processor and the NoC is considered as a bottleneck. A single processor cannot emulate the traffic generated by all application cores for large NoCs at high offered loads, neither it emulates the parallel nature of the application cores. It is neither affordable to connect one processor in each router, because it demands very more hardware resources.

In opposite, the hardware implementation of traffic related components occupies valuable FPGA resources, but it ensures higher communication bandwidth, because the emulation components may be directly connected to the NoC. However it is hard to configure the emulation platform because its components are physically distributed. Figure 3.1 represents this organization for a simple 2x2 mesh NoC. In this architecture a host PC is responsible for configuring the TGs and for retrieving the emulation results from the TRs. An external data link connects the host PC and the FPGA board. A control component is responsible for interfacing the external data link protocol to an on-chip

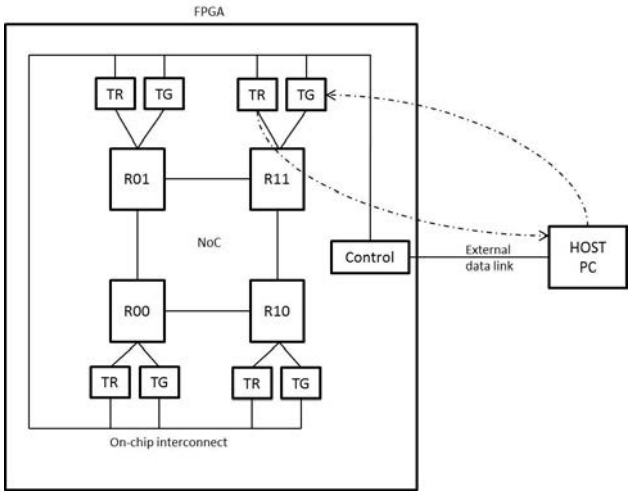


Figure 3.1: Hardware implementation of TGs and TRs.

interconnect, which connects all the emulation components. In this example the host PC configures the TG connected to the router 11 and it retrieves the traffic results from the TR connected to the same router, as depicted by the dashed arrows.

The traffic components implement a spatial-temporal traffic distribution in order to evaluate the performance of a NoC. As stated in chapter 2, there are several traffic generation models used in NoCs performance evaluation. The TRs and TGs are specially designed to handle a set of traffic generation models. In order to execute an emulation scenario, all the emulation components must be configured and, afterward, they must be accessed to extract the emulation results.

Consider the example depicted in Figure 3.2. The left side of this illustration shows a high level description of a traffic scenario. This description is handled by a host in order to generate a set of control packets which are sent to a control unit of an emulation platform. Afterwards, that control unit starts a sequence of internal bus requests in order to configure a target emulation node. This is an example of the configuration of a very simple traffic scenario. After the configuration of all emulation nodes, the traffic generation must be executed. This requires another set of communication flows between a host and an emulation platform. A host must be capable of starting, stopping and resuming the execution of a traffic generation scenario. After the end of the execution, a platform warns a host that it can retrieve the emulation results.

A communication protocol between a host and an FPGA-based NoC emulation platform eases the execution of NoC performance evaluations. This protocol must define a set of standard messages to carry out the control of the emulation components of a platform. The Table 3.1 summarizes the categories of messages of such a protocol. The messages are organized in three categories: configuration, control and feedback. The configuration messages set the parameters of the TGs in charge to produce a traffic model. The control messages manage the execution of a platform. They allow to start, to resume or to stop the execution of a traffic scenario. The feedback messages get the emulation results from the TRs.

In order to assure this protocol inter-operability, it is needed a protocol independent of the traffic model implemented by the emulation nodes. Basically, the messages

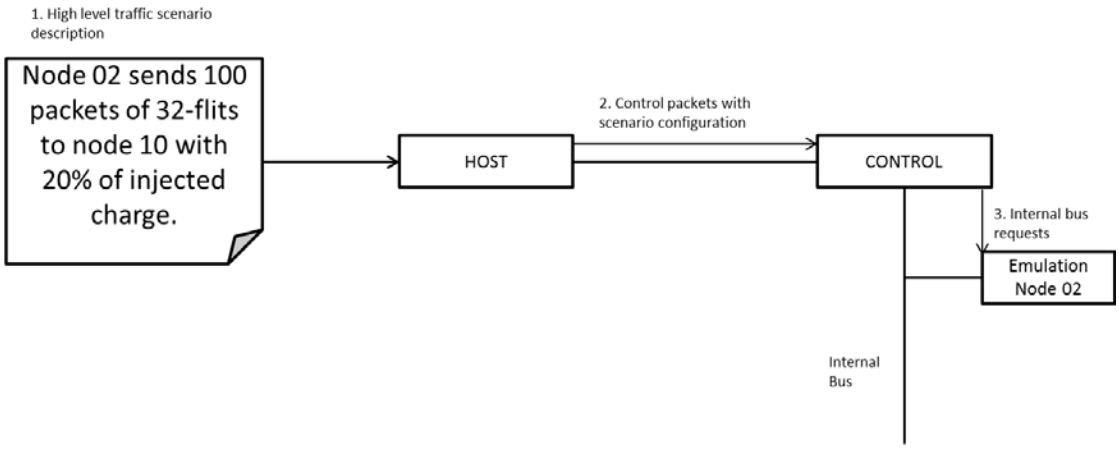


Figure 3.2: A simple example of emulation platform configuration.

Table 3.1: A summary of messages categories for a protocol to manage a FPGA-based emulation platform

Category	Description	Example
Configuration	These messages allow the setup of a traffic scenario parameter on a TG implanted on an emulation node.	Setting the size of the packet exchanged between two nodes.
Control	These messages control the execution of a NoC performance evaluation.	Stopping, resuming, starting, and finishing the execution of an evaluation scenario.
Feedback	These messages allow the retrieving of a traffic scenario results on a TR implanted on an emulation node.	Getting back the average latency between a pair of nodes.

defined by this protocol must be valid for several types of TRs and TGs. It is important to decouple the protocol messages from the data parameters required to specify a traffic model. Thus this protocol is adaptable to different emulation components without modifications on its implementation.

There is no standard protocol to manage the emulation components from the host PC perspective. This protocol could assure the inter-operability of emulation components. This enables the implementation of platforms by the integration of components designed by different manufacturers, as soon as they implement this standard protocol. Moreover, this standard management protocol eases the sharing of those components what could lead to the creation of libraries of IPs specialized in the evaluation of NoCs. This approach eases the scalability and reconfiguration of those platforms which is important for the design of the future many-core embedded systems.

3.2 Analysis of standard computer protocols to manage a NoC emulation platform

In order to adopt a standard protocol to address the management of an FPGA-based emulation platform, it is necessary to investigate the protocols already used in computer systems. Management protocols such as SMBus [smb] or NC-SI [nc-] could be solutions for integrating TGs and TRs. However, these protocols are designed specifically for computer motherboards management. They do not adapt to monitor the traffic components of an emulation platform.

The JTAG is an IEEE standard used to boundary scan printed circuit boards. This protocol is also widely used for debugging complex integrated circuits like high-end microprocessors or FPGAs. The JTAG is based on a daisy-chain bus which reduces its performance. Moreover, this protocol was not designed for management tasks, it is targeted at debugging.

The previous protocols lack flexibility and/or performance to manage a wide range of devices. Herein we analyze the features of network management protocols. Management protocols like WMI[wmi], RDP [rdp], NetFlow [net] and SNMP are largely used for the management of devices connected in computer networks. These protocols are critical for modern computer systems. Monitoring and ensuring those systems reliability in performance is absolutely required. The WMI (Windows Management Instrumentation) is a proprietary protocol designed to run in a Windows operating system. This protocol is used to gather management information about hardware, software, and operating system components. The WMI can be used in all Windows-based applications. The RDP (Remote Desktop Protocol) is also a proprietary protocol targeted to monitor remotely a computer. The NetFlow is a management protocol for Cisco devices. This protocol is specialized on monitoring the traffic flowing in a network. Different from the previous protocols, SNMP is an open protocol which was conceived as a general purpose management protocol for networked devices.

The SNMP is an application level protocol of the TCP/IP stack. It is a standard Internet protocol and its first version is defined in the RFC 1155, RFC 1212 and RFC 1157. The first two of these RFCs describe the SNMP data definition language, which specifies how the features of the managed device are described. The third one describes the protocol operations. The other two versions of this protocol are also defined by RFCs. In the context of this thesis, we are concerned only with the aspects of the first version of this protocol. This version is the simplest one, but it implements the core of network management. The latest versions focus on data security issues which are not relevant in the context of NoC emulation.

The SNMP architectural model is a collection of network management stations (managers) and networks elements (managed devices). The network elements are devices connected on the network like routers, gateways, among others. Each managed device contains an agent, which is responsible for handling the manager requests. An agent has in its possession a list of objects. One object states the operational status of a device. An example of agent's object is the state of a router interface. The agent's objects are organized in a Management Information Base (MIB), which is a repository of managed objects. A MIB has a standard structure in which the managed objects are addressed by an Object Identifier (OID). An object in a MIB is remotely accessed by the operations provided by the SNMP.

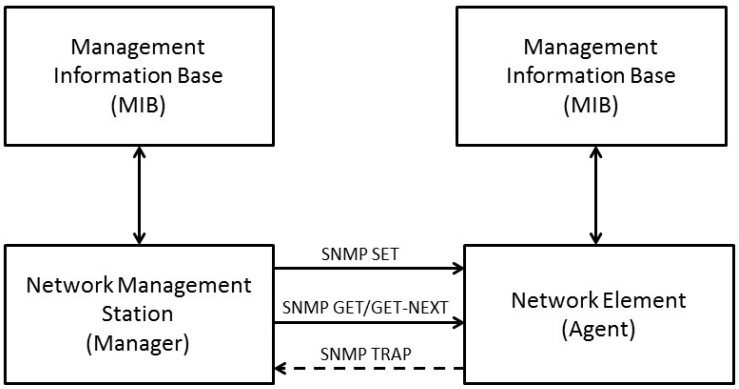


Figure 3.3: SNMP architecture and operations.

The core of SNMP is a set of simple operations that gives managers the ability to change the state of a SNMP-based device. Any device running software that allows the retrieval of SNMP information can be managed. This includes not only hardware devices but also software components, such as databases and web servers. Figure 3.3 depicts the SNMP architecture. A manager and an agent are presented with its respective MIBs. The manager MIB is a copy of the agents MIB. The two copies are updated by the SNMP operations. The manager monitors the agent state as well as it may modify the agent behavior by changing its MIB.

The communication model applied between a manager and a device is the request-response model. A manager accesses the objects stored in an agent MIB through the SNMP GET and SNMP GET-NEXT operations. An agent replies those SNMP operations by sending a SNMP GET RESPONSE filled with the value of the required object. The SNMP SET enables to change the value of a MIB object. The SNMP was designed as a monitoring protocol. In order to give reactiveness to the protocol, the SNMP designers added the SNMP TRAP operation. This operation enables an agent to notify a manager about a specific device event. The SNMP TRAP is sent without previous manager request. Although the SNMP was conceived for computer networks, it is adaptable to the VLSI environment. In [LA04] the SNMP is used to manage a testing platform for SoCs, which uses the P1500 test standard.

3.3 Adapting SNMP to NoC emulation management

The SNMP protocol may be adapted to manage an FPGA-based NoC emulation platform. The adoption of already tested management concepts enables taming the complexity of those platforms. Moreover, the SNMP creates a common interface to manage emulation components from different manufacturers from a host PC. The proposed communication model enables managing an emulation platform components using a set of standard operations. These operations change the platform behavior, in such a way that it is possible to configure and execute several evaluation traffic scenarios without FPGA re-synthesis.

The SNMP is a protocol of the TCP/IP protocols stack, thus it depends on other

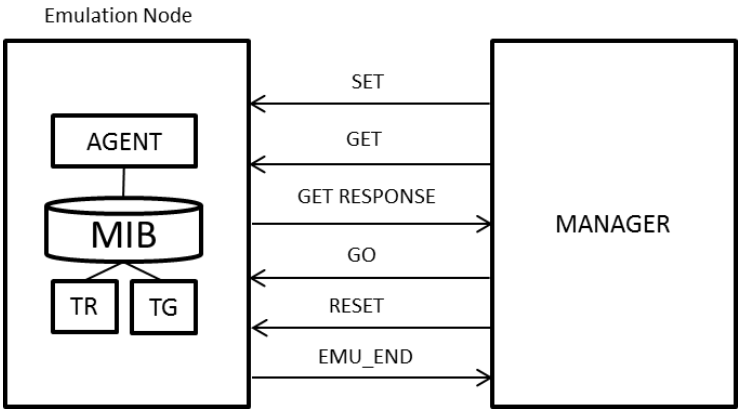


Figure 3.4: SNMP model adapted to NoC emulation.

network protocols to be executed. In order to adapt the SNMP to the NoC emulation context, it is necessary to simplify its implementation. It is convenient to design a management protocol inspired by the SNMP, but this protocol must not depend on the TCP/IP stack. Only the SNMP operations and its communication model must be adapted to the NoC emulation context. That is the reason we called our implementation SNMP-like.

Herein we present an adaptation of the SNMP protocol to manage the emulation of a NoC. At first it is necessary to redefine the role of the main SNMP components: manager, agents, MIB, and managed devices. In this context, the manager is a piece of software running on a host PC. The manager's goal is to configure the emulation components and to retrieve the emulation results. The agent is a hardware component designed to handle the SNMP packets in order to update or retrieve the value of the objects stored in a MIB. Differently from the standard SNMP implementation, this MIB is implemented as a simple bank of registers, which represent the objects of a managed device. The TGs and TRs are the managed devices, which are controlled through a set of managed objects stored in a MIB. These managed objects represent the parameters of a traffic scenario or the network performance metrics.

The agent, MIB, TR and TGs are assembled to create an emulation node. This component is connected to one NoC's router and each router has an associated emulation node. An emulation node is responsible for executing an evaluation scenario during the emulation of a NoC. This component generates data traffic according to a traffic model. It also calculates the performance metrics of a network. Figure 3.4 depicts the organization of the SNMP components adapted to NoC emulation. The communication between the manager and the emulation node is carried out by SNMP operations. The GET, GET RESPONSE and SET operations are standard SNMP operations, which enable the control of a MIB. The GET enables the manager to request the value of a specific object in an emulation node MIB. The GET RESPONSE is the response generated by an emulation node to a GET packet. The SET enables the manager to alter the value of a specific object allocated in a MIB. These operations are the core SNMP operations, which allow the management of a MIB. In order to adapt the SNMP protocol to NoC emulation, three new operations are added. The GO starts the emulation process. The RESET puts the emulation platform in the initial state. The EMU_END indicates that

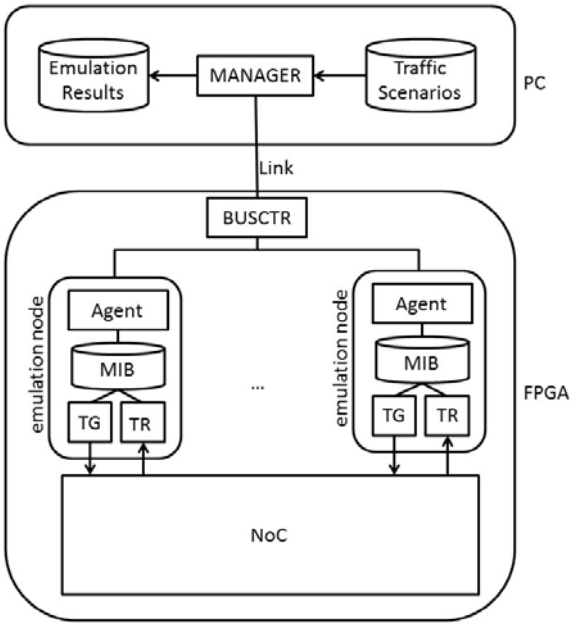


Figure 3.5: Architecture overview.

the emulation is finished and the emulation results are computed. This operation is a specific type of SNMP TRAP operation because it allow the emulation nodes to warn the manager about an event.

The SNMP has been chosen for managing a NoC emulation platform because of its simplicity and its flexibility to deal with devices from different manufacturers. Indeed, to manage a new device using SNMP it is needed only to known the device MIB structure. Moreover, the SNMP RFCs has standardized how a MIB description should be written, which eases the management of SNMP enabled devices as well as the adoption of components from different designers.

3.4 A SNMP-like architecture of an FPGA-based NoC emulation platform

We propose a SNMP-like architecture to manage an FPGA-based NoC emulation platform. A communication protocol between a host PC and a target FPGA is designed. This protocol is capable of configuring the emulation nodes and, then, extracting the emulation results. The emulation nodes are organized as SNMP-enabled devices. They are connected to a communication bus independent of the target NoC.

3.4.1 Architecture overview

The top level architecture of the proposed emulation platform is depicted in Figure 3.5. The emulation platform is composed by synthesizable hardware components and by software components. The hardware components are meant to be implemented in a target FPGA, while the software components are meant to be executed in a host PC.

At first, we present the hardware components. An emulation node is a hardware implementation of a managed device and it is directly connected to a NoC router. It is composed of four components: agent, MIB, TG and TR. The agent is responsible for decoding and executing the management commands. The MIB is an addressable register bank, which contains all the control and status registers that control the behavior of the traffic scenario generation, as well as they store the emulation results. The address of each MIB register is equivalent to the OID onto the standard SNMP implementation. The MIB is accessed by the traffic components through a simple interface, which enables reading and writing on the MIB registers.

The two last hardware components are the TG and TR that are responsible for generating and retrieving data on the traffic scenarios. Those components were adapted from the emulation platform presented in [TFR11]. The TG is connected to the router local input port. The TR is connected to the router local output port. In this implementation, the traffic is generated according to micro-benchmark models like transpose traffic, bit reverse, among others. It is also possible to create traffic scenarios scenarios described as tasks graphs with an arbitrary number of source and destination nodes. The TR processes the traffic and it produces synthetic information about the performance of each communication link. Subsection 3.4.3 presents the traffic models offered by this platform.

The emulation nodes are connected on a dedicated communication bus, which is controlled by the BUSCTR component. This component is responsible for interfacing the platform manager and the managed devices. The BUSCTR decodes the management packets sent by the manager and it retransmits those to the target emulation node through a dedicated communication bus. This internal bus may be implemented using any standard on-chip buses protocols. A multi-hierarchical bus could also be applied according to the number of emulation nodes.

The software components are the manager, traffic scenarios and emulation results. The manager is responsible for controlling the platform and its components in order to execute a NoC emulation. The manager is implemented as a C library that communicates with the emulation platform over a serial link. This library enables easy integration of the emulation platform in third party systems. The communication protocol applied is inspired by the SNMP, but it is not targeted for a TCP/IP implementation. The traffic scenarios component is a set of test cases that may be generated by specific tools as the TGFF [DRW98a], which generates random task graphs. The last component is the emulation results component, which is a synthesis of the performance results gathered during the emulation process. Those results are used to the design space exploration, benchmarking or validation of a NoC implementation.

3.4.2 Protocol implementation

The SNMP has a well-known and widely used management model for computer networks. This model is based on a set of simple operations exchanged in a request-response model. In this work, the SNMP is adapted to a hardware implementation, which is targeted at managing a NoC emulation platform. In this implementation, the emulation nodes are distributed components which are managed through their MIBs. The SNMP provides the operations to handle the MIB from the manager side. Herein we describe the communication protocol between the manager and the emulation nodes.

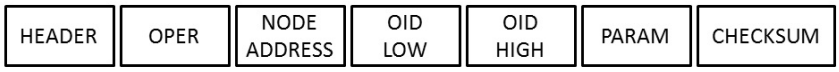


Figure 3.6: Packet format.

A communication protocol between a manager and the emulation nodes is proposed. This protocol implements five operations: SET, GET, GET RESPONSE, GO, RESET, EMU_END. Figure 3.6 depicts the protocol packet format. All packet fields are 8-bit long. The first field represents the start of a packet. The OPER field indicates the requested management operation. Afterward, the emulation node address and the object identifier are represented in the next three fields. The OID indicates the address of a MIB register. The PARAM field is the parameter of the SET operations or the response of the GET RESPONSE operations. At last, the CHECKSUM is an error control field. An incorrect packet is discarded and the destination requires its retransmission.

An example of this protocol execution is presented in Figure 3.7 (a). The data exchanges are represented by the arrows, the number over the arrows indicates the sequence in which the data is exchanged. The manager sends a GET packet to the BUSCTR in order to retrieve the value of the object 0X0001 of node 02. The BUSCTR receives the packet and it sends a request using the internal bus to retrieve the demanded information. The emulation node answers this request also using the internal bus. Afterwards, the BUSCTR sends a GET RESPONSE packet to the manager. This is the most complex packets exchange in this protocol because the other packets do not request a response packet.

Another example is depicted in Figure 3.7 (b). It is a SET packet exchange. The manager sends a SET packet to the BUSCTR in order to modify the value stored in the object 0x000A of node 02. The BUSCTR receives the packet and it sends an internal bus request. The agent of node 02 updates the object. In this case, there is no return packet to the manager. This flow is also valid for GO and RESET packets.

The last packet exchange example is depicted in Figure 3.7 (c). This is the only packet exchange initiated by the agents. The emulation nodes agents send a signal indicating the end of the traffic scenario execution. When all the nodes have finished, the BUSCTR sends an EMU_END packet to the manager.

3.4.3 Traffic models

In order to evaluate a NoC performance, an emulation platform must be capable of generating rich traffic scenarios. The SNMP model is easily adaptable to many traffic models, but in this work two traffic models are used: micro-benchmark and task graph. The proposed traffic models are implemented as two distinct hardware modules, each one with its own MIB. The platform user must select which traffic model will be used before the platform synthesis. The traffic model choice is based on the goals of the NoC evaluation. Furthermore, a resource analysis of both implementations is presented in Subsection 3.5.3. The first traffic model enables the creation of micro-benchmark synthetic traffic as transpose, bit-complement, bit-reversal, among others. These synthetic traffic scenarios are largely used to evaluate the NoC, however they do not resemble to real applications.

The second traffic model enables creating richer traffic scenarios based on task graphs,

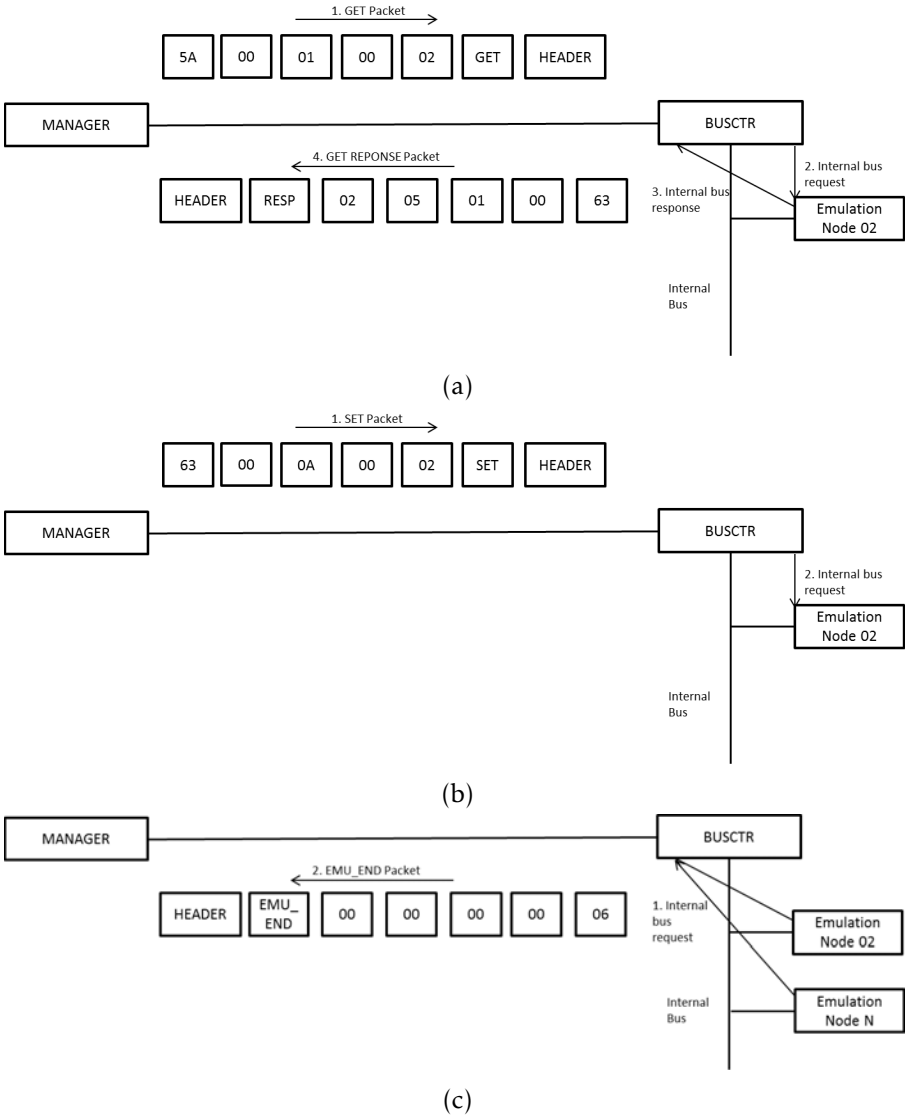


Figure 3.7: Examples of the protocol execution

which are used to model application behavior. In this traffic model, each emulation node possesses a list D of destination nodes. That list is composed by all the other emulation nodes of the system. The communication requirements between one emulation node and its destinations is represented by a list MD. An item in that list is represented by $MD_i = \{npt, pst, ic\}$, where npt is the number of packets to be transmitted, pst is the packet size and ic is the injection rate of these packets. Each emulation node has a source nodes list S and its associated communication requirements list MS. An item in that list is represented by $MS_j = \{npr, psr\}$, where npr is the number of packets to be received and psr is the size of these packets.

The behavior of an emulation node is divided in two parts: reception cycle and transmission cycle. Those cycles are executed concurrently and they are executed several times during an emulation. The reception cycle is executed by the TR and the transmission cycle is executed by the TG. An emulation node has also three more

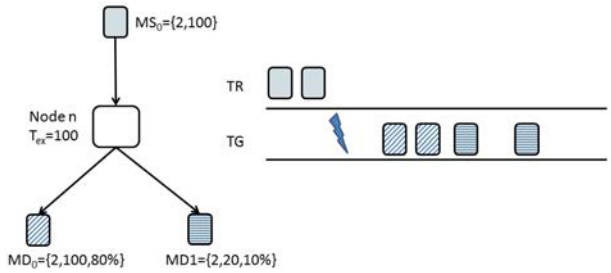


Figure 3.8: Traffic model example.

parameters related to the configuration of the execution cycles, which are: number of iterations, transmission mode, and time between iterations. The number of iterations parameter indicates the number of execution cycles to be run. The transmission mode determines if the TG will generate packets periodically or only when it receives signaling from the TR. The last parameter indicates the time between two transmission cycles.

The reception cycle consists in the actions performed by the emulation node to receive the packets specified by MS. There are a pre-defined quantity of packets to be received during a reception cycle. A new reception cycle starts whenever the emulation node receives all the expected packets of the current reception cycle. At this moment, a signal is sent to the TG module. This signal indicates the end of a reception cycle.

The transmission cycle has two executing modes. In the first mode, the TG generates packets periodically according to the MD specifications. In the second mode, the TG waits for a signal from the TR indicating that all the waited packets were received. After the reception of that signal, the TR waits a predefined number of clock cycles in order to emulate the time to process the received packets. Then, it starts the transmission of the packets as required by the MD. This second executing mode allows this platform to emulate the behavior of tasks on a real application, in which a task generates packets in response to packets previously received.

To illustrate this feature, Figure 3.8 depicts a simple task graph, which contains only four tasks. This illustration shows the communication requirement of the source task and of the two destination tasks. The source task will send two packets of 100 flits each. When the task n's TR receives the packets, it will send a signal to its associated TG. Then, the node n will wait for 100 clock cycles to emulate the task execution time and, afterward, it will send two 100-flits packets to MD_0 with 80% of injected charge and two 20-flit packets to MD_1 with 10% of injected charge. It is important to notice that the TG has a signal counter, thus it keeps track of all reception signals sent by the TR. In this way, if the sources have more throughput than the node n, all the reception signals will be received and the node n will send for each reception signal the packets specified.

3.4.3.1 Generating traffic scenarios using TGFF

The traffic scenarios may be generated by an external tool as the TGFF [DRW98a]. Herein we describe how this tool can be used to generate the parameters of a traffic scenario. The TGFF generates random acyclic task graphs, which are described in an application parameter file. An application parameter file provides all the parameters required to

describe a traffic scenario. For each application task graph, there are three sections in an application parameter, as depicted in Figure 3.9. The 'TASK' section lists all the tasks with their task types. The 'ARC' section describes the connections between task nodes with their arc types. The 'PARA' section gives tables of the required four parameters, which are indexed using the task or arc type:

1. The size of each packet of each transmission between two tasks corresponds to the arc type ('packetSize'). For example, in Fig. 5 from TASK0 to TASK1, the arc type is 37. Using the tables of 'PARA' section, we find that for the type 37 the size of the packet is 21 flits;

2. The number of packets to be sent between two tasks, corresponds to the arc type ('packetNum'). For example, from TASK0 to TASK1, the arc type is 37. Using the tables of 'PARA' section, we find that for the type 37 the number of packets is 36;

3. The data injection charge between two flits of every transmission between two tasks, corresponds to the arc type ('injCharge'). For example, from TASK0 to TASK1, the arc type is 37. Using the tables of 'PARA' section, we find that for the type 37 the injection charge is 76 clock cycles;

4. The period between two iterations, corresponding to the task type ('period'). For example, from TASK0 to TASK1, the arc type is 18. Using the tables of 'PARA' section, we find that for the type 18 the period is 1171 clock cycles;

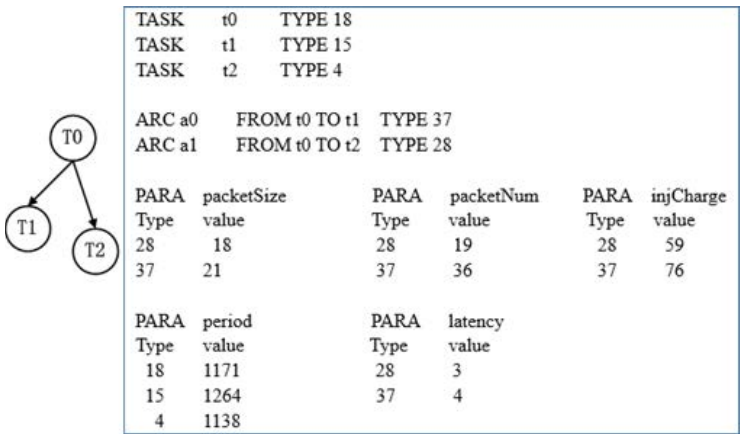


Figure 3.9: Example of application task graph.

The application task graph file is used as a input of the emulation platform execution flow. The manager extracts the traffic scenario parameters from this file. Afterwards, the manager generates the sequence of SNMP packets to run it. This enables the generation of rich evaluation scenarios based on task graphs.

3.4.4 MIB organization

The MIB is the core component in this implementation. It holds the control and status registers of the traffic generation components. Herein we describe the MIB organization

for a task graph traffic model on a 2x2 NoC. Figure 3.10 depicts the MIB, which is organized as a 16-bit linear address memory region. Each register has 32-bit which are byte addressed. There are two distinct addressing regions. The first region holds the registers related to the TG. This region starts at address 0x0000. The second region holds the registers related to the TR and it starts at address 0x1000. The division of the MIB in two regions is due to the concurrent access by the TG and TR. This eases the physical implementation of the MIB as two separate blocks, which may be accessed simultaneously. It is important to notice that, even the MIB has 16-bit addressing mode, only the used registers are physically implemented.

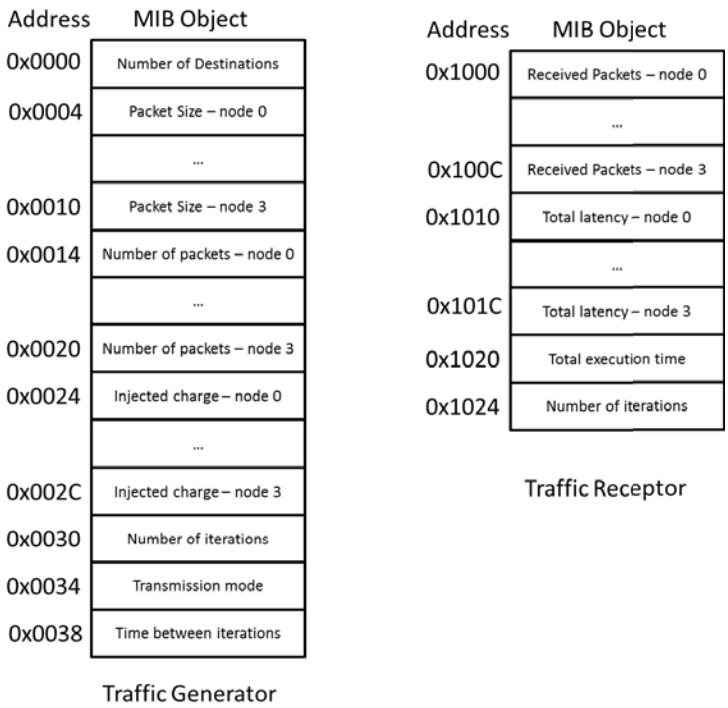


Figure 3.10: MIB organization.

The MIB is potentially a component that requests a lot of hardware resources, because it must hold all the status and control registers of a TR and a TG. In order to reduce the MIB resources requirements, it is necessary to limit the number of implemented registers. The goal is to keep the MIB size independent of the network size. For instance, if a network has N nodes, one node must implement $N-1$ copies of the traffic model registers on the MIB. Each copy describes the traffic requirements for each destination node. This is needed if the traffic model determines that a node must sent packets to all other nodes on the network. The MIB shown in Figure 3.10 has traffic parameters for each router in the network. However, it is possible to design a MIB which contains traffic parameters for a pre-defined number of routers. This reduces the resources utilisation to implement the MIB. Although the traffic model would be limited because each router is capable to send packets only to a pre-defined number of destinations.

It is important to highlight the MIB update process. The manager transforms a traffic scenario description into a series of SET operations, which are further processed in SET

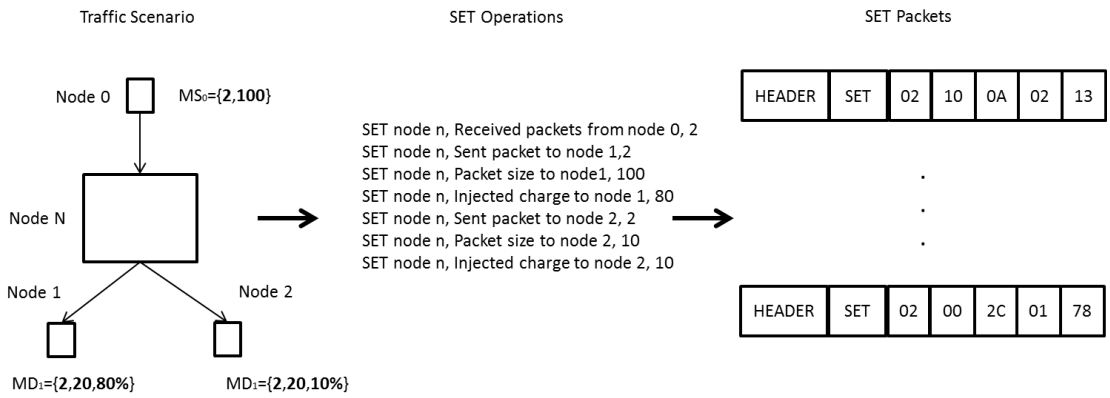


Figure 3.11: From a traffic scenarios to SET operations and SET packets.

packets. These packets are sent to the emulation platform in order to update a node behavior. Figure 3.11 depicts this process. A simple task graph is presented, in which the node n's traffic parameters are highlighted. The node n receives packets from node 0 and it sends packets to node 1 and node 2. The traffic scenario parameters are processed in order to create a list of SET operations. Afterwards, these operations are transformed in a sequence of SET packets, which configure the MIB registers in order to reproduce the traffic requested.

In order to accelerate the configuration of the emulation platform, the manager keeps a copy of each router MIB and it only updates the MIB registers that have distinct values from the values specified by the previous traffic scenario. For instance, after executing the first traffic scenario, the manager starts the configuration step of the second traffic scenario. Then, the manager sends SET packets to change only the MIB registers that are distinct in both traffic scenarios. Those MIB registers that possess the same value are not changed. Thus, the number of SET packets sent is reduced. This feature is called differential MIB update and it is experimentally evaluated in Section 3.5.2.

Figure 3.12 depicts the differential MIB update mechanism. At first, the traffic scenario 01 is configured for the node N. The manager verifies its copy of the MIB, which have default reset values. The traffic parameters which are different between the traffic scenario 01 and the MIB copy are highlighted in gray. As this is the first traffic scenario, all the registers have different values. Afterwards, the manager updates the MIB. Then, after the end of the execution of the first scenario. The execution of a second traffic scenario is requested. Once again the traffic parameters that are outdated in relation to the manager MIB copy are highlighted in gray. The manager updates the MIB, but only the outdated registers.

3.4.5 Execution flow

In this subsection, we explain the execution flow associated with the proposed NoC emulation platform. The execution flow is depicted in Figure 3.13. In order to evaluate the performance of a NoC, a designer must specify a set of traffic scenarios. In the context of this thesis, a traffic scenario describes how each emulation node must generate and receive packets. In this way, for each emulation node the traffic scenario specifies the amount of packets transmitted to each destination node, as well as the packets size,

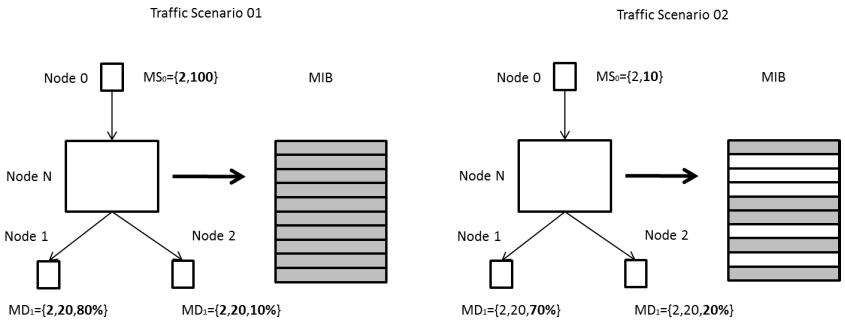


Figure 3.12: Differential MIB update.

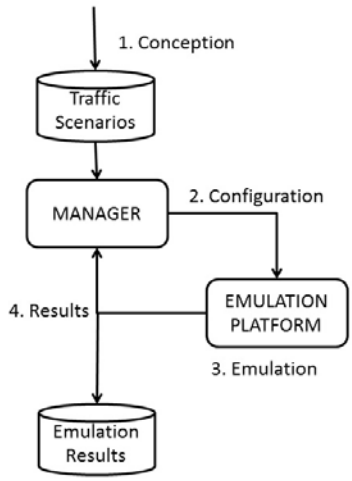


Figure 3.13: Execution flow.

the associated injected charge, among other parameters. The definition of the traffic scenarios is called the conception step. After this step, the execution flow enters in a loop to emulate sequentially all the traffic scenarios. It is important to notice that only one traffic scenario configuration is kept in the emulation nodes MIB.

The manager library is used to transform a traffic scenario in specific SNMP operations needed to configure the emulation platform. At first, each emulation node must be configured with the information gathered from the traffic scenario description. This stage is known as the configuration step. In this step, the manager sends SET packets to change the value of objects stored in the MIB of each emulation node. The updated MIB changes the behavior of the associated node according to the current traffic scenario.

After the configuration step, the manager sends a GO packet and the emulation nodes start sending and receiving data packets. It is the emulation step which lasts as long as necessary to finish all the communication flows described by the traffic scenario. After the emulation nodes receive all the data packets, the emulation platform sends an EMU_END to the manager and the results step begins. On this step, the manager uses GET packets to retrieve the performance results. After this, the results are stored in the emulation results component. The execution flow must continue until all the traffic scenarios are emulated. Then, the emulation results are analyzed in order to find out

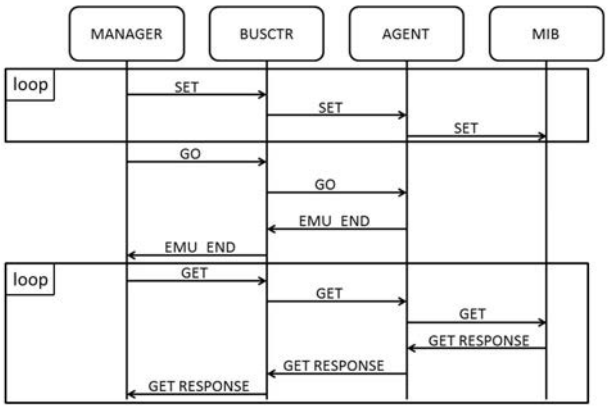


Figure 3.14: Packets exchange during the execution flow.

the NoC performance.

This execution flow entails several packets exchanges, which are depicted on Figure 3.14 as a sequence diagram. The first messages represent the configuration of the emulation platform using the SET packets. A loop represents the sequence of SET packets sent. The messages are sent by the manager and they are redirected to the target MIBs. Afterwards, the manager sends the GO packet, which start the emulation. This packet is sent to the agents. When the emulation scenario is finished, the agents send a EMU_END packet to the manager. Then, a set of GET packets are sent to retrieve the performance metrics, which are sent in GET RESPONSE packets. A loop box indicates that this packets exchanges are repeated until all the performance metrics are retrieved.

3.4.6 Implementation details

This subsection presents the implementation details of the proposed emulation platform. At first, the SNMP hardware components are detailed and, afterward, the SNMP manager library is described.

3.4.6.1 SNMP hardware components

In this subsection, we describe the hardware implementation of the SNMP components: BUSCTR, Agent and MIB. These components are implemented in VHDL using RTL behavioral description.

The BUSCTR treats the SNMP packets from the manager and it adapts them to the internal bus format. This component has an asynchronous full-duplex serial interface to communicate with the manager and it also has a parallel bus interface which is capable to transfer 2 bytes per bus cycle. In order to execute one SNMP operation, the BUSCTR selects the target emulation node and then it takes five bus cycles to transfer the operation and its parameters to the target emulation node. The SNMP operations are executed sequentially by this component. A state machine implements the BUSCTR actions.

The Agent is responsible for decoding and executing the SNMP operations. It has a slave interface to the internal parallel bus and a custom interface to a MIB component.

This interface enables the Agent to read and write in the MIB. A state machine is responsible for implementing the Agent behavior.

The MIB is implemented as a pair of registers banks. The first bank enables read operations and the second one enables write operations. These banks have distinct address spaces. The MIB is connected to a TG and a TR component through a simple interface, which enables the write and read of the MIB registers.

The evaluation of the timing performance for executing each SNMP operation is presented in Section 3.5.1.

3.4.6.2 SNMP manager library

In order to drive the emulation platform, a SNMP manager library written in C is available. This library is easily incorporated into existing and new NoC evaluation tools as a mean to drive the emulation. This subsection outlines the library functions as shown in Table 3.2.

To start the communication with the emulation platform, the *emuNoC_initCOM* function must be called. Afterward, the platform must be reseted before starting an emulation scenario. The *emuNoC_Reset* lets the platform ready for use and it must be called before each emulation scenario. The user creates a scenario based on micro-benchmark with the *emuNoC_scenario* function or a custom scenario is created using the *emuNoC_confNode* function, which must be called for each node participating in this traffic scenario. After the configuration of the emulation nodes, the *emuNoC_GO* function is called to start the emulation. This function returns after the end of the emulation. Then, the performance results are extracted using the *emuNoC_getLatency* function. Afterward, a new traffic scenario is executed or the communication with the platform is closed using the *emuNoC_closeCOM* function.

Table 3.2: SNMP manager library

Operation	Description
emuNoC_initCOM	Initializes the communication with the emulation platform
emuNoC_Reset	Resets the emulation platform
emuNoC_confNode	Configures the communication requirements for one emulation node
emuNoC_scenario	Configure one microbenchmak scenario for this platform
emuNoC_GO	Initializes the NoC emulation
emuNoC_getLatency	Retrieves the average latency for a pair of source and destination nodes
emuNoC_closeCOM	Closes the communication with the emulation platform

3.5 Experiments

In this section, we describe experiments conducted in order to evaluate our proposal. The proposed NoC emulation platform is implemented in a ML605 Virtex 6 evaluation

board for resources analysis and for performance evaluation of the SNMP operations and of the MIB update. The Xilinx ISE 14.3 and the XST are used to synthesize the emulation platform. The target NoC used in these experiments is a Hermes NoC [M+04], in which the XY routing and handshake flow control are applied, as well as a 16-flit buffer is used by each router input port.

3.5.1 SNMP operations evaluation

In order to evaluate this protocol, a workload based on three applications described as task graphs and three pseudo-random acyclic task graphs is executed on a 7x7 NoC. The applications used are MPEG-4, VOPD and multispectral imaging. The synthetic task graphs are created using the TGFF tool [DRW98a]. Each task graph is executed ten times in order to evaluate the emulation platform. The traffic scenario execution time is measured as well as the number of executed GETs and SETs, which enable calculating the time spent on the execution of the SNMP operations. From experiments, it is known that each GET takes 29 clock cycles, each SET takes 30 clock cycles and the FPGA execution frequency is 66 MHz.

The SNMP related measurements are relative only to the execution of those operations by the SNMP dedicated circuits. It does not include the SNMP packet transmission delay, which depends on the link bandwidth between the manager and the FPGA board. Indeed, it does include the SNMP packet handling time, the internal bus latency and the emulation node execution time.

The Table 3.3 depicts the amount of time needed to execute the SNMP operations as well as the average time needed to execute each traffic scenario within a 95% confidence interval displayed in parenthesis. Furthermore, the number of executed GETs, SETs operations and the number of tasks for each application are also presented.

The SNMP operations take only a very small fraction of the time spent to carry out a traffic scenario. For instance, for the MPEG-4 application the SNMP operations represent only 0.03% of the traffic scenario time and only 0.004% for the VOPD. The time spent to execute the traffic scenario is much larger for the task graphs based on real applications. In this case, these applications have more bandwidth requirements than the synthetic ones. However, the time spent on the SNMP operations does not depend on the traffic scenario execution time. Moreover, there is a correlation between the number of tasks and the time spent on SNMP operations. More tasks lead to more time spent on the platform configuration.

The number of executed SETs operations is 36% - 47% bigger than the number of executed GETs for the three task graphs based on real application. For the synthetic task graphs, this difference is larger, it is around 55% - 58%. We expect that the number of executed SETs is bigger than the number of executed GETs for a regular traffic scenario.

3.5.2 MIB update evaluation

In order to evaluate the performance of the differential MIB update which is described in Section 3.4.5, we analyze the amount of data exchanged between the manager and the target FPGA to carry out the evaluation of a NoC. In this experiment, we execute a set of ten transpose traffic scenarios according to the execution flow presented in 3.4.5.

Those scenarios are distinct only because of the injected charge which is consecutively

Table 3.3: SNMP operations execution time

Application	SNMP [ms]	Traffic [ms]	No. of GETs	No. of SETs	No. of tasks
MPEG-4	0.15	430.1 (0.35)	104	232	12
VOPD	0.08	1768.1 (0.19)	52	142	12
Multispectral	0.11	895.2 (0.26)	80	169	14
TGFF 0	0.04	10.7 (1.57)	36	65	9
TGFF 1	0.11	6.3 (0.29)	92	156	14
TGFF 2	0.06	7.0 (0.41)	52	189	7

incremented of 10% for each traffic scenario. The first traffic scenario has 10% of injected charge and the last one have 100% of injected charge. We measure the amount of data transferred to carry out these scenarios for different sizes of NoC. We conduct these experiments with the differential MIB update enabled and also when this features is disabled (linear MIB update).

Figure 3.15 presents the results. The differential MIB update transfers about 54% less data bytes than the linear MIB update. For a 115200 bps serial link and a 4x4 NoC, it takes about 405 ms to execute all the data transfers needed for the ten traffic scenarios when the differential MIB update is applied. In opposite, when the linear update is applied, it takes about 755 ms to accomplish this task. For each router added to the emulation platform, there is an average increment of 33% in the amount of bytes transferred for both update methods.

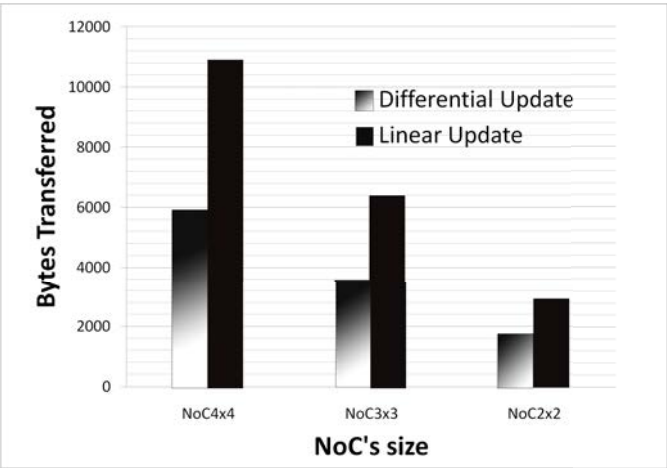


Figure 3.15: MIB update evaluation.

3.5.3 Resources analysis

As described in the subsection 3.4.3, the proposed emulation platform offers two distinct implementations of traffic models. The first one enables the creation of micro-benchmark synthetic traffic scenarios. The second one enables the creation of traffic scenarios based on tasks graphs. Both implementations share the same SNMP architecture, but they use different TGs and TRs components.

At first, we analyze the resources occupation of the micro-benchmark synthetic traffic implementation. Figure 3.16 depicts the FPGA occupation for eight different configurations of the NoC and the emulation platform. The resources occupation of the emulation platform comprises the implementation of the SNMP components, the traffic model components as well as the NoC itself. We observe that for a given NoC size the emulation platform has 2.8 times more LUTs and 4.7 - 5.3 times more Registers than the implementation of only the NoC components. Furthermore, these implementations do not use RAM resources.

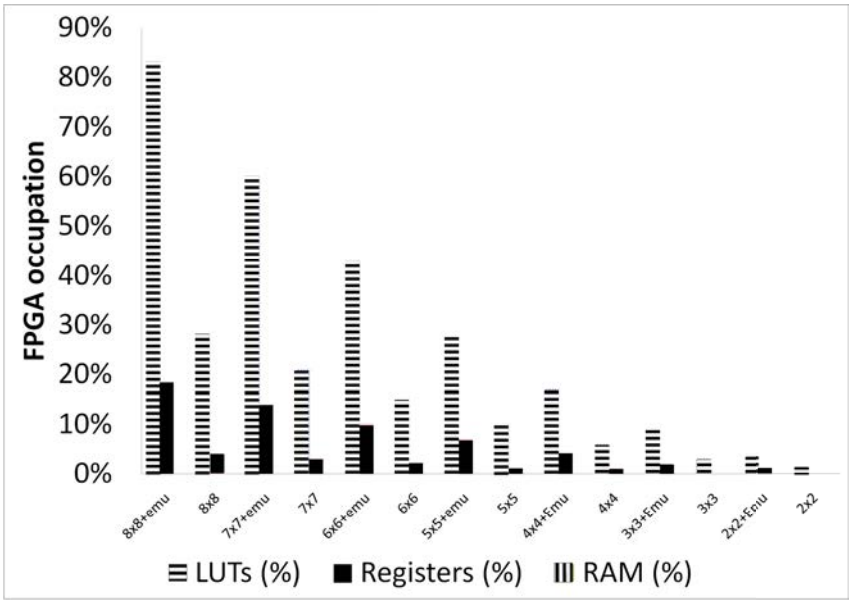
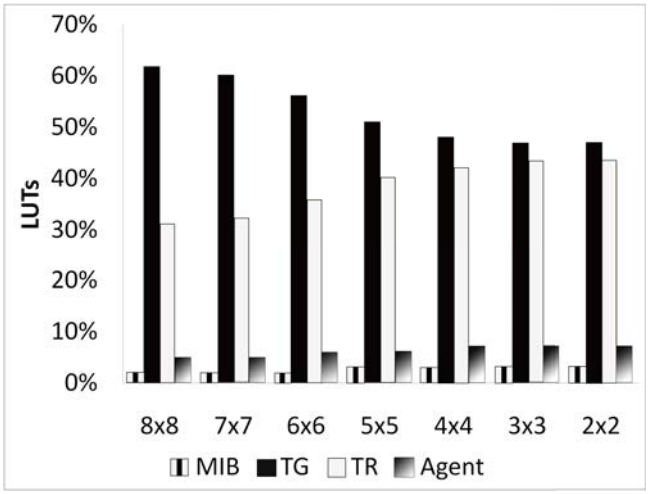


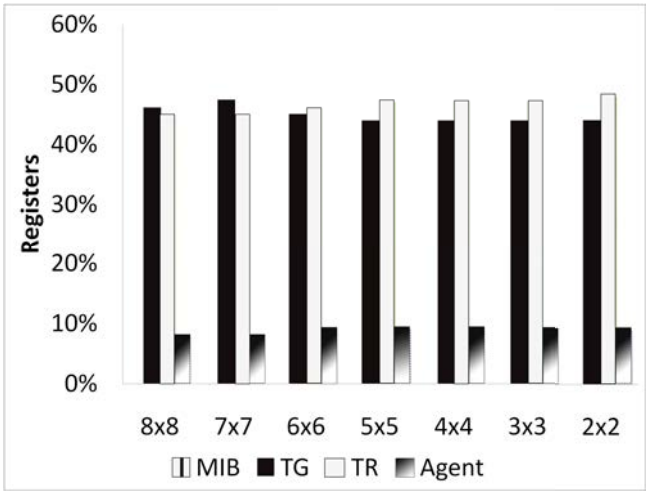
Figure 3.16: FPGA occupation for synthetic traffic implementation

The additional FPGA resources used to implement the emulation platform are due to the SNMP components and the traffic model components. In order to quantify the influence of these components to the overall platform occupation, we analyze the resources occupation of only one emulation node. Figure 3.17 depicts the LUTs and Registers occupation for the MIB, Agent, TR and TG components. We point out that the SNMP components (MIB and Agent) occupy only 7% of the LUTs and 8% of the Registers used to implement one emulation node. Indeed, most of the resources are applied to implement the traffic model components. The TG is the only component whose resources occupation significantly depends on the number of nodes implemented.

Henceforth, the resources occupation of the task graph traffic implementation is analysed. Figure 3.18 depicts the FPGA resources occupation for different configurations of NoC and the emulation platform. We observe that for a given NoC size the emulation platform has 4 times more LUTs and 6 times more Registers than the implementation of only the NoC components. Furthermore, these implementations use RAM resources to implement the MIB registers. For this traffic model, the MIB stores $(20 * number\_of\_destinations + 36)$ 32-bit registers, by the other side, for the synthetic traffic model the MIB stores only $(7 * number\_of\_destinations)$ 32-bit registers. The number of destinations are configured before the platform synthesis. In these experiments the number of destinations is equal to the number of nodes in the NoC, which is



(a)



(b)

Figure 3.17: LUTs (a) and registers (b) occupation share for one emulation node of the synthetic traffic implementation

the worst case possible in relation to resources occupation.

In order to quantify the influence of the SNMP components to the overall platform occupation, we analyze the resources occupation of one emulation node. Figure 3.19 depicts the LUTs and Registers occupation for the MIB, Agent, TR and TG components. The SNMP components represent only 15% - 30% of the LUTs and 6% of the Registers used to implement one emulation node. Indeed, most of the resources are applied to implement the traffic model components. However the MIB has a growing occupation of LUTs which depends on the number of implemented nodes. These resources are used to implement the memory locations for the traffic model parameters. Usually, these memory components are implemented as RAM blocks in the FPGA. When the MIB is synthesized alone the synthesis tool implements them as LUTs. When the whole emulation platform is implemented, the MIB is implemented mostly as RAM blocks.

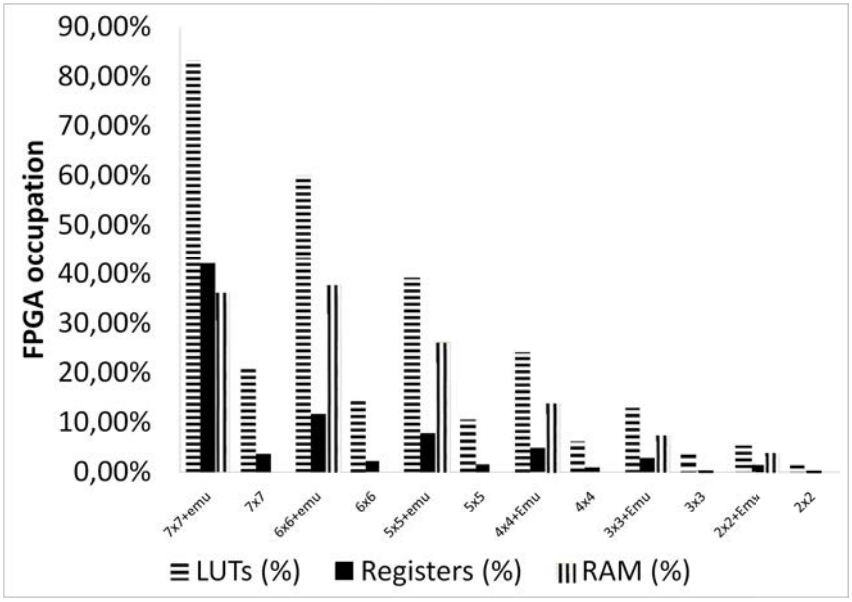


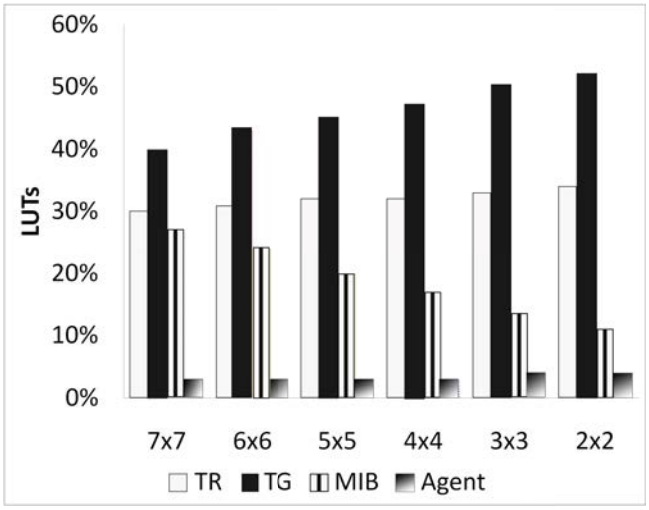
Figure 3.18: FPGA occupation for task graph traffic implementation

3.5.4 Case study: task mapping and mesh topology exploration

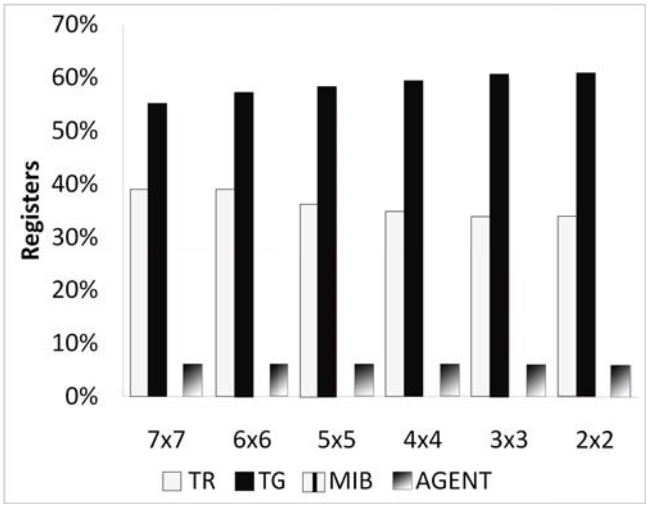
Task mapping strategies on NoC (Network-on-Chip) have a huge impact on the timing performance and power consumption. So does the topology. In this case study, we describe the use of the proposed FPGA-based NoC emulation platform on an exploration flow of task mapping algorithms using different NoC mesh shapes. The proposed SNMP-like management protocol eases the integration of an FPGA-based emulation platform on a NoC exploration flow. Figure 3.20 depicts a task mapping exploration flow based on the proposed FPGA-based NoC platform. It combines NoC configuration parameters and application communication parameters with the task mapping algorithm.

In the first phase of the evaluation flow, a set of evaluation scenarios is created. Using the emulation libraries described in Section 3.4.6.2, the exploration software configures the NoC emulation platform. The FPGA-based emulation platform can accelerate the emulation process and provides the real-time timing results for analytical purposes. The communication is carried out with the proposed SNMP-like protocol. The emulation platform runs the evaluation scenario in phase 2. Afterwards, the exploration software retrieves the evaluation results. These results are analyzed in phase 3 in order to find out the best solution of task mapping.

Several task mapping algorithms and mesh topology shapes were selected to evaluate this exploration flow. Three task mapping algorithms: Branch-and-Bound (BB) [HM05], bandwidth-constraint and latency constraint Branch-and-Bound (BBL) and Template-based Efficient Mapping algorithm (TEM) [WYJL09]. Four shapes of the mesh topology are used: 3x3, 4x4, 3x5 and 2x7 respectively. Two application modeled as a directed task graph are used. By combining the NoC configuration parameters, application communication parameters, and task mapping techniques, the flow generates different task mapping scenarios. By emulating these scenarios on the NoC emulation platform and analyzing the emulation results, the best solution can be found for a specific requirement. The complete exploration space is required for each algorithm. The



(a)



(b)

Figure 3.19: LUTs (a) and registers (b) occupation share for one emulation node of the task graph traffic implementation

experiments show that the exploration flow can help the designer to explore the best task mapping strategy to meet the design requirements of a particular application.

The task mapping scenarios for these benchmarks are generated using PC. The scenarios are then emulated on the ML605 FPGA platform (with the Virtex 6 FPGA). The advantages of such a platform are not only providing logic resources to emulate real communications but also to carry out the performance evaluation in a short time. The main clock frequency of the platform is 66MHz. Its maximum bandwidth can reach 62.94MByte/s. A 7x7 NoC is synthesized on the FPGA-based platform. The SNMP-like protocol enables the configuration of the evaluation scenarios in such a way that is possible to evaluate different shapes without FPGA re-synthesis.

Table 3.4 gives the appropriate task mapping algorithm with the suitable NoC shape for two application benchmarks. The standard used to choose the best solution will differ

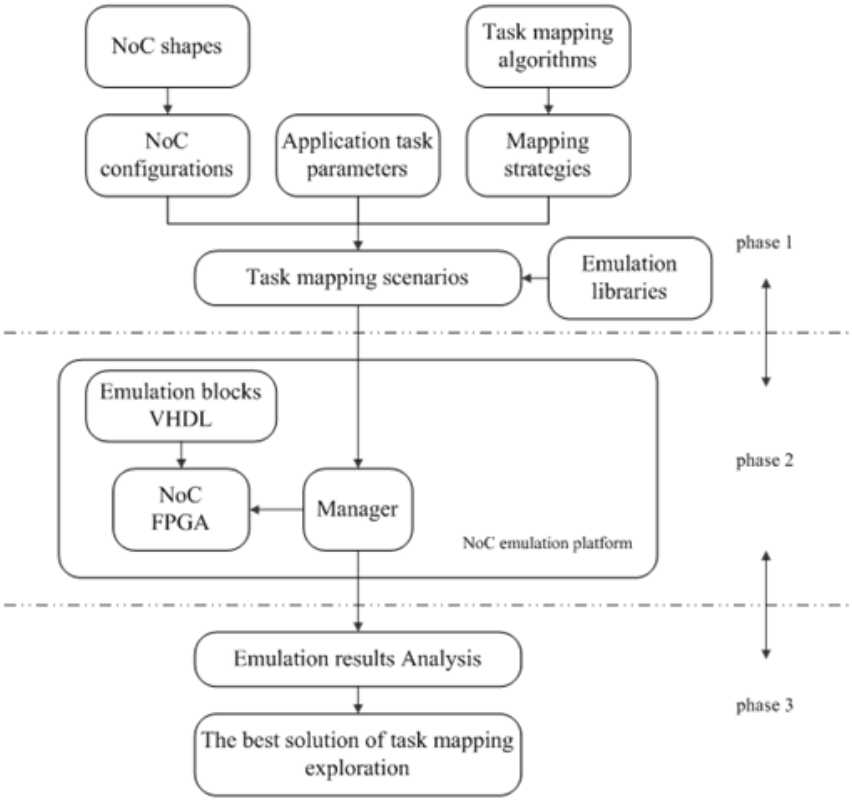


Figure 3.20: Task mapping exploration flow based on FPGA-based NoC platform

depending on the case concerned. It is shown six performance parameters calculated from the results gathered from the FPGA-based platform. The designer may choose the best configuration depending on the application requirements. The emulation platform enables fast exploration of task mapping algorithms.

Table 3.4: Task mapping and mesh topology exploration results according to different performance metrics

	Application 0		Application 1	
	Algorithm	Shape	Algorithm	Shape
Dynamic energy	BB	4x4	BB/BBL	3x5
Longest path latency	BB	4x4	BB	3x5
Total latency	BB	4x4	BB	2x7
Average latency	BB	4x4	BB	2x7
STDEV of latency	BB	3x3	BBL	2x7
Execution time	TEM	4x4	TEM	2x7

3.5.5 Results discussion

The SNMP timing performance evaluation is presented in Subsections 3.5.1 and 3.5.2. The execution of the SNMP operations are independent of the NoC size and these

operations represent a very small fraction (0.004% - 1.7%) of the time spent on the execution of the traffic scenarios. Furthermore, the utilization of the differential MIB update reduces in 54% the amount of data exchanged between the FPGA and the host PC. Thus, it permits fast update of the MIB content. Although, the serial link between the FPGA and the host PC represents a communication bottleneck. A higher bandwidth link could allow faster MIB update and consequently faster emulation execution.

In Subsection 3.5.3, we present the resources analysis for two different implementations of the proposed emulation platform. The results point out that the SNMP components represent a small fraction of the overall emulation platform. The MIB is the only SNMP component that depends on the complexity of the implemented traffic scenarios, because it must hold all the parameters related to the traffic model description. For the ML605 evaluation board, it is possible to implement the emulation platform for a 8x8 NoC (micro-benchmark scenarios) and a 7x7 NoC (task graph scenarios). The low resources occupation of the SNMP components is due to the lightweight circuits in charge to implement the SNMP functions.

The obtained results point out that the SNMP communication model is a fast and lightweight solution to drive a NoC emulation platform. Besides that, the circuitry needed to implement the SNMP components is quite simple.

3.6 Conclusions

We evaluate the SNMP protocol concepts to manage an FPGA-based NoC emulation platform. The implemented architecture enables the easy configuration of the emulation nodes, as well as it defines an inter-operability model for the emulation components based on the MIB description. As a demonstration of the inter-operability the proposed emulation platform integrates two distinct implementations of traffic models, which are defined by two distinct MIBs. The first one enables the creation of micro-benchmark synthetic traffic scenarios. The second one enables the creation of traffic scenarios based on tasks graphs. The experiments highlight that a light version of SNMP is very efficient for a light resources overhead. The proposed platform is also evaluated on a case of study, in which several task mapping algorithms are evaluated on different topologies.

Chapter 4: The synthesis of FPGA-Based NoC emulation platforms from application traces

Emulation is a noble and just passion, full of appreciation.
Friedrich Schiller

Contents

4.1 Overview **54**
4.2 Fundamental definitions **59**
4.2.1 Traces and packets 60
4.2.2 Dependency pattern 62
4.2.3 Dependency table 63
4.2.4 Base Set 65
4.2.5 Message generation table 66
4.3 Problem definiton **66**
4.4 Framework workflow **67**
4.4.1 Patterns extraction 68
4.4.2 Base set extraction 69
4.4.3 Random walk strategy 71
4.4.4 Base set evaluation 71
4.5 Dependency-aware traffic generator **72**
4.6 Experiments **73**
4.6.1 Data storage requirements 73
4.6.2 FPGA occupation 74
4.6.3 Evaluation of the MoC 74
4.6.4 Results analysis and discussions 80
4.7 Conclusions **83**

Trace-based traffic is used in several NoC emulation platforms [Pap11, LPG11, KSPK10] to generate realistic workloads. A trace is a record of a network activity usually extracted from a full-system simulation. These traces include information about the order and time of each packet exchanged by the application nodes. That information is used to reproduce an application behavior in other networks. However, in real systems, there are dependencies between the packets. Some packets are transmitted only after the reception of other packets. The absence of packets dependency information leads to the injection of traffic in a higher injection rate than a real system, because the packets are transmitted without waiting for the occurrence of certain events [MNFA14]. The dependency-aware traces are traces enriched with the packets dependency information. They are more accurate than ordinary traces for a broad range of network architectures, but they demand more data storage resources. The dependency-aware traces are usually used for NoC simulations, because their data storage requirements entail additional complexity to run FPGA-based NoC emulations. It is important to combine the advantages of dependency-aware traces with the speed of FPGA-based NoC emulation. In this chapter, we propose a new traces analysis framework dedicated to the synthesis of dependency-aware traffic generators for NoCs emulation. The packets dependencies are retrieved from application traces in order to build an executable MoC which accurately replaces the application nodes onto the NoC emulation. The rest of this chapiter is organized as follows. The next section presents an overview of the dependency-aware traces. Afterwards, the fundamental definitions of the traces analysis framework are presented in 4.2. The section 4.3 formulates the packets dependencies extraction problem. The section 4.4 describes the proposed framework workflow. The dependency-aware TG architecture is given in section 4.5. The section 4.6 describes the experiments carried out to evaluate this proposition. The conclusions are presented in section 4.7.

4.1 Overview

The FPGA-based emulation tools [GADM+05, LPG11, dLJFR14] accelerate NoC benchmarks as well as design space exploration. These platforms have high accuracy and low execution time compared to NoC simulators. The main propositions of NoC emulators [LPG11, KCdlTR08, dLJFR14, WHS07] focus on synthetic, trace-based or statistical traffic models. The execution of real applications traffic during an FPGA-based NoC emulation is quite challenging. The FPGA resources limitation is the main constraint to the reproduction of application traffic. In some cases communication traces from full-system simulations are available from standard benchmarks. The execution of trace-based traffic is usually simpler than the emulation of all the application cores and a NoC. However there are some pitfalls on trace-based traffic for FPGA-based NoC emulation.

A single trace can easily comprise millions of packets. For example, as stated in [MNFA14], each trace from a Fast Fourier Transform (FFT) benchmark with a problem size of 16 million complex data points results in a trace file size of about 28 GB. The traces size is an important issue for NoC emulation because of the limited internal RAM available on FPGAs. The traces are often allocated on the external RAM, which becomes a bottleneck for generating real-time traffic.

The trace-based execution distorts the injection rate and effects of congestion due to the lack of packets dependency information. To better understand the effects of the lack of that information, we present a simple example. Consider the Table 4.1. This table presents a trace from a full-system simulation. This simulation is executed in a NoC where the latency between a pair of nodes is 2 clock cycles. Figure 4.1a depicts the reproduction of that trace in a NoC with the same latency. The transmission and reception times of each packet are also depicted in that figure. The same trace is reproduced in another NoC. This NoC has 6 clock cycles latency. Figure 4.1b depicts that execution. There are differences between those two executions. The second execution is longer than the first one as well as it has higher average latency. Those differences are due to the fact that each network has distinct latencies.

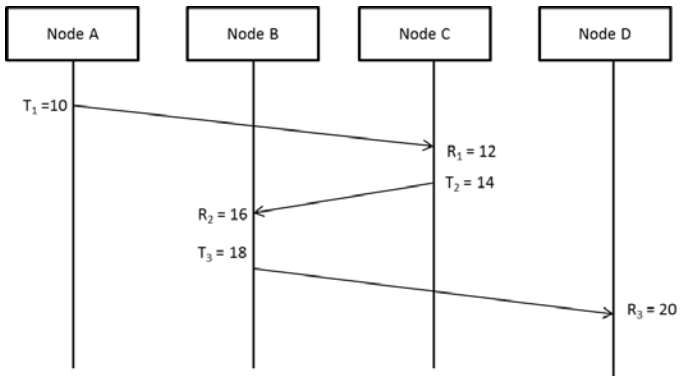
Now we analyse how those differences impact the packets dependency relationships. Assume that there is a dependency relationship between packet 1 and packet 2 and another between packet 2 and packet 3. Henceforth we describe those relationships. Node C receives packet 1, then it gathers its data and executes an operation which takes some execution time. Afterwards, the result of that operation is sent to node B. This node further processes the data and sends the results to node D after some execution time. It is important to notice that the trace does not contain the packets dependency information.

In the execution depicted in Figure 4.1a for the NoC with 2 clock cycles latency, packet 1 is received at time 12. After some execution time, node C sends the operation results to node B at time 14. Node B processes the data and sends the results at time 18. This execution reproduces correctly the behavior of the application because the network used have the same latency as the network used to collect the traces. The traces execution on the second network has some problems as depicted in Figure 4.1b. Using the information coded in the traces the packet 2 (operation results) is sent at time 14. However, the packet 1 (input data) is received at time 16. The operation results are sent before the reception of the input data. The same happens to packet 3 which is sent before the reception of packet 2. In this execution, the packets are injected in a higher rate than the expected one. This is an example of the fragility of trace-based traffic.

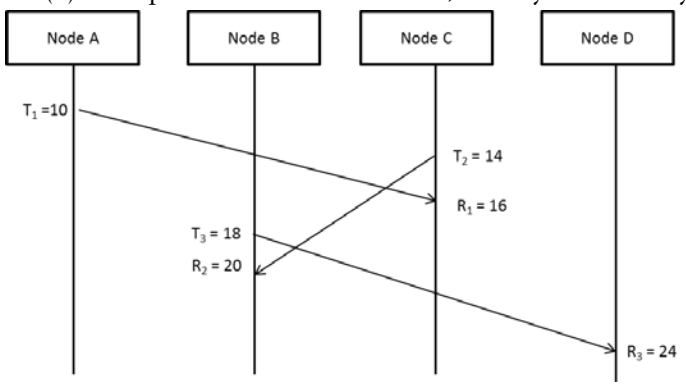
Packet	Time Sent	Time Received	Source	Destination
1	10	12	A	C
2	14	16	C	B
3	18	20	B	D

Table 4.1: Trace example

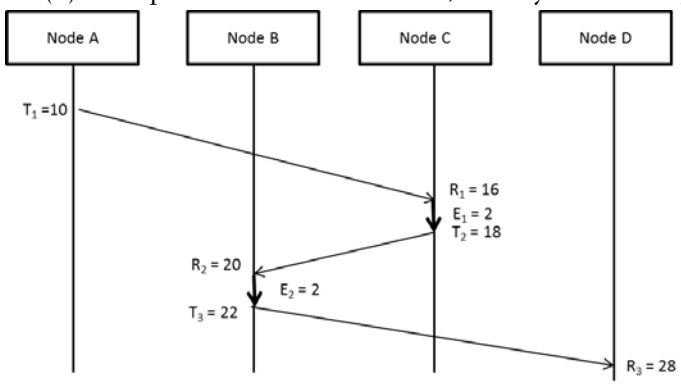
In order to deal with the problems of trace-based execution, some works [NMFA11, HCT+12, TAA+11, MNFA14] propose the utilization of dependency-aware traces. Theses traces are enriched with the packets dependencies information. These traces tend to be bigger than the original ones. However, they are very more accurate than ordinary traces for a broad range of network architectures. Enforcing dependencies ensures a proper interleaving of packets which increases the fidelity of the simulation by eliminating the artificial contention created by trace-based simulation. To better understand the advantages of theses traces, we revisit the last example using a dependency-aware trace. Figure 5c depicts the execution of that trace on the NoC with 6 clock cycles latency. The



(a) Example trace-based execution, latency = 2 clock cycles.



(b) Example trace-based execution, latency = 6 clock cycles.



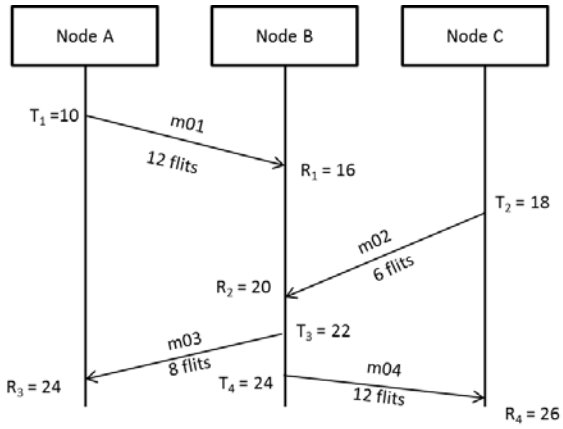
(c) Example of dependency-aware trace execution, latency = 6 clock cycles.

Figure 4.1: Examples of trace-based execution.

transmission and reception times of each packet as well as the execution time to process each packet are also depicted in that figure. In this time, the dependency information is used which leads to a more accurate reproduction of the application behavior. Each node reproduces the behavior specified by the application because they wait to receive the input data. Then, they emulate the execution time of each requested operation. At last, the results are sent on the output packets.

Those three trace execution examples indicate that a trace represents the packets injection rate from the network on which the trace was collected. The same trace used on a different network lead to a packets injection rate different from the one specified by the application. However, the utilisation of the packets dependencies information improves the accuracy of those traces. This conclusion arises a major problem: how to extract the packets dependencies. A natural solution for this problem is to instrument an application code and use a specific simulator to generate dependency-aware traces. However this approach entails significant efforts to modify an application code because some applications are very complex and depend on many libraries and legacy code. Thus, it is important to investigate approaches to extract the packets dependencies from a regular trace.

To illustrate the problem of packets dependencies extraction from a trace, we present a simple example. Figure 4.2 depicts the exchange of some packets by a group of nodes as well as the trace collected from that exchange. Using only the information available in that trace (source, destination, packet length, time sent and time received), we infer seven possible packets dependency relationships. The inference is based on the assumption that there is a casuality relationship between the output packets and some or all input packets. Those relationships are presented in Table 3. We cannot tell which dependency relationships are the true ones. It is clear that we need a procedure to analyze traces in order to retrieve the true packets dependencies.



ID	SOURCE	DEST	LENGTH	TIME SEND	TIME RECEIVED
m01	A	B	12	10	16
m02	C	B	6	18	20
m03	B	A	8	22	24
m04	B	C	12	24	26

Figure 4.2: Example of packets dependency extraction problem.

Herein we analyse the main propositions of dependency-aware traces. In [HGK10],

Dependency	Input Packets	Output packets
1	m01	m03
2	m01	m04
3	m02	m03
4	m02	m04
5	m01,m02	m03,m04
6	m02	m03,m04
7	m01	m03,m04

Table 4.2: Possible packets dependencies.

dependency-aware traces are generated from CMP (Chip Multiprocessor) simulations. These traces are extracted from a full-system simulator which was modified in order to capture the network traces with dependencies. Then, a collection of dependency-aware traces are created by the execution of some applications from PARSEC benchmarks. These applications source code were modified in order to ease the detection of the packets dependencies. A trace reader library is made available in order to ease the utilisation of those traces into new and existing simulators. A similar approach is used in [TAA+11]. In both cases, the network traffic consisted in cache memory references and cache coherence protocol messages. Furthermore, it is not always affordable to modify an application source code in order to extract the packets dependencies.

In [MNFA14] multiple simulations traces from different network configurations are used to create a dependency-aware trace. An application is simulated in a set of NoCs in which each network has different links latencies. From the traces extracted from those simulations, we analyze the packets dependencies patterns. These patterns indicate how output packets must be generated in response to the reception of a set of input packets. Only the patterns that are true for all the input traces are used to build one dependency-aware trace. However, a methodology to determine the number of input traces is not proposed. This proposition is also evaluated only for shared memory systems.

[HCT+12] proposes a dependency-driven table to encode the dependency patterns. The dependencies patterns are extracted from communication traces and they are asserted by observing the packets received within a determined time window. However, there is no clear method to find out this time window parameter. This is an error prone procedure because it can lead to the identification of false dependencies patterns. The dependency-driven table is periodically evaluated in order to generate traffic, but we do not propose a technique to find out the traffic generation period parameter. This parameter has a major effect over the accuracy of the generated traffic.

The previous dependency-aware traces propositions are targeted at software simulations. Some of them also depend on the network architecture used to generate the analyzed traces or they depend on modifications of an application source code. The main challenge of those propositions is the identification of the packets dependencies. This challenge is even bigger when the application source code is not available. In this case, the packets dependencies must be extracted from the analysis of traces from a single full-system simulation. That is the main subject of this chapter. The extraction of packets dependencies from a single trace extracted from an unknown NoC architecture.

In order to extract the dependency relationships from a single trace, we assume some premises. The first premise is that the true packets dependencies are present on the traces. We also assume that some dependencies are false, they are effect of the inject charge distribution on the NoC used to collect the traces. The dependencies are inferred from the information of the packets headers as well as the order and time of the packets on the traces. We evaluate several combinations of packets dependencies in order to find out the best possible combination for a given trace. This procedure is detailed in section 4.4.

A light model of dependency-aware traces can be used to execute applications traffic onto FPGA-based NoC emulations. In this chapter, we establish a trace analysis framework dedicated to synthesize dependency-aware Traffic Generators (TGs) for NoCs emulation. This framework is designed to process traces generated by message passing applications modeled as tasks graphs. The traces are analyzed to create an executable Model of Computation (MoC) which accurately replaces the application nodes in future NoC emulations. The main advantage of this approach is that the proposed MoC requires less hardware resources to hold the packets dependencies than a trace does. Moreover, experimental results highlight that the MoC is more accurate than regular traces for a broad range of NoC architectures.

Despite its advantages, this approach has some drawbacks. The assumption that the true dependencies are coded into the traces is not always correct. Furthermore, the dependencies are asserted from the information from the packets headers and the time and order of the packets. In order to overcome those drawbacks, the framework proposes a procedure to evaluate the accuracy of a MoC generated from traces. That accuracy metric allows a designer to evaluate the consistency of a MoC.

This work's contributions are twofold. The first one is a packet dependencies extraction algorithm for traces, which results in a simple and executable MoC. The second contribution is a synthesizable TG architecture based on the proposed model, which can be used for NoC emulation. In this work we address the packets dependencies extraction problem with a single trace. The proposed framework analysis a single trace from an acyclic message passing application in order to build a packet dependency-aware MoC, which can be synthesize as a TG for NoC emulation. This framework does not depend on the original network architecture neither on the application source code. It also reduces the data storage requirements to execute real applications traffic on FPGAs.

4.2 Fundamental definitions

Simply stated, for a given application trace extracted from a full-system simulation executed in an unknown NoC architecture, our objective is to find for each application node a MoC which could accurately reproduce that application node's communication behavior in future FPGA-based NoC emulations. In the next subsections, we present the main components related to the traces analysis and to the proposed MoC. Herein we present an overview of those components. Figure 4.3 depicts the relationships between those components. At first, we define the structure of the input traces and the representation of a packet exchanged between two nodes. The dependency relationships between the input and output packets of a node correspond to the Dependency Pattern (DPs). The properties and operations of a DP are presented. We also presented the

definition, properties and execution of a Dependency Table (DT), which models the dependent communication behavior of a node by integrating different DPs. The DTs are components of the proposed MoC.

From the DPs an initial DT is created, which must be further processed in order to extract the correct DPs. Then, an important effect related to the DPs is described. This effect is the patterns superposition which imposes difficulties to identify the DPs on the input traces. As a consequence of the patterns superposition, we present the definition of a Base Set which is a solution for the superposition problem. A BS contains a set of DTs generated from a trace. A simple evaluation of the BS elements allows to find out a DT which better emulates the behavior of an analyzed node. The extraction and evaluation of a BS are the main subjects of the trace analysis framework proposed in section 4.4. At last, the second component of the MoC is presented. A Message Generation Table (MGT) models the independent communication behavior of a node. These trace analysis components are proposed in order to generate the DTs and MGTs from the traces of a single full-system simulation.

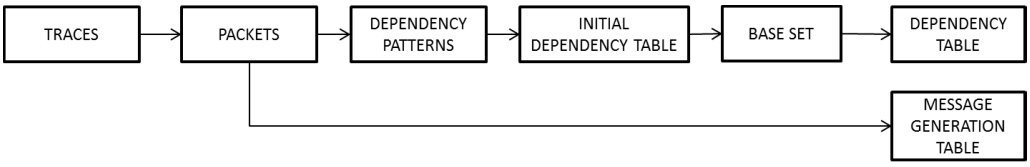


Figure 4.3: Overview of the trace analysis components and proposed MoC.

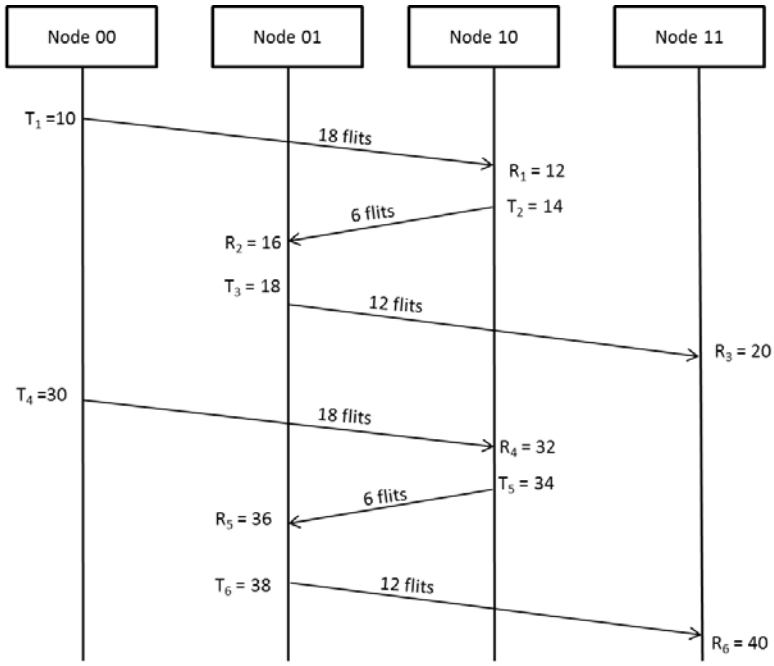
4.2.1 Traces and packets

A trace event $e = (s, d, t, r, l, id)$ is a 6-tuple which consists of source node address s, destination node address d, transmission timestamp t, reception timestamp r, packet length l and an unique identifier id. The traces are generated from the simulation of an application mapped onto a NoC. An application is composed of a set $N = \{1, ..., k\}$ of nodes. A node $n \in N$ has two sets of traces associated: transmission trace T^n_{TX} and reception trace T^n_{RX}.

From T^n_{TX} and T^n_{RX}, the communication behavior of node n is extracted. The proactive communication behavior of n is associated with the packets transmitted before the reception of any packet. These packets do not have dependency relationships. The reactive communication behavior of n is related to the rest of the transmitted packets, which are considered dependent on the received packets. Therefore, we divide a T^n_{TX} in two regions: proactive region $T^n_{TX}(p)$ and reactive region $T^n_{TX}(r)$. The $T^n_{TX}(p)$ is analyzed in order to create a MGT. The $T^n_{TX}(r)$ and T^n_{RX} are analyzed in order to create a DT.

To illustrate the concepts previously presented, we analyze some packets exchanged in a 2x2 mesh NoC. Figure 4.4a depicts that scenario. The node 00 has only a transmission trace with proactive region $T^{00}_{TX}(p)$. This node sends packets to node 10 before receiving any packet. Those packets are related to the proactive behavior of node 00. The node 01 has a reception trace T^{01}_{RX} and a transmission trace with reactive region $T^{01}_{TX}(r)$. This node sends packets to node 11 in response to the packets previously received. Those packets are related to the reactive behavior of node 11. The node 10 has a reception trace T^{10}_{RX} and a transmission trace with reactive region $T^{10}_{TX}(r)$. At last, the node 11 has

only a reception trace T_{RX}^{11}. Figure 4.4b depicts the description of the traces of each node in this example.



(a) Example of packets exchange.

Transmission Trace Node 00

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
1	00	10	18	10	12
2	00	10	18	30	32

Reception Trace Node 01

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
3	10	01	6	14	16
4	10	01	6	34	36

Transmission Trace Node 01

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
5	01	11	12	12	18
6	01	11	12	38	40

Reception Trace Node 10

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
1	00	10	18	10	12
2	00	10	18	30	32

Transmission Trace Node 10

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
3	10	01	6	14	16
4	10	01	6	34	36

Reception Trace Node 11

ID	SOURCE	DEST	LENGTH	TIMESTAMP TX	TIMESTAMP RX
5	01	11	12	12	18
6	01	11	12	38	40

(b) traces description for each node.

Figure 4.4: Example of traces from a 2x2 mesh NoC.

A trace event represents a packet exchange between two nodes. The packets are tagged according to its source, destination and length. The tags are sequentially asserted for each packet in a node's T_{RX}. Figure 4.5 depicts the packets tagging process. In this example, we analyze the T_{RX}^{01} in which three tags are found. The tag 01 represents the set of packets transmitted by node 10 with length 16-flits. The tag 02 represents the packets from the same node which have 32-flits. The tag 03 represents the packets from node 11 with 16-flits. The packets with the same tag are assumed to trigger the same behavior on the destination node. We define as the degree of a T_{RX}^{n} the number of unique

tags presented in the reception trace of node n. A trace with degree three has the tags varying from 1 to 3.

id	source	dest	length	Timestamp TX	Timestamp RX
0	10	01	16	100	160
1	10	01	32	130	190
2	10	01	16	230	260
3	11	01	16	430	490
4	10	01	32	530	590

→

tag
1
2
1
3
2

Figure 4.5: Example of packets tagging process of T_{RX}^{01}.

A packet is synthetically represented by m_t, where t is an integer which represents the packet tag. This notation hides the majority of the parameters of a packet for clarity's sake. It is convenient to represent the case where multiple packets are received in a row, for instance $m_t[N]$ indicates that N copies of m_t were received. Another representation of a packet highlights the transmission timestamp, for instance, $m_t(X)$ indicates that the packet m_t must be sent X clock cycles after the last reception event.

4.2.2 Dependency pattern

A Dependency Pattern (DP) is the synthesis of a repeated communication behavior of an application node. A DP indicates how this node reacts to the reception of a pre-defined set of input packets. It defines the sequence of output packets generated as well as the moment when these packets are sent. Basically, one DP represents a logical rule for generating packets according to the packets previously received.

A $DP(P_{in} \rightarrow P_{out})$ is a mapping from a set of input packets P_{in} to a set of output packets P_{out}, where each output packet must be generated after a while since the reception of P_{in}. The time between the reception and the transmission of each output packet is related to its execution time. For instance, the $DP_A(m_1[2], m_2 \rightarrow m_4(240))$ represents a DP with input packets $P_{in} = \{m_1[2], m_2\}$ and output packets $P_{out} = \{m_4\}$, where the output packet must be sent 240 clock cycles after the reception of the input packets.

Consider the $DP_B(m_3[2], m_4 \rightarrow m_4(560))$. In this example, there are two references to m_4. The first reference is in the input packets and the second on the output packets. It is important to highlight that those references are linked to different packets, because the m_4 in the output packet set is destined to another node on the network. As said before, the packets notation hides the majority of the packets parameters for clarity's sake.

4.2.2.1 DPs properties

The DPs properties are defined bellow:

- **Equality:** A pattern $DP_0(I_0 \rightarrow O_0)$ is equal (=) to pattern $DP_1(I_1 \rightarrow O_1)$, if $I_0 = I_1$ and $O_0 = O_1$. Basically, the input packets sets and the output packets sets of both patterns must be equal, i. e., each packet of each tag must have the same values of packets quantity and execution time. Moreover, the packets order in both packets sets is not relevant because the packets order depends heavily on the

packet injection distribution on the network. Thus, $DP_0(m_2, m_1[2] \rightarrow m_4(240))$ is equal to $DP_1(m_1[2], m_2 \rightarrow m_4(240))$.

- **Similarity:** A pattern $DP_0(I_0 \rightarrow O_0)$ is similar ($\sim$) to pattern $DP_1\{I_1 \rightarrow O_1\}$, if $I_0 = I_1$ and O_0, O_1 have the same packets tags and destinations, but some packets of them have different execution time. For instance, $DP_0(m_2, m_1[2] \rightarrow m_4(240))$ is similar to $DP_2(m_1[2], m_2 \rightarrow m_4(270))$.

- **Difference:** A pattern $DP_0(I_0 \rightarrow O_0)$ is different ($\neq$) than pattern $DP_1\{I_1 \rightarrow O_1\}$, if $\forall m_{ti} \subset I_0, \forall m_{to} \subset I_1 : ti \neq to$. For instance, $DP_0(m_1, m_3[3] \rightarrow m_4(240))$ is different than $DP_1(m_2[2], m_4[3] \rightarrow m_4(240))$ because they do not share any packet with the same tag in their input packets set.

4.2.2.2 DPs merging

Two similar DPs may be merged in order to create a new DP, which replaces the previous DPs. The merge operation $\oplus$ is defined as $DP_0 = (I_0 \rightarrow O_0) \oplus DP_1(I_1 \rightarrow O_1) = DP_2(I_2 \rightarrow O_2)$, where $I_0 = I_1 = I_2$ and O_2 have the same packets tags than O_0 and O_1 but the execution time of an output packet $m_t \in O_2$ is the average of the execution times of $m_t \in O_0$ and $m_t \in O_1$.

For instance, be $DP_0(m_7[2] \rightarrow m_4(240))$ and $DP_1(m_7[2] \rightarrow m_4(260))$ two similar patterns, then $DP_0 \oplus DP_1 = DP_2(m_7[2] \rightarrow m_4(250))$. Notice that the execution time of the output packet is the average of the execution times of the input DPs. This operation is used during the traces analysis to reduce the size of lists of DPs.

4.2.2.3 Alternative notation of a DP

An alternative notation of a DP is herein presented. Be n an application node in which its T^n_{RX} has degree K. It means that node n receives packets like m_i, where $i \in [1..K]$. A set of input packets of n may be represented by a weight vector $v = \{v_1, v_2, ..., v_K\}$, where v_j represents the quantity of packets received with the tag j. Thus, a pattern of n can be represented by $DP_0 = \{v \rightarrow O_0\}$, where v is a weight vector and O_0 is a set of output packets. For instance, a DP of a node with five tags would be represented as $DP_1(\{0, 1, 0, 2, 0\} \rightarrow m_4(240))$. That representation is equivalent to this one $DP_1(m_2[1], m_4[2] \rightarrow m_4(240))$.

4.2.3 Dependency table

A Dependency Table (DT) represents the reactive behavior of an application node by its dependencies relationships between the input and output packets. A DT is a list of DPs and it can be defined as $DT_0(DP_j(v_j, O_j))$, where $j \in [0..K\text{-}1]$. It is important to notice that we use the alternative notation of a DP to highlight the representation of the input packets as a weight vector. Figure 4.6 depicts an example of a DT with three DPs: $DP_0(m_2[2] \rightarrow m_1(120), m_2(200))$, $DP_1(m_1, m_3[3] \rightarrow m_3(100))$, and $DP_2(m_4 \rightarrow m_4(50))$.

4.2.3.1 DTs properties

The DTs properties are defined bellow:

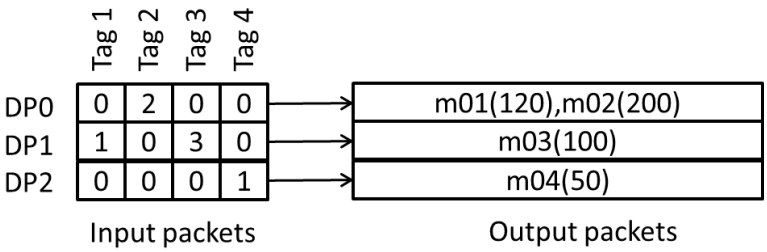


Figure 4.6: Dependency Table (DT) organization

- **Minimality:** A $DT_0(DP_j(v_j, O_j))$ is minimal if $\forall DP_i \in DT_0, \forall DP_k \in DT_0, i \neq k : \neg(DP_i = DP_k)$. Minimality indicates that there is no repeated pattern in a DT.
- **Freedom of conflicts:** A $DT_0(DP_j(v_j, O_j))$ is free of conflicts if $\forall DP_i \in DT_0, \forall DP_k \in DT_0, i \neq k : DP_i \neq DP_k$. Freedom of conflicts indicates that the patterns in a DT do not share any input packet. Moreover, each input packet is linked to only one pattern. Thus, each input packet always generates the same output packets.
- **Degree:** The degree of a $DT_0(DP_j(v_j, O_j))$ is equal to the numbers of elements of the vectors v_j. This number is equal to the number of the tags of the input packets of DT_0. For instance, the DT depicted in Figure 4.6 has degree four.
- **Output Degree:** The output degree of a $DT_0(DP_j(v_j, O_j))$ is a vector v_{od} with all the tags of O_j. For instance, the DT depicted in Figure 4.6 has output degree $v_{od} = \{1, 2, 3, 4\}$.
- **Distinct set:** A pattern DP_i of a DT_0 has a Distinct Set (DS) defined as $ds = \{DP_j : \forall DP_j \in DT_0(DP_j \neq DP_i)\}$. The set ds is composed by all DPs from DT_0 that are different from DP_i. For example, in Figure 4.6, the DS of DP_0 is $ds = \{DP_1, DP_2\}$.

4.2.3.2 Generating traffic with a Dependency Table

The generation of traffic using a DT is straightforward. This demands a status register to hold the quantity of received packets of each tag and registers to hold the input/output packets descriptions for each DP. At the reception of a new packet, the tag field is extracted. Then, the status register is incremented on the position indicated by the tag. Whenever the status register fields are equal or greater than any DP, the associated output packets are scheduled for transmission and the status register is subtracted from the DP values.

Figure 4.7 depicts the use of a DT to generate traffic in response to reception of some input packets. This figure is divided in two sides. The right side presents the organization of a DT with three DPs. The left side represents the evaluation of the patterns of that DT during the reception of some input packets. The evolution of the execution is highlighted by the event column. There are eight events in which the status register is updated. At first the status register indicates that none packet has been received. The first packet received is $m04$ in the event 1. The status register is update in event 2 to demonstrate the reception of that packet. The values of the status register are compared against the weight vectors of the DT. The DP2 is fired because its weight

vectors are equal or smaller than the status register values. In event 3, the weight vector of DP2 is subtracted from the status registers and the packet $m06$ is scheduled to be transmitted. In event 4, the packet $m03$ is received. The status register is updated in the next event, but none DP is fired because there is no DP depending only on $m03$. In event 6, the packet $m02$ is received which fires the DP0 which depends on $m02$ and $m03$. The DP0 is fired latter on event 8. The output packets of DP0 are scheduled in the last event.

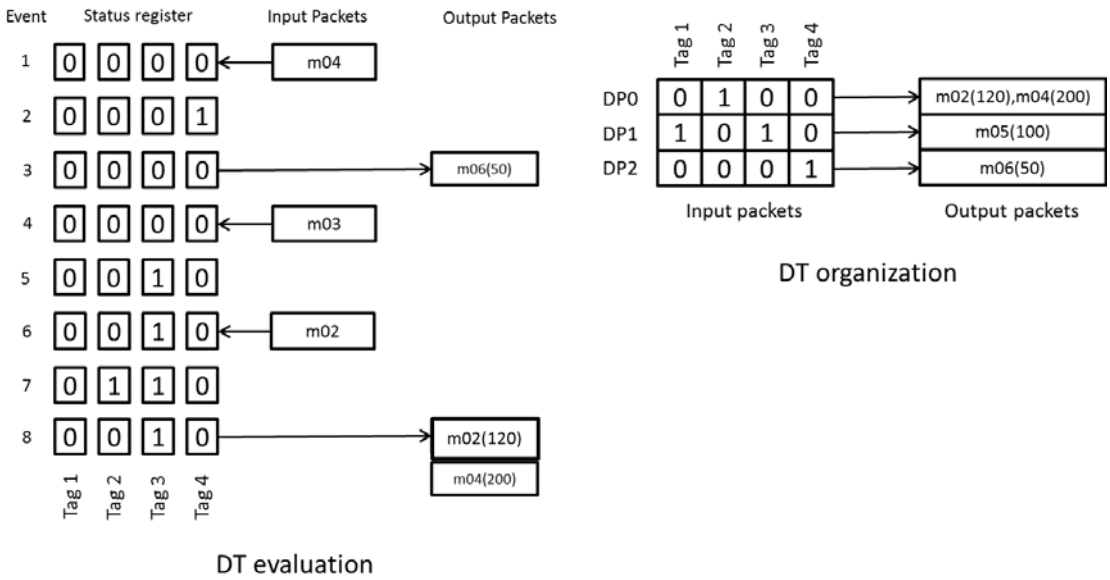


Figure 4.7: DT execution example

4.2.3.3 Patterns superposition

The patterns superposition is a condition characterized by the generation of a sequence of output packets which is the fusion of the output packets of two or more patterns. This happens when a new pattern is triggered before the end of transmission of the output packets of the last trigged pattern. The patterns superposition makes hard to identify the packets dependencies, since some input packets are not related to the next set of sent packets. This leads to the identification of false packets dependencies, which must be handled during the traces analysis.

Figure 4.8 depicts an example of superposition. The patterns $DP_0 = \{m_1, m_2 \rightarrow m_4\}$ and $DP_1 = \{m_3 \rightarrow m_5\}$ are not superposed in (a), but in (b) both patterns happen at same time what creates a false pattern $DP_2 = \{m_1, m_2, m_3 \rightarrow m_4, m_5\}$.

4.2.4 Base Set

A Base Set is used in the traces analysis in order to find a minimal and free of conflicts DT for an application node. A Base set of DT_j is defined as $BS = \{B_i(v_i, O_i)\}$ which is a set of all possible DTs formed by the combination of the DPs presented in DT_j, where:

- $\forall B_i$ has the same degree of DT_j;
- $\forall B_i$ has the same output degree of DT_j;

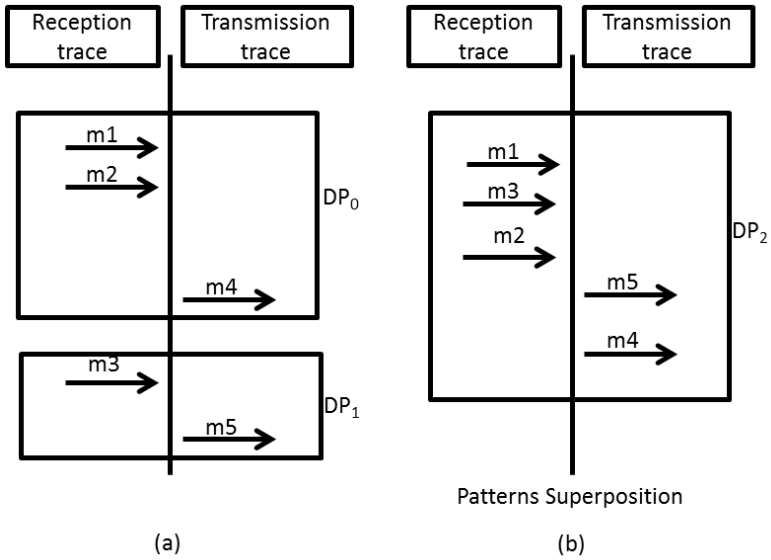


Figure 4.8: Superposition example.

- $\forall B_i$ is minimal;
- $\forall B_i$ is free of conflicts.

A consequence of the definition of the Base Set is that given $B_i \in BS$, the Base Set BS' of B_i is B_i itself. Basically, the Base Set of an element of another Base Set is the element itself, i. e., a BS element cannot be further processed in order to generate other DTs.

Figure 4.9 depicts an example of Base Set. The left side of this picture presents a six patterns DT. Then, the right side presents that DT's Base Set, which contains three DTs each one with two patterns. Each BS element is a functional DT which could be executed in order to generate data traffic. However, it is necessary to verify the fidelity of the traffic generated by a BS element compared to an application node, whose traces were used to generate that BS. A procedure to build a Base Set from a DT is described latter in this chapter. Moreover, we also propose a procedure to evaluate the fidelity of the traffic generated by the BS elements.

4.2.5 Message generation table

The proactive behavior of a node is associated with the packets transmitted before the reception of any packet, i.e., the independent traffic. A Message Generation Table (MGT) is built from the information gathered from traces. A $MGT = \{m_j(send\_time, dest, length)\}$ indicates the number of clocks to send the next packet compared to the last sent packet, as well as the destination node, the packet's length and the tag.

4.3 Problem definiton

Using the previous definitions, the problem of extracting a MoC based on the packets dependencies from application traces can be formulated as follows. Given T^n_{TX} and T^n_{RX} of a node n extracted from a full-system simulation in an unknown NoC architecture,

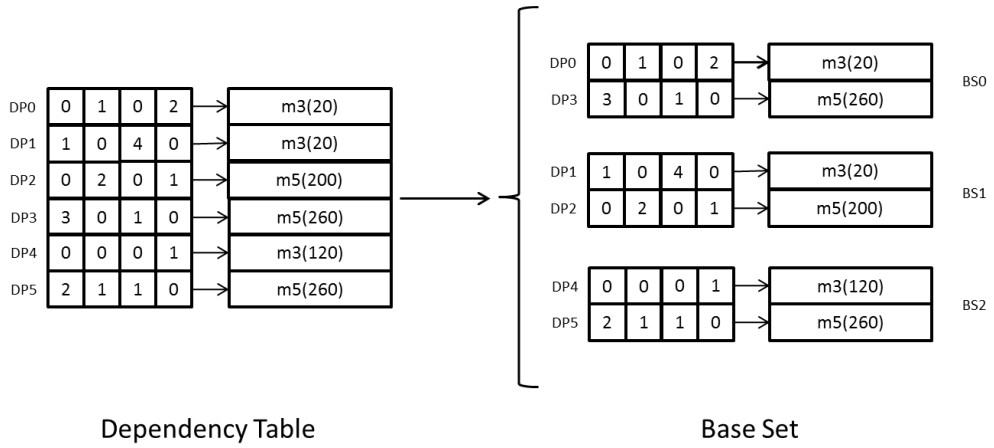


Figure 4.9: Base set example.

find a MGT_n and a DT_n, which is minimal, free of conflicts, and it has the same degree than T^n_{RX}. The MGT_n and DT_n are used to synthesize an architecture of a dependency-aware TG for FPGA-based NoC emulation.

4.4 Framework workflow

The goal of traces analysis is investigating the communication behavior of a NoC-based application. The resources constraint is one of the main issues in FPGA-based NoC emulation. A designer must implement the NoC architecture as well as the traffic model components in one or more FPGAs in order to execute a performance evaluation. It is important that the traffic model components have lightweight resources requirements while they reproduce accurately application traffic. In this work, we address the issue to build lightweight and accurate traffic model components based on a single application trace. This problem is addressed in two steps. The first step is the generation of an executable MoC from a regular application trace. The second step is the design of an architecture of a dependency-aware TG.

We propose a traces analysis framework which retrieves the packets dependencies in order to create a Dependency Table (DT) for an application node. A DT is the synthesis of the reactive behavior of an application node. Afterwards the DT is synthesized as TG for NoC emulation. Figure 4.10 depicts an overview of the traces analysis framework. The first process is extracting a list of dependency patterns (DPs) by the analysis of the reception and transmission traces. One DP represents a logical rule for generating packets according to the packets previously received. The packets sent before any packets reception are grouped as a MGT. The rest of the packets have dependency relationships and from them is created a list of DPs. In the second process, those DPs are merged whenever possible in order to create an initial DT.

The third process is the extraction of the Base Set (BS). A BS is a set of DTs generated from the initial DT. Each DT in the BS models the behavior of the same application node. The BS represents a space of solutions, in which some solutions are more accurate than others. It is necessary to evaluate these DTs in order to find out the most accurate one. The third process is the BS evaluation. The evaluation process uses the original traces

to verify if the evaluated DT is capable of reproducing the behavior registered into the traces. A imprecision index is calculated from this evaluation. The DT with the smallest imprecision index is taken as the correct one.

The next step in the design flow is to build a dependency-aware TG architecture from the MoCs extracted on the previous step. The challenge is to design light but accurate traffic model components. In order to optimize the resources requirements, a specific TG is created for each application task. This TG reproduces only the behavior of a target task thus reducing its resources overhead. The generated TGs can be connected to the target NoC to reproduce the evaluated application behavior.

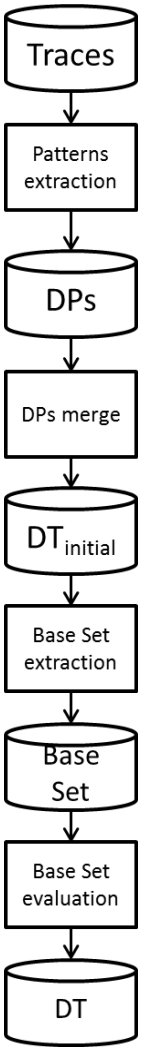


Figure 4.10: Overview of the traces analysis framework.

4.4.1 Patterns extraction

The first step is to create a trace events repository, which consists in the exchanged packets information. A trace event describes a packet exchange between two application nodes. From that repository is possible to get the transmission trace events and the

reception trace events for each application node. The second step is to tag similar trace events. Packets with the same destination, source and size are tagged with the same value.

Figure 4.11 depicts the third step of the traces analysis. The third step consists in creating the DTs and MGTs for each application node. The transmission trace and the reception trace of each application node are analyzed in order of occurrence. All packets sent before the reception of a packet are added to a MGT.

The rest of the packets exchanges have dependencies relationships. Each group of sent packets is related to the last group of received packets in order to create the DPs, which must be analyzed to retrieve the true dependencies of a node. The repeated DPs are eliminated. Similar DPs are merged into simpler DPs. An initial DT is built from the found DPs in the patterns extraction process.

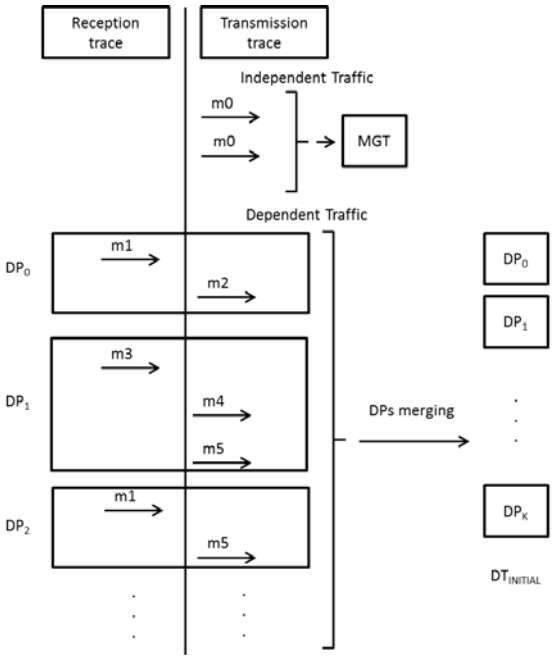


Figure 4.11: Initial traces analysis

4.4.2 Base set extraction

This framework main goal is to identify a minimal and free of conflicts DT for each node. However, after the initial trace analysis, the identified DT usually has logical conflicts due to the patterns superposition. In order to find out the true DT, a base set is extracted from the initial DT. Let's define a base set. Be $DT_{initial}$ a minimal and conflicted DT got from the traces analysis. A base set BS of $DT_{initial}$ is defined as a set of minimal and free of conflicts DTs, which have all the tags from $DT_{initial}$.

A base set represents all possible solutions to model the reactive behavior of a node, but some solutions may be less accurate due to false dependencies. The proposed framework builds the base set in order to evaluate its elements to find out the best possible DT. Given a minimal and conflicted DT, it is possible to extract its base set. The base set extraction algorithm's goal is to create a set of minimal and free of conflicts

DTs. The base set extraction procedure is presented in Algorithm 1. For each DP, the algorithm gets its Distinct Set (DS), which is composed by all DPs that are different from the analyzed DP. Then the Power Set (PS) of DS is generated, which is all the subsets of DS. Afterwards the DP is added into each PS element. The PS elements are evaluated in order to verify if they met the requirements of a DT. Those elements that met the requirements are added into the BS.

Figure 4.12 depicts one iteration of the base set extraction algorithm execution. The $DP_0(m_1 \rightarrow m_2)$ is the first DP. Its DS is formed by $DP_1(m_3 \rightarrow m_4, m_5)$ and $DP_3(m_3 \rightarrow m_4)$. The PS is composed by three elements. The first element $PS_0 = (DP_0, DP_1)$ is evaluated. It is a minimal and free of conflict DT, thus it is added in the BS. The second element $PS_1 = (DP_0, DP_3)$ is evaluated. It is also a minimal and free of conflict DT, however it does not have all the output packets tags registered on the traces. It lacks m_5 packets. Thus, it is not added into the BS. The last element $PS_2 = (DP_0, DP_1, DP_3)$ has a logical conflict, because the packet m_3 triggers two sets of output packets. Then, it is not added into the BS. Afterwards, the first DP is removed from the $DT_{initial}$ and the loop iterates until all the DPs are analyzed.

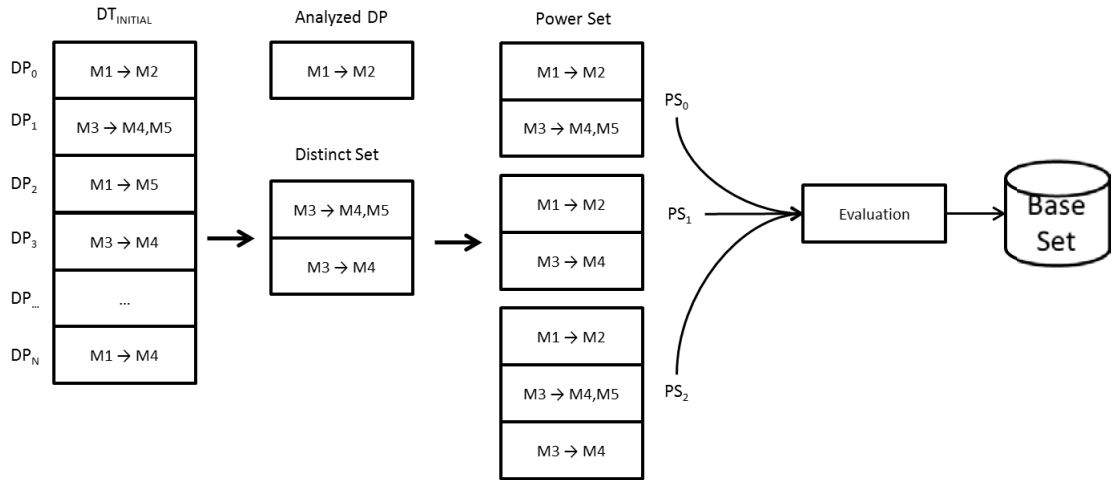


Figure 4.12: One iteration of the Base Set extraction algorithm

Algorithm 1 Base set extraction

```
1: for each dp ∈ DT do
2:     DS ← getDistinctSet(dp,DT)
3:     PS ← getPowerSet(dp,DS)
4:     for each g ∈ randomWalk(PS) do
5:         if testBaseSetElement(g) then
6:             addToBaseSet(g,BS)
7:     removePattern(dp,DT)
8: return BS
```

The PS size grows exponentially according with the size of DS, this makes it hard to search the complete space of solutions. In order to afford the computational costs to analyze a PS, we limit the number of analyzed elements while keeping their represent-

ativeness. We use an oriented random walk strategy to sample the PS elements to be analyzed.

4.4.3 Random walk strategy

The random walk strategy selects a predefined number of PS elements for evaluation. Figure 4.13 depicts an example of the execution of this algorithm. In this case, the PS has 1024 elements which are represented by the biggest arrow. A 10-bit pointer is used to address an element in this PS. The first analyzed region is represented by the dashed line rectangle. It has a predefined length, for this example the first 128 elements are analyzed, the other analyzed regions are represented by rectangles. After the evaluation of the elements from the first region, the algorithm randomly selects a new region. The distance between two regions is determined by a random jump, which has a limited length. The random walk is oriented because it iterates towards the same direction. The regions sizes are randomly selected and they are limited to a maximum value.

The algorithm keeps selecting new regions until evaluated a pre-defined number of elements. In this example, one quart of the PS elements are evaluated after traversing four regions. The region 0 has 128 elements, the region 1 has 60, the region 2 has 50 and the last has 18 elements. Those elements are evaluated in order to create a base set.

The algorithm parameters like region size, jump length and number of evaluated elements may be assigned from the size of the PS. In our experimentations, the number of evaluated elements is defined as 512, the maximum region size is 64 and the maximum jump length is the PS size divide by 50.

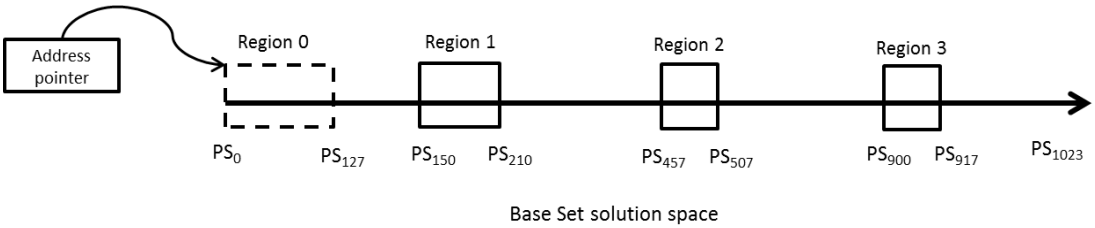


Figure 4.13: Random walk strategy

4.4.4 Base set evaluation

A DT is composed by DPs which specify rules to generate output messages according to a set of input messages. If a DT is composed by the true dependencies, it must generate the same sequence of output messages recorded on the traces, whenever it receives stimuli from the same traces. That is the core of the proposed base set evaluation algorithm. Moreover, if a DT fails to reproduce correctly the recorded behavior, it is possible to measure its degree of imprecision. In some cases, it is impossible to recover the true dependencies, but it is possible to give an approximation with an imprecision indicator. In other cases, the framework will fail in recovering an usable DT.

The algorithm is henceforth described. A data structure called *input_register* is used to hold the number of received messages of each tag. Another data structure called *output_register* is used to hold the number of sent messages of each tag. A pattern is read from the trace. Then, the number of received messages is added up on the correct

positions on the *input_register* variable and the number of sent messages is added up on the *output_register* as depicted in Figure 4.14 (a). The DT is checked in order to identify if any DP has been fired. If one DP has been fired, the number of input messages in this DP is removed from the respective positions on the *input_register* and the DP output messages are subtracted from the *output_register* as depicted in Figure 4.14 (b). These operations are repeated until all the trace patterns are checked.

At the end, an imprecision index is calculated. This index summarizes the DT capacity of reproducing the traces behavior. It is calculate as shown on Equation 4.1, where *number_messages* represents the number of received/sent messages and the modulus sum of *input_register* and *output_register* represents all the received/sent messages that do not correctly fired the DT rules. If the analyzed DT is composed only by the base patterns, the imprecision index is zero or a very small number. The DT with smallest imprecision index is selected as the base DT.

$$imprecision\_index = \frac{\sum |input\_register| + \sum |output\_register|}{number\_messages} \tag{4.1}$$

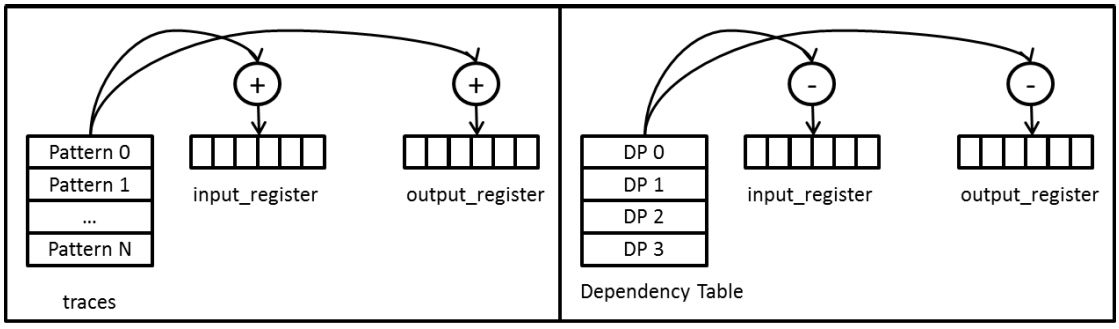


Figure 4.14: Base set evaluation

4.5 Dependency-aware traffic generator

The dependency-aware traffic generator is designed to implement the proposed MoC onto an FPGA. Figure 4.15 depicts its architecture. This component is divided into five blocks: TR, TG, MTGBLOCK, DTBLOCK and arbiter. The TR is responsible for the packets reception and for calculating the traffic measurements. It interfaces with DTBLOCK to indicate the reception of a new packet. The update of the DTBLOCK is carried out after the reception of the packet's header, which contains the packet tag information. The DTBLOCK is the hardware implementation of a DT. It receives the packet tag from the TR, then it updates its internal registers on DTIN and it checks if a DP has been fired. Whenever a DP is fired, the DTBLOCK schedules the transmission of the related output packets on DTOUT. The DTCTR block controls the DTBLOCK interface with the arbiter.

The MTGBLOCK is the hardware implementation of a MGT. It has a small lookup table (LUT) containing the independent packets parameters. The MTGCTR block controls the MTGBLOCK interface with the arbiter in order to schedule packets transmissions. The arbiter uses a simple round-robin scheduler to decide which block sends

packets. The TG creates the packets frames and it sends them to the NoC.

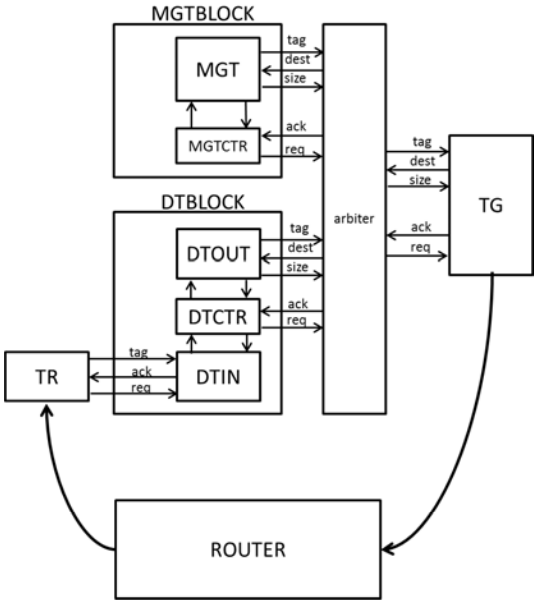


Figure 4.15: Dependency-aware traffic generator

4.6 Experiments

In order to evaluate the proposed framework, a set of CABA simulations is carried out. A workload of 10 synthetic applications is generated by the TGFF tool [DRW98a]. Each application is an acyclic task graph with 36 tasks. The application's independent tasks, the ones which do not receive packets, are modeled as MGTs generated from a Pareto on-off traffic generator. The other tasks are modeled as DTs.

The TGFF models are simulated onto two base architectures (A and B). The architecture A is a 6x6 Mesh NoC, XY routing, 32-flit buffer, credit based flow control. The architecture B is a 4x9 Mesh NoC and it has the same features of the first one. From the simulation we extract the execution traces which are applied on the proposed framework to extract the MoCs. The traces extracted from architecture A are named Models A. The others are named Models B. For each application, there are two traces (A and B) and two MoCs (A and B), which are created from the analysis of those traces. These traces and MoCs are replayed in different NoC architectures in order to verify its accuracy in relation to the original application.

4.6.1 Data storage requirements

The Table 4.3 presents the data storage requirements for both analyzed traffic generators for each target application. In average, the framework requires only 3% of the bytes used by the traces and this value varies between 1.29% and 8.50% depending on the target application. This is due to the synthesis capacity of the MoC proposed, which replaces repeated traffic patterns by simple logical rules.

Application	Traces (Bytes)	Framework (Bytes)	Ratio (Framework/Traces)
01	244,224	3,159	1.29%
02	396,499	15,265	3.85%
03	634,934	11,764	1.85%
04	339,461	11,819	3.48%
05	417,721	7,290	1.75%
06	286,169	7,601	2.66%
07	301,593	12,185	4.04%
08	175,727	15,190	8.50%
09	259,376	15,294	5.90%
10	377,707	3,867	1.02%
Average	343,341.1	10,343.4	3.45%

Table 4.3: Data storage requirements

4.6.2 FPGA occupation

The Table 4.4 presents the synthesis results of the emulation components, the NoC as well as the total resources available on the Virtex 7 (xc7vx485t) FPGA. The synthesis is carried out with Xilinx XST. The EmuCore component represents all the emulation related components. The values presented correspond to one emulation node. The NoC 6x6 correspond to all 36 routers and their interconnections. One router in average occupies 160 registers and 545 LUTs. These values are close to the occupation of one emulation core. The routers usually use very less resources than the application nodes connected to them. Thus, the emulation core is a very lightweight application node. Indeed, one emulation core occupies 2.01% of the registers and 0.75 % of the LUTs of the platform.

The column Platform presents the occupation of the emulation components as well as the NoC. The NoC occupies 39.7% of the platform registers and it occupies 53.4% of the LUTs. The rest of the resources are used on the emulation components. The overall platform uses only 2% of the registers and 12% of the LUTs available on the target FPGA.

	EmuCore	NoC 6x6	Platform	FPGA
Number of slices registers	292 (2.01 %)	5766 (39.7%)	14517 (2%)	607200
Number of slice LUTs	279 (0.75%)	19636 (53.4%)	36775 (12%)	303600

Table 4.4: FPGA occupation

4.6.3 Evaluation of the MoC

In order to evaluate the performance of the MoCs retrieved on the two base architectures, we carry out three sets of simulations on different NoC architectures.

In the first set of simulations, three network parameters are changed: network dimension, buffer size and routing algorithm. The network dimensions are 6x6, 4x9 and 3x12. The routing algorithms are Odd-Even, West-First Minimal, Dynamic XY and Negative First. The buffer sizes are 32-flits, 16-flits and 2-flits. All the other parameters are equal to the original NoC. There are a total of 36 different target networks. These experiments evaluate the accuracy of the traffic generators (framework MoC and trace-based) in 2D mesh NoCs with different configurations (routing algorithm, buffer size, dimensions) than the two base architectures.

In the second set of simulations, a 3D NoC topology is used as target network. The network dimensions are 3x3x4. A XYZ routing algorithm and handshake flow control are used. The buffer sizes are 32-flits, 16-flits and 4-flits. All the other parameters are equal to the original NoC. There are three different target networks for this simulation set. These experiments evaluate the accuracy of the traffic generators (framework MoC and trace-based) in a network with different topology.

In the third simulation set, a 6x6 mesh NoC is used as target network. This network uses XY routing algorithm and 32-flits buffer. Twenty random task mapping are used. These experiments evaluate the accuracy of the traffic generators (framework MoC and trace-based) for distinct task mapping scenarios.

The goal of all those experiments is to verify which traffic generator (framework MoC or trace-based) reproduces more accurately the behavior of the applications. That's why we play those traffic generators in a broad range of NoC configurations. Those NoCs are different from the networks used to extract the traces and the MoCs used in the experiments.

For all the simulation sets, the framework MoC and the traces are executed on the target networks and its performances is compared against the real applications, which are also executed on those networks. The evaluation metrics is the packets latency. We calculate the Mean Absolute Percentage Error (MAPE) of latency. Herein, we describe the method to calculate that metric. From the simulations, we gather the average latency between each communication link (source-destination pair) on the application. The procedure is carried out for each traffic model (application, framework MoC and traces). The application results are considered as the baseline model, thus the error is calculate in reference to them. For each link, the metrics are calculated and the average of these metrics is used to compare the precision error of the traffic models.

4.6.3.1 2D Mesh evaluation

The experiments results are presented classified by routing algorithm. The first routing algorithm analyzed is Odd-Even, which is a non-minimal and semi-adaptive routing algorithm. Figure 4.16 presents the MAPE of latency for different network configurations - (a) 32-flits buffer size, (b) 16-flits buffer size and (c) 2-flits buffer size.

There is an evident advantage for the framework independently of network dimension, buffer size and model. The MAPE of latency, for the framework MoC, varies from 4.73% to 13.08%. However, the trace-based simulations have a bigger variation of the MAPE of latency. It varies from 5.08% to 62.07%. There is a clear difference between the trace-based and the framework models for the 2-flits buffer scenarios. In these cases, the traces lack flexibility to adapt the network configuration. There is also an accuracy variation for the traces when we analyze the precision of Model A and Model

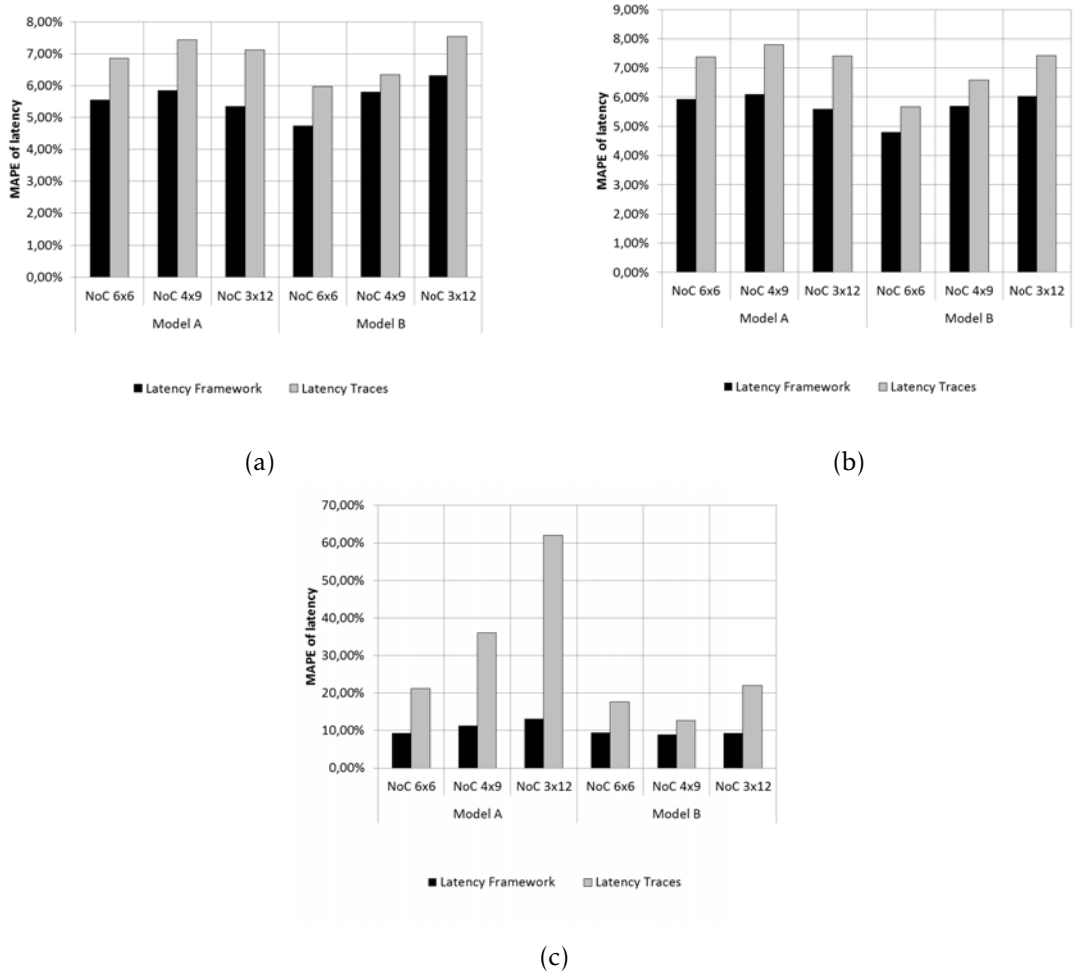


Figure 4.16: Odd-Even routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits.

B, which are extracted from different base architectures. However, for both models, the framework MoC is more fidel to the application behavior.

The second routing algorithm analyzed is a minimal version of West-First, which is a semi-adaptive routing algorithm based on turn-models. Figure 4.17 presents the MAPE of latency for different network configurations - (a) 32-flits buffer size, (b) 16-flits buffer size and (c) 2-flits buffer size.

For the 32-flits and 16-flits buffer scenarios, there no clear differences between the framework MoC and the trace-based simulations. Indeed, the MAPE of latency is very low and it varies only from 2.18% to 3.95%. It is fair enough to say that both models have similar precision. However, for the 2-flits buffer scenarios, there is a clear advantage for the framework models. The traces MAPE of latency varies from 5.03% to 13.53% against a variation of 3.89%-8.20% for the framework MoC.

The third routing algorithm is Dynamic XY (DyX), which is a minimal and semi-adaptive routing algorithm. Figure 4.18 presents the MAPE of latency for different network configuration - (a) depicts the results for 32-flits buffer size, (b) for 16-flits buffer size and (c) for 2-flits buffer size.

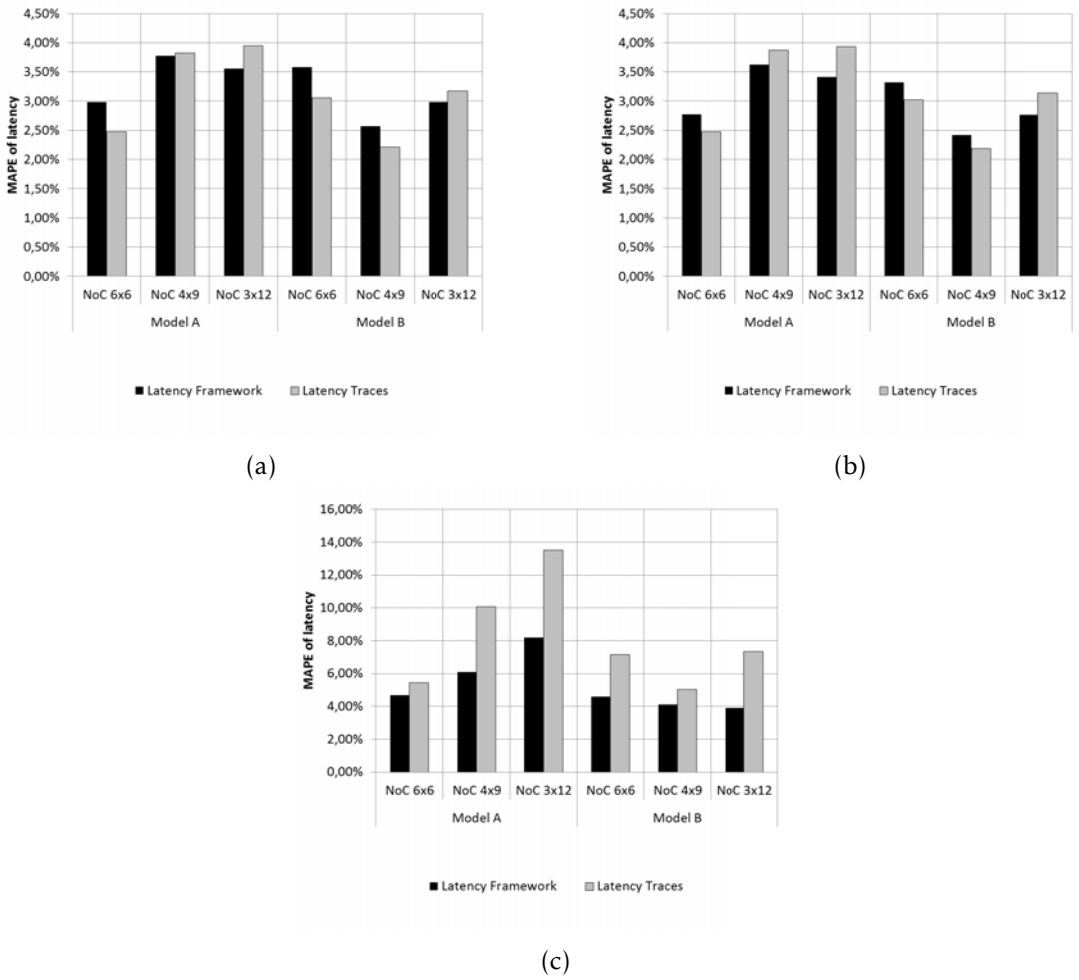


Figure 4.17: West-First routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits.

There are slight differences between the trace-based and framework MoC simulations for the 16-flits and 32-flits buffer scenarios. The MAPE of latency varies only from 1.86% to 3.99%. It is a very low variation. For these scenarios, both models have similar performances. However, for the 2-flits buffer scenarios, the trace-based simulations have worse precision. The MAPE of latency varies from 4.43% to 14.38% against a variation of 3.99%-8.03% for the framework MoC.

The last routing algorithm is Negative First, which is a non-minimal, semi-adaptive turn-model routing algorithm. Figure 4.19 presents the MAPE of latency for different network configuration - (a) depicts the results for 32-flits buffer size, (b) for 16-flits buffer size and (c) for 2-flits buffer size.

For the 32-flits buffer scenario, there is no clear difference between the trace-based and framework. There is a slight difference for the 16-flits buffer scenario, in which the framework has a better precision. However, there is a clear precision difference for the 2-flits buffer scenario. The trace-based simulations have a MAPE of latency between 6.43% and 21.99% against a variation of 4.5%-8.45% for the framework MoC.

The experiments results are also classified by the applications. Figure 4.20 depicts

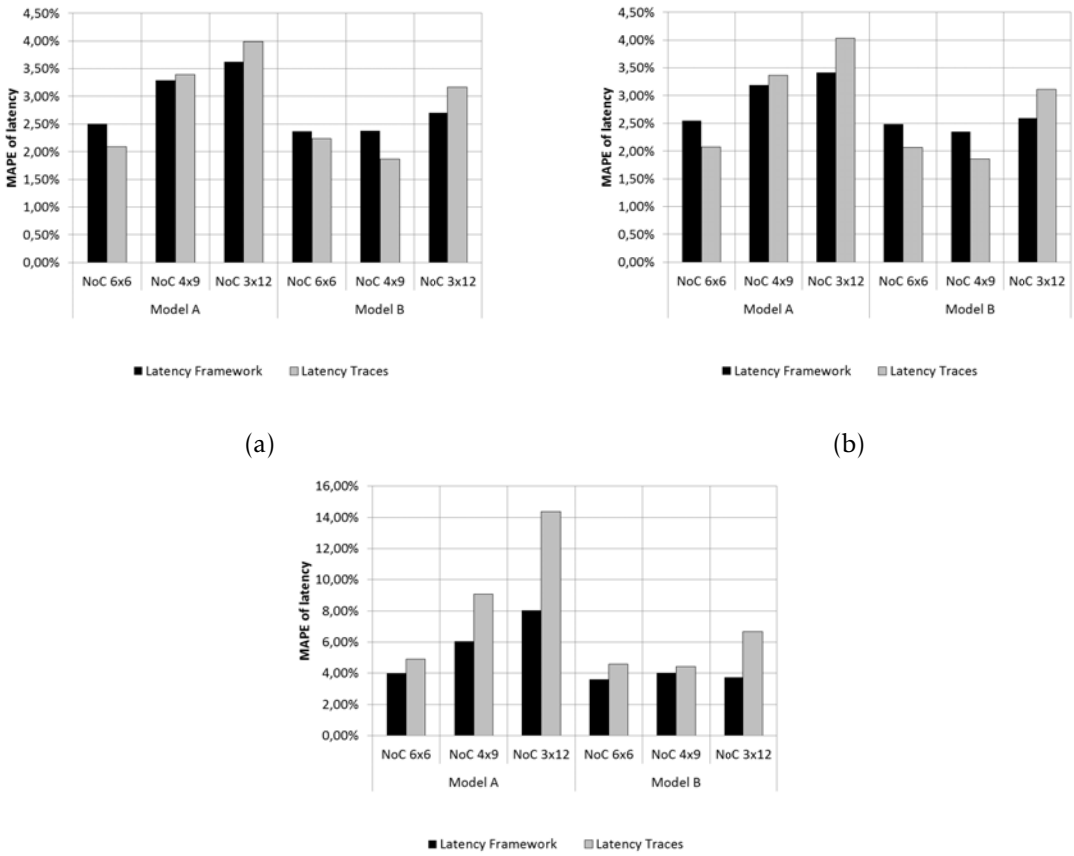


Figure 4.18: DyXY routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits.

the MAPE of latency for each one of the ten applications. It is shown the error for trace-based and framework. These results are the averages of all the experiments carried out in this subsection. For the applications 1, 2, 3 and 10, the framework has a smaller MAPE for Model A and B. There is a clear difference between the framework MoC and the trace-based simulations. For the other applications (4, 5, 6, 7 and 8), the framework has smaller MAPE for Model A, but the MAPE for Model B is very close to the trace-based MAPE. One could say that framework and trace-based have similar performance for these applications on Model B.

For applications 2, 3, 4, 5, 6 and 10, there is a big difference between Model A and Model B for trace-based simulations. The traces behave differently on these scenarios. It suggests that traces gathered from different architectures (Model A and Model B) do not behave similar onto different networks. The same happens for the framework, but in a smaller scale. For the applications 4, 5 and 6, there is a clear difference between Model A and Model B for the framework simulations. It also suggests that the framework MoC depends on the original network architecture, but this dependency is smaller in comparison with the trace-based simulation.

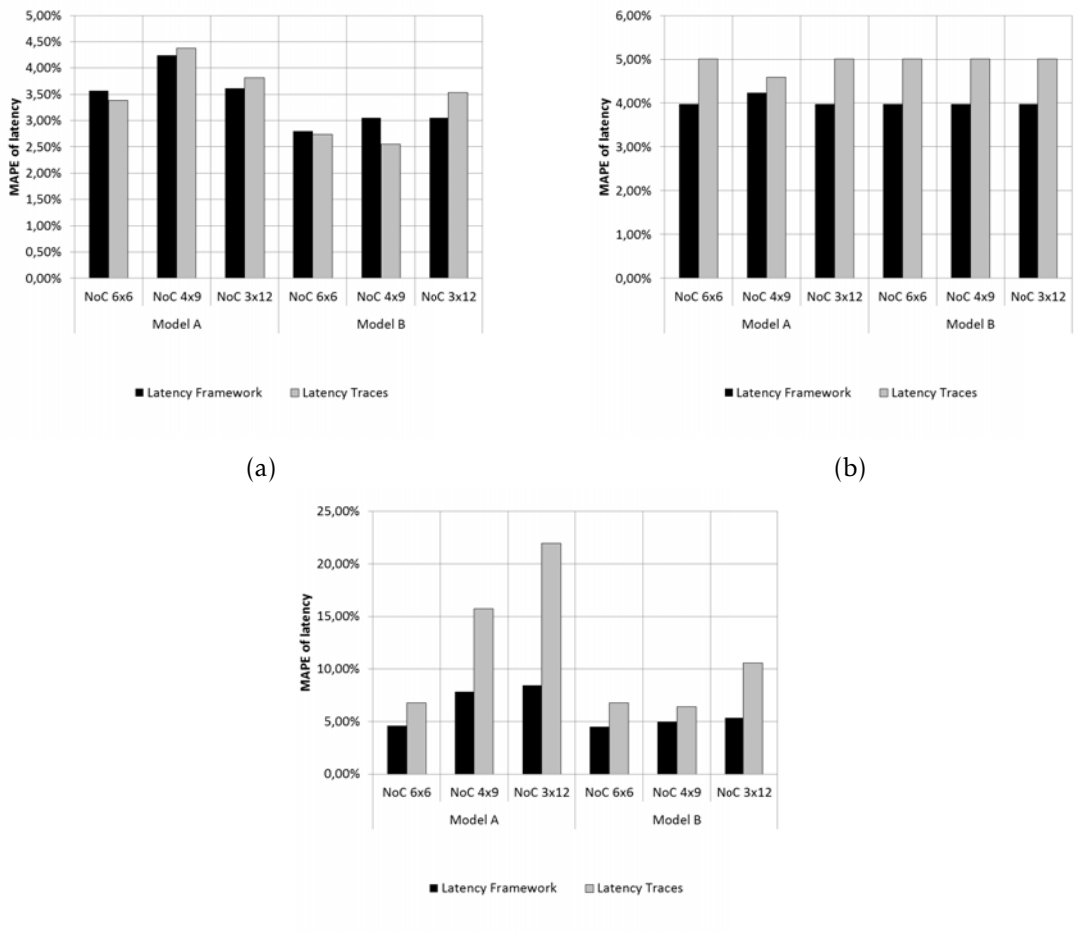


Figure 4.19: Negative-First routing scenarios. (a) 32-flits. (b) 16-flits. (c) 2-flits.

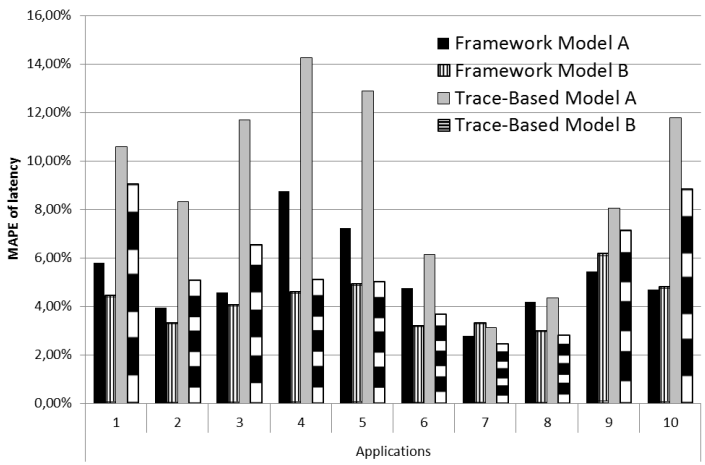


Figure 4.20: Applications errors analysis

4.6.3.2 3D Mesh evaluation

Figure 4.21 depicts the results of the evaluation on a 3x3x4 Mesh NoC for different network configurations - (a) 32-flits buffer size for Model A, (b) 16-flits buffer size for Model A, (c) 4-flits buffer size for Model A, (d) 32-flits buffer size for Model B, (e) 16-flits buffer size for Model B and (f) 4-flits buffer size for Model B. The framework MoC has an error smaller than the traces for all applications, except applications 6 and 7 for Model B. For Model A, the MAPE of latency is in average 12% for the framework MoC and 20% for the traces. The simulations executed using the Model B have smaller MAPE of latency. For Model B, the MAPE of latency is in average 3.3% for the framework MoC and 4.2% for the traces. This evaluation highlights the differences between traces and MoCs extracted from different networks architectures. The original network has an influence over the accuracy of the MoCs when they are executed on different architectures. This difference is smaller on the evaluations carried out in 4.6.3.1, in which the target networks have the same topology of the base architectures.

The error does not vary according to the buffer size. A 3D NoC has more bandwidth than a 2D counterpart. Thus, even for 4-flits buffers there are no increase on the MAPE of latency. This is not true for the 2D mesh evaluation scenarios, in which the 2-flits buffer scenarios show bigger errors than large buffers scenarios.

4.6.3.3 Random task mapping evaluation

Figure 4.22 presents the results of the evaluation onto twenty random task mapping scenarios. The target NoC is 6x6, XY routing, credit based flow control and 32-flit buffer network. Each one of the workload applications is executed under those task mapping scenarios. The MAPE of latency is in average 3.58% for the framework, and it is in average 3.50% for the trace-based simulation. The error is very small because the target network have similar configuration than the base architecture only the tasks placement is different. This highlights that tasks placement has a minor influence on the accuracy of those traffic generators.

4.6.4 Results analysis and discussions

The experiments carried out are designed to evaluate two features of traffic generators: data storage requirements and fidelity in relation to the real applications traffic. The goal in the design of a traffic generator for NoC emulation is to have low data storage requirements while keeping high fidelity traffic. Herein we analyze and discuss the results of the presented experiments.

The data storage requirements for both traffic generators are presented in Table 4.3. The traces demand more data storage space because they hold information about each packet traversing the network. The framework MoC uses DPs to replace repetitive packets exchanges. Those DPs are organized as a DT, which synthesizes the reactive communication behavior of a node. This requires less data storage than holding all the packets information. The framework MoC also holds the data for the MGT, which represent the proactive communication behavior of a node. However, the overall data storage requirement of a MoC is in average only 3% of the data storage for a trace. This requirement reflects in the FPGA occupation of one emulation node, which is the hardware implementation of a framework MoC. As presented in Table 4.4, one

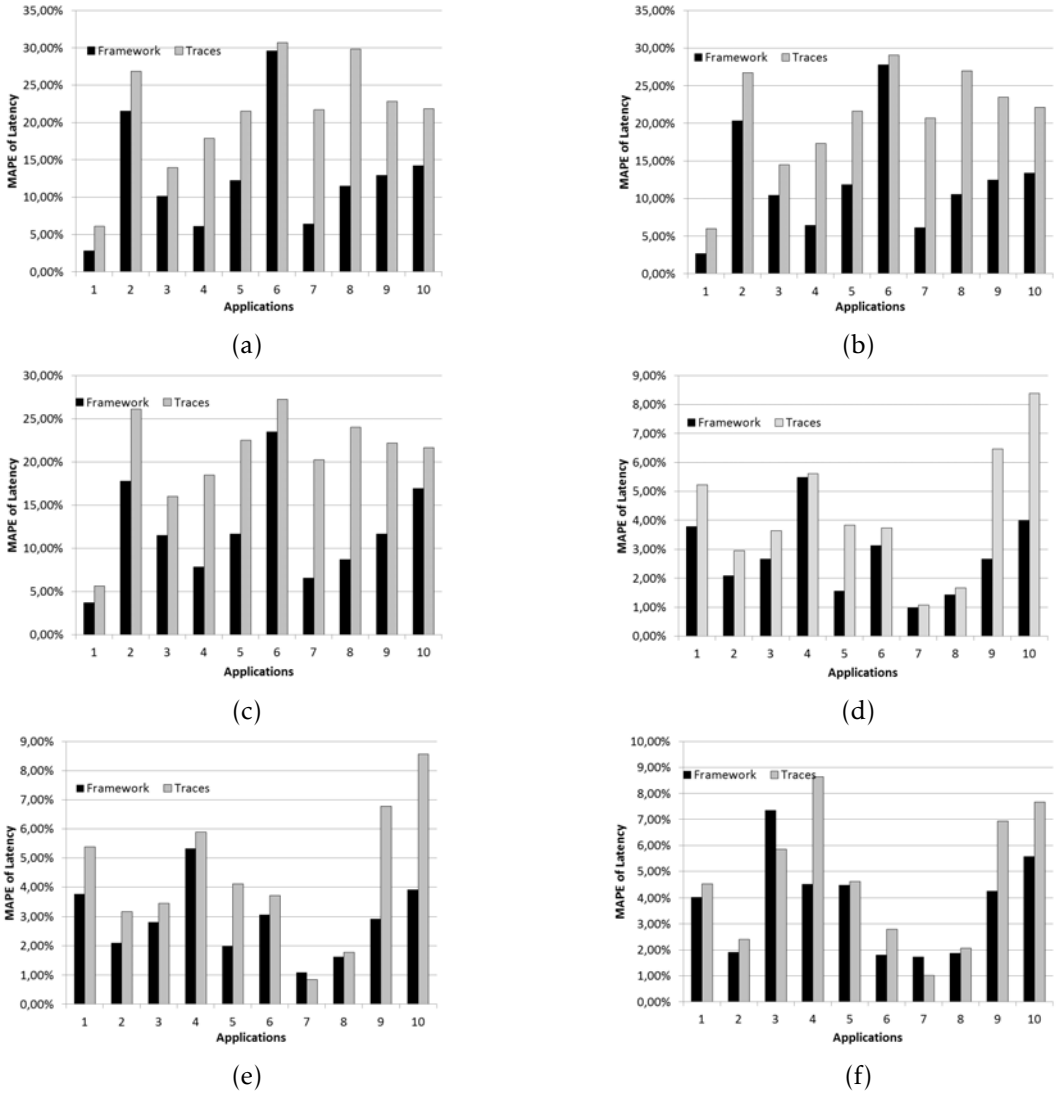


Figure 4.21: 3D Mesh NoC, XYZ routing scenarios. (a) 32-flits Model A. (b) 16-flits Model A. (c) 4-flits Model A. (d) 32-flits Model B. (e) 16-flits Model B. (f) 4-flits Model B.

emulation node occupies only 2.01% of the registers and 0.75 % of the LUTs of the overall platform.

In order to analyze the traffic fidelity, we need to understand what makes a traffic generator produces low fidelity traffic. In section 6.2, we show the effects of the lack of packets dependency information on trace-based traffic. The trace-based execution distorts the injection rate and effects of congestion. This effect is highlighted when the traces are executed in a NoC with different latency as depicted in Figure 4.1. Thus, it is important to comprehend what are the network parameters that influence the packets latency. We present an equation to calculate a packets latency in a not congested wormhole NoC: $latency = Time_{FLIT} * (Packet_{size} - 1) + (Time_{HEADER} + Time_{FLIT}) * N_{hops}$, in which $Time_{FLIT}$ is the number of clock cycles to send one flit, $Packet_{size}$ is the number of flits of a packet, $Time_{HEADER}$ is the number of clock cycles to handle a packet

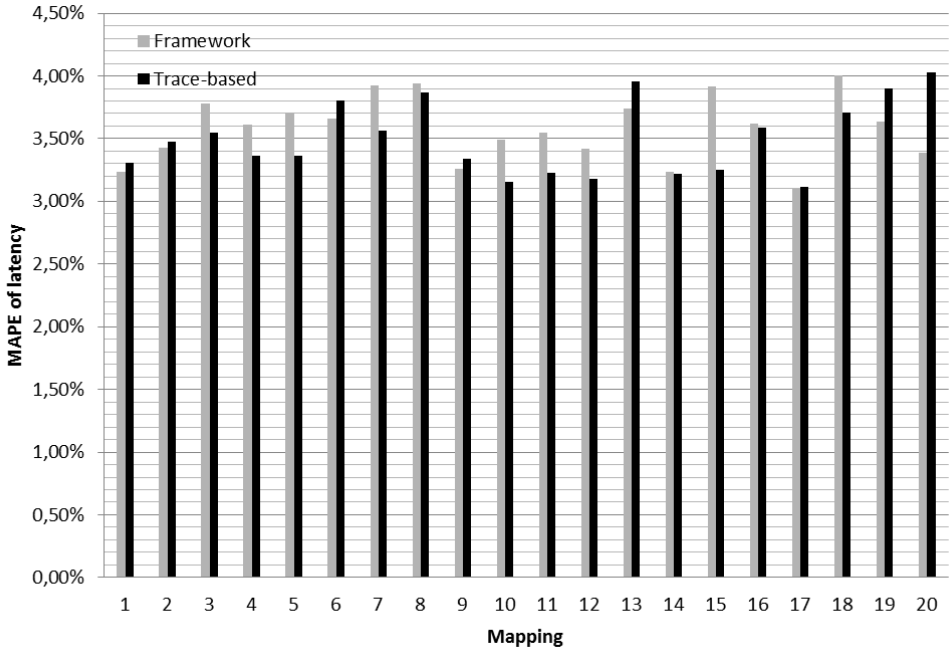


Figure 4.22: Random task mapping evaluation

header, and N_{hops} is the number of routers in the packet path including the source and destination. In our experiments, the first three parameters are unchanged. However the last parameter N_{hops} changes because it depends on the routing algorithm, topology and task placement.

In the experiments carried out, we executed trace-based and framework MoC in a broad range of NoCs. We played with different network parameters like: buffer size, routing algorithm, topology, and task placement. Henceforth, we analyze the impact of each one of these network parameters on the traffic fidelity.

The experiments highlight the influence of the buffer size onto the precision of the trace-based simulations. There is a clear precision difference between trace-based and framework for 2-flits and 4-flits buffer scenarios. However, this difference is very small for 32-flits and 16-flits buffer. A small buffer reduces network bandwidth, which makes the temporal information coded into the traces incorrect. The buffers are important to mitigate the effects of network congestion. However, small buffers are less effective to deal with those effects. Thus, the latency on small buffer networks is bigger. The packets dependency information allows the framework MoC to adapt the traffic generation to the networks constraints. It is interesting to point out that the framework keeps small variations on the errors metrics for different network configurations. The contrary is not true for trace-based simulation as stated for the 2-flits or 4-flits buffer scenarios.

The routing algorithms do not interfere with the simulations precision. Indeed, the simulations results are very similar for all routing algorithms. For the 2-flits buffer scenarios, it is possible to notice that non-minimal algorithms show a bigger loss of precision for the trace-based simulations. This fact is not verified for the framework simulations. A reason for the similarity of routing algorithms would be the tasks placement. The tasks placement is defined by the TGFF tool. Herein we analyze the

average distance between each pair of source and destination for each network. For the 6x6 network the average distance is 2.69 hops, for the 4x9 network it is 2.94 hops and for the 3x12 it is 4.02 hops. An optimal routing algorithm would take 69.4 clock cycles to send a 32-flits packet through a 2.69 hops path, 69.92 clock cycles to route the same packet through a 2.94 hops path, and it would take only 72.12 clock cycles on a 4.02 hops path. The small differences in the nodes distances do not highlight the routing algorithms properties.

The topology changes have also a small impact on the fidelity of the trace-based traffic. The reason is also the distance between the source and destination nodes. This is more evident for the 2D networks. However, the same argument cannot be used to the 3D networks. In those networks, the errors of trace-based traffic are very high. A 3D network is more connected. Thus, the distance between the nodes is smaller than in a 2D network. The differences in the distances highlight the inefficiency of trace-based execution to adapt to the network constraints. By the other side, those differences show that framework MoC adapt the traffic generation according to the constraints of the network used on the evaluation.

The experiments on random task mapping highlight again the issue of the distance between the source and destination nodes. The average distance between the nodes is 2.72 hops which is very close to the distance on the first task mapping which is 2.69 hops. As the nodes distance is close to the original network configuration, the trace-based execution has a performance similar to the framework MoC.

From those experiments, we find some interesting conclusions. At first, the fidelity of the trace-based traffic depends on the differences between the network used to collect the traces and the network under evaluation. In those experiments that the network under evaluation is similar to the network used to collect the traces, the trace-based has performance similar to the framework MoC. However, when the network under evaluation is very different from the original NoC, the framework MoC has a better performance. In order to decide which traffic generator should be used, it is necessary to have a very accurate model of both NoCs. However, this information is not always available. In most of the cases, there is no information about the NoC used to collect the traces. Thus, the framework MoC has an important advantage in relation to ordinary because this framework can be used in a broad range of networks configurations with a good precision. Furthermore, the framework MoC has very small data storage requirements what makes it an interesting option to generate traffic on FPGA-based NoC emulations.

4.7 Conclusions

This work presents a dependency-aware traces framework that synthesizes traffic generators for FPGA-based NoC emulation. This framework explores the packets dependencies in order to build a simple MoC, which can accurately replace applications nodes on future NoC performance evaluations. The experimental analysis highlights that this framework is more accurate than trace-based simulations for a broad variety of network configurations. Furthermore, the proposed framework uses only 3% of the data storage required by the traces.

This framework imposes that the packets dependencies are constant during all the

application execution. As a future work, we intend to evaluate this framework for multi-phases applications, in which the packets dependencies change during the execution. Our proposition is to analysis the traces in order to identify the traces regions in which the dependencies are constant. For each region is built a MoC. The complete execution of the MoCs would reproduce the application behavior.

Chapter 5: Towards the next-generation NoCs: A prospective analysis

The future belongs to those who prepare for it today.

Malcolm X

Contents

5.1 Overview **86**

5.2 A conceptual architecture of a NGNoC **88**

5.2.1 The layers description 90

5.2.2 Internetworking on NGNoCs 93

5.3 Congestion control on NGNoCs **94**

5.3.1 Analysis of congestion control requirements of the conceptual architecture 96

5.3.2 Perspectives on congestion control 97

5.3.3 FlexOE: congestion-aware routing protocol for NoCs 98

5.4 Fault tolerance on NGNoCs **102**

5.4.1 Analysis of fault tolerance requirements of the conceptual architecture 104

5.4.2 A fault tolerant NoC architecture 105

5.5 FPGA-based emulation on NGNoCs **108**

5.6 Conclusions **113**

The growing complexity of embedded systems demands the integration of a high number of IP cores interconnected by advanced on-chip networking mechanisms. These mechanisms must offer low latency and high throughput communication services for multiple data flows even in the presence of hardware faults or network congestion. Moreover, those data flows request different levels of quality of service according to the applications requirements. In the last decade, the research on NoCs has been focused on the development of advanced networking mechanisms like QoS [ACVCGM14, SMG14, W+09, AMP06], congestion control [OW14, CC+12, HH09, HM04], fault tolerance [FHLL14, KTMD14, ZP10, PLB+04], multicast [SGR13, ZWGLB14, EDLT10, LYJ06],

among others. Most of these propositions focus on standard 2D mesh topologies. However, the recent advances in the 3D-IC technology caused a paradigm shift on the on-chip interconnect scenario.

A fully connected 3D-NoC has higher throughput, smaller latency and power consumption than a 2D-NoC [FP07]. Those networks are comprised of multiple layers, each one possibly implementing a different semiconductor technology [MKH+12, GAW+13, PSCC+13, JHH13]. Those mixed technology chips must still offer high level networking services through different layers assuring the performance of intra and inter layer communication flows. Although the 3D technology is not fully mastered, it has strong potential to revolutionize the NoCs scenario introducing a Next-Generation of NoCs (NGNoCs). A NGNoC is a network divided on heterogenous layers of a 3D-IC which are connected by a few transversal links. The layers and the inter-layers connections may have specific physical constraints. Each layer implements an independent NoC with its own topology, number of nodes, routing scheme, among other parameters. Advanced networking mechanisms like QoS, congestion control, multicast communication, fault tolerance are implemented in each layer according to its communication requirements. It is possible that some layers implement completely different mechanisms or even that some of those mechanisms are not implemented at all.

This chapter presents a prospective analysis of NGNoCs architecture by the study of a conceptual embedded system implemented in a 3D-IC. We also present two propositions for those networks. The first proposition is a fault tolerant NoC architecture based upon bypass links. The second one is a congestion-aware routing algorithm. At last, an analysis of FPGA-based emulation tools for NGNoCs is presented.

5.1 Overview

A three-dimension integrated circuit (3D-IC) is composed of multiple layers of active devices stacked above each other and vertically interconnected using Through-Silicon Vias (TSV). The 3D-ICs have attracted a lot of attention in the recent years due to its clear advantages over traditional ICs. Stacking up circuit wafers enables an important reduction on the length and number of global interconnects [PF09]. It increases the circuit performance due to the shorter wire distance, as well as it decreases the power consumption. Moreover, the 3D enables the integration of Complementary Metal Oxide Semiconductor (CMOS) circuits with disparate technologies onto one chip. These mixed-technology chips may integrate logical circuits, memories, Radio Frequency (RF) and Microelectromechanical systems (MEMS) in only one small form factor package. Despite the significant advantages of 3D-ICs, important issues remain open such as thermal mitigation, crosstalk noise analysis, and manufacturing process mastering.

The 3D-IC paradigm combined with the flexibility of the NoCs technology enables the creation of new structures with significant performance improvements over more traditional solutions. Figure 5.1 depicts the architecture of a three-layer 3D Mesh NoC. A 3D-NoC reduces the number of hops to traverse a packet. It increases the overall throughput while decreasing the latency and the power consumption. For instance, in [PF07], it is shown that the porting of a 8x8 2D Mesh to a 4x4x4 3D Mesh leverages a reduction of 40% in the number of hops [FP07]. The same study reports that a 3D-NoC achieves performance improvement of 40% and a decrease of 62% in power consumption

when compared to a traditional 2D NoC for 128-nodes network.

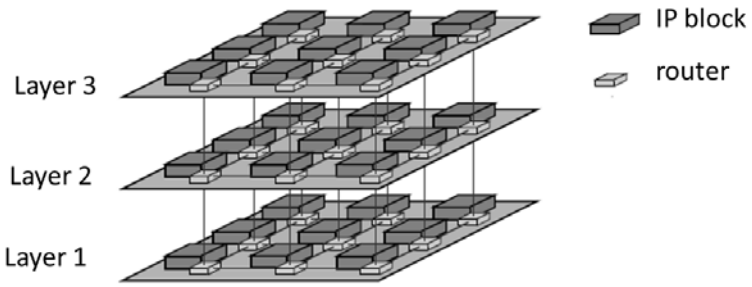


Figure 5.1: 3D Mesh NoC.

The vertical links represents one of the greatest challenges for the design of 3D-NoCs. The TSV is the most promising among the vertical interconnect technologies, because it has the potential do create dense vertical links. Although, the TSVs have a high defect rate [HRF+11] due to the difficult of align the stacked dies, as well as unpredictable phenomena mostly related to thermal compression process used in the wafer stacking process [LML+08]. Moreover, the TSV interconnect pitch uses more area than corresponding horizontal wires. It also adds several costly manufacturing steps, which reduces the production yield. Some propositions focus on the improvement of the reliability of TSVs [LML+08, HRF+11].

A 3D-NoC in which just some routers have vertical links is called vertically partially connected. These networks may reduce the number of TSVs and interconnect heterogeneous layer in a 3D chip. In [D+13], a deadlock-free routing algorithm to vertically partially connected 3D NoCs is proposed. Each layer implements an independent planar topology, which are connected by a few TSVs. A vertical-links assignment algorithm is proposed in [FSP14]. This algorithm improves the average network performance by determining the location and number of TSVs. Figure 5.2 depicts an example of a vertically partially connected 3D-NoC, in which a two layers 3D-NoC is connected by two TSVs. Each layer has a different topology.

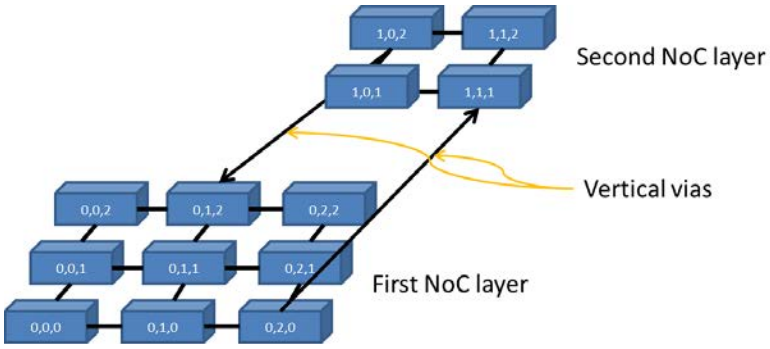


Figure 5.2: Vertically partially connected 3D-NoC.

In the last years, the NoC research community has dedicated efforts to investigate the implementation of advanced networking mechanisms in 3D-NoCs [CCW13, RLL+11, EDP13, HCGSML11, BJS14, SGP+13]. Most of these propositions focus on 3D sym-

metric NoCs, which are an evolution of the 2D mesh topology by simply adding two additional ports to connect with the adjacent layers. Some propositions uses vertical buses to implement the vertical interconnections [SGP+13, RLL+11], but they are a minor variation of the previous ones. The main characteristic of these propositions is the assumption that all layers implement networks with the same capabilities. There is an homogeneity of the network services. Although this assumption is not valid for chips composed of layers of different technology. In these chips the asymmetry of the layers imposes a tailored design for each layer. This has a huge impact on the design of the NoCs implemented in those chips. Those networks will reflect the asymmetry of the chip layers and they will implement different networking services. Although those services must be integrated in order to assure the connectivity of those on-chip networks.

We analyze the perspectives of a NoC implemented in a vertically partially connected 3D-NoC whose layers have disparate semiconductor technology and functions. Those NoCs are considered as the Next-Generation of NoCs (NGNoCs) which will be used on the future multi-core embedded systems. Those networks are intended to deploy several complex networking mechanisms such as Quality of Service (QoS), network management, fault-tolerance and congestion control. However those mechanisms are not homogeneous throughout the chip layers. Thus it is necessary an internetworking protocol to connect those different implementations. Each network implements a 2D deadlock-free routing algorithm and some of the network mechanisms cited previously. An internetworking protocol could integrate the different networks services enabling an efficient communication model on a NGNoC.

Besides the architecture of a NGNoC, it is necessary to prospect performance evaluation tools for those networks. The FPGA-based NoC emulation platforms will play a major role on the performance evaluation of NGNoCs. It is needed advanced strategies to generate high fidelity traffic for NGNoCs emulated on FPGAs, as well as to manage the emulation platform using standard protocols. Furthermore, it is important to investigate new approaches to improve the architecture of those platforms in order to handle the great data volume produced by the circuits probing a NoC performance.

In this chapter, we analyze a conceptual architecture of a NGNoC in order to better understand the challenges involved in the design of those networks. The requirements of the intra and inter communication flows are analyzed, as well as the specification of an internetworking protocol for NGNoCs. From this analysis, we present some propositions for NGNoCs. The first proposition is a fault tolerant NoC architecture based upon bypass links. The second one is a congestion-aware routing algorithm. Both propositions are targeted on the architecture of the horizontal layers of a NGNoC. Furthermore, we also present an analysis of FPGA-based emulation tools for NGNoCs.

5.2 A conceptual architecture of a NGNoC

The future many-core embedded systems will run complex applications such as high definition video encoding and decoding, real-time patterns recognition for imaging applications, among others. Usually these applications will run simultaneously, they will share the processing cores as well as the on-chip interconnect. The NGNoCs are the interconnect technology of the future embedded systems. These networks will

be deployed in 3D-ICs in order to create a heterogeneous chip. In this architecture, each layer will implement a specific function like processing, data storage, high-speed I/O, among others. That heterogeneous chip will be manufactured using chip stacking technology similar to those used in [MKH+12, GAW+13, PSCC+13, JHH13].

In the NGNoCs-based embedded systems, heterogeneous processing cores will be commonplace. Different processors architectures will work together in order to execute advanced applications. Cores with different processing power and energy consumption will be deployed. It enables flexible power consumption management by selecting the active cores according to the system state. Hardware reconfigurability will also be an important feature. Programmable logic devices will be placed to leverage further processing power and flexibility.

The number of cores on a NGNoC-based embedded system will be counted in tens or even a few hundreds deployed in some layers. The inter-layer communication will be a great challenge. Even with the on-going improvements of 3D-IC manufacturing technology, it is plausible that the TSVs will remain a scarce resource. Vertically partially connected NoCs will be the standard architecture. Besides that, the performance asymmetry between vertical and horizontal links will persist. The management of the congestion on the vertical links and the establishment of inter-layers QoS will be key research challenges. Furthermore, the number of stacked layers on a 3D-IC will be limited mostly because of heat dissipation problems. It is coherent to consider a NGNoC deployed in three layers connected by half-dozen vertical links between layers.

In this section, we explore a conceptual architecture of a NGNoC-based 3D-IC for embedded systems applications. This conceptual architecture serves as a study model to explore the high-level communication requirements of a NGNoC. We arbitrarily focus on a 97 cores system deployed on three-layers. The number of cores is not very elevated, but it is enough to determine the communication flows intra- and inter-layers. Moreover, this model includes sufficient details to describe the main functions of a many-core embedded system. Evidently several functional features descriptions are not discussed here because they do not influent the networking architecture.

This system contains three-layers implementing different functions as depicted in Figure 8. The first layer is dedicated to the processing units. Each Processing Element (PE) has its own memory, thus the data sharing is carried out through message passing. The PEs have heterogeneous architectures. They may be general purpose RISC cores, Graphical Processing Units (GPUs) and other IPs dedicated to specific computation like cryptography, among others. Programmable logic devices areas are also implemented in this layer. They enable real-time hardware reconfiguration and they can be used to implement dedicated processing blocks.

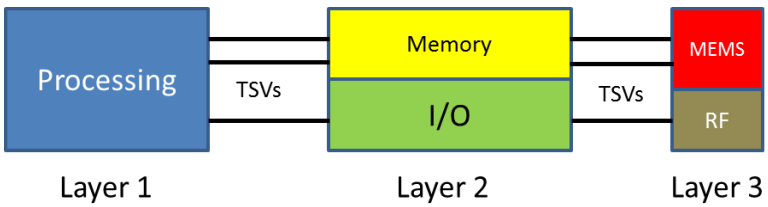


Figure 5.3: Conceptual NGNoC-based 3D-IC architecture.

The second layer has two areas with different purposes. The first area implements

fast storage memory blocks. These blocks are used to storage application data/code. They are also used as a shared secondary memory area for the PEs and other cores. They are connected through a bus to an external memory controller. The second area implements high speed I/O controllers and audio/video processors. They connect the chip to a set of high-end devices like high speed buses, touch screens, high resolution cameras, and networking interfaces. The third and last layer also has two distinct areas. The first one implements a set of MEMS like accelerometers, gyroscopes, etc. The second area implements RF interfaces for short distance communication like Bluetooth or RFID.

Each layer implements a specific NoC architecture with its own features such as: routing algorithm, buffers size, flow control, QoS, among others. A set of special routers connected to the vertical links are responsible to carry the inter-layers traffic. These routers are called elevators, this terminology is suggested in [FSP14]. In order to transmit a packet to another layer, the inter-layer packet must be encapsulated into a standard intra-layer packet destined to the closest elevator router, which is responsible to transmit the packet to the next layer. When the packet arrives on the destination layer, it is unencapsulated and it is transmitted as a regular intra-layer packet until arrive in its destination node. This approach works efficiently with any 2D deadlock free routing algorithm used for intra-layer routing.

5.2.1 The layers description

Henceforth we present the structure of each layer as well as the on-chip networking services requested by each one. The processing layer has 56 cores in which 32 general purpose CPUs, 10 GPUs, 8 FPGAs and 2 cryptographic engines as depicted in Figure 5.4. A 8x7 mesh NoC interconnect these cores. The mix of heterogenous cores enables the implementation of advanced applications. Furthermore, the FPGAs cores add greater flexibility because of their real-time reconfiguration features. Pre-built logical blocks can be programmed onto the FPGAs in order to carry out specific computation tasks. This layer is composed of two symmetrical processing blocks areas, each one containing 28 cores in a 4x7 mesh structure.

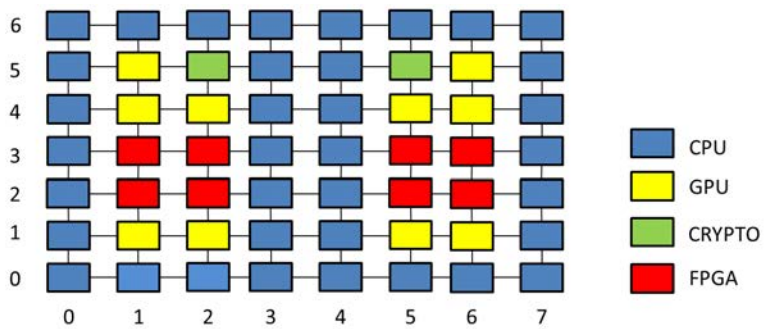


Figure 5.4: Layer 1 architecture.

This network has six TSVs connecting it to the upper layer. The TSVs are unidirectional. There are three to the upstream links and three to the downstream links. The TSVs are connected onto elevator routers which have five bi-directional ports and one unidirectional port connected on a TSV. These routers are spatially distributed in order

to improve the bandwidth of the PEs, four of them are connected onto routers connected to CPUs and two onto routers connected to GPUs. The exact localisation of these elevator routers is not relevant for our discussion. The Table 5.1 describes the connection of the TSVs between this layer and the next one. Although, these vertical links are connected on specific routers, they carry packets of any node from both layers.

TSV	Layer 1	Layer 2	Direction
0	CPU	RAM	$L1 \rightarrow L2$
1	CPU	RAM	$L2 \rightarrow L1$
2	CPU	I/O	$L1 \rightarrow L2$
3	CPU	I/O	$L2 \rightarrow L1$
4	GPU	RAM	$L1 \rightarrow L2$
5	GPU	RAM	$L2 \rightarrow L1$

Table 5.1: TSVs interconnect between layer 1 and layer 2

The CPUs are placed surrounding the other PEs. This arrangement improves the performance because the CPUs use the other PEs as secondary processing units. The PEs placement combined with the TSVs organization allows a perspective of the traffic flows between the cores. The Table 5.2 presents the main traffic flows. The first traffic flow is between the PEs and the memory blocks located on layer 2. Each PE has its own local memory, but they use the external memory blocks as secondary data storage devices. The second traffic flow carries the I/O data to the CPUs cores. The third traffic flow is the communication between the FPGAs and the external memory blocks located on layer 2. The memory blocks hold the bit-stream configuration of the FPGAs. During the FPGA reconfiguration process, a communication flow is carried out to bring the bit-stream from the memory blocks to the FPGA. The next traffic flow is between the CPUs and the MEMS and RF components located on layer 3. This traffic flow is expected to be less intense than the others. The fifth traffic flow models the traffic between the CPUs and the other PEs. The FPGAs, GPUs and cryptographic engines are treated as co-processors. There is a request-response communication model in which the CPU uses the services offered by the other PEs. The last traffic flow is intra-layer between the CPUs cores. This traffic flow carries application data as well as application code during tasks migrations between cores. The two last communication flows may use multicast communication services.

	Traffic Flows	Direction
0	$PEs \longleftrightarrow Memory$	inter-layer
1	$CPUs \longleftrightarrow I/O$	inter-layer
2	$Memory \longleftrightarrow FPGA$	inter-layer
3	$CPUs \longleftrightarrow MEMS/RF$	inter-layer
4	$CPUs \longleftrightarrow otherPEs$	intra-layer
5	$CPUs \longleftrightarrow CPUs$	intra-layer

Table 5.2: Layer 1 traffic flows description

The second layer has 25 cores in which 3 audio/video processors, 5 RAM memory blocks, 10 Flash memory blocks and 7 I/O controllers. A 5x5 mesh NoC interconnects

these cores as depicted in Figure 5.5. The audio/video processors are placed onto this layer because they are I/O bounded cores as well as they also carry intense data traffic within the memory blocks. This layer offers data storage and I/O services to the processing layer. The memory blocks implements fast and dense data storage technology in the order of some gigabits per block. The I/O controllers interface with several high speed devices as well as simpler peripherals.

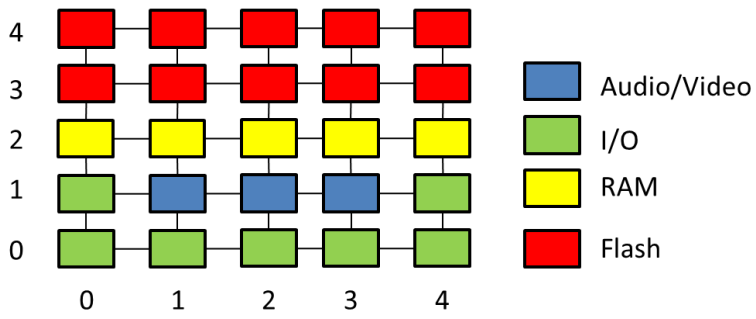


Figure 5.5: Layer 2 architecture.

This layer also has four TSVs connecting it to the upper layer in addition to the vertical links connecting it to the layer 1. The TSVs are unidirectional. There are two to the upstream links and two to the downstream links. The TSVs are distributed in order to improve the bandwidth. The Table 5.3 describes the connection of the TSVs between this layer and the next one. The localization of the TSVs is not detailed, because for the purposes of this model only the amount of TSVs is enough to determine the NGNoC communication flows.

TSV	Layer 2	Layer 3	Direction
0	RAM	MEMS	$L2 \rightarrow L3$
1	RAM	MEMS	$L3 \rightarrow L2$
2	Flash	RF	$L2 \rightarrow L3$
3	Flash	RF	$L3 \rightarrow L2$

Table 5.3: TSVs interconnect between layer 2 and layer 3

The Layer 2 communication flows are partially detailed in Table 5.2. However, there are other communication flows carried out by this layer that are worth to highlight. These communication flows are described in Table 5.4. The Audio/Video processors carry two intra-layer traffic flows in direction of the memory blocks and I/O controllers. The I/O controllers also transfer data to and from the memory blocks. At last, the layer 3 components also may use the memory blocks using the vertical links. This is the only flow between these two layers.

The third layer has 16 cores in which 8 MEMS blocks and 8 RF blocks. A 4x4 mesh NoC interconnects these cores. This is the smallest network and it also carries the simplest communication flows. The implementation technology of this layer is very distinct from the other ones. It is a mix of analog, digital and RF circuits. This layer communication flows are described previously. It does not have intra-layer flows, it basically offers services to the bottom layers.

	Traffic Flows	Direction
0	$Audio/VideoPEs \longleftrightarrow Memory$	intra-layer
1	$Audio/VideoPEs \longleftrightarrow I/O$	intra-layer
2	$I/O \longleftrightarrow Memory$	intra-layer
3	$MEMS/RF \longleftrightarrow Memory$	inter-layer

Table 5.4: Layer 2 traffic flows description

From this conceptual architecture of a NGNoC-based embedded system, we analyze the requirements of a NGNoC in three key aspects: congestion control, fault tolerance, and FPGA-based emulation. The next subsections present a prospective discussion on these subjects as well as they present some evaluated propositions that fit the requirements of this conceptual architecture. It is important to notice that the precise description of the traffic requirements carried out by this NGNoC is not relevant for this analysis because this is a general purpose embedded system platform. It means that this platform may be used to run different applications which have disparate traffic requirements. Nevertheless, the eventual differences on the traffic loads do not nullify the communication flows previously described because they depend only on the placement of the application nodes and the number of vertical links.

5.2.2 Internetworking on NGNoCs

A NGNoC is by definition an on-chip internet because it is composed of several independent NoCs arbitrarily connected by some elevator routers. An internetwork protocol is necessary to integrate those NoCs services. In this subsection, we present a brief specification of this protocol. As described before, each layer of a NGNoC has its own network with specific capabilities. Those networks may or may not implement all the advanced networking mechanisms cited in this chapter. They can even have completely different implementations of the same mechanism. Although, those networks must handle correctly the communication flows from the other layers.

The communication flows of a NGNoC are divided into two categories: intra-layer flows and inter-layer flows. Those flows are treated differently according to the capacities of each layer. In order to enable the integration of the different networks we propose a standard packet frame to encapsulate both communication flows. Figure 5.6 depicts the structure of that packet frame.

This frame is structured for 32-bits flits. The first field is L which indicates if a packet is destined to the same layer (L=0) or to another layer (L=1). The packet structure is different for each type of packet. The inter-layer packets have an additional field to indicate the address of the elevator router. All the other fields are the same for both types of packets. The field M determines if a packet has multi-destinations. A multi-destination packet (M=1) uses a multicast routing strategy and it has multiple copies of the destination address field. In the proposed architecture, only intra-layer packets can be multi-destination. The ToS field specifies the type of service of a packet. This field is used by the QoS mechanism to determine the level of the communication services of a packet. The TL field indicated the total length of a packet in flits. The source and destination address specify a 16-bit address which is valid for all the networks of a NGNoC. The identification field specifies a unique number for each packet between two

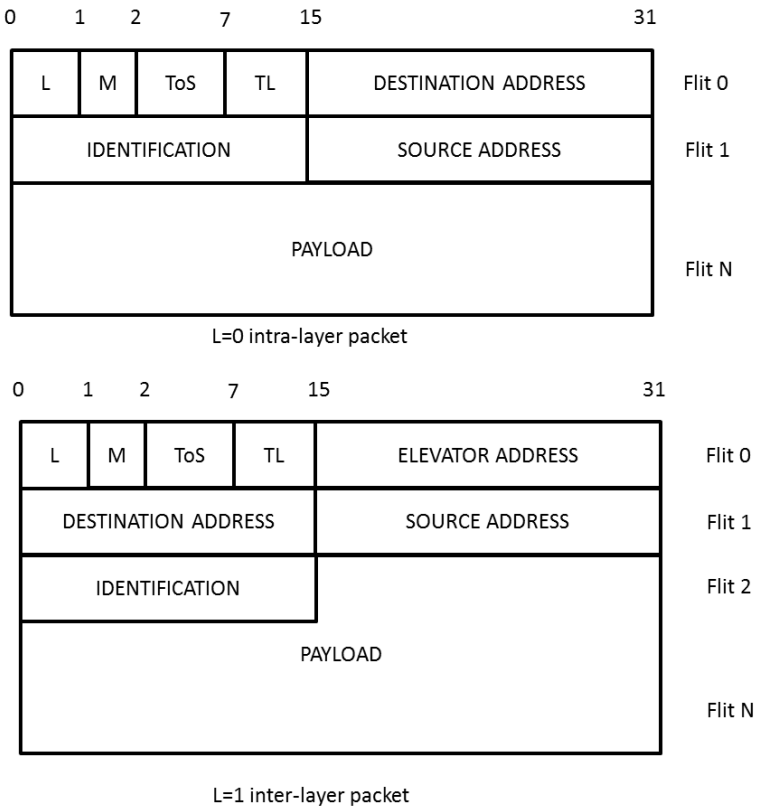


Figure 5.6: NGNoC packet frame.

nodes. This number may be used to put the received packets in order. This is especially useful for networks which use adaptive routing algorithms.

A router of a NGNoC must be capable to handle packets in this format. In some cases, the router may ignore some fields because of its limitations. For instance, the ToS field specifies 128 different levels of services in order to cover the QoS requirements from several NoCs. However, some networks may implement simple QoS strategies dividing the packets in only two categories: best effort (BE) and guaranteed throughput (GT). A packet traversing this network must be handled according to the QoS available, even if its ToS field specifies a better service level. In other cases, a network may not have implemented a QoS mechanism. In this case, the ToS field is completely ignored.

5.3 Congestion control on NGNoCs

The network congestion is an important issue, as it degrades a NoC performance at run-time. A congested NoC suffers from degraded performance caused by the presence of too many packets in a determined network area. The routers' vertical links of a NGNoC are inherently susceptible to be a congested area, because they aggregate the traffic to the neighbours layers. Evidently the other routers may also be congested depending on the traffic distribution. We divide the execution of a congestion control policy in two tasks: congestion detection and congestion-aware routing.

The congestion detection is a non-stop monitoring task usually carried out by a special circuit on the routers. A router parameter is used to measure the intensity of the congestion. Many proposals use different congestion metrics like as: number of used buffer slots, number of free buffers slots, number of waiting input ports, buffers occupation ratio, among others. In bufferless NoCs, we use the time average of link utilisation instead of the buffers fullness to estimate the network congestion [OW14]. The Table 5.5 presents a list of congestion-control proposals and its congestion detection mechanisms as well as its adaptive routing strategies.

Proposal	Congestion Detection	Adaptive Routing
[HM04]	Used slots in adjacent routers input buffers	Odd-Even
[VMPL10]	Used slots in adjacent routers input buffers	DyXY
[KA11]	Used slots in adjacent routers input buffers	Odd-Even
[HH09]	Free buffers slots in adjacent routers and its neighbors	Odd-Even
[WHB10]	Number of waiting input ports on adjacent routers	XY Adaptive
[TMY11]	Number of waiting input ports on adjacent routers	West-First
[MKM10]	Adjacent routers Buffers occupation ration	Odd-Even
[LZJ06]	Used slots in adjacent routers input buffers	XY Adaptive
[CHLW14]	Adjacent routers Buffers occupation ration	Odd-Even
[OW14]	Links utilisation ratio	Odd-Even

Table 5.5: Comparative of congestion-control mechanisms for NoCs

The congestion metrics can be computed in a distributed or centralized way. In distributed congestion metric computing, each router gathers congestion information from adjacent routers through dedicated control links. Usually only adjacent routers share information. It is possible to retrieve congestion information from routers more distant, but it demands longer and more complex wiring or the utilisation of virtual links. Figure 5.7 depicts the interconnections between a group of routers that share congestion information. In a centralized congestion metric computing, a central router is responsible to compute the congestion metric of a cluster of routers. Afterwards, this central router updates the congestion metric of each member of its cluster. This approach requires less wiring than the previous one.

Congestion-aware routing is a category of adaptive routing that uses congestion information to dynamically determine the path for a packet. The Table 5.5 presents the main adaptive routing algorithms used for congestion-aware routing. These routing algorithms are more complex than simple deterministic ones. However, their adaptiveness relieves the congestion effects. The majority of the routing algorithms presented in

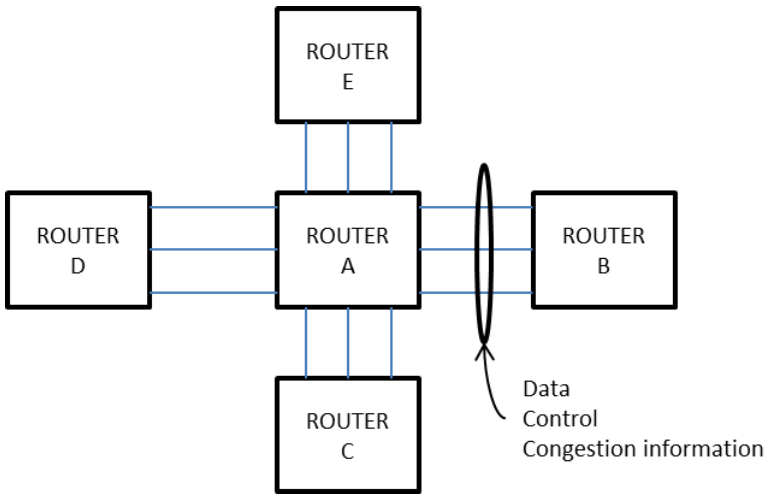


Figure 5.7: Adjacent routers interconnect to share congestion information.

the proposals are semi-adaptive, i. e., they limit the number of paths between a source and destination in order to achieve deadlock freedom.

5.3.1 Analysis of congestion control requirements of the conceptual architecture

In our conceptual NGNoC architecture, the layer 1 has the greatest number of routers and it also executes most of the communication flows. The implementation of congestion-control mechanisms may help to separate the intra- and inter-layer traffic. Elevator routers are more susceptible to be congested because they concentrate the inter-layer traffic. A congestion-aware routing algorithm automatically diverts the intra-layer traffic from the elevator routers. We expect an intense traffic between the PEs which are in a short distance because the applications may exploit PEs clustering to run computing intensive applications. The medium and long distance traffic will be mostly packets destined to the elevator routers.

The layer 2 has a medium number of routers, but it concentrates the greatest number of TSVs. This layer has an intense traffic because we expect that the memory blocks are used by components in the three layers. Once again the adoption of a congestion control mechanism helps to deviate the traffic from the congested zones, which are likely to be the zones surrounding the elevator routers. The last layer is the simplest one, it has only 16 nodes. Most of the traffic flows of this layer are directed to the other layers. The adoption of a congestion control mechanism is not mandatory. A simple networking scheme meets the softer communication requirements of this layer.

A standard congestion control mechanism mitigates the congestion around the elevator routers by changing the path of the intra-layer traffic flows. However, it can not deal with the TSVs congestion itself. As stated before, the TSVs are very scarce resources. There are few vertical links per layer. Whenever a node sends an inter-layer packet, this packet must be redirected to the closest elevators router. This simple scheme requires very low resources to be implemented, but it does not handle the TSVs congestion.

Consider the example depicted in Figure 9. The black squares depict the elevator

routers. The gray squares are routers sending inter-layer packets. The arrows represent the traffic flows. In this example, one elevator router is carrying traffic from six routers, while the other one is carrying traffic of just one router. This load inequality causes network congestion around the first elevator router. In order to solve this problem, it is necessary to determine the elevator router according to its congestion information and position. The dynamic assignment of the elevator router is a challenge because all the other routers must have knowledge of each elevator state. This is even more challenging for big networks with a few TSVs as the case of the layer 1.

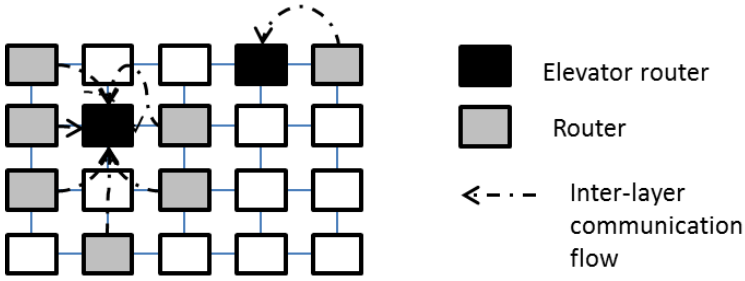


Figure 5.8: Example of inefficient TSVs utilisation.

One could propose to add extra wires between the routers to share the elevators congestion information. Although apparently simple, this solution incurs additional complexity due to the quantity and length of this wiring. A better solution is to explore the networking infrastructure available to share important network information as the elevators congestion status.

5.3.2 Perspectives on congestion control

We briefly introduce a simple piggybacking mechanism to share congestion information from the elevator routers. This mechanism is not fully explored in the context of this thesis, but it is targeted as one of our future research works. The piggybacking is a data communication technique in which a receiver node piggybacks the acknowledge information onto outgoing data packets destined for the transmitting node. It is used in several MAC protocols. We propose to piggyback the congestion information onto the data packets passing through the elevator routers or its adjacent routers. Whenever the packet containing the congestion information is handled by a router, this router gets the congestion information and it updates a data table, in which it is hold the status of all elevator routers. When an inter-layer packet is transmitted, that data table is consulted in order to determine which elevator will take care of this packet.

The advantage of this mechanism is the exploration of the usual data traffic to propagate congestion control information. The congestion information can be coded in just one flit. The drawback is that it is not effective in case of low traffic load or in scenarios where the traffic passing through the elevator routers do not traverse all the network routers. Because of this, the elevators data table in each router must be reset periodically. Moreover, whenever the table is outdated the closest elevator router must be chosen.

Besides the vertical link congestion control, it is necessary to deal with the intra-layer congestion. We propose to implement a congestion-aware routing algorithm FlexOE,

which is based on a simple and flexible scheme of prioritized sets of rules, as well as the network load balance. These sets of rules are based on the Odd-Even turn model, minimal paths checking, congestion information from adjacent routers and availability of output path. The load balance is achieved because the proposed algorithm does not choose always the lowest latency path. The output path is chosen through the verification of the sets of rules. It is important to notice that the paths verification sequence is updated every clock cycle of the router hardware, in such a way the traffic is distributed for many paths. The originality of the proposed algorithm is to cope with the congestion on NoCs by a prioritized scheme of set of rules, which allows combining different types of information from the router and the network.

5.3.3 FlexOE: congestion-aware routing protocol for NoCs

The FlexOE is an adaptive routing algorithm that selects the output path for a given packet based on a simple prioritized scheme of sets of rules. These sets of rules are based on descending priority criteria. The first criterion must be a multi-path routing algorithm. The other can be some network or router metrics that can be used to select one of the output paths pre-selected by the first criterion. The sets of rules are analyzed sequentially. The algorithm starts checking each output path against the most astringent set of rules. If no output path attends this rules set, the next rules set is checked, and so on. The output paths are checked in a pre-determined order. The order is cyclically changed for each given clock period.

For this work, we have selected four criteria to implement this congestion-aware routing. The first criterion checks if an output path respects the Odd-Even turn model restrictions. The Odd-Even model is designed for deadlock-free and multi-path routing in 2D meshes. In this model, columns in a 2D mesh are alternatively designated as odd or even. There are two main rules to ensure deadlock-free routing in the Odd-Even turn model: 1) a packet is prohibited to make an EAST-NORTH or EAST-SOUTH turn at any node located in an even column; and 2) a packet is prohibited to take a NORTH-WEST or SOUTH-WEST turn at any node located in an odd column.

The second criterion verifies if an output path is a minimal one, in other words, an output path which makes the packet closer to the destination. The third criterion checks if an output path has low congestion. The congestion level is assessed by the number of waiting input ports and the number of busy output ports. The fourth criterion verifies if an output path is free of use. The router control circuit maintains information about the state of the output ports.

Figure 5.9 illustrates how those sets of rules are organized. The first set of rules is composed of all four rules. In its turn, the second one is based on the three most priority rules (the least important rule is taken out). The third one is based on the second set of rules without the least important rule of the second set, and so on.

Figure 5.10 depicts how the checking sequence changes over time. For instance, on the first clock cycle the checking sequence is: SOUTH, NORTH, WEST, and EAST. The sequence is updated every clock cycle of the router hardware. In this way, the FlexOE is developed to spread the network load out through a subset of the available output paths chosen by a multi-path routing algorithm. This sub-set is defined by the sequential verification of the sets of rules.

This algorithm is implemented by a dedicated block in each router, which is respons-

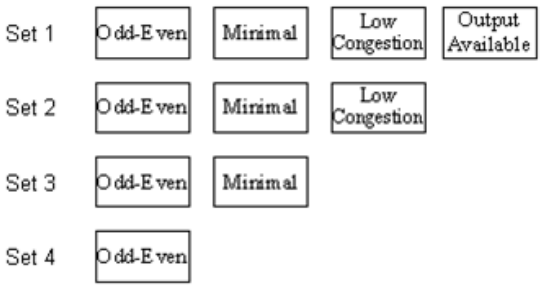


Figure 5.9: Organization of the sets of rules.

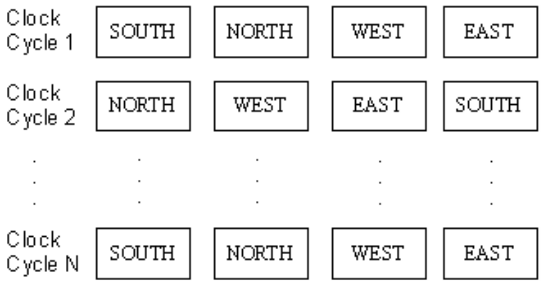


Figure 5.10: Output paths sequence updates.

ible for checking each rule, as well as the sets of rules. This block gathers congestion information from adjacent routers through dedicated links. The congestion information is updated every clock cycle of the router hardware. Each rule is evaluated for each output path by a specific circuit in the dedicated block. When a packet is chosen to be routed, the destination information is used to verify the rules Odd-Even and Minimal. At the same time, the Low Congestion rule is verified by comparing the congestion information of the adjacent routers with a congestion threshold. The Output Available rule is the simplest one, because the router maintains control of the state of each output port.

The dedicated block updates a rules table. Each output port has an index, which is updated every clock cycle of the router hardware. This index indicates the priority of an output port. For instance, the port with index equal to 0 has the highest priority and the port with index equal to 3 has the lowest priority. The rules table indicates what rules are true for each output path. After the table's update, the rule checker analyses in a parallel way the table to identify the output path. The analysis of the table takes only one clock cycle of the router hardware. The output port selected is the one that checked the highest set of rule. If there are more than one output ports in this situation, the one with the highest priority is chosen.

This dedicated block structure is depicted in Figure 5.11. The 1s in the rules table indicates that one rule is asserted as true; otherwise its value is 0. The output ports NORTH and WEST have checked the highest set of rules, in this case the set of rules 2 (Odd-Even, Minimal and Low Congestion). However, the output port WEST has highest priority (index equal to 2), while the output port NORTH has the lowest priority (index equal to 3). Thus, the WEST port is selected.

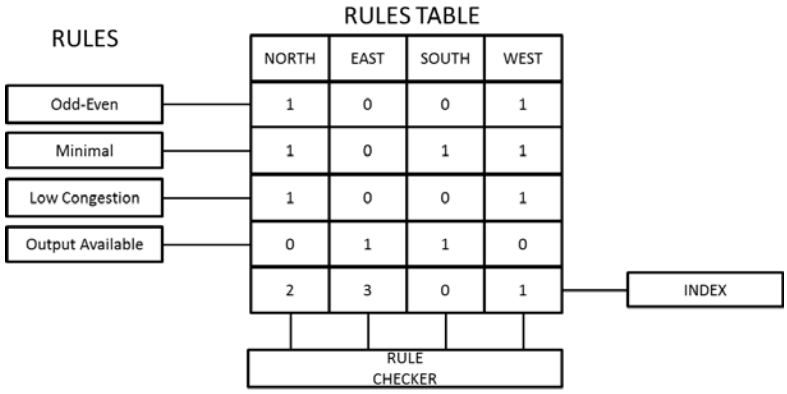


Figure 5.11: Set of rules evaluation.

In order to evaluate this proposition, a set of experiments is carried out by integrating the FlexOE on a 5X5 2D mesh Hermes NoC. The classic routing algorithms: XY, DyXY and DyAD-OE are implemented, in order to carry out the performance analysis. Figure 5.12 shows the relation between injected charge and the average total latency for a test scenario, in which each router transmits a set of 30 packets. The average total latency to transmit a set of packets is measured with the injected charge from 10% to 100%. The total latency is the amount of clock cycles necessary to transmit all the flits of a set of packets. It is calculated by subtracting the arrival time of the last flit of the last packet by the departure time of the first flit of the first packet. The transpose traffic model is applied on this test scenario.

The FlexOE has the lowest values of the average total latency for injected charges between 30% and 70%. For the higher values of injected charge, the FlexOE and DyXY reach the saturation point because the average total latency does not reduce while the injected charge increases.

The FlexOE is capable of accelerating the transmission of a set of packets, even if the average latency of each packet is not lower than the other propositions. The FlexOE spreads out the injected packets on the network for multiple paths. Accelerating the transmission of a set of packets can significantly reduce the total execution time of an application.

Figure 5.13 shows the relation between the total average latency and the size of the packets in flits. The FlexOE presents the lowest levels of average total latency; its values are about 10% lesser than the other algorithms for packets size between 32 and 256 flits. This demonstrates the property of FlexOE of reducing the time to transmit a set of packets, instead of reducing the individual latency of each packet. This feature can be useful on applications that need to receive a set of packet before starting processing it.

Figure 5.14 presents the number of registers and LUTs for different sizes of NoCs based on the algorithms presented. The cost of implantation of FlexOE is compatible with the other algorithms.

The FlexOE is a flexible congestion control mechanism which fits the requirements of the layer 1 and 2 of the proposed conceptual NGNoC architecture. Furthermore, this protocol rules scheme can be extended to cover other network parameters like fault-tolerance or QoS.

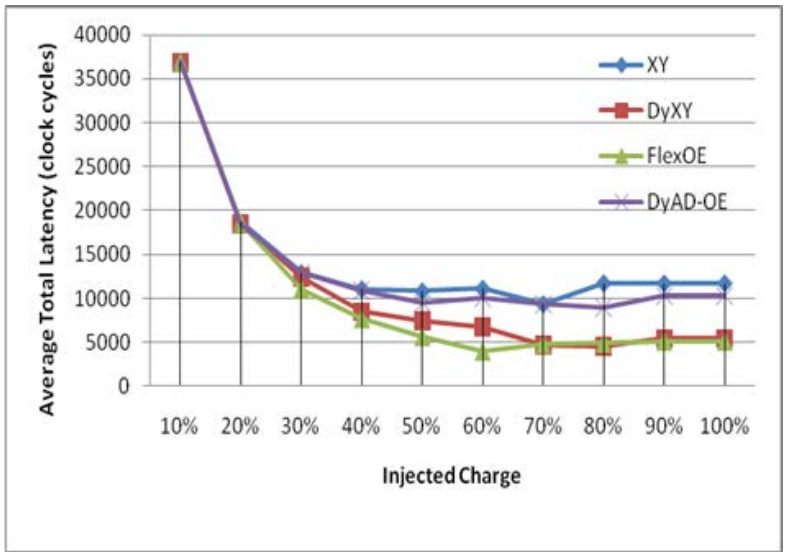


Figure 5.12: Evaluation under transpose traffic.

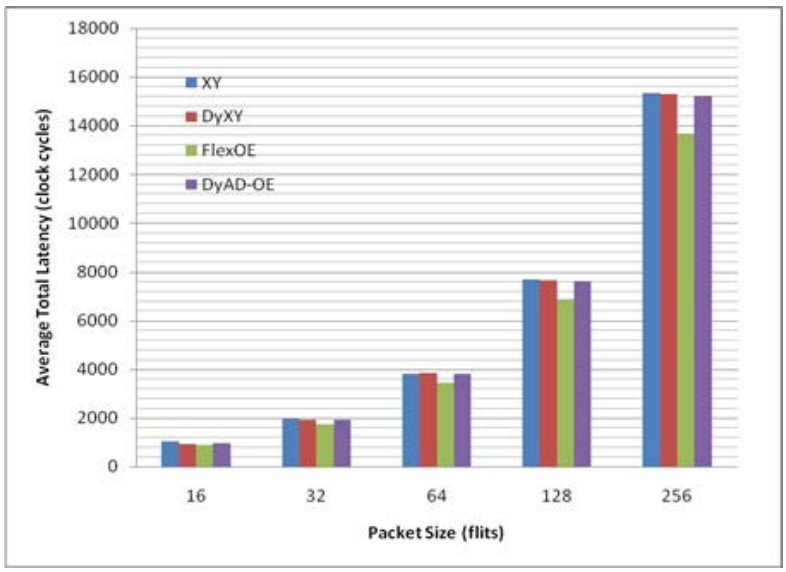
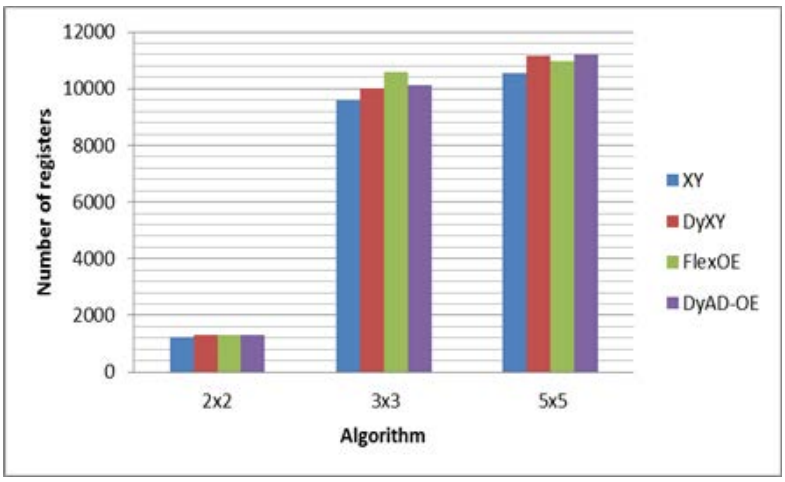
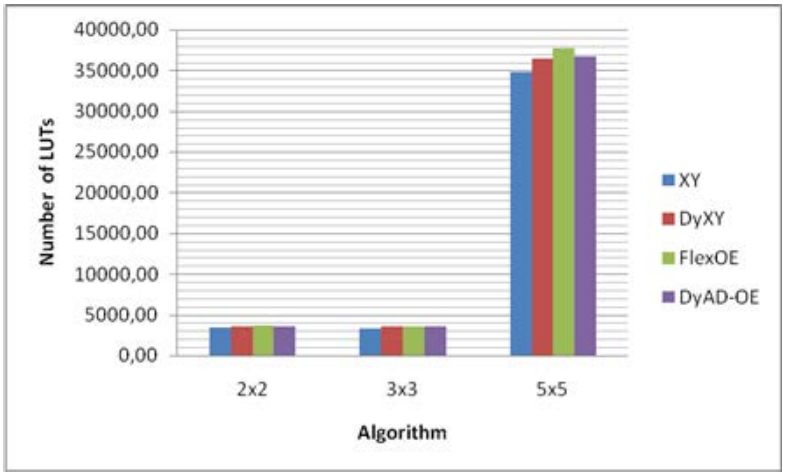


Figure 5.13: Average total latency for uniform transpose (30 packets, 50% injected charge).



(a)



(b)

Figure 5.14: FPGA resources analysis for a Virtex 6. (a) number of registers for implementing a NoC. (b) number of LUTs for implementing a NoC.

5.4 Fault tolerance on NGNoCs

A NGNoC is subject to diverse faults arising from different sources. For example, a fault in only one transistor of a NoC may cause grave failure in one chip [LKRD+10]. This is even more dangerous for a multi-layer NoC architecture as those conceived to the NGNoCs. The vertical links are the most sensitive components in a NGNoC due to the inherent manufacturing problems of 3D-ICs. The TSV is the most promising among the vertical interconnects. Although, the TSVs have a high defect rate [HRF+11] due to the difficult of align the stacked dies, as well as unpredictable phenomena mostly related to thermal compression process used in the wafer stacking process [LML+08].

Some propositions focus on the improvement of the reliability of TSVs through the exploration of redundant vertical links. In [LML+08], TSVs-based multi-bit links are used to implement a defect-tolerance technique, which significantly improve the yield. In [HRF+11], a fault-tolerant vertical link design is proposed, which enabled the

implementation of 3D chips without the need of adding redundant TSVs. Despite that, the other NGNoC components also deserve attention. We analyze the main propositions of fault tolerance mechanisms for NoCs.

In general, the proposed fault tolerance mechanisms for NoCs can be divided into two groups. The first group deals with this problem by applying routing algorithms capable of computing alternative routes as a method of avoiding routing through faulty routers. The second group uses architectural solutions in order to deal with the failures. In relation to the first group, we emphasize the stochastic routing algorithms. These algorithms provide fault tolerance by sending several copies of a packet by different routes, which increase the probability of reception of these packets. However, these algorithms cause an increase in energy consumption and network congestion. Several algorithms are based on this strategy [ZPG07, PLB+04].

Some fault tolerance routing algorithms are based upon the turn model [GN92], which are used to construct partially adaptive routing algorithms, free of deadlocks, without the need of additional virtual channels. This model is based upon the analysis of direction changes, in which the packets can take in the network and in the cycles that these are able to form. The routing algorithms West-first, North-last and Negative first, are based upon this model. In [Chi00], a proposal is made for the "Odd-Even Turn Model" that compared to other algorithms based upon the turn model provides a degree of superior adaptability. The turn models can be applied to circumnavigate the faulty routers in the NoC, however, due to the restrictions in change of direction, there are no guarantees that the packets will be able to arrive at the destination for all possibilities of network faults. The Odd-Even turn model can also be utilized to supply load balancing in a NoC, through its capacity to route packets by multiple paths.

NARCO [ZP10] is a fault tolerant routing algorithm for NoCs, based upon the knowledge of states of the neighboring routers and upon the replication of messages controlled by the rate of failures. The adaptability of the algorithm results from the utilization of turn models, more specifically, the Odd-Even and Inverted Odd-Even models that allow circumnavigation of the faulty routers, but do not guarantee that the packets will be delivered. In addition to this, the replication of the packets is triggered when the rate of router faults exceeds a pre-established limit. At this moment in time, the packets are duplicated at the router source, whereby each packet is routed into one distinct virtual channel, each one of these using one of the two Turn-models quoted above.

The architecture Vicis [DFB+12] uses two distinct approaches to implement fault tolerance in NoCs. The first approach corresponds to an architectural solution and consists of a router that comprises of reconfigurable components that are able to tolerate diverse faults. The second approach now consists in carrying out the reconfiguration of the route tables of the routers, utilizing an algorithm of distributed routing. However, the NoC reconfiguration stage consumes on average 1000 clock cycles, during which, the network remains unavailable.

Another architectural solution is presented in [KMAMP08]. This solution consists of a light mechanism for fault tolerance of NoCs based upon standard backup paths, which are designed to maintain, even in the presence of faults, both the connectivity of non-faulty routers and the ones of healthy processing cores. This mechanism provides alternative paths, internal to the router, that connect the appropriated ports to the processing core. These paths serve to circumnavigate faulty components of the router

architecture. In the worse fault scenario, a uni-directional ring network provides connectivity between all the routing cores. However, the proposed mechanism in [KMAMP08] has its performance evaluated in relation only to a conventional NoC, whereby comparisons with other architectures or fault tolerant routing algorithms have yet to be carried out.

5.4.1 Analysis of fault tolerance requirements of the conceptual architecture

In our conceptual NGNoC architecture, the layers 1 and 2 have the greatest number of routers and they also carry out most of the communication flows. The implementation of fault tolerance mechanisms for this layer is imperative. It demands the implementation of a fault tolerant router architecture, in which a BIST (Build-In Self-Test) mechanism evaluates the router's internal circuits. This mechanism evaluates the status of the main router's components like: buffers, crossbar, routing and arbitration control. Even with the implementation of hardware redundancy schemes in the routers level, it is possible that a router stops to work properly. The adoption of a fault-tolerant routing scheme is mandatory. As stated on the sub-section 5.3, we proposed the utilisation of FlexOE as the routing scheme for layer 1 and for layer 2. We must adapt the proposed algorithm to also handle routers faults.

The FlexOE is a flexible routing scheme to control the congestion on a NoC. We propose to use the failure signal from the BIST circuit to improve the FlexOE rules scheme. Figure 10 depicts the proposed change. The most important FlexOE rule is replaced by the verification of the failure signal of the adjacent router in conjunction with the Odd-Even routing. In this way, if the failure signal is not active and the output port is selected by Odd-Even then this rule is enabled. Otherwise, the output port is not available for routing. That is a very simple and light weight change, which adds an important feature to this NGNoC.

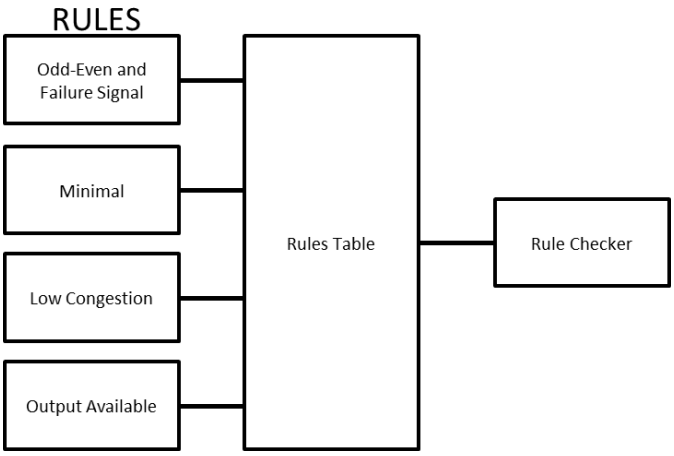


Figure 5.15: Fault-tolerant version of FlexOE.

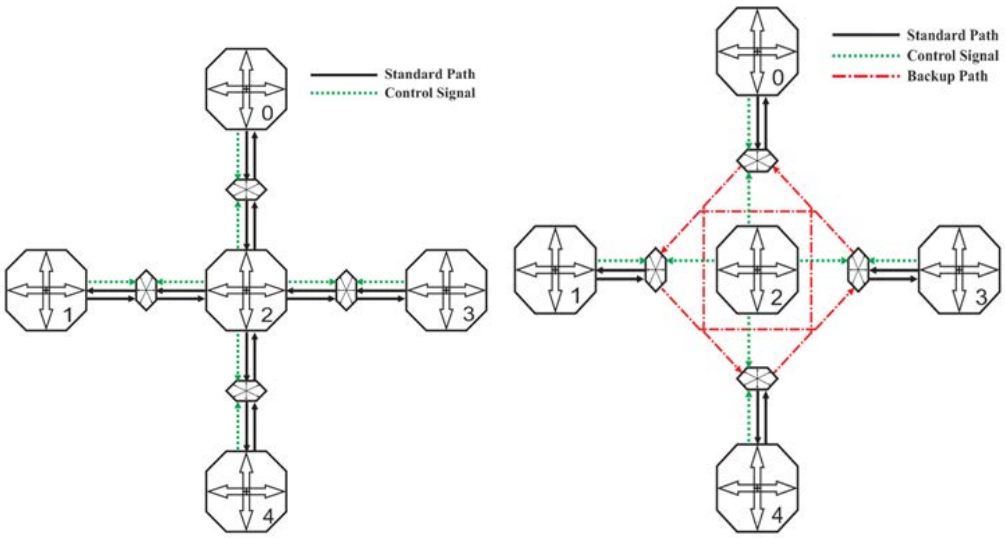
Differently from the other layers, the layer three has a small number of routers and it also carries simpler communication flows. For this layer, we propose a fault tolerant NoC architecture based on external links redundancy and on a variant of the XY algorithm.

The results of performance evaluation show that the proposed architecture guarantees 100% success in packet delivery for test scenarios up to 19% of faulty routers, while maintaining packets latency up to 90% lower than the reference algorithms.

5.4.2 A fault tolerant NoC architecture

The fault tolerant NoC architecture proposed in this work is characterized by the utilization of additional data paths external to the router, which allow circumnavigation of the faulty routers. The intention of this approach is to allow communication between the functional routers of the network. Each router must contain a BIST mechanism that evaluates its internal circuits. When this mechanism detects a router which is no longer capable of performing its functions correctly, a fault signal is sent to the Control Path Backup Mechanism (CPBM) circuits. These circuits work as simple multiplexers and, whilst the failure signal is active, all of the signals that were directed to each one of the router input ports are physically routed to the input ports of the neighboring routers. On the other hand, the signals that left the output ports of the faulty router are replaced by the output port signals of the neighboring routers.

Figure 5.16 illustrates the two fault scenarios for a router with its respective CPBM circuits. In the first scenario (a), when there are no router faults, the bypass circuit directs the signals directly to the input ports and to the output ports of the router, as in the usual way. In the second scenario (b), when there is now a presence of router fault, the bypass circuit reconfigures the organization of the input and output port signals in a way that the router at fault is bypassed, hence isolating it from the network but preserving the connectivity of the healthy routers.



(a) (b)

Figure 5.16: a) Connection details between routers in no-fault scenario. b) Connection details between routers in a faulty scenario.

Although the utilization of the CPBM increases the resilience of this NoC , this mechanism does not guarantee the connection for all cases of router faults. For example,

in case of one router at the edge of the NoC fails, it is not possible to use the bypass approach since those routers do not have all their five ports connected to other routers. In order to overcome this situation, it is necessary to apply the CPBM in a topology similar to the previously mentioned Torus, with some modifications. Figure 5.17 presents this topology, whereby all the routers possess their ports connected in such a way that each router at the edges of the NOC is fully connected. The proposed topology favors the CPBM utilisation, but it demands the implementation of long communication wires. The wires connecting the edge routers are very much longer than the other ones. Therefore, this topology is more appropriated for small scale NoCs.

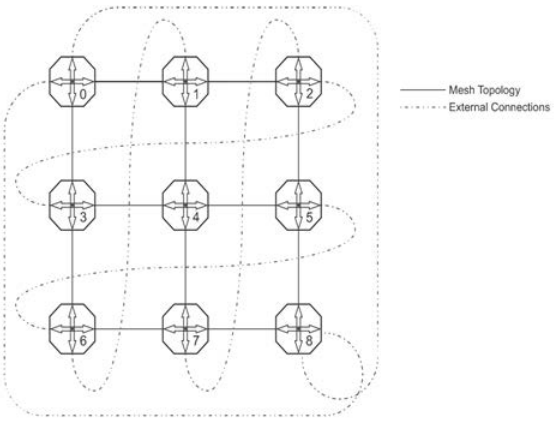


Figure 5.17: The proposed topology.

Besides the fault tolerance mechanism and the topology presented, there is a necessity for a routing algorithm capable to, even in the presence of faults, guarantee the functioning of the network. The proposed algorithm for this architecture is based upon the algorithm XY.

Initially the algorithm directs the flits by the axis X, and when the flit encounters the same column of the router destination, the transition to axis Y is completed. The differential of this algorithm is perceived when the router responsible for making the transition from axis X to axis Y for the determined route is faulty. In this scenario, the CPBM reconfigures the data paths in order to by-pass the faulty router, and the flit is directed to the router in the adjacent column, but on the same line. This router is then responsible for the directing of the flits to the axis Y. This situation is detected when the packet comes from a CPBM port and not from a routers output port, as shown on Algorithm 2.

The algorithm XY is free from deadlocks thanks to its self-imposed transition restrictions. The algorithm XY possesses only four possible transitions. However, the modified version of this algorithm used in this work adds other transitions. These additional transitions permit the creation of cyclic dependencies (deadlocks) between flits. In order to guarantee the absence of deadlocks, the proposed algorithm utilizes two virtual channels. The first channel is used as a standard for the exchange of messages. The second virtual channel is only used in the scenario as described in the previous paragraph, whereby this scenario adds transitions alongside those that already exist in the usual XY algorithm, which in turn, could form cycles (deadlocks) on the network. To block this, some transitions are routed by a virtual auxiliary channel.

Algorithm 2 Routing algorihtm

```
1: Source node:  (sx,sy);Destination node(dx,dy);
2: Current node:  (cx,cy);Incomming port:  inPort;
3: Outcome port:  outPort;
4: if (cx == dx) & (cy == dy) then
5:     Deliver packet to the local port and exit;
6: else
7:     if dx > cx then
8:         outPort ← EAST
9:     else
10:        if dx < cx then
11:            outPort ← WEST
12:        else
13:            if dx == cx then
14:                if dy > cy then
15:                    outPort ← NORTH
16:                else
17:                    if dy < cy then
18:                        outPort ← SOUTH
19: if inPort == CPBM_PORT then
20:     if dy > cy then
21:         outPort ← NORTH
22:     else
23:         if dy < cy then
24:             outPort ← SOUTH
25:     send(outPort, CV1)
26: else
27:     send(outPort, CV0)
```

Figure 5.18 presents an example of routing of a packet in a 3x3 NoC that utilizes the proposed architecture in this work. In the scenario presented, routers 0 and 5 are faulty and one packet is transmitted from router 2 to router 6. Even with two faulty routers, it was still possible to deliver the packet to its destination. It is important to note that in cases in which a router fails during a packet transmission, or in other words, when various flits of the packet are in transmission or stored in the router buffers, the proposed architecture would have the capacity to support this failure. In such case, it is necessary to apply a timeout mechanism in each router, so that the packet may be re-transmitted by another route.

In order to evaluate the performance of the proposed architecture, test scenarios were described in detail within the OMNeT++ simulator and the HNoCs [BIZCK12] library. In each scenario, it was measured the latency metrics and packet delivery success rates for a 4x4 network with 19% of faulty routers, in addition to a supplied load variation from 10% up to 50%. The choice of faulty routers followed uniform probability distribution. Moreover, each scenario was repeated 31 times (which is a standard value in order to calculate the confidence interval) within a confidence interval of 95%.

In order to compare the performance of the architecture, fault tolerant versions of

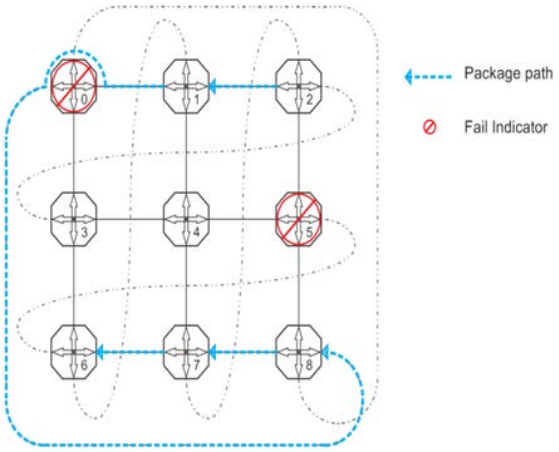


Figure 5.18: Routing of a packet sent by router 2 to router 6 when routers 0 and 5 are faulty.

the routing algorithms based upon Turn-models, Odd-Even and Inverted Odd-Even, as well as an implementation of the stochastic routing algorithm NARCO, in which each node has knowledge of the state of its immediate neighbors, were used.

Figures 5.19 present the results for the first two scenarios of test, in which 5%, 12% and 19% of faulty routers exist, respectively. In these scenarios, our proposal presented a performance well superior compared to the others. This is due to the simplicity of the proposed routing algorithm, which is based upon the algorithm XY. Furthermore, the proposed architecture guarantees 100% success in packet delivery for test scenarios up to 19% of faulty routers.

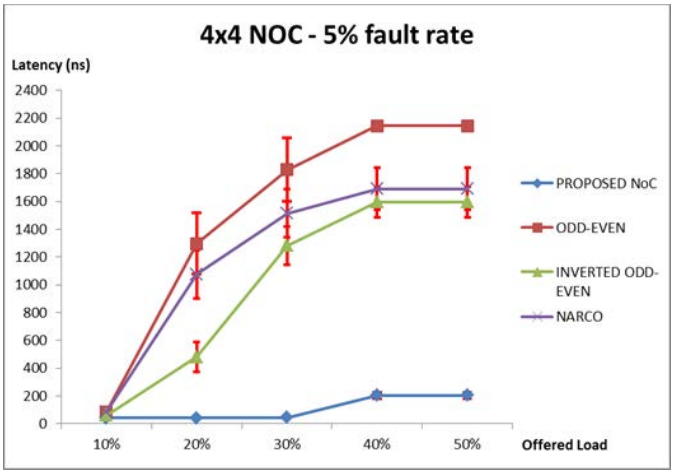
The proposed fault-tolerant NoC architecture is aimed at small dimensions networks, but it can be useful for the layer 3 of the conceptual architecture of NGNoCs.

5.5 FPGA-based emulation on NGNoCs

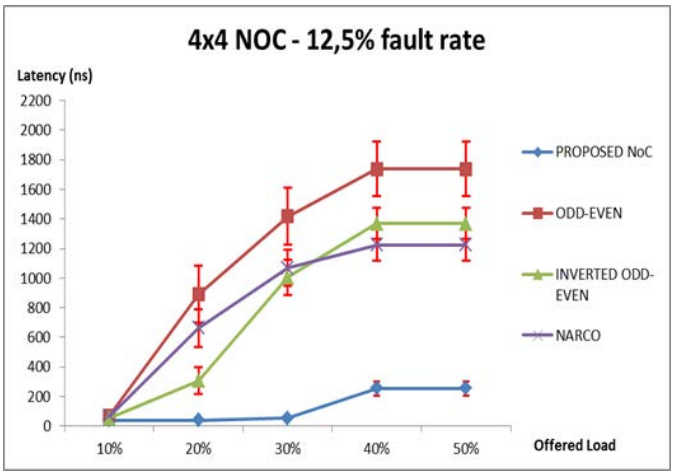
The FPGA-based emulation tools will play a major role onto the development of NGNoCs. The complexity of these networks demands advanced evaluation tools capable of verifying their features like QoS, congestion control, fault tolerance mechanisms, among others. The verification of these features is a key concern in FPGA-based NoC emulation. The main component of a NoC is the router. The router implements the majority of the networking functions. During a NoC prototyping, one is interested in validating a network with minimum changes on the routers circuits.

In order to check if a NoC works correctly, it is required the addition of special instrumentation circuits on the surroundings of the network routers. The additional instrumentation circuits must be used carefully, because they may change the network components behavior or even add serious errors. An interesting solution is placing the instrumentation circuits on the routers external links. This approach does not add changes on the routers circuits, thus it must infer the overall network performance by analyzing the observable behavior of each router.

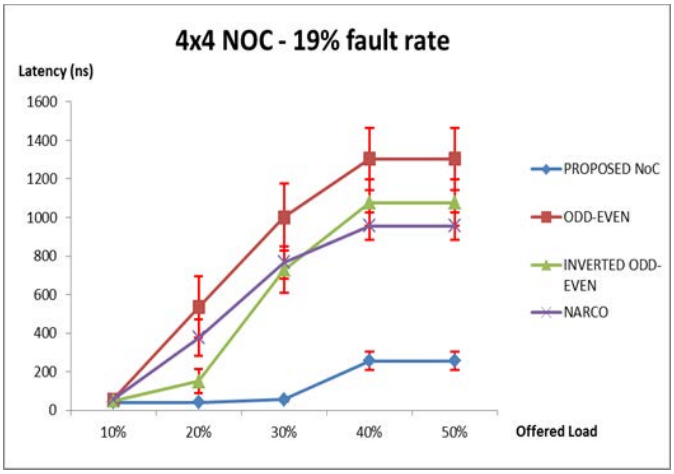
Those Link Monitoring Circuits (LMCs) are polyvalent. They can be used to count the number of flits traversing a link in order to calculate the dynamic power consumption or



(a)



(b)



(c)

Figure 5.19: Average latency for (a) 5% of faulty routers, (b) 12% of faulty routers and (c) 19% of faulty routers.

to determine if there is network congestion. Or even they can be used to inject permanent or transient faults during the NoC evaluation. Those LMCs must be accessible from the emulation Manager, which must configure them and extract the data generated during the emulation. Figure 11 depicts how these components may be add to the SNMP-like architecture presented in chapter 3. In this architecture, the LMCs are connected to the emulation node's MIB what makes them visible to the manager. The dashed lines represent the links to the MIB, which must be modified to hold the new registers related to the LMCs. Evidently more hardware resources are used to implement this new architecture. LMCs can be also used to monitor the vertical links connecting the NGNoC layers.

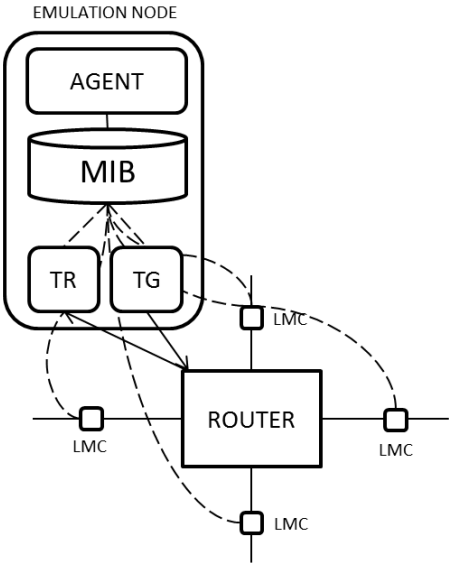


Figure 5.20: Emulation node with LMCs connections.

The LMCs add more flexibility to an FPGA-based emulation tools, but at same time they add a great design challenge: the augmentation of data volume generated during the emulation. The proposed SNMP-based emulation platform uses a very simple strategy to recovery the emulation results. In this architecture, the results are gathered after the execution of a traffic scenario. In order to do this, the emulation nodes must hold those results during the emulation. For some performance metrics like average latency this is not a big problem.

However when the links monitoring is added, some performance metrics may require a lot of data storage resources. For instance, a designer may be interested in the analysis of the evolution of dynamic power consumption during a traffic scenario. In order to do this, it is necessary to known how many flits traversed a given link for a determined time window. This flit count is repeated thousands or hundreds of thousands of times during one traffic scenario. Thus the emulation node is required to hold all those flit counts to calculate the desired performance metric. Of course it is not feasible to hold all that data for each router link on an emulation node MIB. It is necessary to continually send these metrics to the manager during the emulation. This on-going traffic between emulation nodes and the manager points to a necessary improvement of the interconnect between those components.

The future FPGA-based NoC emulators will carry out continuous communication flows with the platform manager during a NoC emulation. These communication flows are due to the great data volume produced by the circuits probing a NoC performance. The utilisation of an extra NoC to interconnect the emulation nodes may be an interesting solution for carrying out that traffic. However, this goes against one of the biggest constraints in FPGA-based emulation: the scarcity of hardware resources. In order to implement an extra NoC, one must design a very lightweight on-chip interconnect specially conceived for this application.

Herein we analyze architecture of a FPGA-tuned NoC which is conceived to interconnect the nodes of an FPGA-based NoC emulator. The SNMP protocol proposed in this thesis is used to manage the emulation nodes. This NoC is a perspective design for the emulation of the NGNoCs. This approach is not fully developed in the context of this thesis. However, it is targeted as one of our future research works.

We focus on the abundance of wires provided by FPGAs to propose a flit-routing scheme. In this scheme, each flit encodes data and control information necessary to route it to its destination node. Moreover, this strategy matches perfectly with our proposition of SNMP protocol because each flit carries one SNMP operation. Figure 5.21 depicts the fields of one flit. The flit has 61-bits divided in four fields: emulation node destination address with 10-bits length, SNMP operation with 3-bits length, the object identifier with 16-bits length, and the parameter value of the SNMP operation with 32-bits length.

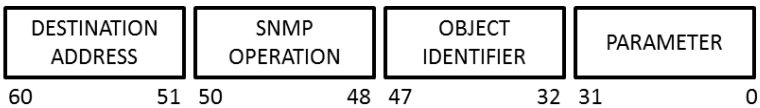


Figure 5.21: Flit organization.

The flit-routing enables the utilization of smaller buffers. Indeed, it is possible to consider a one flit buffer. This approach reduces the hardware resources used to implement one router. Another solution to reduce the resources overhead is to implement multi-local port routers. These routers have multiple instances of local ports, which are connected to different emulation nodes. For instance, one router with three local ports demands a 7x7 crossbar to interconnect its ports. This is similar to the crossbar used on router of 3D NoC topologies. Figure 5.22 depicts that router organization. In order to implement the additional local ports, it is necessary to change the nodes addressing. Usually a node is addressed by its X and Y coordinates, it is necessary to add a new field to indicate the destination port. This modified address is already previewed on the flit organization.

The multi-local port routers reduce the number of routers needed to connect the emulation nodes. However, it is possible to further reduce the amount of routers by connecting additional emulation nodes to unused ports of the routers on the network borders. Figure 5.23 depicts this organization. In this example, a 3x3 NoC connects 39 emulation nodes. This approach combined with the flit-routing of SNMP packets enables a potential reduction on the resources used to implement this NoC.

The utilisation of an extra NoC to interconnect the emulation nodes has the potential to increase the performance of an FPGA-based emulation platform. However, in order to

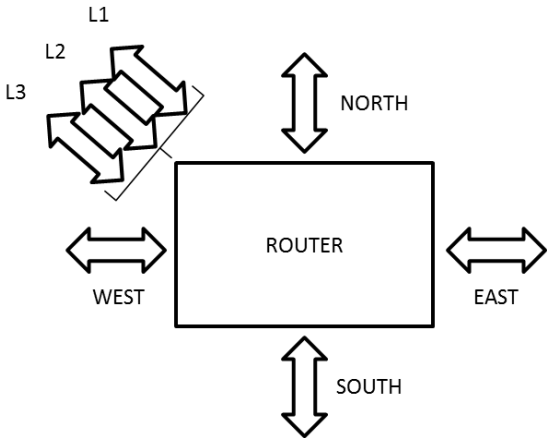


Figure 5.22: Multi-local port router organization.

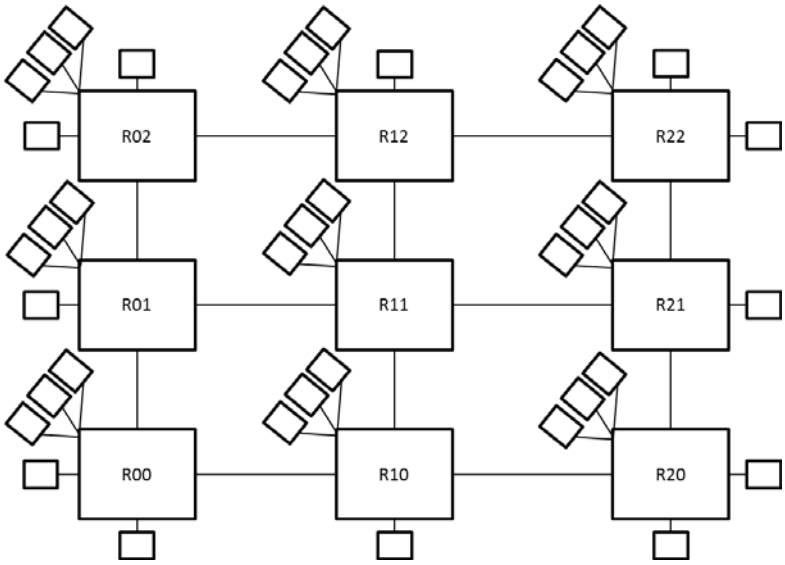


Figure 5.23: NoC topology to connect the emulation nodes.

profit from the extra bandwidth provided by this network it is necessary to improve the data link between the FPGA and the manager. High-end FPGAs have some high speed optical links which could be used to increase the bandwidth of the external data link.

Another key challenge for the emulation of NGNoCs is the accurate reproduction of application traffic. In Chapter 4 is presented a dependency-aware TG architecture based on a MoC extracted from applications traces. This approach does not depend on the original network architecture and it reduces the data storage requirements to execute real applications traffic on FPGAs. However, the NGNoCs are based on heterogeneous IPs like processors, memories blocks, MEMS, RFs, among others. It is necessary to analyze if the proposed MoC is capable to extract the communication behavior of those diverse components. Although this TG architecture proposition is a plausible candidate to the traffic generation on NGNoCs emulation, it is demanded a deeper analysis of its limitations. This analysis is also targeted as one of our future research works.

At last, it is important to highlight the importance of multi-FPGA platforms for the NGNoCs emulation. Those platforms are the natural solution to emulate a multi-layer NoC. Even though there are yet some open questions. For instance, there is a lack of design tools capable to automatically synthesize those platforms effectuating the correct partition of the NoC and emulation components on multiple FPGAs. The interconnect between the FPGAs is also another important issue. The performance asymmetry between the external links and the on-chip links limits the usability of those platforms.

5.6 Conclusions

The future many-core embedded systems will demand advanced on-chip networking features. These systems will be deployed in mixed-technology 3D-ICs. The NGNoCs are the interconnect technology of the future embedded systems. Therefore, these networks will be divided on heterogeneous layers connected by transversal links. The layers and the inter-layer connections may have specific physical constraints. Each layer implements an independent NoC with its own topology, number of nodes, QoS, among other parameters. This chapter presented a prospective analysis of those networks. From a conceptual NGNoC architecture, it is analyzed three subjects: congestion control, fault tolerance and FPGA-based emulation. For each subject, we have proposed mechanisms to deal with the main concerns for the NGNoCs. Some of those propositions are treated as future works and therefore they are not fully described or evaluated. Evidently there are other subjects concerning the NGNoCs which are not discussed in this chapter but they also deserve attention of the research community like QoS, network management, among others.

Chapter 6: Conclusions and perspectives

Amidst the worldly comings and goings, observe how endings become beginnings.
Lao Tzu

Contents

6.1 Conclusions 115
6.2 Future work 117

6.1 Conclusions

The FPGA-based emulation platforms will play a major role in the design process of the NGNoCs. Advanced networking features like QoS, congestion control, fault tolerance mechanisms will be a common place in NGNoCs. These features must be deeply evaluated during a network prototyping. The complexity of those emulation platforms increases according to the features implemented on the NoCs. Thus the number of emulation components will grow significantly, what exposes the need of an efficient management of them. Moreover, standard protocols to manage FPGA-based NoC emulation platforms are an elegant solution to deal with the complexity of these systems as well as to provide high-level interfaces to control the execution of evaluation scenarios. Realistic workloads are also a key concern in the performance evaluation of NGNoCs. The generation of realistic traffic is a challenge on FPGA-based emulation. Although it is vital to evaluate new NoC architecture with applications traffic, because they provide very more accurate results.

This thesis objective is the synthesis of FPGA-based NoC emulation platforms as well as a prospective analysis of the requirements of the NGNoCs. We have proposed different solutions to deal with key challenges of those platforms. At first, we have proposed a light version of SNMP protocol to manage the emulation of a NoC. A model based on the SNMP principles was implemented. The emulation components were organized as agents in which their behavior is remotely controlled by changing the registers in a MIB. The simple request-response communication model enables the creation of an execution flow. This execution flow entails the emulation nodes configuration, traffic scenario execution and the retrieve of the evaluation results. These tasks are executed using SNMP operations.

In addition to the proposed protocol and execution flow, We implement an architecture of an FPGA-based NoC emulation platform for two different traffic models. This platform enables the emulation of NoCs with up to 64 nodes for a Virtex-6. Several experiments were carried out to evaluate the number of executed SNMP operations for different applications, the resources utilisation for several NoC sizes, the time required to update the agents MIB. The experiments highlight that a light version of SNMP is very efficient for a light resources overhead. Furthermore, it was also presented a case of study in which all these proposals are used in order to carry out a DSE of a NoC in relation to the task mapping and mesh topology.

Another contribution of this thesis is a traces analysis framework to synthesize TGs. It is known that packets dependency information improves the accuracy of trace-based simulations. However, these dependency-aware traces require more data storage resources than regular traces. We have proposed a framework to analyze applications traces in order to extract their packets dependencies. These packets dependencies are further processed to create a MoC, which is easily implemented by a dependency-aware TG architecture for FPGA-based NoC emulation. The framework extracts the packets dependencies in the form of Dependency Patterns (DPs), which represent the relationship between the input and output packets.

The framework evaluates several combinations of DPs in order to find the best possible combination to create a Dependency Table (DT). A DT is a model of the reactive behavior of an application node. The proactive behavior is modeled as a small packets table. The proposed framework is evaluated for a broad range of NoC configurations. We evaluate its precision for different routing algorithms, buffer sizes, flow control mechanisms, for 2D and 3D topologies as well as random task mapping. The experimental analysis highlights that this framework is more accurate than trace-based simulations for a broad variety of network configurations. Furthermore, the proposed framework uses only 3% of the data storage required by the traces.

We have also analyzed the perspectives of the NGNoCs. From a conceptual model of a next-generation embedded system, we have extracted high level communication requirements of an on-chip communication infrastructure implemented on a 3D-IC. These communication requirements enabled a prospective analysis of the networking features for three key areas: congestion control, fault tolerance and FPGA-based emulation. We have proposed some networking mechanisms for these subjects. At first, the FlexOE which is an adaptive routing algorithm that selects the output path for a given packet based on a simple prioritized scheme of sets of rules. This algorithm is used to deal with the congestion control and fault tolerance for a NoC implemented in one of the layers of a 3D-IC.

We have also proposed a piggyback protocol to spread the vertical links congestion information throughout a NoC. A new fault tolerant architecture for small and medium sizes NoC is also presented. This architecture is based on external bypass links which allow circumnavigation of the faulty routers. The results of performance evaluation show that the proposed architecture guarantees 100% success in packet delivery for test scenarios up to 19% of faulty routers, while maintaining packets latency up to 90% lower than the reference algorithms. At last, we analyze the requirements of the FPGA-based emulation platforms for the NGNoCs.

6.2 Future work

The future works of this thesis are organized in three axes: traces analysis, architectures for the NGNoCs and FPGA-based emulation.

The work carried out to formulate the proposed traces analysis framework allows us to lay the groundwork for more advanced investigations. We are interested in reformulating the base set extraction/evaluation problem as a combinatorial optimization problem. In this way, classical optimization heuristics could be used in order to extract the DT. We believe that this may lead to more accurate DTs. Furthermore, the proposed framework imposes that the packets dependencies are constant during all the application execution. We intend to evaluate this framework for multi-phases applications, in which the packets dependencies change during the execution. Our proposition is to analysis the traces in order to identify the traces regions in which the dependencies are constant. For each region is built a MoC. The complete execution of the MoCs would reproduce the application behavior.

Another perspective for the traces analysis proposition is the study of the traces gathered from the simulation of shared memory chip multi-processors (CMPs). By now, we have focused on message passing application, which do not share a common memory region. The CMPs are deeply used in industrial applications. Their on-chip traffic is composed of cache coherence protocols packets. This research may lead to new communication models to replace the processing cores as well as the cache modules on the NoC performance evaluations.

The architecture of the NGNoCs was treated as a prospective analysis in this thesis. However, we are interested in advancing in the propositions of new mechanisms for these networks. More specifically, we intend to spend efforts in order to adapt data diffusion protocols to the context of NGNoCs. We believe that is important to extract an global view of the network status during its execution. This global view enables the monitoring of important information like congested regions, faulty components, and power consumption, among others. The piggyback mechanism mentioned previously is the first approach in this direction. However, we are inclined to investigate other solutions used in different networks like the Wireless Sensor Networks (WSNs), which are known for using interesting techniques such as data fusion, data based routing, gossip algorithms, etc. Besides that, we intend to continue investigating fault tolerant mechanisms for NGNoCs. We also have presented a scheme of a NoC to interconnect the nodes of an FPGA-based NoC emulation platform. We intend to implement this architecture in order to evaluate its potential for the emulation of the NGNoCs.

There are also perspectives of international collaborations which were created from the research works developed during this thesis:

- with the research team of Suiyng Yao from Tianjin University, China, for the development of FPGA-based NoC emulation platforms;
- with the research team of Helano Castro from Federal University of Ceara, Brazil, for the development of fault tolerant mechanisms for NoCs.

Glossary

CABA	Cycle Accurate Bit Accurate
CMOS	Complementary Metal Oxide Semiconductor
MEMS	Microelectromechanical systems
DT	Dependency Table
DP	Dependency Pattern
DSE	Design Space Exploration
ECC	Error Correcting Code
FPGA	Field Programmable Gate Array
FFT	Fast Fourier Transform
HDL	Hardware Description Language
MoC	Model of Computation
MGT	Message Generation Table
NoC	Network-On-Chip
PCB	Printed Circuit Board
OSI	Open Systems Interconnect
QoS	Quality of Service
RF	Radio Frequency
SAF	Store-and-Forward
SNMP	System Network Management Protocol
SoC	System-On-Chip
TDM	Time Division Multiplexing
TG	Traffic Generator
TSV	Through-Silicon Vias
TR	Traffic Receptor
VCT	Virtual Cut-Through

List of Publications

International Conferences and Symposiums

[1] Otávio Alcântara de Lima Jr., Virginie Fresse and Frédéric Rousseau. Evaluation of SNMP-like protocol to manage a NoC emulation platform. In 2014 International Conference on Field-Programmable Technology, Shangai, Chine, 2014. ***[Published]***

[2] Otávio Alcântara de Lima Jr., Virginie Fresse and Frédéric Rousseau. FlexOE: A congestion-aware routing algorithm for NoCs. In 2013 International Symposium on Rapid System Prototyping, Montreal, Canada, 2013. ***[Published]***

[3] Helano Castro and Otávio Alcântara de Lima Jr. A fault tolerant NoC architecture based upon external router backup paths. In 2013 International Conference on New Circuits and Systems Conference, Paris, France, 2013. ***[Published]***

Journals

[4] Ke Pang, Virginie Fresse, Suying Yao and Otávio Alcântara de Lima Jr. Task mapping and mesh topology exploration for an FPGA-based network on chip. In Microprocessors and Microsystems, may, 2015, pages 189–199, 2015. ***[Published]***

Appendix : Resumé en Français

1.0 - Introduction

Le développement accéléré de la microélectronique permet l'intégration de dizaines ou de centaines de cœurs logiques complexes dans un System-on-Chip (SoC). Ces SoCs sont constitués d'un ensemble de blocs de calculs, mémoires, blocs logiques spécialisés qui partagent une infrastructure de communication. Les besoins croissants de communication de ces systèmes exigent des architectures de communication flexibles et parallèles. Les réseaux sur puce (NoCs) apparaissent comme la technologie la plus prometteuse pour la communication de SoCs modernes, car ils ont une plus grande flexibilité par rapport à d'autres solutions telles que les connexions point à point et les bus de communication. Les NoCs possèdent un vaste espace de conception concernant différents aspects comme la topologie, le routage, le contrôle de flux et la qualité de services entre autres.

Un concepteur de SoC doit faire plusieurs choix architecturaux puis réaliser des évaluations de performances pour trouver la meilleure configuration d'un NoC. En plus, tous les aspects (choix des IPs, interfaces de communication, services réseaux) du système complet doivent être testés et validés. Une partie significative des coûts de conception d'un SoC est consacrée à l'exploration de l'espace de conception et à la validation de l'architecture de communication. Les outils de conception spécialisés pour les NoCs peuvent être utilisés pour réduire les coûts de conception. La simulation de haut niveau [CCG+04, HAAN+07] est utilisée dans les premières étapes du processus de développement pour une exploration de l'espace de conception plus rapide, mais moins précise. Normalement ces simulations sont écrites en langages de haut niveau comme le C, C++ et Java. Les simulations CABA (Cycle Accurate and Byte Accurate) ont une plus grande précision au détriment d'un temps d'exécution très élevé. Ces simulations sont utilisées pour valider un modèle HDL.

Les outils d'émulation basés sur FPGA sont une technique prometteuse pour la validation d'une nouvelle architecture de NoC. Ces plateformes peuvent réduire considérablement le temps d'évaluation tout en conservant un niveau de précision élevé. La limitation des ressources d'un FPGA est une contrainte qui détermine la taille maximale du NoC émulé. L'émulation d'un système complet n'étant pas toujours possible, un émulateur utilise des composants spécifiques afin d'injecter et d'extraire des paquets sur un réseau. Les générateurs de trafic (TG) prennent en charge la génération de trafic basé sur des applications réelles ou des modèles synthétiques. Les récepteurs de trafic permettent de récupérer les paquets et de faire des calculs de performances.

Les futurs NoCs offriront des fonctionnalités avancées afin de répondre aux exigences

croissantes des applications de systèmes embarqués. La complexité des plateformes d'émulations sur FPGA augmente en fonction de la complexité du NoC émulé, ce qui nécessite une gestion efficace des composants de ces plateformes. Il n'existe pas, à l'heure actuelle, des protocoles standard conçus pour la gestion des composants d'émulation de NoCs. En plus, les propositions des plateformes d'émulation sur FPGA n'offrent pas l'interopérabilité des composants d'émulation. L'adoption d'un protocole standard peut faciliter l'intégration des composants d'émulation créés par différents concepteurs. Ce protocole doit fournir une interface commune pour les composants d'émulation du point de vue du logiciel de gestion fonctionnant sur le PC hôte en charge de l'émulation.

L'émulation de NoC sur FPGA dépend fortement des modèles de génération de trafic, qui précisent la répartition spatio-temporelle d'un ensemble de paquets de données. Les NoCs devenant l'architecture de communication par défaut pour les SoCs, les modèles de génération de trafic deviennent très importants pour la conception de systèmes embarqués. Les modèles de génération de trafic traditionnels sont basés sur des charges de travail synthétiques, qui ne représentent pas fidèlement les caractéristiques des applications réelles. Il est important de valider des nouvelles architectures de NoC avec des charges de travail réalistes, car elles fournissent des résultats beaucoup plus précis. La synthèse des TGs basés sur le trafic d'applications est une préoccupation majeure pour l'émulation des NoCs.

L'exécution des charges de travail réalistes sur une plateforme d'émulation de NoC sur FPGA est assez difficile. Les ressources matérielles du FPGA sont faibles et doivent être utilisées à bon escient. Dans certains cas, les nœuds d'application ne sont pas disponibles pour l'émulation alors que des traces de communication le sont. Une trace peut facilement comprendre des millions de paquets. La taille des traces est une question importante pour l'émulation de NoC en raison du nombre et de la capacité des RAM internes sur FPGA. Les traces sont souvent allouées dans la RAM externe, ce qui devient un goulot d'étranglement pour la génération du trafic en temps réel. En outre, l'exécution basée sur les traces déforme le taux et les effets de la congestion à cause du manque d'information des dépendances de paquets. Certains auteurs [NMFA11, HCT+12, TAA+11] proposent l'utilisation des traces en tenant compte des dépendances. Ces traces contiennent alors des informations de dépendances de paquets, ce qui les rend plus précises pour être exécutées dans une large gamme de configurations de NoC. Toutefois, elles exigent plus d'espace de stockage de données que les traces régulières.

Les futurs systèmes embarqués multi-cœurs exigent plusieurs mécanismes de mise en réseau complexes tels que la tolérance aux pannes, la gestion de réseau, le contrôle de congestion et la QoS entre autres. Les NoCs des générations futures (NGNoCs pour Next Generation NoC) mettront en œuvre ces mécanismes avancés de mise en réseau. Ces réseaux seront déployés sur des composants 3D. Par conséquent, ces réseaux seront répartis sur des couches hétérogènes reliées par des liaisons transversales fiables. Les couches et les connexions inter-couches peuvent avoir des contraintes physiques spécifiques. Chaque couche implémente un NoC indépendant avec sa propre topologie, son propre nombre de nœuds et sa QoS.... De nombreuses questions de mise en œuvre découlent de cette nouvelle tendance technologique, et nous pouvons citer entre autre : le contrôle de congestion intra et inter -couches et les mécanismes de tolérance aux pannes. Il est également important d'envisager les nouvelles méthodes d'évaluation de performances de ces mécanismes avancés au moyen d'outils d'émulation de NoC sur FPGA.

La contribution principale de cette thèse est la synthèse de plateformes d'émulation de NoC sur FPGA pour les futurs systèmes embarqués multi-cœurs. De nouvelles stratégies pour la génération de trafic pour les NoCs émulés sur FPGA sont proposées, ainsi que la gestion de la plateforme d'émulation en utilisant un protocoles standard inspiré sur des réseaux informatiques sont proposées. De plus, une analyse prospective des architectures NGNoCs est également effectuée.

2.0 - L'état d'art

Les NoCs semblent la technologie la plus prometteuse pour les communications sur puce. Ils permettant une plus grande flexibilité et offre un parallélisme que des solutions telles que les bus et les connexions point-à-point. De nombreux outils et plateformes industriels utilisent la technologie NoC, comme Intel SCC [VHR+08], Polaris [HDH+10] and Tilera64 [BEA+08].

Malgré ses avantages d'utilisation évidents, un NoC est une architecture simple à quatre composantes : les cœurs de traitement, les interfaces réseau (NIs), les routeurs et les liens. La figure 1 représente l'architecture NoC mise en œuvre sur un maillage 2x2. Les cœurs de traitement sont les clients des services de communication du NoC. Ils utilisent le réseau pour transmettre et recevoir des informations. Ces cœurs sont multiples, mais sont généralement des processeurs, DSP, des blocs mémoire ou des blocs IP spécialisés. Les NIs sont responsables de l'interface entre les cœurs et le réseau, et également responsables de la communication de bout en bout du réseau. Ils fournissent des services de communication différenciés tels que les services orientés connexion, sans connexion et la qualité de service. Les routeurs sont utilisés pour transmettre des paquets à travers un réseau, et pour fournir des mécanismes de contrôle de flux. Ils sont reliés aux NIs sur les ports locaux. Enfin, les liens sont les fils physiques interconnectant les ports des routeurs créant une topologie du réseau.

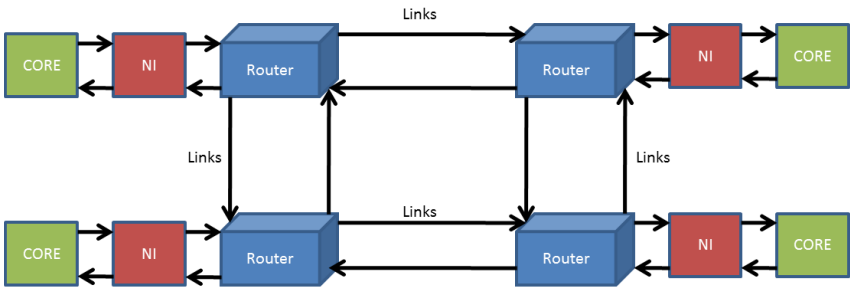


Figure 1: Architecture d'un NoC 2x2 maillé.

Dans cette partie, le modèle du NoC est comparé au modèle de référence OSI de réseaux d'interconnexion. Le modèle NoC possède quatre couches qui sont extraites/inspirées des sept couches du modèle OSI. La figure 2 illustre la relation entre les couches du NoC et le modèle de référence OSI. La couche système comprend les services des deux premières couches OSI. Cette couche représente les besoins/requêtes des cœurs de traitement. La couche de l'interface réseau est chargée de contrôler les services de communication de bout en bout. Cette couche représente les services des

couches OSI de session et de transport. Ensuite, la couche réseau fournit les mêmes services que la couche réseau du modèle OSI. Ce service de couches est mis en œuvre par le routeur. Enfin, la couche liaison comprend les liens logiques et physiques qui relient les composants du NoC.

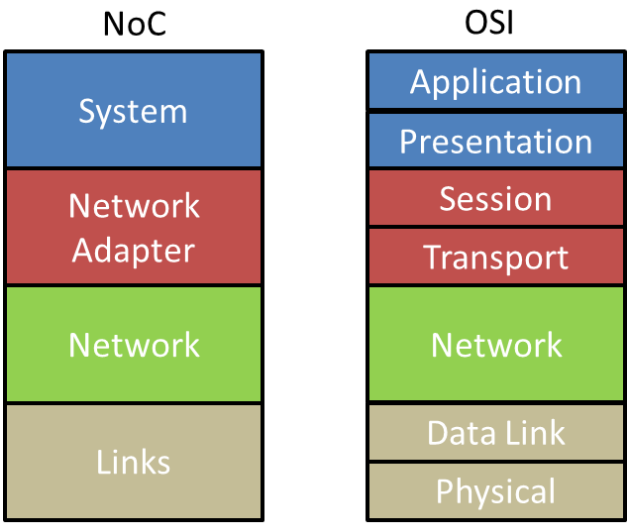


Figure 2: Modèle de couche d'un NoC.

2.1 - Les Plateformes d'évaluation de performance de NoC sur FPGA

Les plateformes d'évaluation de performances de NoC sur FPGA sont des outils importants pour la conception d'un NoC. En effet, le potentiel des FPGA pour réaliser des évaluations rapides et précises permet la conception de plusieurs architectures de plateformes d'évaluation. Nous avons simplement divisé ces plateformes en deux catégories selon leur objectif : le prototypage et la simulation. Les plateformes de prototypage sont utilisées pour émuler et vérifier la conception matérielle d'un NoC. Les plateformes de prototypage à base de FPGA sont complexes parce qu'elles modélisent précisément tous les composants d'un NoC. De même, les plateformes de simulation sur FPGA accélèrent l'évaluation d'un NoC en utilisant un modèle de réseau moins précis et plus léger car seulement quelques composants d'un NoC sont modélisé grossement. Ces plateformes sont utilisées pour une première simulation (plus rapide) d'une architecture NoC déjà connue. Malgré les différences entre ces deux approches, il est possible de décrire une organisation générale pour ces deux plateformes, dans la figure 3.

Les principaux composants d'une plateforme d'évaluation de performance sont représentés dans la figure 3. L'hôte est un ordinateur connecté à une carte FPGA via un lien de données. Le logiciel s'exécutant sur l'hôte est responsable de la mise en place de la plateforme selon les scénarios d'évaluation prédéfinis. Ces scénarios peuvent être générés par un autre outil logiciel afin d'évaluer un NoC. L'hôte permet également d'extraire les résultats de l'évaluation de la plateforme. Ces résultats peuvent être aussi analysés par d'autres outils. Nous identifions les tâches de l'hôte comme les tâches de gestion du système : ces tâches gèrent la configuration d'une plateforme et offrent les

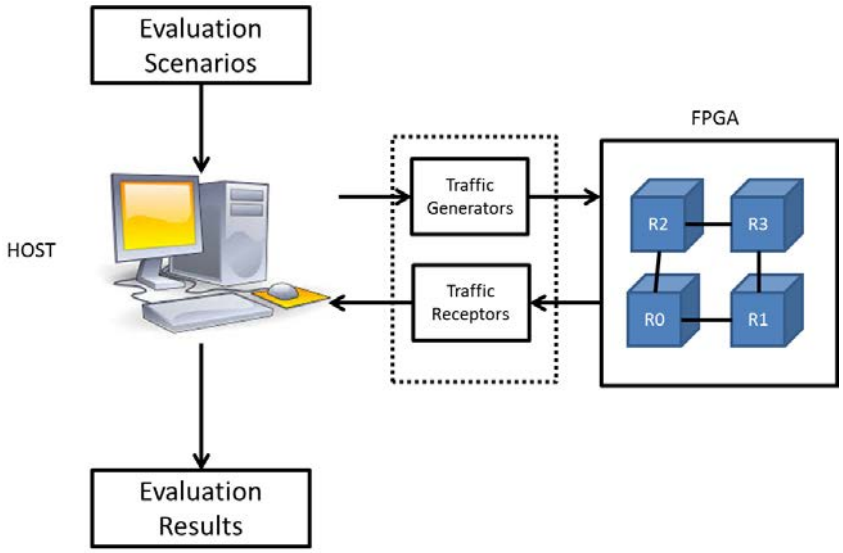


Figure 3: L'organisation d'une plateforme d'évaluation de performances de NoC sur FPGA.

services de la plateforme à d'autres outils dans le processus de conception d'un NoC.

Les générateurs de trafic (TG) sont responsables de la génération de trafic basé sur des modèles réels ou synthétiques. Les récepteurs de trafic (TR) permettent de récupérer les paquets et de calculer des indicateurs de performances. Dans une évaluation complète du système, ces composants sont remplacés par les vrais cœurs d'applications. Dans d'autres scénarios d'évaluation, des composants spécifiques sont utilisés, notamment le TG et le TR. Ces composants ne sont pas intégrés dans le FPGA dans la figure 3, car il existe plusieurs stratégies d'intégration pour ce type de blocs. Ils peuvent être mis en œuvre de manière logicielle dans un cœur de processeur, ou matérielle comme des blocs IP sur le FPGA. Une évaluation des performances d'un NoC dépend des modèles de génération de trafic, qui spécifient la distribution spatio-temporelle du trafic de données. Il est important de valider les nouvelles architectures de NoC avec des charges de travail réalistes, car elles fournissent des résultats plus précis. Les modèles traditionnels de génération de trafic synthétique ne représentent pas fidèlement les caractéristiques des applications réelles [GK10]. L'exécution des applications réelles dans l'émulation de NoC sur FPGA est assez complexe. Ces deux composants (TG et TR) correspondent au modèle de génération de trafic.

Le dernier composant est le NoC, qui est le composant à évaluer. Le modèle de NoC utilisé sur une plateforme dépend de l'objectif de l'évaluation. Les plateformes de prototypage ciblent la validation du réseau complet et cette validation implique des efforts importants pour la mise en œuvre et la vérification. D'un autre côté, les plateformes de simulation utilisent des modèles plus simples pour accélérer l'évaluation. Ces simulations peuvent être utilisées pour l'exploration de l'espace de conception (DES) dans les premières étapes de développement. En outre, les plateformes de simulation peuvent appliquer les techniques de Time Division Multiplexing (TDM) qui permettent l'utilisation d'un modèle d'un seul routeur pour simuler l'ensemble du

réseau. Cette technique permet d'augmente la taille du réseau final implanté sur FPGA. Ces implémentations architecturales correspondent au modèle du réseau.

2.2 - Exemples des Plateformes

Le tableau 1 présente quelques-unes des principales plateformes d'évaluation de NoC sur FPGA. Dans ce tableau, il est possible de recueillir des informations intéressantes sur les différentes approches pour évaluer les NoCs sur FPGA. Dans un premier temps, le nombre de nœuds évalué dépend du modèle de réseau. Les plateformes utilisant des techniques de virtualisation permettent l'évaluation de grands réseaux. Cependant, la mise en œuvre directe (Direct-Mapping) possède généralement un petit nombre de nœuds principalement lorsqu'elle est combinée avec l'émulation des cœurs de l'application. Un autre point important est la fréquence d'exécution. Les plateformes utilisant la mise en œuvre directe possèdent des fréquences élevées d'exécution. Par contre, les plateformes de simulation et de virtualisation travaillent à faibles fréquences d'exécution. Il existe trois exceptions [CSK14], [PHM11] and [Pap11], mais les fréquences affichées sont les fréquences du moteur de simulation. Elles ne correspondent pas à la fréquence d'exécution du routeur virtuel, fréquence qui est beaucoup plus faible en raison de l'exécution séquentielle des routeurs virtuels.

2.3 - Défis ciblés de la recherche sur l'évaluation de performance de NoCs sur FPGA

A partir de l'analyse des principales caractéristiques des plateformes d'évaluation de NoC sur FPGA, il est possible de mettre en évidence deux défis clés de la recherche : l'interopérabilité et la génération de charge de travail réalistes. Ces deux défis ne sont pas suffisamment explorés dans les récentes publications scientifiques. Néanmoins, ils mènent à des améliorations importantes pour les NoCs actuels et ceux des prochaines générations.

2.3.1 - La problématique de l'interopérabilité

Les plateformes d'émulation de NoCs sur FPGA utilisent des TGs et TRs qui sont mis en œuvre sous forme de composants matériels ou logiciels. Pour exécuter un scénario d'évaluation, tous les composants de l'évaluation doivent être configurés et par la suite, doivent être accessibles pour extraire les résultats d'émulation. Il n'existe aucune interface standard pour gérer les composants d'émulation du point de vue du PC hôte. Les propositions précédentes des plateformes d'évaluation sur FPGA n'abordent pas les questions liées à l'interopérabilité des composants des plateformes d'évaluation. L'adoption d'un protocole standard facilite l'intégration de composants d'émulation créés par différents concepteurs.

La gestion des TGs et TRs impose des défis. L'adoption des concepts de gestion déjà testés permet de maîtriser la complexité d'une plateforme d'émulation de NoCs. En outre, un protocole de gestion crée une interface commune pour gérer les composants d'émulation de différents fabricants à partir d'un PC hôte. Les protocoles de gestion tels que SMBus [smb], NC-SI [nc-], WMI [wmi], RDP ite RDP, NetFlow [net] et SNMP pourraient être des solutions envisageables pour intégrer et configurer des TGs et TRs. Il est nécessaire d'étudier les caractéristiques de ces protocoles pour évaluer leur

Plateforme	Modèle de réseau	Génération de trafic	Nombre de Routers	FPGA	Fréquence	Année
[CSK14]	Virtual	Synthetic	1024	Virtex 7	42 MHz	2014
[FGTR12]	Multi-FPGA	Synthetic	18	2 Virtex 5	-	2012
[HGM+12]	Direct-Mapping	Application Cores	9	Virtex 5	-	2012
[TFR11]	Direct-Mapping	Synthetic	36	Virtex 5	-	2012
[PAK+11]	Virtual	Application Cores	16	Virtex 5	3.15 MHz	2011
[KC11]	Virtual	Application Cores	64	Virtex 2	-	2011
[PHM11]	Simulation	Synthetic	576	Virtex 6	300 MHz	2011
[Pap11]	Virtual	Traces	256	Virtex 6	152 MHz	2011
[LPG11]	Direct-Mapping	Traces/Synthetic	25	Virtex 5	-	2011
[WHLB10]	Simulation	Synthetic	49	Virtex 6	50 MHz	2010
[KSPK10]	Direct-Mapping	Traces	64	Virtex 2	45 MHz	2010
[L+10a]	Multi-FPGA	Application Cores	16	5 Virtex 5	-	2010
[LK09]	Direct-Mapping	Application Cores	9	Virtex 2	50 MHz	2009
[LH08]	Multi-FPGA	Application Cores	48	4 Virtex 2	-	2008
[KMSP08]	Multi-FPGA	Synthetic	16	2 Virtex 2	-	2008
[KCdlTR08]	Direct-Mapping	Synthetic	16	Virtex 2	-	2008
[WHS07]	Virtual	Synthetic	64	Virtex 2	91.6kHz	2007
[GADMB07]	Direct-Mapping	Synthetic	6	Virtex 2	50MHz	2007

Table 1: Plateformes d'évaluation de performance de NoC sur FPGA

adéquation dans le contexte d'émulation NoC. L'interopérabilité des composants des plateformes d'évaluation s'élève comme un défi clé de la recherche sur la conception de NoCs. Fournir des protocoles standard est essentiel pour faciliter le partage de blocs IP ciblés sur l'évaluation de NoCs.

2.3.2 - La problématique de la génération de charges de travail réalistes

L'amélioration de la précision de la charge de travail utilisée pour effectuer des évaluations de NoC est essentielle au développement de la prochaine génération de NoC. Cependant, les modèles de génération de trafic traditionnels basés sur les charges de travail synthétiques ne représentent pas fidèlement les caractéristiques des applications réelles [GK10]. La façon la plus précise pour évaluer un NoC est d'exécuter des applications réelles. Cependant, ces simulations dites « système-complet » nécessitent beaucoup de ressources FPGA et ne sont pas envisageables tellement la complexité est importante. Pour surmonter ce problème, les concepteurs utilisent des traces de simulations système complet, qui sont plus rapides à exécuter. Cependant, l'information temporelle obtenue des traces n'est pas valable pour un large éventail d'architectures. En tenant compte de dépendances, les traces sont plus précises, mais elles exigent plus de ressources de stockage. Les traces générées par la modélisation statistique du trafic présentent les mêmes inconvénients que les traces obtenues par une simulation système complet. Il est nécessaire de prospecter sur de nouvelles approches pour la synthèse des architectures de TGs basés sur des traces d'applications, architectures plus légères et précises.

3.0 - Un protocole basé sur SNMP pour la gestion d'une plateforme d'évaluation de NoC

Dans cette section, nous présentons une adaptation du protocole SNMP pour la gestion d'une plateforme d'évaluation de NoC sur FPGA.

3.1 - Analyse des protocoles informatiques pour la gestion d'une plateforme d'émulation de NoC

Afin d'adopter un protocole standard pour la gestion d'une plateforme d'émulation sur FPGA, il est nécessaire d'explorer les protocoles déjà utilisés dans les systèmes informatiques. Les protocoles de gestion tels que SMBus [smb] ou NC-SI [nc-] pourraient être des solutions envisageables pour l'intégration de TGs et TRs. Cependant, ces protocoles sont conçus spécifiquement pour la gestion des cartes mères d'ordinateur. Ils ne sont pas conçus pour surveiller les composants de trafics dans une plateforme d'émulation.

Le JTAG est une norme IEEE utilisée pour contrôler le bon fonctionnement des entrées sorties des cartes de circuits imprimés. Ce protocole est également largement utilisé pour le débogage des circuits intégrés complexes comme des microprocesseurs ou FPGA. Le JTAG est basé sur un bus en cascade, ce qui réduit ses performances. En outre, ce protocole n'a pas été conçu pour les tâches de gestion, il est plutôt ciblé pour le débogage.

Les protocoles précédents manquent de souplesse et/ou de performance pour gérer un large éventail de dispositifs. Ici, nous analysons les caractéristiques des protocoles de gestion de réseau. Les protocoles de gestion comme WMI [wmi], RDP [**?**], NetFlow [net] et SNMP sont largement utilisés pour la gestion des périphériques connectés à des réseaux informatiques. Ces protocoles sont essentiels pour les systèmes informatiques modernes. Assurer une surveillance et la fiabilité de ces systèmes dans la performance est absolument nécessaire. Le WMI (Windows Management Instrumentation) est un protocole propriétaire conçu pour fonctionner dans un système d'exploitation Windows. Ce protocole est utilisé pour recueillir des informations de gestion sur les matériels, les logiciels et les composants du système d'exploitation. Le WMI peut être utilisé pour toutes les applications Windows. Le RDP (Remote Desktop Protocol) est également un protocole propriétaire ciblé pour surveiller à distance un ordinateur. Le NetFlow est un protocole de gestion des périphériques Cisco. Ce protocole est spécialisé dans le contrôle de la fluidité du trafic dans un réseau. Contrairement aux protocoles antérieurs, le protocole SNMP est un protocole ouvert qui a été conçu comme un protocole de gestion à usage général pour les périphériques en réseau.

Le modèle architectural SNMP est un ensemble de stations de gestion de réseau (gestionnaires) et les éléments de réseau (dispositifs gérés). Les éléments de réseau sont des appareils connectés sur le réseau comme des routeurs ou des passerelles. Chaque périphérique géré contient un agent, qui est responsable du traitement des demandes du gestionnaire. Un agent a en sa possession une liste d'objets. Un objet indique l'état de fonctionnement d'un dispositif. Un exemple d'objet d'un agent est l'état d'une interface de routeur. Les objets de l'agent sont organisés dans un Management Information Base (MIB), qui est un référentiel des objets gérés. Une MIB possède une structure standard dans laquelle les objets gérés sont traités par un identificateur d'objet (OID). Un objet dans une MIB est accessible à distance par les opérations fournies via le protocole SNMP.

Le modèle de communication appliqué entre un gestionnaire et un dispositif est le modèle requête-réponse. Un gestionnaire accède aux objets stockés dans la MIB d'un agent à travers l'instruction SNMP GET et les opérations de SNMP GET-NEXT. Un agent répond à ces opérations SNMP en envoyant SNMP GET RESPONSE contenant la valeur de l'objet demandé. L'instruction SNMP SET permet de modifier la valeur d'un objet MIB. Le protocole SNMP est conçu comme un protocole de surveillance. Afin d'ajouter de la réactivité au protocole, les concepteurs de SNMP ont ajouté l'opération SNMP TRAP. Cette opération permet à un agent d'émettre une notification à un gestionnaire sur un événement de périphérique spécifique. Le TRAP SNMP est envoyé sans la demande du gestionnaire. Bien que le SNMP ait été conçu pour les réseaux informatiques, il est adaptable dans un environnement VLSI. Dans [LA04] le protocole SNMP est utilisé pour gérer une plateforme de test pour les systèmes sur puce en utilisant la norme de test P1500.

3.1.1 - Une architecture basée sur SNMP pour une plateforme d'émulation de NoC sur FPGA

Nous proposons une architecture basée sur le protocole SNMP pour gérer une plateforme d'émulation de NoC sur FPGA. Un protocole de communication entre un ordinateur hôte et un FPGA cible est développé. On peut alors configurer les nœuds d'émulation puis extraire les résultats d'émulation. Les nœuds d'émulation sont organisés comme

des périphériques SNMP. Ils sont reliés à un bus de communication indépendant du NoC ciblé.

3.1.2 - Présentation de l'architecture

L'architecture de haut niveau de la plateforme d'émulation proposée est illustrée dans la figure 4. La plateforme d'émulation est constituée de composants matériels synthétisables et de composants logiciels. Les composants matériels sont destinés à être mis en œuvre dans un FPGA cible, tandis que les composants logiciels sont développés pour être exécutés sur un ordinateur hôte.

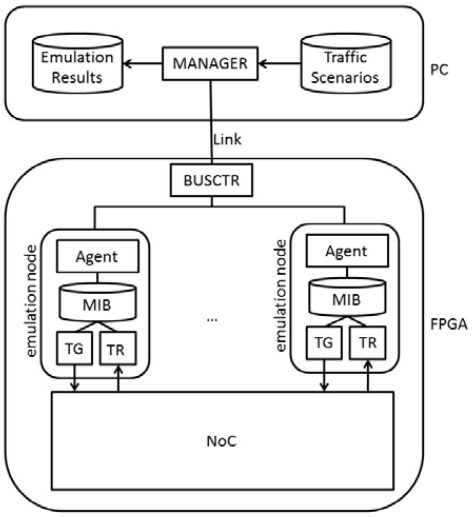


Figure 4: Aperçu de l'architecture d'interface SNMP entre le PC hôte et le NoC à émuler.

Dans un premier temps, nous présentons les composants matériels. Un nœud d'émulation est une implémentation matérielle d'un dispositif géré qui est directement relié à un routeur du NoC. Ce noeud est composé de quatre éléments : l'agent, la MIB, le TG et le TR. L'agent est chargé de décoder et d'exécuter les commandes de gestion. La MIB est une banque de registres adressables, qui contient tous les registres de contrôle et d'état. Ces registres contrôlent le comportement de la génération de scénario de trafic, et stockent les résultats d'émulation. L'adresse de chaque registre MIB est équivalente à l'OID dans la mise en œuvre du SNMP standard. La MIB est accessible par les composants de trafic à travers une interface simple, ce qui permet la lecture et l'écriture dans les registres de la MIB.

Les deux derniers composants matériels sont les TG et TR qui sont responsables de la génération et de la récupération de données dans les scénarios de trafic. Ces composants ont été adaptés à la plateforme d'émulation présentée dans [TFR11]. Le TG est connecté au port local d'entrée d'un routeur. Le TR est relié au port local de sortie d'un routeur. Dans cette mise en œuvre, le trafic est généré en fonction des modèles de micro-benchmark comme le trafic transpose, bit reverse, et autres types de trafic. Il est également possible de créer des scénarios de trafic, scénarios décrits comme des graphes de tâches avec un nombre arbitraire de nœuds source et destination. Le TR reçoit et traite le trafic puis produit des informations synthétiques relatives à la performance de chaque lien de communication.

Les nœuds d'émulation sont connectés à un bus de communication dédié, qui est commandé par le composant BUSCTR. Ce composant est responsable de l'interface entre le gestionnaire de plateforme et les périphériques gérés. Le BUSCTR décode les paquets de gestion envoyés par le gestionnaire et les retransmet au nœud d'émulation cible à travers un bus de communication dédié. Ce bus interne peut être implémenté en utilisant un standard de protocoles de bus sur puce. Un bus multi-hiérarchique pourrait également être utilisé en fonction du nombre de nœuds d'émulation.

Les composants logiciels constituent le gestionnaire, les scénarios de trafic et les résultats d'émulation. Le gestionnaire est responsable du contrôle de la plateforme et de ses composantes afin d'émuler un NoC. Le gestionnaire est implémenté sous forme d'une bibliothèque C qui communique avec la plateforme d'émulation via une liaison série. Cette bibliothèque permet l'intégration facile de la plateforme d'émulation dans des systèmes tiers. Le protocole de communication appliqué est inspiré par le protocole SNMP, mais il n'est pas ciblé pour une mise en œuvre TCP/IP. Le composant des scénarios de trafic est un ensemble de cas de tests qui peuvent être générés par des outils spécifiques comme l'outil TGFF [DRW98b], qui génère des graphes de tâches aléatoires. Le dernier composant est le composant des résultats d'émulation, qui est une synthèse des résultats de performance recueillis au cours du processus d'émulation. Ces résultats sont utilisés pour l'exploration spatiale, l'étalonnage ou la validation d'une mise en œuvre d'un NoC.

3.2 - Discussion des résultats

L'exécution des opérations SNMP est indépendante de la taille des NoC et ces opérations représentent une très faible fraction (0,004 % - 1,7 %) du temps consacré à l'exécution des scénarios de trafic. En outre, l'utilisation de la mise à jour différentielle de la MIB réduit de 54 % la quantité de données échangées entre le FPGA et l'ordinateur hôte. Ainsi, elle permet la mise à jour rapide du contenu de la MIB même si la liaison série entre le FPGA et le PC hôte représente un goulot d'étranglement. Un lien avec une bande passante plus élevée pourrait permettre la mise à jour plus rapide de la MIB et par conséquent l'exécution d'émulation plus rapide.

L'analyse des ressources pour les deux implémentations de la plateforme d'émulation proposée, indique que les composants SNMP ne représentent qu'une petite fraction de la plateforme globale d'émulation. La MIB est le seul composant SNMP qui dépend de la complexité des scénarios de trafic mis en œuvre, car elle doit contenir tous les paramètres relatifs à la description de modèle de trafic. Pour la carte d'évaluation ML605, il est possible de mettre en œuvre la plate-forme d'émulation pour le NoC 8x8 (scénarios micro-benchmark) et un NoC 7x7 (scénarios de graphe de tâches). La faible occupation des ressources des composants SNMP est due aux circuits légers qui sont en charge de mettre en œuvre les fonctions SNMP.

Les résultats obtenus indiquent que le modèle de communication SNMP est une solution rapide et légère pour conduire une plate-forme d'émulation de NoC. En outre, les structures pour mettre en œuvre les composants SNMP sont assez simples.

4.0 - La synthèse des plateformes d'émulation de NoC à partir de traces d'application

Les outils d'émulation de NoCs sur FPGA [GADM+05, LPG11] accélèrent des « benchmark » ainsi que l'exploration de l'espace de conception. Ces plateformes sont d'une haute précision et ont un faible temps d'exécution par rapport aux logiciels de simulation de NoC. Les principales propositions d'émulateurs de NoCs [**?**, KCdlTR08, WHS07] ciblent des modèles de trafic synthétique, à base de traces ou statistiques. L'exécution du trafic des applications réelles au cours d'une émulation de NoC est assez complexe. La limitation des ressources d'un FPGA est le principal obstacle à la reproduction de trafic d'application. Dans certains cas, des traces de communication obtenues à partir de simulations système complet sont disponibles. L'exécution de trafic basé sur traces est généralement plus simple que l'émulation système complet. Cependant, quelques pièges apparaissent avec l'utilisation du trafic basé sur traces pour l'émulation de NoC sur FPGA.

Une seule trace peut facilement comprendre des millions de paquets. Par exemple, comme indiqué dans [MNFA14], une application de référence FFT avec une taille de problème de 16 millions de points de données complexes génère un fichier de trace d'environ 28 Go. La taille des traces est une question importante pour l'émulation du NoC en raison de la taille limitée de RAM interne disponible sur FPGA. Les traces sont souvent allouées sur la RAM externe, ce qui présente un goulot d'étranglement pour la génération du trafic en temps réel.

L'exécution basée sur traces déforme le taux d'injection et les effets de la congestion due au manque d'informations de dépendance des paquets. Pour mieux comprendre les effets de l'absence de cette information, nous présentons un exemple simple. Considérons le tableau 4.1. Ce tableau présente une trace d'une simulation système complet. Cette simulation est exécutée sur un NoC où la latence entre une paire de nœuds est de 2 cycles d'horloge. La figure 5a représente la reproduction de cette trace dans un NoC avec la même latence. Les temps de transmission et de réception de chaque paquet sont également représentés sur cette figure. La même trace est reproduite dans un autre NoC. Ce NoC possède 6 cycles d'horloge de latence. La figure 5b illustre cette exécution. Des différences apparaissent entre ces deux exécutions. La deuxième exécution est plus longue que la première et a également une latence moyenne plus élevée. Ces différences sont dues au fait que chaque réseau présente des latences distinctes.

Ensuite, nous analysons comment ces différences impactent les relations de dépendances de paquets. Supposons qu'il existe une relation de dépendance entre les paquets 1 et 2 et une autre entre les paquets 2 et 3. Le nœud C reçoit le paquet 1, il rassemble les données du paquet et exécute une opération qui demande un temps d'exécution défini. Ensuite, le résultat de cette opération est envoyé au nœud B. Ce nœud traite les données et envoie les résultats au nœud D après un certain temps d'exécution. Il est important de noter que la trace ne contient pas les informations de dépendance des paquets.

Dans l'exécution représentée sur la figure 5a pour le NoC avec 2 cycles d'horloge de latence, le paquet 1 est reçu au cycle 12. Après un certain temps d'exécution, le nœud C envoie les résultats de l'opération au nœud B au cycle 14. Le noeud B traite les données et envoie les résultats au cycle 18. Cette exécution reproduit correctement le comportement de l'application parce que le réseau présente la même latence que celle du réseau utilisé pour recueillir les traces. L'exécution des traces sur le deuxième réseau

présente des problèmes, représentés sur la figure 5b. En utilisant l'information codée dans les traces, le paquet 2 (résultats des calculs) est envoyé au cycle 14. Cependant, le paquet 1 (données d'entrée) est reçu au cycle 16. Les résultats de l'opération sont envoyés avant la réception des données d'entrée. Le même phénomène se produit avec le paquet 3 qui est envoyé avant la réception du paquet 2. Dans cette simulation, les paquets sont injectés avec un taux supérieur à celui attendu. Cet exemple montre la fragilité de trafic basé sur trace.

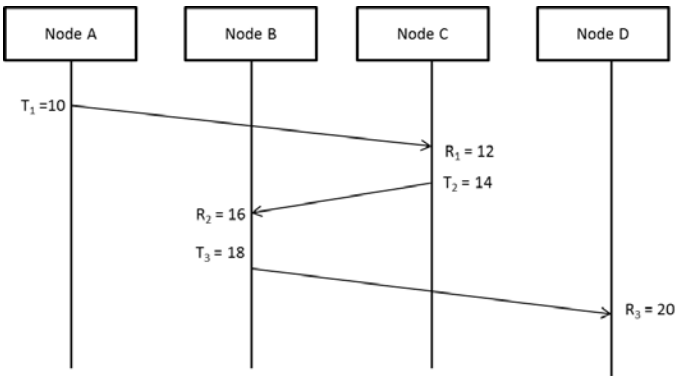
Paquet	Cycle Transmission	Cycle Réception	Source	Destination
1	10	12	A	C
2	14	16	C	B
3	18	20	B	D

Table 2: Exemple de trace

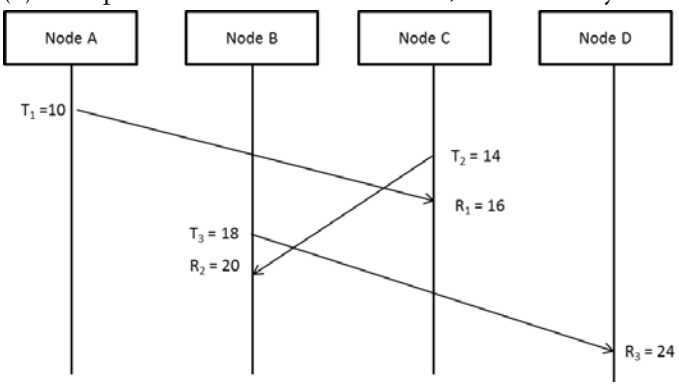
Afin de faire face aux problèmes d'exécution basée sur des traces, certains travaux [NMFA11, HCT+12, TAA+11, MNFA14] proposent l'utilisation de traces en tenant compte de dépendances. Ces traces sont alors enrichies avec les informations de dépendance des paquets. Elles sont donc plus volumineuses que les traces d'origine, mais elles sont plus précises pour une large gamme d'architectures de réseau. La présence des informations de dépendance assure un entrelacement approprié des paquets, ce qui augmente la fidélité de la simulation en éliminant la congestion artificielle créée par la simulation basée sur traces. Pour mieux comprendre les avantages de ces traces, nous revisitons le dernier exemple en tenant compte de dépendance. La figure 5c représente l'exécution de cette trace sur le NoC avec 6 cycles d'horloge de latence. Les temps de transmission et de réception de chaque paquet, ainsi que le temps d'exécution pour traiter chaque paquet sont également représentés dans cette figure. Dans ce cas, le fait d'utiliser les informations de dépendance conduit à une reproduction plus précise du comportement de l'application. Chaque nœud reproduit le comportement spécifié par l'application, car ils attendent la réception des données d'entrée. Ensuite, ils reproduisent le temps d'exécution exigé pour chaque opération. Enfin, les résultats sont envoyés sur les paquets de sortie.

Dans ces trois exemples, le taux d'injection de paquets présent dans les traces est recueilli et dépend donc du réseau utilisé pour générer les traces. Cette même trace, utilisée sur un autre réseau, va présenter un taux d'injection de paquets déformé. Cependant, l'utilisation des informations de dépendances des paquets améliore la précision de ces traces. En conclusion, une question majeure se pose : comment extraire les dépendances de paquets ? Une solution naturelle à ce problème est l'instrumentation du code d'application et l'utilisation d'un simulateur pour générer des traces en tenant compte des dépendances. Toutefois, cette approche implique des efforts considérables pour modifier le code d'application, certaines applications étant très complexes et dépendent de nombreuses bibliothèques et de code hérité. Ainsi, il est important d'étudier les approches utilisées pour extraire les dépendances de paquets à partir d'une trace.

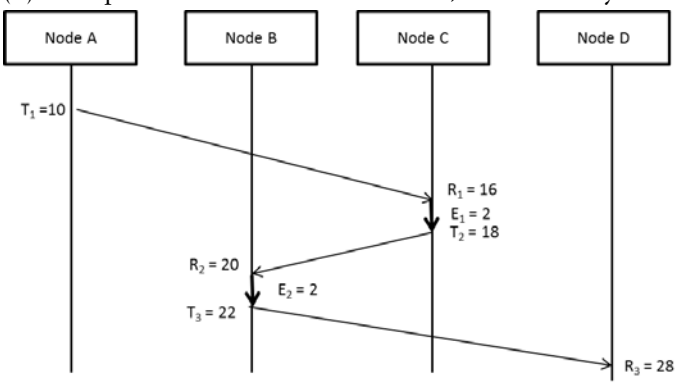
Pour illustrer le problème de l'extraction des dépendances des paquets à partir d'une trace, nous présentons un exemple simple. La figure 6 représente l'échange de certains



(a) Exemple d'exécution basée sur trace, latence = 2 cycles d'horloge.



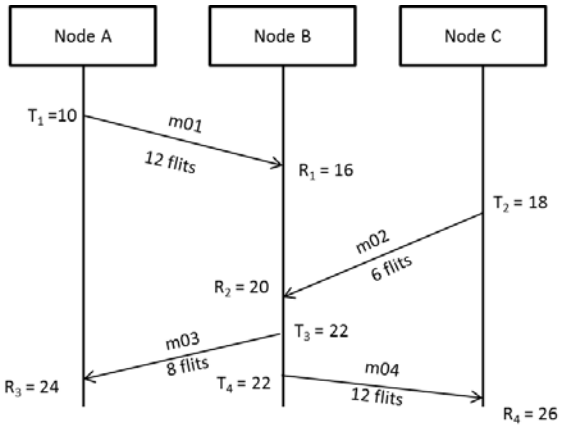
(b) Exemple d'exécution basée sur trace, latence = 6 cycles d'horloge.



(c) Exemple d'exécution basée sur trace en tenant compte de dépendances, latence = 6 cycles d'horloge.

Figure 5: Exemples d'exécutions basées sur traces.

paquets par un groupe de nœuds ainsi que la trace recueillie par cet échange. En utilisant seulement les informations disponibles dans cette trace (la source, la destination, la longueur des paquets, le temps d'envoi et le temps de réception), nous en déduisons sept relations possibles de dépendance des paquets. La conclusion est basée sur l'hypothèse selon laquelle il existe une relation de causalité entre les paquets de sortie et un ou plusieurs (ou tous) les paquets d'entrée. Ces relations sont présentées dans le tableau **??**. Nous ne pouvons pas dire lesquelles de ces relations de dépendance sont les vraies. Il est évident qu'une procédure d'analyse des traces est nécessaire afin de récupérer les vraies dépendances des paquets.



ID	SOURCE	DEST	LENGTH	TIME SEND	TIME RECEIVED
m01	A	B	12	10	16
m02	C	B	6	18	20
m03	B	A	8	22	24
m04	B	C	12	22	26

Figure 6: Exemple du problème d'extraction de dépendances de paquets

Dépendances	Paquets d'entrée	Paquets de sortie
1	m01	m03
2	m01	m04
3	m02	m03
4	m02	m04
5	m01,m02	m03,m04
6	m02	m03,m04
7	m01	m03,m04

Table 3: Les dépendances possibles.

Les précédentes propositions relatives aux dépendances dans les traces [HGK10, TAA+11, MNFA14, HCT+12] ciblaient des simulations logicielles. Certaines d'entre elles dépendaient aussi de l'architecture du réseau utilisé pour générer les traces analysées ou dépendaient des modifications d'un code d'application. Le principal défi de ces propositions est l'identification des dépendances de paquets. Ce défi est encore plus grand lorsque le code d'application n'est pas disponible. Dans ce cas, les dépendances des

paquets doivent être extraites de l'analyse des traces d'une seule simulation de système complet. Ceci constitue le sujet principal de ce chapitre, qui est donc l'extraction de dépendances des paquets d'une seule trace recueillie d'une architecture de NoC inconnue.

Afin d'extraire les relations de dépendance à partir d'une seule trace, nous considérons certains principes. Le premier principe est que les vraies dépendances des paquets sont présentes dans les traces. Nous supposons également que certaines dépendances sont fausses, elles sont le résultat de la distribution de la charge d'injection sur le NoC utilisé pour recueillir les traces. Les dépendances sont déduites à partir des informations d'en-têtes des paquets, ainsi que l'ordre et la durée des paquets sur les traces. Nous évaluons plusieurs combinaisons de dépendances des paquets afin de trouver la meilleure combinaison possible pour une trace donnée.

Un modèle léger de traces avec dépendances peut être utilisé pour exécuter le trafic des applications pour des émulations de NoC sur FPGA. Dans ce chapitre, un environnement d'analyse de traces est proposé pour synthétiser les générateurs de trafic (TG) en tenant compte des dépendances pour l'émulation des NoCs. Cet environnement est conçu pour traiter des traces générées par des applications modélisées sous forme de graphes de tâches. Les traces sont analysées afin de créer un Modèle de Calcul (MoC) qui remplace avec précision les nœuds de l'application dans les futurs émulations de NoC. Le principal avantage de cette approche est que la réalisation matérielle du MoC obtenu nécessite moins de ressources matérielles qu'une trace régulière. En outre, les résultats expérimentaux mettent en évidence que le MoC est plus précis que des traces régulières pour un large panel d'architectures de NoC.

4.1 - Un outil d'analyse des traces

L'objectif de l'analyse des traces est d'étudier le comportement communicatif d'une application basée sur un NoC. La contrainte des ressources est l'un des principaux problèmes dans l'émulation de NoC sur FPGA. Un concepteur doit mettre en œuvre l'architecture NoC ainsi que les composants du modèle de trafic dans un ou plusieurs FPGA pour exécuter une évaluation de performances. Il est important que les composants du modèle de trafic aient des besoins faibles en ressources tout en reproduisant avec précision le trafic des applications. Dans ce travail, nous abordons la question de la construction des composants légers et précis, de modèles de trafic basés sur une seule trace d'une application. Ce problème est traité en deux étapes. La première étape est la génération d'un MoC à partir d'une trace de l'application régulière. La deuxième étape est la conception d'une architecture d'un TG en tenant compte des dépendances.

Un outil d'analyse des traces est proposé pour récupérer les dépendances de paquets puis pour créer un tableau de dépendance (DT). Un DT est la synthèse du comportement réactif d'un nœud d'application. Le DT est ensuite synthétisé comme TG pour l'émulation d'un NoC. La figure 7 donne un aperçu d'outil d'analyse des traces. Le premier processus consiste à extraire une liste de motifs de dépendance (DPs) par l'analyse des traces de réception et de transmission. Une DP représente une règle logique utilisée pour générer des paquets à partir des paquets reçus précédemment. Les paquets envoyés avant toute réception de paquets sont regroupés sous forme de MGT . Le reste des paquets présente des relations de dépendance qui permet de créer une liste de DPs. Ces DPs sont fusionnés dans la mesure du possible afin de créer un DT initial.

Le second processus consiste en l'extraction d'une Base d'Ensembles (BS pour Base Set). Une BS est un ensemble de DT générés par le DT initial. Chaque DT dans une BS modélise le comportement du même nœud d'application. La BS représente un espace de solutions, dans lequel des solutions sont plus précises que d'autres. Il est nécessaire d'évaluer ces DT afin de trouver la plus précise. Le troisième processus consiste à évaluer la BS. Le processus d'évaluation utilise les traces originales afin de vérifier si le DT évalué est capable de reproduire le comportement enregistré dans les traces. Un indice d'imprécision est calculé à partir de cette évaluation. Le DT avec l'indice le plus petit est considéré comme le bon.

La prochaine étape dans le flot de conception est la construction d'une architecture de TG en tenant compte de dépendances à partir des MoC extraits dans l'étape précédente. Le défi consiste à concevoir des composants de modèles de trafic légers mais précis. Afin d'optimiser les besoins en ressources, un TG spécifique est créé pour chaque tâche d'application. Ce TG reproduit seulement le comportement d'une tâche cible pour réduire l'utilisation des ressources. Le TG généré peut être connecté au NoC cible pour reproduire le comportement de l'application évaluée.

4.2 - Analyse des résultats

Les expériences sont menées pour évaluer deux caractéristiques de générateurs de trafic : les besoins de stockage de données et la fidélité par rapport au trafic réel des applications. L'objectif dans la conception d'un générateur de trafic pour l'émulation NoC est d'avoir un espace minimal pour le stockage de données, tout en gardant une haute-fidélité de trafic. Nous analysons et discutons ci-après des résultats des expériences présentées.

Les traces exigent plus de ressources matérielles pour stocker ses données , car elles contiennent des informations sur chaque paquet traversant le réseau. L'outil utilise des DPs pour remplacer les échanges répétitifs de paquets. Ces DPs sont organisés sous forme d'un DT qui synthétise le comportement réactif de communication d'un nœud. Le stockage de données d'un DT est moins important que le stockage des traces complètes. L'outil contient également les données pour le MGT qui représentent le comportement proactif de communication d'un nœud. Toutefois, les obligations de stockage de données d'outil MoC représente en moyenne 3 % du stockage de données pour une trace. Cette exigence se retrouve dans l'occupation FPGA d'un nœud d'émulation, qui est la mise en œuvre matérielle d'un outil MoC. Un nœud d'émulation occupe seulement 2,01 % des registres et 0,75 % des LUT de la plateforme globale.

Afin d'analyser la fidélité du trafic, nous avons besoin de comprendre pourquoi un générateur de trafic produit un trafic de basse fidélité. Dans la section 6.2, nous montrons les effets de l'absence des informations de dépendance de paquets sur le trafic basé sur les traces. L'exécution basée sur les trace déforme le taux et les effets de la congestion. Cette déformation est mise en évidence lorsque les traces sont exécutées dans un NoC avec une latence différente, comme illustré dans la figure 4.1. Ainsi, il est important de comprendre quels sont les paramètres du réseau qui influent sur la latence des paquets. Nous présentons une équation pour calculer un temps de latence des paquets sur un NoC non congestionné : $= latenceTime_{FLIT} * (Packet_{size} - 1) + (Time_{HEADER} + Time_{FLIT}) * N_{hops}$ houblon , dans lequel $Time_{FLIT}$ est le nombre de cycles d'horloge pour envoyer un flit, $Packet_{size}$ est le nombre de flits d'un paquet, $Time_{HEADER}$ est le nombre de cycles d'horloge pour traiter un en-tête de paquet, et N_{hops} est le nombre de routeurs dans

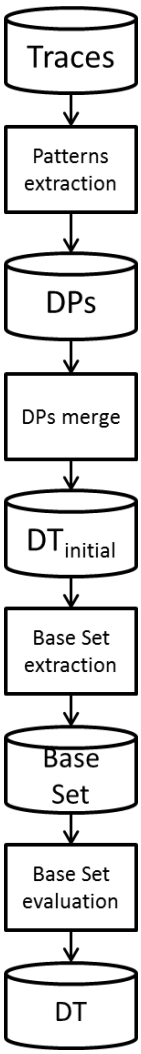


Figure 7: Vue de l'ensemble de l'outil d'analyse de traces.

le chemin d'un paquet comprenant la source et la destination. Dans nos expériences, les trois premiers paramètres restent inchangés, cependant, le dernier paramètre N_{hops} change car il dépend de l'algorithme de routage, de la topologie et du placement des tâches.

Au cours des expériences réalisées, nous avons exécuté des traces et des outil MoC pour un large éventail de NoCs. Nous avons joué avec différents paramètres de réseau tels que : la taille des « buffer », l'algorithme de routage, la topologie, et le placement des tâches. Désormais, nous analysons l'impact de ces paramètres de réseau sur la fidélité du trafic.

Les expériences mettent en évidence l'influence de la taille des « buffers » sur la précision des simulations basées sur des traces. Il existe une nette différence de précision entre les traces et les « outil » pour des scénarios 2-flits et 4-flits. Cependant, cette différence est très faible pour des scénarios 32-flits et 16-flits. Une petite taille de « buffer » réduit la bande passante du réseau, ce qui rend l'information temporelle codée dans les traces incorrectes. Les « buffers » sont des éléments importants dans l'atténuation des

effets de la congestion du réseau. Toutefois, les petites tailles de « buffer » sont moins efficaces pour palier à ces effets. Ainsi, la latence sur les réseaux de petits « buffers » est plus grande. Les informations de dépendances des paquets permettent aux MoC d'adapter la génération de trafic vers les réseaux. Il est intéressant de souligner que le « outil » présente de petites variations sur les erreurs pour différentes configurations de réseau. Le contraire n'est pas vrai pour la simulation basée sur les traces comme indiqué pour les scénarios 2-flits ou 4-flits.

Ces expériences nous amènent à quelques conclusions intéressantes. Dans un premier temps, la fidélité du trafic basé sur les traces dépend des différences entre le réseau utilisé pour recueillir les traces et le réseau en cours d'évaluation. Quand le réseau en cours d'évaluation est similaire au réseau utilisé pour recueillir les traces, la trace possède une performance similaire au MoC. Toutefois, lorsque le réseau en cours d'évaluation est très différent du NoC d'origine, le MoC a une meilleure performance. Afin de décider quel générateur de trafic doit être utilisé, il est nécessaire d'avoir un modèle très précis des deux NoC. Cependant, ces informations ne sont pas toujours disponibles. Dans la plupart des cas, il n'y a pas d'informations du NoC utilisé pour recueillir les traces. Ainsi, le outil MoC présente un avantage important par rapport aux traces ordinaires parce qu'il peut être utilisé pour un large panel de configurations de réseaux avec une bonne précision. En outre, le MoC requiert un petit espace de stockage de données, ce qui en fait une option intéressante dans la génération du trafic pour les émulations de NoC sur FPGA.

5.0 - Vers les NoC des générations futures : une analyse prospective

La complexité croissante des systèmes embarqués exige l'intégration d'un grand nombre de coeurs IP interconnectés par des mécanismes avancés de réseaux sur puce. Ces mécanismes doivent offrir une faible latence et des services de communication à haut débit pour plusieurs flux de données, même en présence de défauts matériels ou de congestion du réseau. En outre, ces flux de données exigent différents niveaux de qualité de service en fonction des besoins des applications. Au cours de la dernière décennie, la recherche sur les NoC a été axée sur le développement de mécanismes de mise en réseau avancés tels que la QoS [ACVCGM14, SMG14, W+09, AMP06], le contrôle de congestion [OW14, CC+12, HH09, HM04], la tolérance aux pannes [FHLL14, KTMD14, ZP10, PLB+04], le multicast [SGR13, ZWGLB14, EDLT10, LYJ06] entre autres. La plupart de ces propositions se concentrent sur des topologies standard de maillage 2D. Cependant, les progrès récents dans la technologie de circuits 3D a provoqué un changement de paradigme sur le scénario d'interconnexion sur puce.

Un NoC 3D présente un débit plus élevé, une plus petite latence et une plus faible consommation d'énergie qu'un NoC 2D [FP07, PF07]. Ces réseaux sont constitués de plusieurs couches, chacune d'entre elle peut mettre en œuvre une technologie différente de semi-conducteurs. Ces puces à technologie mixte doivent offrir des services de réseau de haut niveau à travers différentes couches assurant la performance des flux de communication intra et inter-couche. Bien que la technologie 3D ne soit pas entièrement maîtrisée, elle possède un fort potentiel pour révolutionner le contexte de NoCs et pour introduire une nouvelle génération de NoC (NGNoCs pour Next Generation NoC). Un

NGNoC est un réseau divisé en couches hétérogènes dans un circuit 3D, ces couches sont reliées par quelques liens transversaux. Les couches et les connexions inter-couches peuvent avoir des contraintes physiques spécifiques. Chaque couche implémente un NoC indépendant avec sa propre topologie, son nombre de nœuds, et son schéma de routage, parmi d'autres paramètres. Les mécanismes avancés de mise en réseau tels que la QoS, le contrôle de congestion, la communication multicast, et la tolérance aux pannes sont mis en œuvre dans chaque couche, en fonction des besoins de communication. Il est possible que certaines couches mettent en œuvre des mécanismes complètement différents ou même que certains de ces mécanismes ne soient pas du tout utilisés.

Dans cette section, nous explorons une architecture conceptuelle d'un circuit 3D de type NGNoC pour des applications de systèmes embarqués. Cette architecture conceptuelle sert de modèle d'étude pour explorer les exigences de communication de haut niveau d'un NGNoC. Nous nous concentrons sur un système constitué de 97 cœurs déployés sur trois couches. Le nombre de cœurs n'est pas très élevé, mais il suffit pour déterminer le flux de communication intra- et inter-couches. En outre, ce modèle comprend suffisamment de détails pour décrire les principales fonctions d'un système embarqué multi-cœurs. Plusieurs caractéristiques fonctionnelles ne sont pas abordées ici car elles n'influent pas sur l'architecture du réseau.

Ce système contient trois couches d'application différentes comme le montre la Figure 8. La première couche est dédiée aux blocs de calculs. Chaque bloc de calcul (PE) possède sa propre mémoire, ainsi le partage des données s'effectue par passage de messages. Les PEs ont des architectures hétérogènes, pouvant être des cœurs d'usage général, des RISC, des unités GPU (Graphical Processing Unit) et d'autres IPs dédiés au calcul spécifique comme la cryptographie. Des blocs programmables logiques sont également mis en œuvre dans cette couche. Ils permettent la reconfiguration matérielle en temps réel et peuvent être utilisés pour mettre en œuvre des blocs de traitement dédiés.

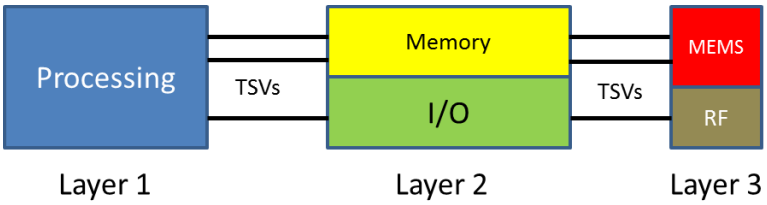


Figure 8: Architecture conceptual de NGNoC sur un circuit 3D.

La seconde couche comprend deux domaines d'utilisation différents. Le premier domaine est dédié au stockage et met en œuvre des blocs de mémoire rapide. Ces blocs sont utilisés par l'application pour le stockage des données et du code. Ils sont également utilisés comme une zone de mémoire partagée secondaire pour les PEs et d'autres cœurs. Ces blocs sont reliés à un contrôleur de mémoire externe par un bus. Le deuxième domaine implémente des contrôleurs d'E/S rapides et des processeurs audio/vidéo. Ils relient la puce à un ensemble de dispositifs externes comme des bus rapides, des écrans tactiles, des caméras haute résolution et des interfaces réseau. La troisième et dernière couche a également deux domaines distincts. Le premier met en œuvre un ensemble de MEMs comme des accéléromètres, gyroscopes, etc. Le deuxième domaine implémente des interfaces RF pour la communication courte distance comme

le Bluetooth ou RFID.

5.1 - Analyse de requêtes de contrôle de congestion de l'architecture conceptuelle

Dans notre architecture conceptuelle de NGNoC, la couche 1 possède le plus grand nombre de routeurs et exécute aussi la plupart des flux de communication. La mise en œuvre de mécanismes de contrôle de congestion peut aider à séparer le trafic intra et inter-couche. Les routeurs « ascenseurs » sont responsables des liaisons entre les couches qui sont susceptibles d'être plus congestionnées. Un algorithme de routage en tenant compte de congestion écarte automatiquement le trafic intra-couche des routeurs « ascenseurs ». On s'attend à un trafic intense entre les PEs qui sont sur une courte distance parce que les applications peuvent exploiter le regroupement de PE pour exécuter des applications informatiques intensives. Le trafic à distance moyenne et longue sera principalement des paquets destinés aux routeurs « ascenseurs ».

La couche 2 intègre un nombre moyen de routeurs, mais concentre le plus grand nombre de TSVs (pour Through-Silicon Vias). Cette couche possède un trafic intense, car on s'attend à ce que les blocs de mémoire soient utilisés par les composants des trois couches. Une fois de plus, l'adoption d'un mécanisme de contrôle de congestion aide à dévier le trafic en provenance des zones congestionnées, qui sont susceptibles d'être les zones entourant les routeurs « ascenseurs ». La dernière couche est la plus simple, elle ne dispose que de 16 nœuds. La plupart des flux de trafic de cette couche sont dirigés vers les autres couches. L'adoption d'un mécanisme de contrôle de congestion n'est pas obligatoire. Un système de gestion de réseau simple impose des exigences de communication moins strictes sur cette couche. Prenons l'exemple illustré à la figure **??**. Les carrés noirs représentent les ascenseurs routeurs. Les carrés gris sont des routeurs qui envoient des paquets inter-couches. Les flèches représentent les flux de trafic. Dans cet exemple, un routeur ascenseur transporte le trafic à partir de six routeurs, tandis que l'autre transporte un trafic d'un seul routeur. Cette inégalité de la charge provoque la congestion du réseau autour du premier routeur ascenseur. Afin de résoudre ce problème, il est nécessaire de choisir le routeur ascenseur en fonction de ses informations de congestion et sa position. L'affectation dynamique du routeur ascenseur est un défi parce que tous les autres routeurs doivent avoir une connaissance de l'état de chacun des ascenseurs. Ceci est encore plus difficile pour les grands réseaux avec quelques TSV, ce qui est le cas de la couche 1.

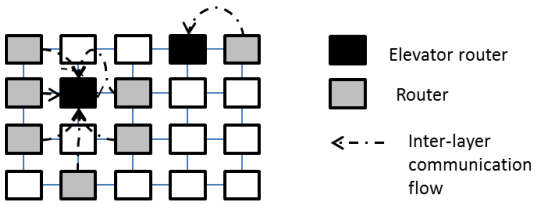


Figure 9: Exemple de l'utilisation de TSVs.

On pourrait proposer d'ajouter des fils supplémentaires entre les routeurs pour partager des informations de congestion des ascenseurs. Bien que simple en apparence, cette solution entraîne une complexité supplémentaire en raison de la quantité et la

taille de ce câblage. Une meilleure solution est d'explorer l'infrastructure de mise en réseau disponibles pour partager des informations de réseau importantes comme l'état de la congestion des ascenseurs.

5.2 - Perspectives sur le contrôle de congestion

Nous introduisons brièvement un mécanisme de « piggybacking » simple pour partager des informations de la congestion des routeurs ascenseurs. Ce mécanisme n'est pas complètement exploré dans le contexte de ce mémoire de thèse, mais il est envisagé en tant que l'un de nos futurs travaux de recherche. Le « piggybacking » est une technique de communication de données dans lequel un nœud récepteur ajoute des informations de reconnaissance (« acknowledge ») sur les paquets de données sortants destinés au nœud de transmission. Ce mécanisme est utilisé dans plusieurs protocoles MAC. Nous proposons de greffer les informations de la congestion aux paquets de données passant par les routeurs ascenseurs ou ses routeurs adjacents. Chaque fois que le paquet contenant les informations de congestions est géré par un routeur, ce routeur obtient ces informations et met à jour une table de données, dans laquelle est maintenu l'état de tous les routeurs ascenseurs. Quand un paquet inter-couche est transmis, ce tableau de données est consulté afin de déterminer quel ascenseur s'occupera du paquet.

L'avantage de ce dispositif est l'exploration du trafic de données habituel pour propager les informations de contrôle de congestion. Les informations de congestion peuvent être codées dans un seul flit. L'inconvénient est que cette solution n'est pas efficace en cas de faible charge de la circulation ou pour des scénarios où le trafic passant par les routeurs ascenseurs ne traversent pas tous les routeurs du réseau. En raison de cela, les tableaux de données des ascenseurs dans chaque routeur doivent être réinitialisés périodiquement. En outre, chaque fois qu'une table n'est pas à jour, le routeur ascenseur le plus proche doit être choisi.

Au-delà du contrôle de congestion de liens verticaux, il est nécessaire de faire face à la congestion intra-couche. Nous proposons de mettre en œuvre un algorithme basé sur un schéma simple et flexible d'ensembles de règles, en tenant compte de la congestion, ainsi que l'équilibre de la charge réseau. Ces ensembles de règles sont basés sur le modèle « turn model » Odd-Even, les chemins minimaux, l'information de la congestion des routeurs et la disponibilité du chemin de sortie. L'équilibrage de charge est atteint parce que l'algorithme proposé ne choisit pas toujours le chemin avec la plus faible latence. Le chemin de sortie est choisi grâce à la vérification des ensembles de règles. Il est important de noter que la séquence de vérification des chemins est mise à jour à chaque cycle d'horloge, de manière à ce que le trafic soit distribué pour de nombreux chemins. L'originalité de l'algorithme proposé est de faire face à la congestion dans le NoCs par un régime prioritaire d'ensemble de règles, ce qui permet de combiner différents types d'informations à partir du routeur et le réseau.

5.3 - L'analyse de requêtes sur la tolérance aux pannes de l'architecture conceptuelle

Dans notre architecture conceptuelle de NGNoC, les couches 1 et 2 possèdent le plus grand nombre de routeurs et réalisent également la plupart des flux de communication. La mise en œuvre de mécanismes de tolérance aux pannes pour cette couche est im-

pérative et consiste en une architecture de routeur tolérant aux pannes, dans laquelle un mécanisme BIST (Build-in Self-Test) évalue la circuiterie interne du routeur. Ce mécanisme évalue l'état des principaux composants du routeur comme les buffers, le « crossbar » , le routage et le contrôle de l'arbitrage. Même avec la mise en œuvre de redondance matérielle au niveau des routeurs, il est possible qu'un routeur soit en panne. L'adoption d'une solution de routage avec une tolérance aux pannes est obligatoire. Nous avons proposé l'utilisation du FlexOE comme schéma de routage pour la couche 1 et pour la couche 2. Nous devons adapter l'algorithme proposé pour gérer également les routeurs en panne.

L'algorithme FlexOE propose un schéma de routage flexible pour contrôler la congestion dans un NoC. Nous proposons d'utiliser le signal de défaillance du circuit BIST pour améliorer l'ensemble des règles FlexOE. La figure 10 représente la modification proposée. La règle du FlexOE la plus importante est remplacée par la vérification du signal de défaillance du routeur adjacent associée au routage Odd-Even. De cette façon, si le signal de défaillance n'est pas actif et que le port de sortie est sélectionné par Odd-Even, cette nouvelle règle est activée. Dans le cas contraire, le port de sortie n'est pas disponible pour le routage. Ceci constitue un changement relativement simple et léger, mais qui ajoute un caractéristique importante au NGNoC.

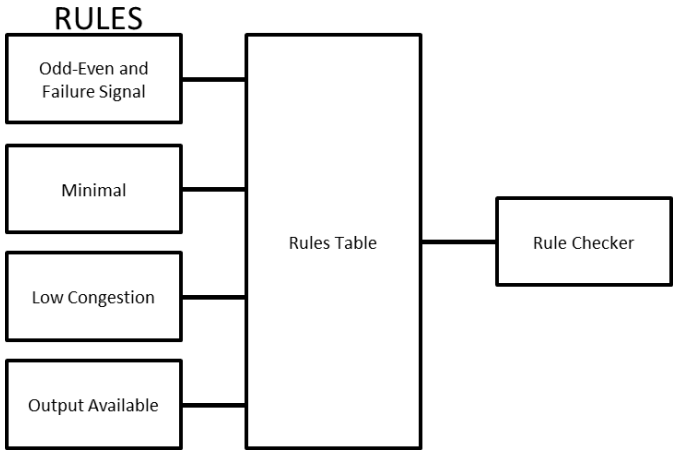


Figure 10: Version tolérante aux pannes du FlexOE.

Contrairement aux autres couches, la couche 3 possède un petit nombre de routeurs et porte également des flux de communication plus simples. Pour cette couche, nous proposons une architecture NoC tolérante aux pannes sur la base de liens externes redondants et sur une variante de l'algorithme XY. Les résultats de l'évaluation de performances montrent que l'architecture proposée garantit le succès à 100 % dans la livraison de paquets pour les scénarios de test allant jusqu'à 19 % de routeurs défectueux, tandis que les paquets ont une latence jusqu'à 90 % plus faible par rapport aux algorithmes de référence.

5.4 - L'émulation à base de FPGA sur NGNoCs

Les outils d'émulation à base de FPGA vont jouer un rôle primordial dans le développement de NGNoCs. La complexité de ces réseaux exige des outils d'évaluation de pointe

capables de vérifier leurs caractéristiques comme la qualité de service, le contrôle de congestion et les mécanismes de tolérances aux pannes entre autres. La vérification de ces caractéristiques est une préoccupation majeure dans l'émulation du NoC sur FPGA. Le composant principal d'un NoC est le routeur qui met en œuvre la plupart des fonctions de mise en réseau. Lors d'un prototypage de NoC, on s'intéresse à la validation d'un réseau avec un minimum de changements sur la circuiterie des routeurs.

Afin de vérifier le bon fonctionnement d'un NoC, il est nécessaire d'ajouter des composants d'instrumentation spéciaux aux environs des réseaux routeurs. Les composants d'instrumentation supplémentaires doivent être utilisés avec prudence, car ils peuvent modifier le comportement des éléments du réseau ou même ajouter des erreurs graves. Une solution intéressante est de placer les composants d'instrumentation sur les liens externes des routeurs. Cette approche n'ajoute pas de changements dans la structure des routeurs, mais elle permet de déduire la performance globale du réseau en analysant le comportement observable de chaque routeur.

Ces circuits de surveillance (LMCs pour Link Monitoring Circuits) sont polyvalents. Ils peuvent être utilisés pour compter le nombre de flits traversant un lien pour calculer la consommation dynamique d'énergie ou pour déterminer s'il y a congestion dans un réseau. Ils peuvent être également utilisés pour injecter des défauts permanents ou transitoires lors de l'évaluation du NoC. Ces LMCs doivent être accessibles à partir du gestionnaire d'émulation, qui doit les configurer et extraire les données générées lors de l'émulation. La figure 11 montre comment ces composants peuvent être ajoutés à l'architecture SNMP, architecture présentée dans la section 6.2. Dans cette architecture, les LMCs sont connectés à la MIB du nœud d'émulation, ce qui les rend visibles du gestionnaire. Les lignes en pointillées représentent les liens vers la MIB, qui doit être modifiée pour maintenir les nouveaux registres liés aux LMC. Bien évidemment un plus grand nombre de ressources matérielles est utilisé pour mettre en œuvre cette nouvelle architecture. Un LMC peut également être utilisé pour surveiller les liens verticaux reliant les couches d'un NGNoC.

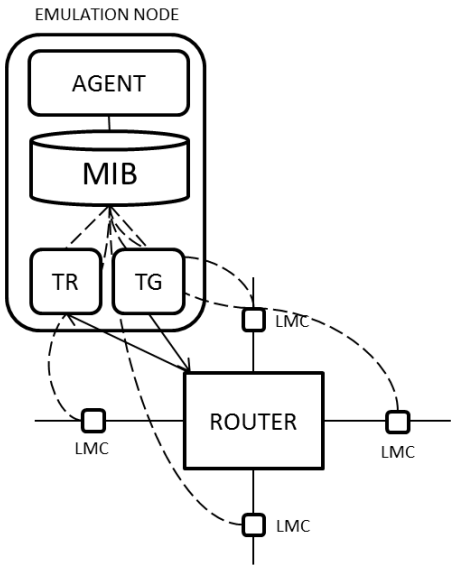


Figure 11: Un nœud d'émulation avec des liens LMCs.

Les LMC apportent plus de flexibilité pour un outil d'émulation à base de FPGA, mais en même temps ils ajoutent un grand défi de conception : l'augmentation du volume de données générées lors de l'émulation. Le projet de plateforme d'émulation SNMP utilise une stratégie très simple pour la récupération des résultats d'émulation. Dans cette architecture, les résultats sont récupérés après l'exécution d'un scénario de trafic. Pour ce faire, les nœuds d'émulation doivent conserver ces résultats au cours de l'émulation. Pour certains indicateurs de performances comme la latence moyenne, ceci ne constitue pas un gros problème.

Toutefois, lorsque le suivi des liens est ajouté, certains indicateurs de performances peuvent nécessiter beaucoup de ressources de stockage de données. Par exemple, un concepteur peut être intéressé par l'analyse de l'évolution de la consommation dynamique d'énergie lors d'un scénario de trafic. Pour ce faire, il est nécessaire de tenir compte du nombre de flits qui traversent une liaison donnée pour une fenêtre de temps déterminée. Ce comptage de flits est répété des milliers ou des centaines de milliers de fois au cours d'un scénario de trafic. Ainsi, le nœud d'émulation est obligé de contenir tous ces comptages de flits pour calculer la métrique de performance souhaitée. Bien sûr, il n'est pas possible de maintenir toutes ces données pour chaque lien d'un routeur dans la MIB d'un nœud d'émulation. Il est nécessaire d'envoyer continuellement ces mesures au gestionnaire au cours de l'émulation. Ce trafic se poursuit entre les nœuds d'émulation et le gestionnaire pointe l'amélioration nécessaire de l'interconnexion entre ces composants.

6.0 - Conclusions

Les plateformes d'émulation à base de FPGA joueront un rôle majeur dans le processus de conception des NGNoCs. Des fonctionnalités avancées de réseau telles que la QoS, le contrôle de congestion, les mécanismes de tolérances aux pannes seront des services communs à tous les NGNoCs. Ces caractéristiques devront être évaluées en profondeur lors du prototypage d'un réseau. La complexité de ces plateformes d'émulation augmente selon les caractéristiques de mises en œuvre sur ces réseaux. Ainsi, le nombre de composants d'émulation va croître de manière significative, ce qui entraîne la nécessité d'une gestion efficace entre ces composants. En outre, des protocoles standard pour la gestion des plateformes d'émulation de NoC sur FPGA sont une solution élégante pour faire face à la complexité de ces systèmes ainsi que pour fournir des interfaces de haut niveau pour contrôler l'exécution de scénarios d'évaluation. Les charges de travail réalistes sont également une préoccupation clé dans l'évaluation de performances des NGNoCs. La génération de trafic réaliste est un défi quant à l'émulation sur FPGA. Cependant il est essentiel d'évaluer les nouvelles architectures de NoC avec des trafics réalistes, car ils fournissent des résultats bien plus précis.

L'objectif de cette thèse est la synthèse de plateformes d'émulation sur FPGA pour les NoCs ainsi qu'une analyse prospective des exigences des NGNoCs. Nous avons proposé différentes solutions pour faire face aux principaux défis de ces plateformes. Dans un premier temps, nous avons proposé une version allégée du protocole SNMP pour gérer l'émulation d'un NoC. Un modèle basé sur les principes du SNMP a été mis en œuvre. Les composants d'émulation ont été organisés comme des agents dans lesquels leur comportement est contrôlé à distance en changeant les registres dans la MIB. Le modèle

de communication simple de requête-réponse permet la création d'un flux d'exécution. Ce flux d'exécution implique la configuration des nœuds d'émulation, l'exécution du scénario de trafic et l'extraction des résultats de l'évaluation. Ces tâches sont exécutées en utilisant des opérations SNMP.

Au-delà du protocole proposé et son flux d'exécution, une plateforme d'émulation de NoC sur FPGA est proposée. Cette plateforme permet l'émulation des NoCs contenant jusqu'à 64 nœuds pour un Virtex-6. Plusieurs expériences ont été menées pour évaluer le nombre d'opérations SNMP exécutées pour différentes applications, l'utilisation des ressources pour plusieurs tailles NoC et le temps nécessaire pour mettre à jour la MIB des agents. Les expérimentations mettent en évidence qu'une version allégée du SNMP est très efficace pour un faible ajout de ressources. De plus, il a également été présenté un cas d'étude dans lequel toutes ces propositions sont utilisées afin de réaliser l'exploration l'exploration de l'espace de conception d'un NoC.

L'outil évalue plusieurs combinaisons de motifs de dépendance afin de trouver la meilleure combinaison pour créer un tableau de dépendances. Ce tableau est un modèle du comportement réactif d'un nœud de l'application. Le comportement proactif est modélisé sous forme d'une petite table de paquets. L'outil proposé est évalué pour un large panel de configurations de NoC. Sa précision est évaluée pour différents algorithmes de routage, tailles de buffer, des mécanismes de contrôle de flux, de topologie 2D et 3D ainsi que de placements aléatoires des tâches. L'analyse expérimentale souligne que cet outil est plus précis que des simulations basées sur traces pour une grande variété de configurations de réseau. En outre, l'outil proposé n'utilise que 3 % de la taille de stockage de données requise par les traces.

Nous avons également analysé les perspectives des NGNoCs. D'un modèle conceptuel d'un système de génération future, nous avons extrait des exigences de communication au niveau d'une infrastructure de communication sur puce mise en œuvre sur un circuit 3D. Ces exigences de communication ont permis une analyse prospective des fonctionnalités de mise en réseau pour trois domaines clés : le contrôle de congestion, la tolérance aux pannes et l'émulation à base de FPGA. Nous avons proposé des mécanismes de mise en réseau pour ces sujets.

References

[ACVCGM14] E. Alceu Carara, N.L. Vilar Calazans, and F. Gehm Moraes. Differentiated communication services for NoC-Based MPSoCs. *IEEE Transactions on Computers*, 63(3):595–608, March 2014.

[Ala03] M.A. Alam. A critical examination of the mechanics of dynamic NBTI for PMOSFETs. In *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, pages 14.4.1–14.4.4, December 2003.

[AMP06] A. Agarwal, M. Mustafa, and A.S. Pandya. QOS driven network-on-chip design for real time systems. In *Canadian Conference on Electrical and Computer Engineering, 2006. CCECE '06*, pages 1291–1295, 2006.

[ASA+12] E. Antunes, M. Soares, A. Aguiar, F.S. Johann, M. Sartori, F. Hessel, and C. Marcon. Partitioning and dynamic mapping evaluation for energy consumption minimization on NoC-based MPSoC. In *2012 13th International Symposium on Quality Electronic Design (ISQED)*, pages 451–457, March 2012.

[BB09] J. H. Bahn and N. Bagherzadeh. A generic traffic model for on-chip interconnection networks. In *The First International Workshop on Networks-on-Chip Architectures*, 2009.

[BDM+05] M. Briere, E. Drouard, F. Mieyeville, D. Navarro, I. O'Connor, and F. Gaffiot. Heterogeneous modelling of an optical network-on-chip with SystemC. In *The 16th IEEE International Workshop on Rapid System Prototyping, 2005. (RSP 2005)*, pages 10–16, June 2005.

[BEA+08] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, Liewei Bao, J. Brown, M. Mattina, Chyi-Chang Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 - processor: A 64-core SoC with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 88 –598, February 2008.

[BIZCK12] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny. HNOCS: modular open-source simulator for heterogeneous NoCs. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 51–57, 2012.

[BJS14] P. Bahrebar, A. Jalalvand, and D. Stroobandt. Adaptive multicast routing method for 3D mesh-based networks-on-chip. In *System-on-Chip Conference (SOCC), 2014 27th IEEE International*, pages 70–75, September 2014.

[CC+12] H.-L. Chao, Y.-R. Chen, et al. Congestion-aware scheduling for NoC-based reconfigurable systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1561 –1566, March 2012.

[CCG+04] M. Coppola, S. Curaba, M.D. Grammatikakis, G. Maruccia, and F. Papariello. OCCN: a network-on-chip modeling and simulation framework. In *DATE 2004*, pages 174–179. IEEE Comput. Soc, 2004.

[CCM09] E. Carara, N. Calazans, and F. Moraes. Managing QoS flows at task level in NoC-based MPSoCs. In *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 133 –138, October 2009.

[CCW13] Chih-Hao Chao, Kun-Chih Chen, and An-Yeu Wu. Routing-based traffic migration and buffer allocation schemes for 3-d network-on-chip systems with thermal limit. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(11):2118–2131, November 2013.

[CdLJ13] H.S. Castro and O.A. de Lima Jr. A fault tolerant NoC architecture based upon external router backup paths. In *New Circuits and Systems Conference (NEWCAS), 2013 IEEE 11th International*, pages 1–4, June 2013.

[Chi00] G.-M. Chiu. The odd-even turn model for adaptive routing. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):729–738, July 2000.

[CHLW14] E.-J. Chang, H.-K. Hsin, S.-Y. Lin, and A.-Y. Wu. Path-congestion-aware adaptive routing with a contention prediction scheme for network-on-chip systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(1):113–126, January 2014.

[CM08] E. Carvalho and F. Moraes. Congestion-aware task mapping in heterogeneous MPSoCs. In *International Symposium on System-on-Chip, 2008. SOC 2008*, pages 1–4, November 2008.

[COM08] C.-L. Chou, U.Y. Ogras, and R. Marculescu. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879, October 2008.

[CSK14] T.V. Chu, S. Sato, and K. Kise. KNoCEmu: high speed FPGA emulator for kilo-node scale NoCs. In *2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs (MCSoc)*, pages 215–222, September 2014.

[D+13] F. Dubois et al. Elevator-first: A deadlock-free distributed routing algorithm for vertically partially connected 3D-NoCs. *IEEE Transactions on Computers*, 62(3):609–615, March 2013.

[DFB+12] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, Jin Hu, and G. Chen. A reliable routing architecture and algorithm for NoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(5):726 –739, May 2012.

[dLJFR13] O. A. de Lima Jr., V. Fresse, and F. Rousseau. FlexOE: a congestion-aware routing algorithm for NoCs. In *2013 International Symposium on Rapid System Prototyping (RSP)*, pages 51–57, October 2013.

[dLJFR14] O. A. de Lima Jr., V. Fresse, and F. Rousseau. Evaluation of SNMP-like protocol to manage a NoC emulation platform. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 199–206, December 2014.

[DM04] G. De Micheli. Reliable communication in systems on chips. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 77–77, July 2004.

[DRW98a] R.P. Dick, D.L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *(CODES/CASHE '98) Proceedings of the Sixth International Workshop on Hardware/Software Codesign, 1998*, pages 97–101, March 1998.

[DRW98b] R.P. Dick, D.L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *(CODES/CASHE '98) Proceedings of the Sixth International Workshop on Hardware/Software Codesign, 1998*, pages 97–101, March 1998.

[EDLT10] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and H. Tenhunen. HAMUM - a novel routing protocol for unicast and multicast traffic in MPSoCs. In *2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 525–532, February 2010.

[EDP13] M. Ebrahimi, M. Daneshtalab, and J. Plosila. Fault-tolerant routing algorithm for 3D NoC using hamiltonian path strategy. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1601–1604, March 2013.

[FGTR12] V. Fresse, Z. Ge, J. Tan, and F. Rousseau. Case study: Deployment of the 2D NoC on 3D for the generation of large emulation platforms. In *2012 23rd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 23–29, October 2012.

[FHLL14] B. Fu, Y. Han, H. Li, and X. Li. ZoneDefense: a fault-tolerant routing for 2-d meshes without virtual channels. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(1):113–126, January 2014.

[FP07] B. Feero and P.P. Pande. Performance evaluation for three-dimensional networks-on-chip. In *IEEE Computer Society Annual Symposium on VLSI, 2007. ISVLSI '07*, pages 305–310, March 2007.

[FSP14] S. Foroutan, A. Sheibanyrad, and F. Petrot. Assignment of vertical-links to routers in vertically-partially-connected 3-d-NoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(8):1208–1218, August 2014.

[FTHJ09] S. Foroutan, Y. Thonnart, R. Hersemeule, and A Jerraya. Analytical computation of packet latency in a 2D-mesh NoC. In *Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference, 2009. NEWCAS-TAISA '09*, pages 1–4, June 2009.

[FXH11] W. Fan, L. Xiang, and S. Hui. The QoS mechanism for NoC router by dynamic virtual channel allocation and dual-net infrastructure. In *2011 International Conference on Computational Problem-Solving (ICCP)*, pages 1 –5, October 2011.

[GADM+05] N. Genko, D. Atienza, G. De Micheli, L. Benini, J.M. Mendias, R. Hermida, and F. Catthoor. A novel approach for network on chip emulation. In *IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005*, pages 2365–2368 Vol. 3, 2005.

[GADMB07] N. Genko, D. Atienza, G. De Micheli, and L. Benini. Feature - NoC emulation: a tool and design flow for MPSoC. *IEEE Circuits and Systems Magazine*, 7(4):42–51, 2007.

[GAW+13] S.K. Goel, S. Adham, Min-Jer Wang, Ji-Jan Chen, Tze-Chiang Huang, A. Mehta, F. Lee, V. Chickermane, B. Keller, T. Valind, S. Mukherjee, N. Sood, Jeongho Cho, H.H. Lee, Jungi Choi, and Sangdoo Kim. Test and debug strategy for TSMC CoWoS #x2122; stacking process based heterogeneous 3D IC: a silicon case study. In *Test Conference (ITC), 2013 IEEE International*, pages 1–10, September 2013.

[Gha82] P.B. Ghate. Electromigration-induced failures in VLSI interconnects. In *Reliability Physics Symposium, 1982. 20th Annual*, pages 292–299, March 1982.

[GK10] P. V. Gratz and S. W. Keckler. Realistic workload characterization and analysis for networks-on-chip design. In *4th Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2010.

[GN92] C.J. Glass and L.M. Ni. The turn model for adaptive routing. In , *The 19th Annual International Symposium on Computer Architecture, 1992. Proceedings*, pages 278 –287, 1992.

[HAAN+07] H. Hossain, M. Ahmed, A. Al-Nayeem, T.Z. Islam, and M.M. Akbar. Gpnocsim - a general purpose simulator for network-on-chip. In *International Conference on Information and Communication Technology, 2007. ICICT '07*, pages 254–257, 2007.

[HCGSML11] K.-Y. Hsieh, B.-C. Cheng, R.-T. Gu, and K. Shu-Min Li. Fault-tolerant mesh for 3D network on chip. In *Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT), 2011 6th International*, pages 202–205, October 2011.

[HCT+12] Y.S.-C. Huang, Y.-C. Chang, T.-C. Tsai, Y.-Y. Chang, and C.-T. King. Attackboard: A novel dependency-aware traffic generator for exploring

NoC design space. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 376–381, 2012.

[HDH+10] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson. A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 108 –109, February 2010.

[HGK10] J. Hestness, B. Grot, and Stephen W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, page 31–36, New York, NY, USA, 2010. ACM.

[HGM+12] G. Heck, R. Guazzelli, F. Moraes, N. Calazans, and R. Soares. HardNoC: a platform to validate networks on chip through FPGA prototyping. In *2012 VIII Southern Conference on Programmable Logic (SPL)*, pages 1–6, March 2012.

[HH09] P.-T. Huang and W. Hwang. An adaptive congestion-aware routing algorithm for mesh network-on-chip platform. In *SOC Conference, 2009. SOCC 2009. IEEE International*, pages 375 –378, September 2009.

[HKB12] J. Heisswolf, R. Konig, and J. Becker. A scalable NoC router design providing QoS support using weighted round robin scheduling. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 625 –632, July 2012.

[HM04] J. Hu and R. Marculescu. DyAD - smart routing for networks-on-chip. In *In ACM/IEEE Design Automation Conference*, page 260–263, 2004.

[HM05] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, April 2005.

[HRF+11] C. Hernandez, A. Roca, J. Flich, F. Silla, and J. Duato. Fault-tolerant vertical link design for effective 3D stacking. *Computer Architecture Letters*, 10(2):41–44, July 2011.

[JHH13] M.H. Jabbar, D. Houzet, and O. Hammami. Heterogeneous stacking of 3D MPSoC architecture: Physical implementation analysis and performance evaluation. In *2013 5th Asia Symposium on Quality Electronic Design (ASQED)*, pages 127–135, August 2013.

[KA11] H.S. Kia and C. Ababei. A new fault-tolerant and congestion-aware adaptive routing algorithm for regular networks-on-chip. In *2011 IEEE Congress on Evolutionary Computation (CEC)*, pages 2465 –2472, June 2011.

[KC11] W.-C. Ku and T.-F. Chen. Accelerating manycore simulation by efficient NoC interconnection partition on FPGA simulation platform. In *2011 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4, April 2011.

[KCdlTR08] Y.E. Krasteva, F. Criado, E. de la Torre, and T. Riesgo. A fast emulation-based NoC prototyping framework. In *International Conference on Reconfigurable Computing and FPGAs, 2008. ReConFig '08*, pages 211–216, 2008.

[KLJ13] AE. Kiasari, Zhonghai Lu, and A Jantsch. An analytical latency model for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, January 2013.

[KMAMP08] M. Koibuchi, H. Matsutani, H. Amano, and T. Mark Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Second ACM/IEEE International Symposium on Networks-on-Chip, 2008. NoCS 2008*, pages 13 –22, April 2008.

[KMSP08] A.-M. Kouadri-Mostefaoui, B. Senouci, and F. Petrot. Large scale on-chip networks : An accurate multi-FPGA emulation platform. In *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008. DSD '08*, pages 3–9, 2008.

[KSPK10] G. Krishnaiah, B. V N Silpa, P.R. Panda, and A Kumar. FastFwd: an efficient hardware acceleration technique for trace-driven network-on-chip simulation. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 247–256, October 2010.

[KTMD14] C. Killian, C. Tanougast, F. Monteiro, and A. Dandache. Smart reliable network-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):242–255, February 2014.

[L+10a] P. Liu et al. Building a multi-FPGA-based emulation framework to support networks-on-chip design and verification. *International Journal of Electronics*, pages 1241–1262, October 2010.

[L+10b] S.-H. Lo et al. QoS aware BiNoC architecture. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1 –10, April 2010.

[LA04] O. Laouamri and C. Aktouf. Enhancing testability of system on chips using network management protocols. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1370–1371 Vol.2, 2004.

[LH08] X. Li and O. Hammami. Multi-FPGA emulation of a 48-cores multi-processor with NOC. In *Design and Test Workshop, 2008. IDT 2008. 3rd International*, pages 205–208, 2008.

[LK03] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro Symposium on Digital System Design, 2003. Proceedings*, pages 180–187, September 2003.

[LK09] T. Le and M.A.S. Khalid. NoC prototyping on FPGAs: a case study using an image processing benchmark. In *IEEE International Conference on Electro/Information Technology, 2009. eit '09*, pages 441–445, June 2009.

[LKRD+10] P. Lotfi-Kamran, A. M. Rahmani, M. Daneshtalab, A. Afzali-Kusha, and Z. Navabi. EDXY: a low cost congestion-aware routing algorithm for network-on-chips. *Journal of Systems Architecture*, 56(7):256–264, 2010.

[LML+08] I. Loi, S. Mitra, T.H. Lee, Shinobu Fujita, and L. Benini. A low-overhead fault tolerance scheme for TSV-based 3D network on chip links. In *IEEE/ACM International Conference on Computer-Aided Design, 2008. ICCAD 2008*, pages 598–602, November 2008.

[LPG11] S. Lotlikar, V. Pai, and P.V. Gratz. AcENoCs: a configurable HW/SW platform for FPGA accelerated NoC emulation. In *2011 24th International Conference on VLSI Design (VLSI Design)*, pages 147–152, 2011.

[LYJ06] Z. Lu, B. Yin, and A. Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006*, pages 6 pp.–, March 2006.

[LZJ06] M. Li, Q.-A. Zeng, and W.-B. Jone. DyXY - a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 849 –852, 2006.

[M+04] F. Moraes et al. HERMES: an infrastructure for low area overhead packet-switching networks on chip. *Integr. VLSI J.*, 38(1):69–93, October 2004.

[MKH+12] N. Maeda, Y.S. Kim, Y. Hikosaka, T. Eshita, H. Kitada, K. Fujimoto, Y. Mizushima, K. Suzuki, T. Nakamura, A. Kawai, K. Arai, and T. Ohba. Development of ultra-thinning technology for logic and memory heterogeneous stack applications. In *3D Systems Integration Conference (3DIC), 2011 IEEE International*, pages 1–4, January 2012.

[MKM10] A. Mehranzadeh, A. Khademzadeh, and A. Mehran. FADyAD- fault and congestion aware routing algorithm based on DyAD algorithm. In *2010 5th International Symposium on Telecommunications (IST)*, pages 274 –279, December 2010.

[MNFA14] K. MacDonald, C. Nitta, M. Farrens, and V. Akella. PDG_GEN: a methodology for fast and accurate simulation of on-chip networks. *IEEE Transactions on Computers*, 63(3):650–663, March 2014.

[nc-] Network controller sideband interface (nc-si) specification. http://www.dmtf.org/sites/default/files/standards/documents/DSP0222_1.0.1.pdf. Accessed: 2015-05-30.

[net] Cisco systems netflow services export version 9. http://www.ietf.org/rfc/rfc3954.txt. Accessed: 2015-05-30.

[NMFA11] C. Nitta, K. Macdonald, M. Farrens, and V. Akella. Inferring packet dependencies to improve trace based simulation of on-chip networks. In *2011 Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 153–160, 2011.

[OW14] G. Oxman and S. Weiss. Simple method to reduce congestion in bufferless network-on-chip. *Electronics Letters*, 50(8):581–583, April 2014.

[PAK+11] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, pages 406–417, February 2011.

[Pap11] M.K. Papamichael. Fast scalable FPGA-based network-on-chip simulation models. In *2011 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 77–82, July 2011.

[PF07] V.F. Pavlidis and E.G. Friedman. 3-d topologies for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(10):1081–1090, October 2007.

[PF09] V. F. Pavlidis and E. G. Friedman. 3-d circuit architectures. In Vasilis F. Pavlidis and Eby G. Friedman, editors, *Three-dimensional Integrated Circuit Design*, pages 185–246. Morgan Kaufmann, Boston, 2009.

[PFYDLJ15] K. Pang, V. Fresse, S. Yao, and O. A. De Lima Jr. Task mapping and mesh topology exploration for an FPGA-based network on chip. *Microprocessors and Microsystems*, 39(3):189–199, May 2015.

[PHM11] M.K. Papamichael, J.C. Hoe, and O. Mutlu. FIST: a fast, lightweight, FPGA-friendly packet latency estimator for NoC modeling in full-system simulations. In *2011 Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 137–144, May 2011.

[PLB+04] M. Pirretti, G.M. Link, R.R. Brooks, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *IEEE Computer society Annual Symposium on VLSI, 2004. Proceedings*, pages 46 – 51, February 2004.

[PP12] I. Pratomo and S. Pillement. Gradient - an adaptive fault-tolerant routing algorithm for 2D mesh network-on-chips. In *2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8, October 2012.

[Pra10] Grot B. Gratz P.V. Hu J. Prabhu, S. Ocin tsim- DVFS aware simulator for NoCs. In *SAW 1,2010*, 2010.

[PSCC+13] S.L. Pei-Siang, Fa Xing Che, Chong Ser Choong, M.C.B. Rong, V.N. Sekhar, V.S. Rao, and Chai Tai Chong. Heterogeneous three-layer TSV chip stacking assembly with moldable underfill. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 3(11):1960–1970, November 2013.

[Qia13] Da-Cheng J. Bogdan P. Chi-Ying T. Marculescu D. Marculescu R. Qian, Z. SVR-NoC: a performance analysis tool for network-on-chips using learning-based support vector regression model. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 354–357, March 2013.

[rdp] Remote desktop protocol, basic connectivity and graphics remoting. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RDPBCGR].pdf. Accessed: 2015-05-30.

[RLL+11] A.-M. Rahmani, P. Liljeberg, K. Latif, J. Plosila, K.R. Vaddina, and H. Tenhunen. Congestion aware, fault tolerant, and thermally efficient inter-layer communication scheme for hybrid NoC-bus 3D architectures. In *2011 Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 65–72, May 2011.

[SC13] P. K. Sahu and S. Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):60–76, January 2013.

[SGP+13] J. Sepulveda, G. Gogniat, R. Pires, C. Pedraza, Wang Chau, and M. Strum. QoS 3D-HoC hybrid-on-chip communication structure for dynamic 3D-MPSoCs. In *2013 IEEE Fourth Latin American Symposium on Circuits and Systems (LASCAS)*, pages 1–4, February 2013.

[SGR13] A. Shalaby, V. Goulart, and M.E.-S. Ragab. Study of application of network coding on NoCs for multicast communications. In *2013 IEEE 7th International Symposium on Embedded Multicore Socs (MCSoC)*, pages 37–42, September 2013.

[SMAG12] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens. A TDM NoC supporting QoS, multicast, and fast connection set-up. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1283 –1288, March 2012.

[smb] System management bus, howpublished = http://smbus.org/, note = Accessed: 2015-05-30.

[SMG14] R.A. Stefan, A. Molnos, and K. Goossens. dAElite: a TDM NoC supporting QoS, multicast, and fast connection set-up. *IEEE Transactions on Computers*, 63(3):583–594, March 2014.

[Sta01] J.H. Stathis. Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits. *IEEE Transactions on Device and Materials Reliability*, 1(1):43–59, March 2001.

[SWP06] V. Soteriou, H. Wang, and L.-S. Peh. A statistical traffic model for on-chip interconnection networks. In *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006*, pages 104–116, September 2006.

[SZ12] L.X. Shi and Z. Zhang. A hybrid NoC-Based MPSoC simulator. *Advanced Engineering Forum*, 6-7:238–242, September 2012.

[TAA+11] F. Trivino, F.J. Andujar, F.J. Alfaro, J.L. Sanchez, and A. Ros. Self-related traces: An alternative to full-system simulation for NoCs. In *2011 International Conference on High Performance Computing and Simulation (HPCS)*, pages 819–824, 2011.

[TFR11] J. Tan, V. Fresse, and F. Rousseau. Generation of emulation platforms for NoC exploration on FPGA. In *2011 22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 186–192, 2011.

[TMY11] M. Taniguchi, H. Matsutani, and N. Yamasaki. Design and implementation of on-chip adaptive router with predictor for regional congestion. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, volume 2, pages 22 –27, August 2011.

[VHR+08] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29 –41, January 2008.

[VM04] G.V. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, 2004.

[VMPL10] M. Valinataj, S. Mohammadi, J. Plosila, and P. Liljeberg. A fault-tolerant and congestion-aware routing algorithm for networks-on-chip. In *2010 IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 139 –144, April 2010.

[W+09] X. Wu et al. QoS router with both soft and hard guarantee for network-on-chip. In *NORCHIP, 2009*, pages 1 –6, November 2009.

[WB12] C. Wang and N. Bagherzadeh. Design and evaluation of a high throughput QoS-Aware and congestion-aware router architecture for network-on-chip. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 457 –464, February 2012.

[WHB10] C. Wang, W.-H. Hu, and N. Bagherzadeh. Congestion-aware network-on-chip router architecture. In *2010 15th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, pages 137 –144, September 2010.

[WHLB10] C. Wang, W.-H. Hu, S.E. Lee, and N. Bagherzadeh. Area and power-efficient innovative network-on-chip architecurte. In *2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 533–539, February 2010.

[WHS07] P.T. Wolkotte, P.K.F. Hölzenspies, and G. J M Smit. Fast, accurate and detailed NoC simulations. In *First International Symposium on Networks-on-Chip, 2007. NOCS 2007*, pages 323–332, 2007.

[WJS11] D. Wang, N. E. Jerger, and J. G. Steffan. DART: a programmable architecture for NoC simulation on FPGAs. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, page 145–152, New York, NY, USA, 2011. ACM.

[WLV+14] D. Wang, C. Lo, J. Vasiljevic, N.E. Jerger, and J.G. Steffan. DART: a programmable architecture for NoC simulation on FPGAs. *IEEE Transactions on Computers*, 63(3):664–678, March 2014.

[wmi] Windows management instrumentation remote protocol. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-WMI].pdf. Accessed: 2015-05-30.

[WYJL09] X. Wang, M. Yang, Y. Jiang, and P. Liu. Power-aware mapping for network-on-chip architectures under bandwidth and latency constraints. In *4th International Conference on Embedded and Multimedia Computing, 2009. EM-Com 2009*, pages 1–6, December 2009.

[YA08] Q. Yu and P. Ampadu. Adaptive error control for NoC switch-to-switch links in a variable noise environment. In *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08*, pages 352–360, October 2008.

[YA12] Q. Yu and P. Ampadu. Dual-layer adaptive error control for network-on-chip links. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(7):1304–1317, July 2012.

[ZP10] Y. Zou and S. Pasricha. NARCO: neighbor aware turn model-based fault tolerant routing for NoCs. *IEEE Embedded Systems Letters*, 2(3):85 –89, September 2010.

[ZPG07] H. Zhu, P.P. Pande, and C. Grecu. Performance evaluation of adaptive routing algorithms for achieving fault tolerance in NoC fabrics. In *on Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf*, pages 42 –47, July 2007.

[ZWGLB14] M. Zhong, Z. Wang, H. Gu, and S. Le Beux. An improved minimal multicast routing algorithm for mesh-based networks-on-chip. In *2014 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 775–779, August 2014.