Architecture reconfigurable de système embarqué auto-organisé

Slavisa Jovanovic

Nancy-Université
Université Henri Poincaré

U.F.R. Sciences et Techniques, Mathématiques, Informatique, Automatique
École doctorale Informatique Automatique Électronique Électrotechnique Mathématiques
Département de formation doctorale Électronique Électrotechnique

Architecture reconfigurable de système embarqué auto-organisé

THÈSE

présentée et soutenue publiquement le 6 novembre 2009

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité Instrumentation et Microélectronique)

par

Slaviša JOVANOVIĆ

Composition du jury

Président :	M. Michel PAINDAVOINE, Professeur, Université de Bourgogne, Dijon
Rapporteurs :	M. Ian O'CONNOR, Professeur, INL - Ecole Centrale de Lyon, Lyon M. Olivier SENTIEYS, Professeur, ENSSAT, Lannion
Examinateurs :	M. Christophe BOBDA, Professeur, Université de Potsdam, Allemagne M. Camel TANOUGAST, Maître de Conférences, Université Paul Verlaine, Metz M. Serge WEBER, Professeur, UHP, Nancy 1

Laboratoire Instrumentation Electronique Nancy

Laboratoire d'Instrumentation Électronique de Nancy
Faculté des Sciences - 54506 Vandoeuvre-lès-Nancy

Remerciements

J'adresse mes sincères remerciements à Monsieur Ian O'CONNOR, Professeur à l'École Centrale de Lyon ainsi qu'à Monsieur Olivier SENTIEYS, Professeur à l'École Nationale Supérieure des Sciences Appliquées et de Technologie de Lannion, qui m'ont fait l'honneur de juger cette thèse en qualité de rapporteurs.

Je tiens également à remercier Monsieur Christophe BOBDA, Professeur à l'Université de Potsdam et Monsieur Michel PAINDAVOINE, Professeur à l'Université de Bourgogne d'avoir accepté d'examiner ce travail et de participer à ce jury.

J'adresse mes remerciements à Monsieur Serge WEBER, Professeur à l'Université Henri Poincaré de Nancy et directeur du Laboratoire LIEN, de m'avoir accueilli au sein du Laboratoire LIEN et de m'avoir accordé sa confiance au sein de l'équipe « Architecture » qu'il dirige. Je tiens également à le remercier d'avoir accepté d'être mon directeur de thèse.

Je souhaite particulièrement remercier Monsieur Camel TANOUGAST, Maître de Conférences à l'université de Paul Verlaine de Metz, de m'avoir encadré durant cette thèse, pour ses qualités et sa rigueur scientifique, sa disponibilité et son soutien permanent qui m'ont beaucoup aidé tout au long de ce travail.

Je remercie tous mes collègues et membres du LIEN pour leur aide ou les conseils qu'ils ont pu m'apporter à un moment ou un autre. Sans oublier Patrice, pour son sens de l'humour, ses blagues et son café, rendant nos pauses café des moments détendus et conviviaux.

Je tiens à remercier mes parents et tous mes amis pour leur soutien et leurs questions « répétitives » concernant mon sujet de travail et son utilité finale.

Enfin, je souhaite terminer ces remerciements par une attention particulière à Jasna, mon épouse, pour son soutien permanent, sa patience et sa compréhension tout au long de ces trois années de thèse.

Slaviša JOVANOVIĆ
septembre 2009

« Un voyage de mille kilomètres commence toujours par un premier pas »
Lao Tseu

Je dédie ce travail à mes parents :
Stanka et Velizar

Table des matières

Table des figures

Liste des tableaux

Résumé

Afin de répondre à une complexité croissante des systèmes de calcul, due notamment aux progrès rapides et permanents des technologies de l'information, de nouveaux paradigmes et solutions architecturales basées sur des structures auto-adaptatives, auto-organisées sont à élaborer. Ces dernières doivent permettre d'une part la mise à disposition d'une puissance de calcul suffisante répondant à des contraintes de temps sévères (traitement temps réel). D'autre part, de disposer d'une grande flexibilité et adaptabilité dans le but de répondre aux évolutions des traitements ou des défaillances non prévues caractérisant un contexte d'environnement évolutif de fonctionnement du système. C'est dans ce cadre que s'insèrent les travaux de recherche présentés dans cette thèse qui consistent à développer une architecture auto-organisée de type *Reconfigurable MPSoC (Multi processor System on Chip)* à base de technologie FPGA.

La stratégie de recherche proposée consiste à développer une approche architecturale mettant en œuvre le concept d'auto-adaptativité, d'auto-organisation par l'exploitation de la reconfiguration dynamique (RD) des circuits FPGA tout en recherchant à satisfaire la notion d'*Adéquation Algorithme - Architecture*. L'objectif de ces travaux est d'élaborer le principe d'une auto-organisation (AO) matérielle en vue d'aboutir à des architectures de calcul embarqué capables de s'auto-restructurer par reconfiguration, s'auto-gérer de manière autonome et sans contrôle externe tout en répondant à des contraintes multiples de conception ou de traitement (contrainte de temps ou puissance de calcul, flexibilité, tolérance aux fautes, etc.). L'approche architecturale de calcul embarqué auto-organisée proposée repose à la fois sur une structure de communication reconfigurable de type *NoC* (*Network on Chip*) adaptée à la technologie FPGA et sur des structures originales de communications *informationnelles* contribuant à mettre en œuvre les principes d'auto-adaptation, d'auto-organisation.

Le manuscrit de cette thèse est organisé de la manière suivante :

- Une première partie présente et identifie les principales caractéristiques du concept d'auto-organisation et sa transposition dans le domaine de la conception matérielle des systèmes embarqués à base de technologies reconfigurables. Plus précisément, dans un premier temps, nous proposons une définition d'un système auto-organisé et ses principales propriétés associées (*adaptabilité, autonomie, robustesse, flexibilité, anticipabilité, contrôle décentralisé*). Ensuite, nous montrons l'intérêt de l'utilisation de la technologie reconfigurable dynamique des circuits FPGA pour la mise en œuvre matérielle de systèmes possédant de telles propriétés. Nous recensons alors les principaux besoins pour leurs réalisations et nous montrons la nécessité d'élaborer d'une part de nouveaux concepts architecturaux basés sur des structures de communication

adaptées à la technologie reconfigurable FPGA. D'autre part, de développer des mécanismes d'apprentissage au cours du fonctionnement de tels systèmes.

- Ensuite, un nouveau concept architectural associé aux propriétés d'auto-organisation définies dans cette première partie est proposé. Cette approche architecturale est caractérisée par la mise en œuvre du principe d'auto-organisation à travers le développement matériel de nouvelles structures de communications informationnelles internes, circulaires localement et globalement au système auto-organisé proposé (notion de *flux* et de *descripteur*). Ce concept original permet des échanges d'informations entre les modules constituant le système, sans un contrôle centralisé et tout en générant une « conscience » interne parmi l'ensemble des entités le constituant, dans le but de répondre à tout changement ou évolution du système. Ce concept est également détaillé à travers un exemple concret du fonctionnement auto-organisé du système proposé, lors d'une défaillance temporaire ou permanente d'entités le constituant, par changement ou substitution fonctionnelle de ses modes de fonctionnement.

- La deuxième partie présente et détaille la conception d'une structure adaptée de communication sur puce (*Network on Chip - NoC*) pour les systèmes reconfigurables auto-organisés à base de technologie FPGA. Après avoir rappelé les principales caractéristiques des réseaux sur puce et justifié les raisons pour lesquelles les NoC « classiques » actuels ne sont pas adaptés à des systèmes reconfigurables auto-organisés, nous proposons deux nouvelles architectures de routeur sur puce (*CuNoC* et *QNoC*) correspondant à une évolution pour architecture NoC adaptée aux systèmes reconfigurables. Une analyse et une comparaison détaillées de leurs principaux avantanges et inconvénients sont également présentées. Un algorithme de routage spécifiquement développé pour une architecture reconfigurable NoC associée à ces routeurs est également détaillé.

- En vue de valider l'approche architecturale globale du système auto-organisé proposé et associé aux concepts, mécanismes et structures de communication sur puce adoptés, nous présentons les résultats de l'exécution d'une auto-organisation de notre architecture dans le cas d'un traitement d'images temps réel défaillant au cours du temps tout en maintenant une *Adéquation Algorithme - Architecture*. L'objectif principal de cette partie est une validation expérimentale de l'ensemble des aspects conceptuels abordés au cours de ces travaux de recherche.

- Enfin, nous discutons et dressons le bilan de nos travaux en présentant les apports des concepts architecturaux proposés et en dégageant les perspectives de ce travail ainsi que les amélioration à apporter.

Mots-clés: *Auto-adaptation et Auto-organisation matérielles, Calcul embarqué, Systèmes reconfigurables, Réseaux sur puce reconfigurables (RNoC), Algorithme de routage tolérant aux fautes, FPGA.*

Abstract

The growing complexity of computing systems, mostly due to the rapid progress in Information Technology (IT) in the last decade, imposes on system designers to orient their traditional design concepts towards the new ones based on self-organizing and self-adaptive architectural solutions. On the one hand, these new architectural solutions should provide a system with a sufficient computing power, and on the other hand, a great flexibility and adaptivity in order to cope with all non-deterministic changes and events that may occur in the environnement in which it evolves. Within this framework, a reconfigurable MPSoC self-organizing architecture on the FPGA reconfigurable technology is studied and developped during this PhD.

The proposed research methodology is there to develop a hardware architectural approach based on the concepts of self-adaptivity and self-organization by exploiting the property of dynamic reconfiguration of the FPGA circuits and by seeking to satisfy the *Algorithm / Architecture* matching. The objective of this work is to integrate the main properties of the self-organization principle in the FPGA reconfigurable computing architectures. These architectures should be capable of self-restructuring through reconfiguration and self-managing in an independent manner and without any external controls while satisfying multiple design and processing constraints (real-time constraints, computing power, flexibility, fault tolerance, etc.). The proposed self-organizing hardware architecture is based on a reconfigurable *Network-on-Chip (NoC)* structure adapted to the FPGA technology and on a hardware design approach allowing integration of self-managing properties into the hardware architecture.

This manuscript is organized as follows :

- In the first part, we first of all identify and detail the main properties of the self-organization. After that, these properties are transposed to the domain of reconfigurable hardware architectures. Here, we propose definitions of a self-organizing system and its main properties (adaptability, autonomy, robustess, flexibility, anticipation, decentralized control). Then, we show the interest of using the dynamically reconfigurable FPGA technology for a hardware implementation of self-organizing properties. We also list the main needs for these implementations : a new hardware design approach based on self-organizing properties, a reconfigurable communication structure adapted to the FPGA circuits of type NoC and a learning mechanism allowing the systems to learn from their experiences.

- Then in the second part, we propose a new hardware design approach based on the self-organizing properties (defined and detailed in the first part). This approach is characterized by two structures : a dynamic circular structure called *flux* and a static descriptive structure called *descriptor*. The use of this hardware design approach in hardware systems allows an exchange of information between the systems' modules without any external control and also allows handling and responding to all environnemental changes and events firstly at the local level,

and secondly, if necessary, at the system's level. This design approach is detailed in the case of a permanent or transient fault of a system's module where the faulty module is functionnaly replaced by its direct neighbouring modules.

- A communication structure adapted to the FPGA circuits of type NoC is presented and detailed in the third part of this manuscript. Fistly, we give an overview of basic NoC approaches and their main characteristics as well as the main reasons why these classic NoCs are not very suited for the use in the FPGA circuits. Then, we propose two new router architectures (*CuNoC* and *QNoC*) for the FPGA reconfigurable systems. The main advantages and drawbacks of these two architectures and their comparison are detailed and given in this part. In the final section of this part, a routing algorithm used for both architectures is presented and discussed.

- In order to validate experimentally all the aspects proposed in the previous parts, we applied them to an image processing application which was implemented and executed on an FPGA platform. We present the main steps and details of the implementation of the proposed hardware design approach into the given application. The experimental results are also discussed as well as some performance evaluations of the given application.

- In the last part, we discuss a summary of this work and we present its main contributions. Some perspectives and future work based on this research are also given in this part.

Keywords: *Self-organization and self-managing in hardware, Embedded computing, Reconfigurable systems, Reconfigurable Network-on-chips, Fault-tolerant routing algotithm, FPGA*

Glossaire

AO : Auto-Organisation
ASIC : Application Specific Integrated Circuit ,

CoNoChi : Configurable Network-on-Chip
CPU : Central Processing Unit
CS : Circuit Switching

DyNoC : Dynamic Network on Chip

FPGA : Field Programmable Gate Array,

ICAP : Internal Configuration Access Port

LUT : Look Up Table

MPSoC : Multiprocessor System-on-Chip

NI : Network Interface

OSI : The Open Systems Interconnection Reference Model

PE : Processing Element
PIR : Packet Injection Rate
PS : Packet Switching

QoS : Quality of Service

RD : Reconfiguration Dynamique,
ReNoC : Reconfigurable NoC

SAF : Store-and-Forward
SAR : Système Auto-organisé Reconfigurable
SNS : Smart Network Stack
SoC : System on Chip,

SRAM : Static Random Acces Memory
ST : Switching Technique

VCT : Virtual Cut Through

WS : Wormhole Switching

Introduction générale

1 Contexte général

Grâce aux progrès de la microélectronique, les technologies de l'information ont pénétré tous les aspects de notre vie quotidienne. En effet, l'avènement des circuits intégrés, permettant de réaliser des fonctions de plus en plus complexes, des traitements de plus en plus rapides tout en consommant de moins en moins d'énergie et ainsi permettre une portabilité, a profondement changé nos modes de vie. De nos jours, nous disposons de nouveaux dispositifs dits « concentré de technologie » regroupant sur quelques centimètres carrés une variété de fonctions complexes (codeur-décodeur *GSM*, décodeur *MP3, MPEG4, GPS* etc.), d'utilité quotidienne et présentés sous forme de matériel de poche. Ainsi, avec la miniaturisation à outrance et l'ère du développement de la nanoélectronique où la dimension caractéristique devient de l'ordre de dizaine nanomètres, les tendances actuelles dans la production et la consommation massive de ces « bijoux technologiques » ne cessent de croître et d'occuper une place de plus en plus importante dans nos modes de vie sociétale dite « moderne ».

Afin de répondre à cette demande massive, les concepteurs des circuits intégrés ont recours à la conception et à la fabrication de circuits spécifiques *ASIC*, qui permettent une solution d'intégration maximale par rapport au progrès technologique du moment. Cependant, généralement ce type de conception et de fabrication s'effectue au détriment de coûts fixes, de temps et de coûts de développement élevés et d'un degré de flexibilité très faible. En effet, la solution *ASIC* n'est viable que pour un nombre restreint d'utilisations. Une autre solution disposant d'une plus grande flexibilité, des performances acceptables et d'un coût nettement moins élevé par rapport aux *ASIC*, est présentée par la technologie reconfigurable de type *FPGA*. Grâce à la versatilité de ces circuits *FPGA*, les systèmes microélectroniques à base de ces technologies apportent une plus grande flexibilité. De plus, ils résolvent les principaux inconvénients des *ASIC* grâce à leurs facilités d'utilisation et leur rapidité de mise en œuvre (prototypage) qui ont fortement contribué à leur large succès ces dernières décennies au point de remplacer les *ASIC* de densité en termes de portes logiques d'environ un million et pour des fabrications de petites et moyennes séries. Toutefois l'utilisation de ces systèmes est limitée par leur taille et par leurs performances équivalentes à environ un tiers de celles des circuits *ASIC*.

Actuellement, la technologie reconfigurable dynamiquement de type *FPGA* est considérée comme une nouvelle alternative pour la réalisation d'architectures de traitement numérique de l'information [SW01]. En effet, elle permet de mettre en œuvre le paradigme du concept de « calcul reconfigurable ». Bien que ce concept soit déjà relativement ancien [EV62], la démocratisation de la technologie *FPGA* a conduit à un regain d'intérêt pour l'exécution d'une stratégie de calcul reconfigurable. En pratique, cela s'est traduit initialement par l'association de ces technologies à un processeur pour évoluer de nos jours vers un couplage plus étroit se traduisant par l'intégration d'une matrice *FPGA* et de processeurs sur une même puce [BRM+99, Tan01]. L'objectif est de profiter simultanément des performances d'une technologie permettant une logique câblée tout en gardant une forte flexibilité à travers la disposition d'une structure microprogrammée [CH02]. Avec l'évolution actuelle des systèmes complets intégrant des modules de nature différente sur une même puce (*SoC - System on a chip*), les technologies reconfigurables deviennent primordiales. En effet, les *SoC* ayant à l'origine une flexibilité matérielle limitée (structure de type *ASIC*) ont besoin à la fois de gérer leurs fonctionnements de manière autonome et de s'adapter à des modifications de l'environnement dans lequel ils évoluent. C'est dans ce contexte que la technologie *FPGA* permet une flexibilité à travers des phases de reconfiguration dynamique (globales ou partielles) tout en conservant un minimum de performances. Cette flexibilité s'exprime à travers des changements spatio / temporels de la structure du circuit permettant ainsi une grande souplesse matérielle [Tan01].

2 Motivation

Les systèmes de traitement d'information, étant de plus en plus complexes et hétérogènes, deviennent difficilement appréhendable. Ceci est dû notamment au progrès rapide en technologie de l'information au cours de ces dernières années. En effet, les besoins en terme de puissance de calcul et de traitement ne cessent d'augmenter. Afin de faire face aux nouveaux besoins et aux nouvelles exigences, de nouveaux paradigmes et solutions architecturales basées sur des structures auto-adaptatives, auto-organisées sont à élaborer. Ces nouvelles approches doivent permettre aux systèmes de faire face de manière plus autonome et plus flexible à tous les changements dynamiques prévus et imprévus. Des approches architecturales basées sur des mécanismes d'auto-organisation et inspirées du milieu naturel caractérisé par une organisation spontanée, s'imposent comme des solutions logiques à ces nouveaux challenges. Ces nouvelles approches doivent permettre d'une part la mise à disposition d'une puissance de calcul suffisante répondant à des contraintes de temps sévères (traitement temps réel) ; d'autre part, de disposer d'une grande flexibilité et adaptabilité dans le but de répondre aux évolutions des traitements ou des défaillances non prévues caractérisant un contexte d'environnement évolutif de fonctionnement du système.

Les technologies reconfigurables de type *FPGA* ont permis de répondre dans une certaine mesure à l'optimisation des ressources matérielles tout en garantissant une puissance de calcul suffisante pour des applications temps réel. L'atout principal d'une telle solution technologique est l'apport d'une flexibilité permettant des traitements adaptés. Cependant, il ne demeure pas moins déterministe dans la mesure où la conception basée sur cette technologie doit planifier les calculs successifs à mettre en œuvre. Or, dans le cas d'un traitement de flot de données non déterministe où une évolution non précise et non connue des traitements existants, une simple solution architecturale reconfigurable ne permet pas une réponse adaptée à ces traitements évolutifs. Néanmoins, la reconfigurabilité dynamique des technologies *FPGA* est sans doute l'aspect le plus prospectif des travaux autour des systèmes adaptatifs et auto organisés. Afin que le concept d'auto-organisation soit utilisé et appliqué dans un système sur puce reconfigurable, des adaptations du concept d'auto-organisation doivent être menées en prenant en compte les propriétés des technologies reconfigurables.

Actuellement, la plupart des travaux portant sur la conception d'architectures reconfigurables dynamiquement sont destinés à une gamme d'application et de traitements spécifiques. En effet, aucune approche proposée ne prend en considération les aspects d'auto-organisation possibles d'un système et plus particulièrement dans la conception des systèmes reconfigurables. Considérant les environnements, dans lesquels les architectures de calcul évoluent, de plus en plus « changeables » et non-déterministes et que chaque changement exige une réponse adéquate et efficace. Des solutions architecturales doivent être développées afin de permettre d'associer au sein d'une même architecture de traitement à la fois :

- la versatilité structurelle de la technologie microélectronique (reconfiguration dynamique des circuits *FPGA*) permettant d'assurer une adaptabilité structurelle des architectures de traitement tout en maintenant des performances de traitement élevées ;
- l'introduction des concepts relatifs au principe d'auto-organisation permettant une réponse du système la plus appropriée à tous changements environnementaux du système à travers une auto-adaptabilité.

C'est dans ce contexte que se situent ces travaux de recherche. Les travaux menés et présentés dans ce manuscrit ont été initiés dans le cadre du Collège Doctoral Franco-Allemand intitulé « *Système de Calcul Reconfigurable, Adaptable, Autonomique et Organique* » (« *Autonomische, Organische und Reconfigurable Rechensysteme* ») débuté en octobre 2006 et en collaboration entre le groupe « Computer Engineering » dirigé par le Prof. Christophe BOBDA de l'*Institut für Informatik (IFI)*[1] de l'université de Potsdam, le laboratoire *LIEN*[2] de la faculté des sciences et techniques de Nancy 1 et le laboratoire *LICM*[3] de l'université Paul Verlaine de Metz. L'objec-

[1] Institut für Informatik - Universität Potsdam, `http://www.cs.uni-potsdam.de`
[2] Laboratoire d'Instrumentation Électronique de Nancy, `http://www.lien.uhp-nancy.fr`
[3] Laboratoire Interface Capteurs Microélectronique de Metz, `http://www.licm.sciences.univ-metz.fr`

tif principal de ces travaux de recherche est de développer une nouvelle approche architecturale pour la conception de systèmes auto-organisés à base de technologies reconfigurables.

3 Contribution

Le but de ce travaux de recherche est d'apporter des solutions architecturales innovantes basées principalement sur le principe d'auto-organisation et permettant la mise en œuvre d'une architecture embarquée auto-organisée adaptée à la technologie reconfigurable. Plus précisément, ces travaux de thèse consistent à développer une architecture auto-organisée de type reconfigurable *MPSoC* (*Multi Processor System on Chip*) à base de technologie reconfigurable *FPGA*. Pour ce faire, les caractéristiques principales du principe d'auto-organisation ont été identifiées et transposées dans le domaine de la conception de systèmes reconfigurables. Ainsi, un système reconfigurable auto-organisé a été défini et les principaux besoins pour sa réalisation ont été recensés.

La stratégie de recherche proposée dans ces travaux de thèse consiste à développer une approche architecturale mettant en œuvre les concepts d'auto-adaptativité, d'auto-organisation par l'exploitation de la reconfiguration dynamique (*RD*) des circuits *FPGA* tout en recherchant à satisfaire la notion d'*Adéquation Algorithme - Architecture*. L'objectif de ces travaux est d'élaborer le principe d'une auto-organisation (*AO*) matérielle en vue d'aboutir à des architectures de calcul embarqué capables de s'auto-restructurer dynamiquement, s'auto-gérer de manière autonome et sans contrôle externe tout en répondant à des contraintes multiples de conception ou de traitement (contrainte de temps ou puissance de calcul, flexibilité, tolérance aux fautes, etc). L'approche architecturale de calcul embarqué auto-organisée proposée repose à la fois sur une structure de communication reconfigurable de type *NoC* (*Network on Chip*) adaptée à la technologie *FPGA* et sur des structures originales de communications informationnelles contribuant à mettre en œuvre les principes d'auto-adaptation, d'auto-organisation. L'ensemble de ces points ont été validés non seulement au niveau de simulations mais également au niveau expérimental sur un exemple concret d'application de traitement d'images temps réel. Les travaux menés pendant cette thèse sont originaux en plusieurs points. D'abord, parce qu'ils portent sur la transposition matérielle des aspects du principe d'auto-organisation des systèmes. Ensuite, parce que les travaux effectués prennent en compte les spécificités de la technologie reconfigurable *FPGA* afin d'aboutir à un prototype opérationnel.

4 Plan du manuscrit

Le manuscrit de cette thèse est structuré de la manière suivante :

Dans **le premier chapitre** nous recensons de manière non-exhaustive la notion d'auto-organisation, d'émergence et les principales propriétés associées (*adaptabilité, autonomie, robustesse, flexibilité, anticipabilité, contrôle décentralisé*). Nous proposons une définition d'un système auto-organisé et / ou émergent. Nous présentons quelques mises en oeuvre pratiques de tels systèmes. Ensuite, nous montrons l'intérêt de l'utilisation de la reconfiguration dynamique des circuits *FPGA* pour la mise en œuvre matérielle de systèmes possédant de telles propriétés. Enfin, nous concluons ce chapitre d'introduction sur l'intérêt d'exploiter les technologies *FPGA* pour la conception de systèmes embarqués auto-organisés.

Dans **le deuxième chapitre** nous rappelons les principales caractéristiques de la reconfiguration dynamique à base de technologie *FPGA*. Ensuite, nous proposons une définition d'un système reconfigurable auto-organisé. Puis, une transposition des caractéristiques et des propriétés d'auto-organisation est menée dans le cadre d'une mise en œuvre dans un système reconfigurable à base de circuits *FPGA*. Enfin, nous concluons par un recensement des principaux besoins pour la conception de systèmes reconfigurables auto-organisés et nous montrons la nécessité d'élaborer d'une part de nouveaux concepts architecturaux basés sur des structures de communication adaptées à la technologie *FPGA*. D'autre part, de développer des mécanismes d'apprentissage au cours du fonctionnement de tels systèmes.

Le troisième chapitre présente l'approche architecturale adoptée permettant une solution matérielle de mise en œuvre des propriétés d'auto-organisation dans les systèmes sur puce reconfigurables (*RSoC* ou reconfigurable *MPSoC*). Dans un premier temps, un rappel des principaux types de répartition de contrôle généralement utilisés dans la conception des systèmes est présenté. Quelques exemples sont cités. Ensuite, nous proposons une approche architecturale caractérisée par la mise en œuvre du principe d'auto-organisation à travers le développement matériel de nouvelles structures de communications informationnelles internes, circulaires localement et globalement au système auto-organisé proposé (notion de *flux* et de *descripteur*). Ce concept original permet des échanges d'informations entre les modules constituant le système, sans un contrôle centralisé et tout en générant une « conscience » interne parmi l'ensemble des entités le constituant, dans le but de répondre à tout changement ou évolution du système. Ce concept est également détaillé à travers un exemple concret du fonctionnement auto-organisé du système proposé au cours d'une défaillance temporaire ou permanente d'entités le constituant, par changement ou substitution fonctionnelle de ses modes de fonctionnement.

Dans le **quatrième chapitre** nous présentons et détaillons la conception d'une structure adaptée de communication sur puce (*Network on Chip - NoC*) pour les systèmes reconfigurables auto-organisés à base de technologie *FPGA*. Après avoir rappelé les principales caractéristiques des réseaux sur puce et justifié les raisons pour lesquelles les *NoC* « classiques » actuels ne sont pas adaptés à des systèmes reconfigurables auto-organisés, nous proposons deux nouvelles architectures de routeur sur puce (*CuNoC* et *QNoC*) correspondant à une évolution

pour architecture *NoC* adaptée aux systèmes reconfigurables. Une analyse et une comparaison détaillées de leurs principaux avantages et inconvénients sont également présentées. Un algorithme de routage spécifiquement développé pour une architecture reconfigurable *NoC* associée à ces routeurs est également détaillé. Nous concluons ce chapitre en résumant quelques travaux en perspective à effectuer sur les *NoC* reconfigurables.

En vue de valider l'approche architecturale globale du système auto-organisé proposé et associée aux concepts, mécanismes et structures de communication sur puce adoptés (et présentés dans les chapitres précédents), le **cinquième et dernier chapitre** présente les résultats de l'exécution d'une auto-organisation de notre architecture dans le cas d'un traitement d'images temps réel défaillant au cours du temps tout en maintenant une *Adéquation Algorithme - Architecture*. Nous détaillons les démarches que nous avons effectuées pour appliquer les structures principales de notre approche architecturale. L'objectif principal de ce chapitre est une validation expérimentale de l'ensemble des aspects conceptuels abordés au cours de ces travaux de recherche dans le but de rendre un système capable de gérer de manière autonome et intelligente les éventuelles perturbations provenant de l'environnement dans lequel il évolue.

Enfin, une **conclusion générale** discute et dresse le bilan de ces travaux de recherche en présentant les apports des concepts architecturaux proposés et en dégageant des perspectives.

Chapitre 1

Les systèmes auto-organisés et / ou émergents

1.1 Introduction

D'innombrables exemples de systèmes naturels montrant une organisation de « haut niveau » sont présents autour de nous : des oiseaux qui volent en escadrille, le développement de colonnies de bactéries, la variation périodique des populations dans un système « prédateur proie », le déplacement cohérent d'un banc de poisson, les fourmilières, etc.

La notion de ces comportements n'est pas spécifique au vivant, elle s'observe également dans certains systèmes chimiques alimentés en continu, à la formation des dunes, des rivages, des galaxies, des systèmes planétaires, des sociétés, etc. Au niveau microscopique, les éléments constituant ces systèmes obéissent à des règles spécifiques à première vue relativement simples. Au niveau macroscopique, leurs comportements globaux dépassent le simple ensemble de règles de leurs parties. En effet, les comportements de ces systèmes émergent d'un ensemble de comportements individuels et d'interactions entre les parties constitutives du système. C'est la raison pour laquelle tous ces systèmes donnent l'impression d'avoir des comportements spontanés, irrationnels, machinaux, d'avoir une structure décentralisée et d'être caractérisés par une organisation dissimulée. C'est pourquoi, on définit ces spécificités et particularités comme une « émergence » et / ou une « auto-organisation ». En effet, soit l'interaction des éléments du système est considérée comme une émergence en vue d'une adaptation définie comme une auto-organisation ; soit l'interaction est une auto-organisation synonyme d'une émergence.

D'une manière générale et malgré la variété des caractérisations de ces systèmes, on remarque un ensemble d'attributs les spécifiant tels que : « organisation », « émergence », « adaptation », « dynamique », et ils sont souvent associés et précédés du préfixe « auto ». Cependant, on constate que les termes les plus souvent utilisés pour désigner le comportement global de ces

systèmes sont « l'auto-organisation » et « l'émergence ». Les principes et les propriétés fondamentales liés à ces deux notions ainsi que la distinction entre elles font l'objet de ce chapitre.

La suite de ce chapitre est organisée de la manière suivante. Dans une première partie nous recensons de façon non-exhaustive la notion d'auto-organisation, d'émergence et les propriétés associées. Nous proposons une définition d'un système auto-organisé et / ou émergent. Nous présentons quelques mises en œuvre pratiques de tels systèmes. Ensuite, nous montrons l'intérêt de l'utilisation d'une technologie reconfigurable de type *FPGA* pour la réalisation de tels systèmes.

1.2 Auto-organisation

Le problème majeur parmi toutes les définitions que l'on rencontre dans la littérature est souvent la désignation d'un même terme en lui attribuant des propriétés parfois très différentes. Notre objectif n'est pas de donner une définition consistante de ce terme, mais plutôt d'analyser les contextes d'apparition de ce concept avec tous les attributs qui lui sont attachés. Ensuite, de proposer une réalisation matérielle d'un tel système en mettant en œuvre ces propriétés grâce aux technologies reconfigurables de type *FPGA* à travers l'exécution de procédés de reconfiguration dynamique que l'on trouve dans les systèmes reconfigurables [Tan01].

1.2.1 Définitions courantes

Avant de présenter les définitions des systèmes possédant des propriétés d'auto-organisation les plus souvent rencontrées dans la littérature et d'assigner des définitions à ce principe dans le domaine des systèmes reconfigurables, nous rappelons les définitions des racines des mots suivants : *« organiser »*, *« organisation »* et *« auto »*.

Le dictionnaire français Hachette [HAC] définit les termes *« organiser »*, *« organisation »* et *« auto »* de la manière suivante :

organiser (verbe)

1. *Mettre en place (les éléments d'un ensemble) en vue d'une fonction, d'un usage déterminés. Organiser un service.*
2. *Préparer, monter. Organiser un voyage, un spectacle. Régler, aménager. Organiser ses loisirs, son temps.*
3. *v. pron. Devenir organisé. Les secours s'organisent. Prendre ses dispositions pour agir efficacement.*

organisation (nom féminin)

1. *Manière dont un corps est organisé ; structure. Organisation des reptiles, d'une cellule.*
2. *Action d'organiser. Voulez-vous vous charger de l'organisation de la fête ?*
3. *Manière dont un ensemble quelconque est constitué, réglé. Organisation judiciaire.*
4. *Association, groupement.*

auto- *Élément, du gr. autos, « soi-même ».*

D'après ces définitions, la notion d'« auto-organisation » d'un système pourrait désigner un phénomène de structuration réglementé des éléments constituant le système et de leurs associations selon une vision fonctionnelle, dont le but est de délivrer un « service attendu ».

D'autres définitions du terme « auto-organisation » sont également proposées. L'encyclopédie en ligne [WIK] définit le terme « auto-organisation » de la façon suivante :

Auto-organisation :
L'auto-organisation est un phénomène de mise en ordre croissant, et allant en sens inverse de l'augmentation de l'entropie ; au prix bien entendu d'une dissipation d'énergie qui servira à maintenir cette structure. Le terme auto-organisation fait référence à un processus dans lequel l'organisation interne d'un système, habituellement un système hors équilibre, augmente automatiquement sans être dirigée par une source extérieure. Typiquement, les systèmes auto-organisés ont des propriétés émergentes (bien que cela ne soit pas toujours le cas).

Le dictionnaire en ligne de l'Office Québécois de la langue française [GRD] donne une définitions plus courte :

Auto-organisation (Domaine : informatique)
équivalent anglais : *self-organization*
Définition :
Capacité d'un système d'organiser (arranger) sa structure interne.

De ces définitions, nous pouvons déduire que le concept d'auto-organisation caractérise la notion d'organisation interne d'un système par ses propres moyens et sans aucun contrôle extérieur. Par la suite, nous adoptons cette caractérisation comme propriété principale de la notion d'auto-organisation d'un système.

1.2.2 Premiers principes d'auto-organisation : genèse

1.2.2.1 Le principe d'auto-organisation par *Ashby*

La première apparition du terme d'auto-organisation a lieu dans les années 40 avec les travaux de *William R. Ashby*, psychiatre-ingénieur anglais [Ash47, Ash62]. En effet, il a été introduit les premières définitions de l'auto-organisation et les termes relatifs associés. Dans ces travaux, il a été considéré :

- d'une part, que l'organisation d'un système présentait une dépendance fonctionnelle de ses états futurs par rapport aux états actuels et entrées éventuelles du système. Nous pouvons y voir une analogie avec les systèmes numériques séquentiels par opposition aux systèmes combinatoires.
- d'autre part, qu'un système auto-organisé organise lui-même sa structure interne sans directives extérieures.

Plus précisement, les travaux présentés spécifient la notion du terme « organisation » en lui attribuant les propriétés de « conditionnalité » et de « réductibilité ». Il est considéré que dans un système quelconque la notion d'organisation est présente dès qu'une relation entre deux entités A et B constituant le système devient conditionnelle sur une valeur ou un état de l'entité C faisant également partie du système. Une approche réductionniste a été adapté en considérant que le système est complexe et est composé de plusieurs entités (système par « morceaux »). Une autre fonctionnalité définie dans ses travaux est la notion de « bonne » et « mauvaise » organisation. Un système quelconque dans un environnement quelconque est considéré de « bonne organisation » si l'ensemble des paramètres environmentaux pouvant affecter le système considéré sont en relation avec la fonction globale de ce système. Dans le cas contraire, le système est considéré de « mauvaise organisation ». Il souligne qu'il n'y a pas de « bonne organisation » d'un point de vue absolue et que ce terme est relatif et dépendant à la fois de l'environnement, de l'ensemble de paramètres environnementaux, du point de vue considéré, etc.

Au principe de l'auto-organisation, *Ashby* assigne deux significations. La première concerne le système dont le point de départ présente ses éléments constitutifs qui sont initialement indépendant les uns des autres et qui au fil du temps agissent vers un état de connections les uns avec les autres. Ce système est « auto-organisé » dans le sens où il change d'état « éléments constitutifs séparés » vers l'état « éléments constitutifs connectés ». Comme exemple de tel système auto-organisé, on trouve l'embryon du système nerveux central dont le développement part des cellules n'ayant pas ou presque aucun effet les unes sur les autres et qui au fil du temps évolue en formant des synapses et des dentrites vers un système connecté et conscient de ses parties constitutives.

La deuxième signification attribuée au principe d'auto-organisation concerne les systèmes dont l'organisation change, évolue d'une « mauvaise » vers une « bonne » organisation préala-

blement définies. Comme exemple de ce genre d'auto-organisation, *Ashby* cite l'exemple d'un enfant qui commence par un état d'esprit l'incitant à mettre le feu partout jusqu'au moment où un « changement » dans son état d'esprit se produit et l'enfant prend conscience des risques liés au feu et cesse de le faire. De plus, *Ashby* prouve analytiquement qu'il n'y pas de systèmes étant auto-organisés de cette manière au sens littéral du terme. C'est-à-dire, des systèmes qui passent « soudainement » d'une « mauvaise » à une « bonne » organisation. Il constate que le changement soudain d'un système d'une « mauvaise » vers une « bonne » organisation peut se produire uniquement dans le cas où ce changement est initié de l'extérieur à partir d'une entrée supplémentaire du système considéré. De manière plus formalisée, si on considère un ensemble S d'états dans lequel un système quelconque peut se trouver, l'organisation de ce système est définie par une fonction $f : S \rightarrow S$ transformant l'ensemble S dans lui-même. Pour qu'un système puisse passer d'une organisation f vers une organisation g (caractérisé par une variable $\alpha(t)$ ayant comme valeur initiale f et comme valeur finale g), il doit exister un événement externe qui permettra d'effectuer ce changement. En effet, comme indiqué précédemment, ce changement d'organisation ne peut pas être uniquement à l'origine des changements actuels du système. De plus, *Ashby* souligne que pour qu'un système soit considéré comme auto-organisé, l'ensemble des états S doit être élargi par la cause étant à l'origine de cet évènement externe. Sinon le système est contradictoire à sa définition.

Ces premières définitions de la notion d'auto-organisation ont été par la suite complétées à la fois par les travaux suivants de *Ashby* et par les travaux de *Mesarovic* et *Lendaris* [Mes62, Ash62, Len64]. Ces premières définitions deviendront par la suite la référence à toutes nouvelles définitions de la notion d'auto-organisation rencontrées dans la littérature.

1.2.2.2 Le principe d'auto-organisation par *Lendaris*

Une approche plus mathématique que celle proposée par *Ashby* a été présentée par *Lendaris* [Len64]. Cette approche est applicable pour tous les systèmes de traitement d'information possédant des entrées / sorties par lesquelles ces informations arrivent et sortent traitées du système.

Un système S est alors défini par un groupe de trois ensembles $[I, O, R]$ des valeurs présentant les entrées paramétrables et non-paramétrables (ensemble I), les sorties (ensemble O) et les relations entre les entrées paramétrables et les sorties (ensemble R). Il définit également la notion de *domination* d'un système A vers un système B comme étant les relations entre deux systèmes où les entrées paramétrables du système B dit dominé présentent les sorties du système A dominant, sans relations entre les sorties du système B dominé et les entrées du système A dominant. Sur ces bases, il a prouvé par théorème que pour un système ainsi défini, il existe un ensemble D de relations entre les entrées paramétrables et les sorties qui par la règle de domination fait du système, un système auto-organisé par rapport à cet ensemble de

relations. Bien que cette approche sur le principe d'auto-organisation soit mathématiquement prouvée, elle reste limitée dans la mesure où elle n'apporte pas de réponses ou de précisions sur les caractéristiques principales des systèmes intégrant le principe d'auto-organisation.

1.2.3 Notion d'auto-organisation des systèmes

Les approches proposées par *Ashby* et *Lendaris* restent relativement abstraites. Les premiers travaux avoir introduit de manière pragmatique la notion d'auto-organisation dans les systèmes sont les travaux de *Von Neumann* sur les automates cellulaires [Neu66] et les travaux de *Wiener* fondant les bases de la cybernétique [Wie48]. Les automates cellulaires présentent des modèles numériques discrets où l'état d'une cellule évolue en fonction de l'état des cellules voisines [Neu66]. Les travaux de *Wiener* portent sur l'étude en parallèle de l'organisation, des mécanismes de contrôle et de la communication dans les systèmes vivants et artificiels. Ces travaux ont été suivi par les travaux de *Turing* [Tur52] sur la morphogenèse (conception de modèles où le couplage de réactions chimiques et de la diffusion des réactifs produit des motifs en bandes) et les travaux de *Prigogine* sur les structures dissipatives (structures stationnaires hors équilibre où la dissipation d'énergie entretient une organisation locale) [Pri68]. Tous ces travaux ont permis de souligner l'importance des rétroactions, des non linéarités et du caractère ouvert et hors équilibre des systèmes pour qu'il y apparaisse des formes stables et reproductibles sans un plan d'ensemble ni prescription extérieure.

1.2.3.1 Définition d'un système auto-organisé

Actuellement, les définitions récentes du principe d'auto-organisation des systèmes rencontrées dans la littérature sont plus ou moins cohérentes. Les principales différences portent sur certaines caractéristiques de ce concept. D'une manière générale, un système auto-organisé est défini par le lemme suivant : « Un système auto-organisé est un système complexe et modulaire. » En se basant sur la notion de modularité et de complexité d'un système, des définitions compréhensibles de ce principe sont apparues. Une liste non-exhaustive de définitions relatives au principe d'auto-organisation est donnée ci-dessous :

Définition 1 *[Dem98] : « L'auto-organisation d'un système se rapporte exactement à ce qui est suggéré par son nom : propriété de systèmes qui semblent s'organiser sans direction, manipulation, ou commande externe. »*

Cette définition met l'accent sur l'absence de contrôle extérieur dans la gestion d'un système auto-organisé. Cette propriété d'un système d'organiser sa structure interne par ses propres moyens et sans être dirigé de l'extérieur donne au système un caractère « auto-organisé ». D'autres définitions [Ger05] mettent en avant de nouvelles propriétés d'un système, telles que les propriétés de « *dynamique* » et d'« *interactivité* » :

Définition 2 *« Un système auto-organisé est un système dont les éléments constitutifs* ***inter-agissent*** *afin d'atteindre* ***dynamiquement*** *une fonction globale. »*

La conséquence de l'interactivité entre les modules constituant le système et de leur dynamique est la fonction globale du système correspondant au service que le système délivre à son environnement. Cette fonction globale du système n'est pas « déterminable » à partir des fonctions locales de chacune des entités. Elle présente simplement le résultat de l'interactivité de toutes les fonctions locales des éléments du système. Le rôle d'un système auto-organisé est donc de s'adapter à toutes les perturbations extérieures sans dégrader le fonctionnement du système. De plus, le fonctionnement global du système auto-organisé n'est pas centralisé d'un point de vue de contrôle, car cela ne répond pas à la dynamique exigée d'un tel système.

Une définition plus complète introduisant la notion d'adaptabilité, à partir des définitions précédentes, a été présentée dans les travaux de *De Wolf* et *Holvoet* [WH05] :

Définition 3 *« L'auto-organisation des systèmes est un processus adaptatif et dynamique dans lequel les systèmes acquièrent et maintiennent leur structure eux-mêmes, sans aucun contrôle externe. »*

De Wolf et *Holvoet* constatent que la structure d'un système peut être aussi bien spatiale, temporelle que fonctionnelle. Ils mettent également l'accent sur la gestion autonome du système, sans manipulation et interférence avec des éléments extérieurs au système. Cela ne signifie pas que le système reste insensible aux changements des paramètres environnementaux. Bien au contraire, il s'adapte de façon autonome et indépendante à son environnement. Selon *De Wolf* et *Holvoet*, un système dit auto-organisé est capable de « croître » par rapport à certains critères tels que le degré d'organisation de sa structure et le degré de fonctionnement. Ainsi, en partant d'une structure initiale moins adaptée aux exigences d'un fonctionnement voulu, le système croît vers une structure et un fonctionnement permettant de délivrer un service souhaité avec « moins d'efforts » et plus d'efficacité.

D'autres définitions mettent l'accent plutôt sur les caractéristiques attendues d'un système auto-organisé. Par exemple, *Heylighen* et *Gershenson* proposent une définition plus explicite du principe d'auto-organisation [HG03]. Plus précisément, un système auto-organisé doit non seulement adapter son comportement aux changements éventuels par modification de sa structure, mais également générer lui-même sa structure de fonctionnement. En effet, une structure ne présente qu'une disposition des composants constituant le système d'une manière particulière avec des fonctions préalablement définies. Par conséquent, l'auto-organisation est une structuration fonctionnelle qui apparaît et se maintient de façon spontanée. Les systèmes auto-organisés doivent être caractérisés par un contrôle distribué, généralement enfoui dans tous les composants du système afin d'éviter que la défaillance d'un élément contrôleur entraîne un dysfonctionnement complet du système. De plus, la propriété de *robustesse* est à considérer.

Un système auto-organisé doit être capable de résister à une grande variété d'erreurs, de perturbations, voire même de perte temporelle ou destruction complète d'une de ses entités. Bien évidemment, lorsque le degré d'endommagement atteint un certain niveau, le système commence à fournir une fonction principale dégradée jusqu'à ne plus répondre au service attendu si l'endommagement augmente de nouveau. L'adaptation aux changements environnementaux s'effectue par changement ou réorganisation de structure interne du système tout en veillant à ce que des mécanismes d'apprentissage utilisés permettent au système une mémorisation de résolutions. *Heylighen* et *Gershenson* soulignent également l'importance de l'interactivité des entités du système à la fois au niveau local entre les éléments voisins et au niveau système.

A partir de ces définitions, nous proposons ci-dessous une définition synthétique et plus globale d'un système auto-organisé.

Définition de l'auto-organisation des systèmes :
« *On définit l'auto-organisation d'un système comme la capacité d'un système complexe et modulaire à se restructurer sans contrôle extérieur par l'interactivité des éléments le constituant dans l'objectif de s'adapter aux changements imprévus de son environnement ou d'optimiser son fonctionnement selon les critères préétablis* »

Le principe d'auto-organisation des systèmes repose donc sur des propriétés principales dont les caractéristiques sont détaillées dans la section suivante.

1.2.3.2 Propriétés des systèmes auto-organisés

Les principales caractéristiques d'un système auto-organisé sont les suivantes :

– ***La complexité***. La propriété de complexité est une des principales caractéristiques d'un système auto-organisé qui par ailleurs apparaît dans les premiers travaux initiaux relatifs à l'auto-organisation [Ash47]. La propriété de complexité présente elle-même un concept complexe et difficile à définir. Il n'y a pas vraiment de définition générale de la propriété de complexité puisque ce terme peut prendre différentes significations dans des contextes très variés [Edm95]. Néanmoins, un système est considéré complexe si il est composé de plusieurs éléments *interactifs* de sorte que le comportement global du système ne peut pas être déduit des comportements individuels des éléments le constituant [Sim06]. Un exemple de système présentant la propriété de complexité est une cellule biologique. Une cellule est un système « vivant », qui manifeste la « vie » au niveau macroscopique tandis que ces éléments microscopiques constitutifs ne manifestent aucune forme de vie. A ce niveau, la vie « émerge » de la complexité de la cellule et des interactions des éléments qui la forment. Ces propriétés d'un système qui n'apparaissent pas aux niveaux microscopiques mais seulement au niveau système et qui présentent le résultat des fortes

interactions de ses éléments constitutifs sont appelées propriétés *émergentes* [And72].

- ***L'interactivité***. Les éléments constituant un système complexe sont caractérisées par un degré très fort d'interactivité, tant au niveau local entre les éléments voisins directs qu'au niveau global [Ger05]. En d'autres termes, les éléments d'un système échangent des « messages », analysent ensemble les changements des paramètres environnementaux et prennent des décisions afin de s'adapter aux changements ou prédisent de nouveaux changements en réorganisant leur structure interne. Le comportement de système porte en grande partie sur les interactions de ses éléments constitutifs. Avec la complexité, le degré d'interactions entre les éléments devient tellement important qu'il rend impossible l'observation du tel système comme la simple somme de ses éléments. On dit que le comportement d'un tel système « émerge » des interactions des éléments le constituant. Par ailleurs, on définit la complexité d'un système comme une fonction à la fois du nombre d'éléments constituant le système, du nombre d'interactions entre eux, de la complexité de chaque élément et de la complexité des interactions entre les éléments [Ger02].
- ***La dynamique***. Un système auto-organisé peut être considéré comme un processus dynamique évoluant dans le temps. Étant donné qu'un système auto-organisé évolue dans un environnement variable et dynamique, le système lui-même doit posséder la même dynamique que l'environnement dans lequel il évolue. Cette dynamique lui permet de réagir dans des délais raisonnables à tous changements pouvant survenir. De plus, un système est caractérisé par une dynamique et une réaction à des changements plus importantes si son état est « loin de l'équilibre » (*far-from-equilibrium*) [Hey03]. Plus précisement, l'équilibre d'un système est caractérisé par l'absence « d'entropie », c'est à dire par le fait que dans l'état d'équilibre le système ne dissipe pas d'énergie [NP77]. Autrement dit, dans l'état d'équilibre la consommation d'énergie d'un système est ramenée au minimum possible. Si il n'y a pas d'énergie supplémentaire « injectée » de l'extérieur, le système reste « coincé » dans son état d'équilibre. C'est pourquoi le système, doit se maintenir loin de son équilibre, dans un état qui lui permet un échange constant d'énergie entre ses éléments constitutifs et son environnement, correspondant à l'état plus sensible, mais plus réactif et dynamique aux changements. Au lieu de réagir à chaque changement par un *feedback* négatif qui ramenerait le système dans son état d'équilibre, le système se maintient loin de l'équilibre où il peut produire une large variété d'actions menant à plusieurs configurations stables.
- ***L'adaptabilité***. L'adaptabilité d'un système est sa capacité de maintenir sa fonction globale dans la mesure du possible tout en adaptant son comportement aux perturbations provenant de son environnement. Pour pouvoir maintenir le comportement voulu d'un système en présence de perturbations prévues ou imprévues, le système doit soit réagir avant que les perturbations se produisent (*feedforward*, [HJ01]) par anticipation à ces

éventuelles perturbations, soit après qu'elles se soient produites (*feedback*) en essayant de « neutraliser » les éventuelles dommages que ces perturbations peuvent provoquer. A ces deux types d'adaptations pouvant être combinées nous rajoutons une approche asymétrique provoquant un effet tampon entre perturbations et réactions du système (*buffering*).

- ***La robustesse et la flexibilité***. Les propriétés de robustesse et de flexibilité sont souvent employées dans le cadre d'adaptabilité des systèmes auto-organisés. On attend d'un système auto-organisé qu'il fasse face à tous les changements soit environnementaux soit internes et qu'il organise sa structure interne d'une manière autonome et spontanée. Pour répondre à tous changements, un tel système doit disposer d'une large variété de comportements différents [Ger05]. Le comportement d'un système évolue d'un comportement moins « organisé », moins « adapté » (par rapport à plusieurs paramètres tels que ressources utilisées, etc.) vers un comportement plus approprié, plus accordé aux exigences du système. Le système doit d'une part avoir un nombre suffisant d'actions et de comportements (propriété de robustesse) pour répondre à toutes les perturbations possibles ; d'autre part il doit être capable de choisir l'action et le comportement les plus appropriés à la perturbation survenue (propriété de flexibilité) [Hey03]. La variété d'actions et de comportements d'un système est assurée par son maintien loin de l'équilibre dans un ou plusieurs états possibles dans lequel le système peut se trouver (voir propriété de dynamique). Une très grande variété d'actions et de comportements peut rendre le système incontrôlable, tandis qu'une grande sélectivité par action rend le système moins flexible [BGZ04].
- ***L'anticipabilité***. Un système disposant de la propriété d'anticipabilité est défini comme système possédant un modèle prédictif de lui-même et / ou de son environnement, afin de permettre de changer d'état à un instant donné tout en restant en accord avec son modèle prédéfini [Ros85]. Un système auto-organisé doit être capable de prévoir des changements et des perturbations, puis d'adopter son comportement global afin d'y répondre dans un délai raisonnable. L'anticipabilité peut être considérée comme un cas particulier de l'adaptabilité où le système ne doit pas forcement passer par un changement pour pouvoir y répondre. Les changements produits par l'anticipation d'un évènement peuvent être considérés comme une sorte de préadaptation avant que cet évènement se produise. Il est évident que l'aspect le plus particulier des systèmes disposant de la propriété d'anticipabilité mis au premier plan par cette définition présente leur dépendance non seulement sur les états futurs mais aussi sur les états précédents du système. Dans [PHF07] deux types d'adaptabilité des systèmes sont définis : une adaptabilité explicite où le système prédit intérieurement et représente ses états futurs au sein du système et une adaptabilité implicite où le système coordonne son comportement avec les états futurs sans les représenter explicitement dans le système.

- ***Le contrôle décentralisé***. Pour exercer une fonction précise ou pour s'adapter à des changements environnementaux, chaque système doit disposer d'un mécanisme de contrôle. Plus précisément, d'un module dont la fonction principale consiste à gérer et coordonner tous ses entités. Le contôle d'un système peut être soit centralisé, sous forme d'un module unique, soit décentralisé, c'est-à-dire reparti et distribué dans plusieurs entités du système. Le contrôle centralisé n'est pas adapté pour les systèmes auto-organisés, car en général un système auto-organisé est constitué d'entités de nature hétérogène et de forte dynamique. Le contrôle centralisé pour de tels systèmes est impossible [Ser04]. Le principe du contrôle décentralisé réside dans le fait que seuls les mécanismes locaux (mécanismes de chaque module du système) sont utilisés dans la gestion globale du système. Autrement dit, les éléments constituant le système ne disposent pas d'informations sur le fonctionnement global du système. Les seules informations dont ils disposent sont les données relatives à leurs fonctionnements propres et celles de leurs voisins directs. Pour l'échange d'informations, la distribution du contrôle doit s'appuyer sur une forte interactivité des modules (voir la propriété d'interactivité). Par conséquent, le contrôle décentralisé contribue à l'« émergence » de la fonction globale du système qui ne peut pas être déduite comme la simple somme des comportements de chacun des entités du système [WH05]. C'est pourquoi, la propriété de contrôle décentralisé est souvent associée à la notion d'*émergence*.
- ***« L'augmentation de l'ordre d'un système » (Increasing in order)***. Une des propriétés les plus importantes d'un système auto-organisé est que son comportement manifeste une organisation, une structuration et un arrangement de ses entités de telle manière à permettre l'accomplissement des fonctions globales possibles par système. Le fait d'effectuer une fonction spécifique parmi plusieurs possibles réduit l'espace des états du système. Le groupe d'espace d'états utilisé pour effectuer une fonction s'appelle *attracteur* (*attractor*) et représente un état stable du système exprimant une structure possible de ce dernier et se manifestant à travers une fonction correspondant à une structure de système donnée [Luc97]. Le système peut avoir plusieurs *attracteurs*, plusieurs états stables correspondant à ses différentes fonctions et son état peut évoluer d'un *attracteur* vers un autre. L'évolution du système d'un *attracteur* vers un autre est considéré comme un changement d'ordre du système. Si le changement d'ordre du système mène vers une structure spatiale, temporelle ou fonctionelle mieux adaptée aux besoins du système pour une fonction donnée, on parle de l'augmentation de l'ordre du système. Une approche plus formelle basée sur la *complexité statistique* (*statistical complexity*) permettant de décrire l'augmentation de l'ordre d'un système est présentée dans [Sha01]. La *complexité statistique* est une manière de mesurer la complexité d'un système, effectuant une fonction précise, à travers une *mémoire historisée*. Par exemple, la *mémoire historisée* d'un

système devient plus importante si le nombre des entités du système augmente. L'augmentation de l'ordre d'un système implique qu'il démarre d'un état initial totalement aléatoire ou semi-organisé (par exemple un état sans mémoire historisée) [GH00]. Il est aussi possible que démarrant d'un état initial ou d'un état quelconque le système se déplace vers un état caractérisé par moins d'ordre par rapport à son état précèdent. Un système sans ordre ou possédant trop d'ordre dans son organisation ne peut pas produire un comportement utile. Il est possible que le comportement d'un système évolue jusqu'à un état très complexe dans lequel il ne peut plus effectuer des fonctions utiles. Les systèmes se trouvant entre ces deux extrémités peuvent manifester des comportements plus flexibles et organisés [Hak00, Lan90].

– ***L'autonomie***. Le système auto-organisé doit s'organiser « spontanément », sans aucun contrôle externe, sans aucune interférence avec l'extérieur [Hey89, MRF+03]. Cela ne signifie pas qu'un tel système n'a aucun lien avec l'extérieur. Bien au contraire, un tel système possède des entrées / sorties à travers lesquelles il reçoit / envoie des données depuis ou vers l'extérieur. Cependant, ces données ne correspondent pas à des instructions de contrôle. Ceci signifie qu'en présence de perturbations environnementales, ce système n'attend pas de signaux de contrôle extérieurs pour adaptater son comportement par une réorganisation éventuelle de sa structure. Le système lui-même à travers ses éléments constitutifs initie d'une manière autonome et independante des actions à effectuer afin de répondre à des changements potentiels.

1.2.4 Notion d'émergence dans les systèmes auto-organisés

A la plupart des définitions et descriptions du principe d'auto-organisation ou de ses propriétés est souvent associé le terme d'émergence. Dans la littérature, certains auteurs constatent que l'émergence fait partie intégrale de l'auto-organisation. Tandis que d'autres notent que l'émergence peut exister indépendamment de l'auto-organisation et être très bien combinée avec les propriétés propres à l'auto-organisation [WH05].

La figure 1.1 illustre un système auto-organisé et ses propriétés associées de manière non-exhaustive, ainsi que le domaine d'intersection entre les propriétés du principe de l'auto-organisation et de l'émergence. A cette liste de propriétés spécificiée par la figure 1.1, nous trouvons également les propriétés de « *micro-macro effet* », « *nouveauté radicale* », « *cohérence* » et « *bi-directionnalité* » [WH05]. Ces propriétés liées à l'émergence complètent celles déjà définies comme des propriétés des systèmes auto-organisés (figure 1.2). La section suivante spécifie la notion d'émergence et détaille ses propriétés principales et les points de distinction principaux par rapport à l'auto-organisation.

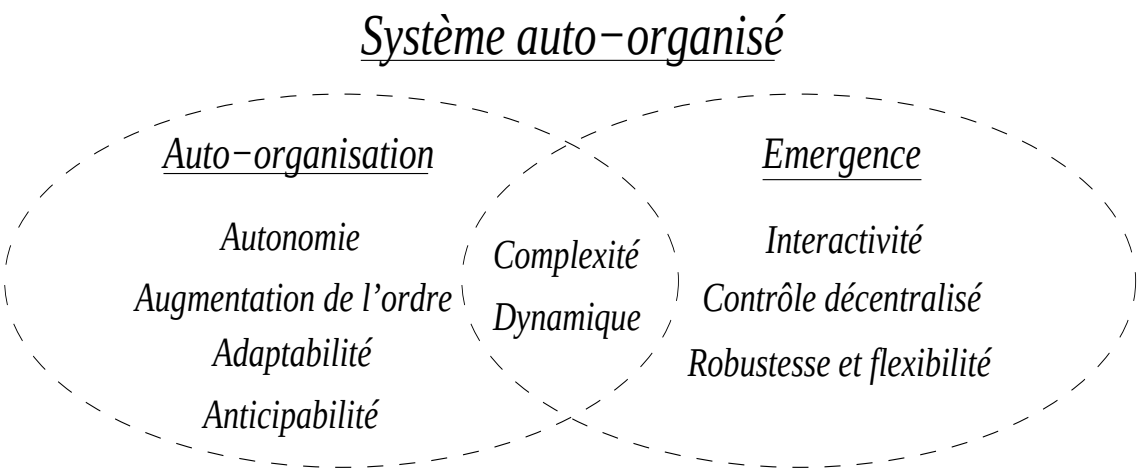


FIG. 1.1 – Propriétés partagées entre l'auto-organisation et l'émergence.

1.2.4.1 L'émergence - définition

Le dictionnaire français Hachette [HAC] définit les termes « émerger » et « émergence » de la manière suivante :

émerger v. intr.

1. *Se dégager, sortir d'un milieu après y avoir été plongé*
2. *ASTRO Réapparaître après avoir été occulté, en parlant d'un astre.*
3. *Fig. Sortir de l'ombre, apparaître plus clairement.*
4. *Fig., fam. Sortir du sommeil, d'une situation difficile. Avoir du mal à émerger.*

émergence n. f.

1. *Action d'émerger ; état de ce qui émerge. Point d'émergence d'une source : l'endroit par où elle sort.*
2. *PHYS Point d'émergence (d'un rayon lumineux).*
3. *ASTRO Émersion.*
4. *Fig. Apparition soudaine de quelque chose, arrivée au premier plan de quelqu'un.*

L'encyclopédie en ligne [WIK] définit le principe d'émergence de la façon suivante :

Émergence - *Un phénomène qu'on trouve dans les domaines physiques, biologiques, écologiques, socio-économiques et autres systèmes dynamiques comportant des rétroactions. Elle est caractérisée par :*

1. *l'ensemble fait plus que la somme de ses parties. Ceci signifie qu'on ne peut pas prédire le comportement de l'ensemble par la seule analyse de ses parties.*
2. *l'ensemble adopte un comportement caractérisable sur lequel la connaissance détaillée de ses parties ne renseigne pas.*

À partir d'un certain seuil critique de complexité, ces systèmes voient apparaître de nouvelles propriétés, dites propriétés émergentes. Celles-ci deviennent observables lorsqu'elles vont dans le sens d'une organisation nouvelle (voir aussi auto-organisation).

Le point commun entre ces définitions est qu'elles attribuent à la notion d'émergence d'un système le fait d'apparition plus ou moins spontanée et soudaine de nouvelles propriétés d'ordre supérieur d'un système.

L'émergence n'est pas une nouvelle notion. Les postulats à l'origine de l'émergence sont « le tout est plus grand que la somme de ses constituants », apparu à l'époque antique, ou encore le terme allemand « Gestalt » signifiant *forme ou configuration d'éléments unifiés dans un ensemble dont la structure ne peut pas être déduite de ses constituants*. De même, nous trouvons aussi des traces écrites de *Thalès* et *Anaximandre* des concepts d'émergence et d'auto-organisation. Ces premières notions ont servi de base pour expliquer certains phénomènes non décomposables.

La première utilisation du terme d'émergence dans le cadre des systèmes dynamiques fût

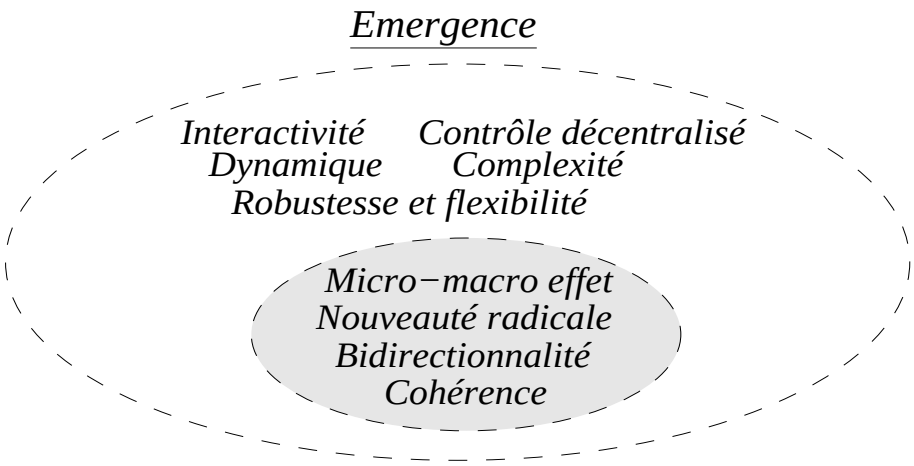


FIG. 1.2 – Propriétés d'un système émergent.

en 1875 par le philosophe anglais G. H. Lewes [Lew75]. Il est le premier à avoir introduit le terme « émergent » pour distinguer les « phénomènes traçables », caractérisés par une séquence d'étapes les produisant et nommés *les résultants*, des « phénomènes intraçables » pour lesquels il n'est pas possible d'établir les étapes les produisant et appelés *les émergents*. Cependant, il faut attendre le milieu du XIXe siècle pour voir apparaître un mouvement de pensée (appelé « proto-émergentisme ») autour du concept d'émergence. Les proto-émergentistes cherchaient principalement à définir *l'émergence* afin de distinguer et différencier les phénomènes non explicables et explicables grâce à des théories ou modèles. Ainsi, il en résulte une vision du processus d'émergence comme une « boîte noire » possédant des entrées bas niveaux et des sorties de plus hauts niveaux dont la relation entre ces entrées-sorties n'est pas connue.

Ce fût qu'à la deuxième moitié du XXe siècle qu'un nouveau mouvement de pensée (appelé « néo-émergentisme ») commence à explorer cette boîte noire [Gol99]. Il en résulte de nouvelles recherches sur le principe d'émergence, liées à la *théorie de la complexité* et prenant racines dans les approches de la *dynamique des systèmes* en physique, en mathématiques et en informatique. La diversité d'approches sur lesquelles les nouvelles recherches se sont basées a permis la création de plusieurs écoles de recherche du principe d'émergence en parallèle [Hol98, New96, Hak84, Nic89]. De l'ensemble de ces travaux, des premières définitions de la notion d'émergence ont pu être élaborées et caractérisées.

Une définition courante et élaborée de la notion d'émergence est donnée par *Goldstein* [Gol99]. Il définit l'émergence comme un phénomène d'apparition de structures originales et cohérentes, de modèles et de propriétés du processus d'auto-organisation dans les systèmes complexes. Ici l'émergence est un processus dynamique à un niveau global du système (*macro level*), issu des composantes et des processus de *micro-niveau* dont les interactions mutuelles locales le font surgir. Il attribue également au principe de l'émergence les propriétés de « nouveauté radicale », « cohérence », « corrélation », « dynamique » dont les significations sont détaillées dans la section suivante.

1.2.4.2 Propriétés propres à l'émergence

Plutôt que de prétendre donner une définition exacte et exhaustive de l'émergence, nous donnons, à partir de définitions tirées de la littérature, les particularités qui nous semblent fondamentales pour cerner de façon précise la notion d'émergence :

- ***L'effet micro-macro***. C'est la caractéristique la plus importante de l'émergence et la plus citée explicitement dans la littérature [Gol99, Hey89, Hey03, Ode02a, Par01, Ode02b]. La signification qui lui est attribuée est relative à des propriétés, comportements, structures ou des modes de fonctionnement d'un système qui apparaissent soudainement au niveau global (le plus haut niveau d'observation - *macro-level*), comme le résultat d'interactions des constituants du système au niveau local (le plus bas niveau - *micro-level*). Autrement

dit, le comportement global du système émerge des interactions de ses constituants et apparaît « soudainement » au premier plan tout en « cachant » les comportements locaux de ses constituants.

- ***« Nouveauté radicale » (radical novelty)***. L'émergence est caractérisée par l'apparition d'une nouveauté (propriétés, structures, formes ou fonctions) ne pouvant pas être décrite, expliquée ou prédite à partir des conditions de base définies aux niveaux inférieurs [VdV97]. Cette caractéristique de l'émergence est souvent appelée la *nouveauté radicale*, étant donné que le comportement du système n'est pas explicitement décrit par ses constituants. Cela provient directement de la définition de l'émergence de Goldstein [Gol99].
- ***La cohérence (coherence)***. C'est une caractéristique qui se rapporte à la corrélation logique et cohérente de tous les constituants du système [Gol99, Ode02b]. Les émergents du système (les parties *intraçables*) étant produits comme des résultats de fortes interactions du système, font également partie du système, aussi bien que les constituants dont les interactions les font apparaître. Ces parties émergentes, manifestées à travers un comportement nouveau et radical du système (en particulier d'un niveau *macro*), doivent être corrélées avec les constituants du système caractérisés par des comportements bien définis afin de préserver un sens logique dans le comportement global du système. Le comportement global d'un système ne peut pas être présenté comme la somme des comportements de tous ses constituants. Autrement dit, la cohérence met en corrélation les comportements des constituants du système avec le comportement global du système. Les composants de plus bas niveaux avec les composants de plus haut niveau. Cette propriété est aussi appelée *la fermeture organisationnelle (organisational closure)* [Hey03].
- ***L'interactivité (interactivity)***. Dans la plupart des définitions de la notion d'émergence, un des mots clé relatif à cette propriété est l'interactivité des constituants du système [Gol99, Hol98]. Sans interactions entre les constituants d'un système il n'y a pas d'émergence. Pour faire apparaître des propriétés émergentes d'un système, ses constituants doivent interagir (plus ou moins) entre eux. En effet, la simple juxtaposition de constituants (fonctionnement des constituants indépendamment les uns des autres) ne suffit pas. De plus, les propriétés de *micro-macro effet* et de la *nouveauté radicale* représentent la conséquence directe des fortes interactions des constituants du système [Cru94].
- ***La dynamique (dynamic)***. Cette propriété caractérise l'émergence comme un processus dynamique évoluant dans le temps [Cru94, Ode02b, Ode02a]. Les émergents (les parties émergentes d'un système) apparaissent à un moment donné comme le résultat des interactions des constituants du système. Ces émergents qui redonnent un nouveau sens au comportement global du système (c'est-à-dire une nouveauté dans le comportement par rapport à l'état précédent du système) ne surgissent pas aussitôt que le système se met

en marche. Il faut attendre que le degré d'interactivité, le degré d'échange d'informations entre les constituants du système atteigne un seuil suffisant pour faire apparaître les nouvelles formes de comportements du système. Cette évolution de comportement d'un système est caractérisée par la propriété de *dynamique* d'un système.

- ***Le contrôle décentralisé***. La propriété de contrôle décentralisé est associée à l'émergence. La totalité du contrôle d'un système est répartie et délocalisée dans ses constituants. Ce type de contrôle est basé seulement sur les mécanismes locaux. Aucune partie du système n'est chargée explicitement du contrôle global du système. Toutes les actions des constituant du système sont donc contrôlables, tandis que le comportement global du système ne l'est pas [WH05]. Ainsi, en distribuant le contrôle, afin de compenser le manque de contrôle centralisé le degré d'interactivité de constituants du système augmente cela favorisant l'apparition des propriétés émergentes.
- ***La bidirectionnalité (Two-way link)***. Les parties émergentes d'un système étant en rapport étroit avec les constituants les faisant surgir sont reliées à ces derniers d'une manière bidirectionnelle. Cela signifie que non seulement les constituants de bas niveau d'un système (*micro level parts*) influencent les parties émergentes (*macro level parts*) mais également les parties émergentes agissent dans l'autre sens. Les parties émergentes ont des effets causaux sur les parties de bas niveau. A titre d'exemple, nous pouvons citer la formation de chemin chez les fourmis. Le chemin créé par un groupe de fourmis présente la partie émergente (le niveau *macro*), qui influence le mouvement des fourmis au niveau *micro* parce que les fourmis sont guidées par les phéromones.
- ***La robustesse et la flexibilité***. Le fait qu'un système soit doté de contrôle décentralisé et qu'aucun de ses constituants ne représente un comportement global du système, laisse supposer qu'aucun de ses constituants ne présente un point de défaillance [Ode02b, Ode02a]. La défaillance d'un élément constituant le système ne provoquera pas sa défaillance complète. Bien évidemment, elle entraînera une perte de performances du système, sa dégradation fonctionnelle graduelle mais en aucun cas la qualité des sorties du système ne sera soudainement détériorée. Cette flexibilité rend le système robuste par substitution de ses constituants tout en gardant les structures émergentes.

1.2.5 L'auto-organisation et l'émergence des systèmes

Les principes d'auto-organisation et d'émergence sont similaires et mettent l'accent sur les différents aspects du comportement d'un système. La propriété essentielle de l'auto-organisation d'un système est son comportement adaptable, le conduisant de manière autonome vers un autre comportement caractérisé par la notion de l'« augmentation de l'ordre ». D'un autre côté, l'émergence d'un système est décrite comme une modification de son comportement global, radicalement nouveau par rapport aux comportements des entités constituant le système. Ces deux

principes sont dynamiques, robustes et évoluent dans le temps. Le principe d'auto-organisation d'un système est robuste par rapport à sa capacité d'adaptation et de maintien de *l'augmentation de l'ordre* (robustesse aux changements). Tandis que la robustesse de l'émergence est caractérisée par la flexibilité de la structuration des entités le constituant, qui sont également principalement à l'origine des propriétés d'émergence. Ainsi, la défaillance d'une entité d'un système ne doit pas nuire au maintien des propriétés d'émergence du système.

Les principes d'auto-organisation et émergence peuvent coexister ensemble (chacun influe sur différentes caractéristiques du système) ou isolément dans un système [WH05]. La figure 1.3a illustre un système auto-organisé sans émergence. Il symbolise une autonomie, une adaptation aux changements environnementaux, une capacité d'augmentation de son ordre par rapport à des exigences (fonctions) définies et attendues. Cependant, un tel système ne possède aucunes caractéristiques liées au principe d'émergence (*nouveauté radicale, micro-macro effet, contrôle décentralisé,* etc). La figure 1.3b illustre un système manifestant uniquement des propriétés de l'émergence. Ces propriétés résultent des interactions entre les entités constituant le système. Ces entités peuvent évoluer sans entraîner un changement structurel du système initial. La figure 1.3c donne une illustration générique d'un système auto-organisé. Plus précisément un système possédant à la fois les propriétés de l'auto-organisation et de l'émergence. Cette illustration représente la plupart des systèmes rencontrés et mettant en œuvre simultanément ces deux principes.

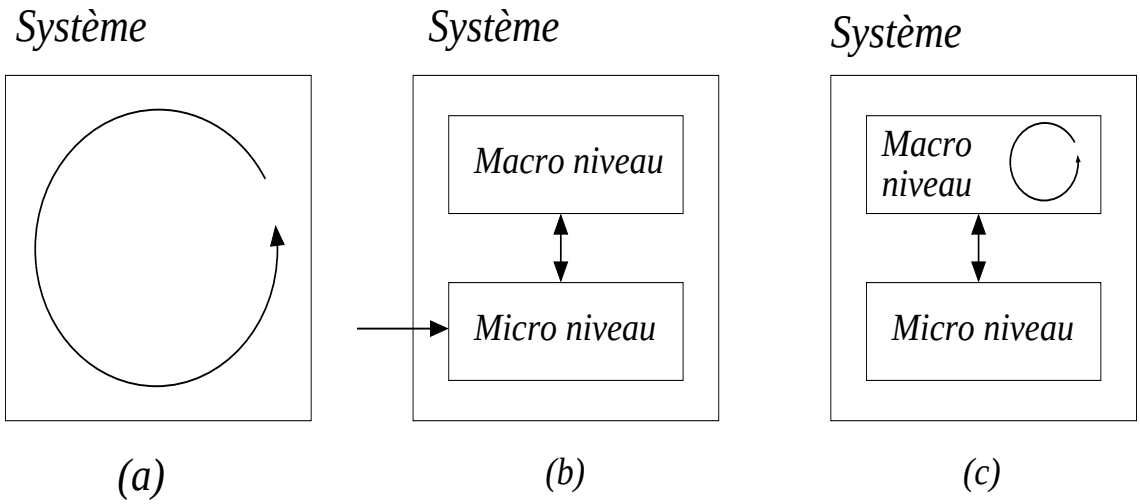


FIG. 1.3 – Illustrations des systèmes auto-organisés et/ou émergents

Dans la littérature, certains auteurs considèrent que les propriétés d'émergence d'un système sont la conséquence directe du processus d'auto-organisation au niveau *micro* du système [Hey89, MZ04]. Tandis que d'autres considèrent l'auto-organisation comme le résultat des processus d'émergence dans un système [Par01, PB04].

Par la suite, nous considérerons un système auto-organisé comme un système possédant à la fois les propriétés propres à l'auto-organisation et à l'émergence.

1.3 Mise en œuvre de l'auto-organisation et / ou émergence

Parmi les systèmes mettant en œuvre les principes d'auto-organisation et / ou d'émergence nous trouvons les *systèmes de calcul inspirés organique* (*organic computing*), les *systèmes de calcul autonome* (*autonomic computing*) et les *systèmes multi-agents* (*multi-agent systems*). Ces systèmes sont caractérisés par une complexité et une taille très élevées. D'une manière générale, ils sont caractérisés par une composition et une structuration de plusieurs entités simples ou de complexité moindre.

1.3.1 Systèmes de calcul autonome - *Autonomic computing*

La complexité croissante des systèmes à concevoir et les attentes de plus en plus exigeantes des systèmes en terme d'organisation et de gestion dans des environnements évolutifs, ont conduit aux développements de nouvelles approches de conception. Un exemple d'une telle approche est la conception des *systèmes de calcul autonome* (*autonomic computing systems*) [KJCD03]. Cette approche, à l'initiative de la société IBM, met au premier plan à la fois l'hétérogénéité grandissante des composants constituant les systèmes actuels et leurs interconnectivités de plus en plus complexes rendant les méthodes de conception existantes incapables à anticiper ou à prévoir les interactions des constituants des systèmes lors de leur fonctionnement. Ainsi, dans le but de répondre à de nouvelles exigences, une nouvelle approche inspirée des systèmes naturels (systèmes biologiques, sociétés) caractérisés par une grande autonomie et une autogestion est considérée [KJCD03]. D'où le terme *autonome* associé à ces systèmes en référence au système nerveux autonome capable de gérer « automatiquement » toutes les fonctions du corps humain (respiration, digestion, fréquence cardiaque, tension artérielle, etc).

La base d'un système de calcul autonome est un système autogestionnaire, libérant des administrateurs du système de la supervision de travaux de bas niveau tout en fournissant un comportement optimal et adapté aux besoins actuels de l'environnement dans lequel il exerce [KJCD03, NB07]. Les *systèmes de calcul autonome* sont gérés et contrôlés sans intervention directe ou indirecte humaine et reposent sur un ensemble de règles générales définissant le comportement global du système. Les systèmes de calcul autonome sont caractérisés par quatre grands principes (les *auto - * propriétés*, eng. *self-* properties*) :

– *L'auto-configuration* (*self-configuration*) : configuration automatique des composants du système,

- *L'auto-optimisation* (*self-optimization*) : contrôle automatique des ressources assurant un fonctionnement optimal du système dans des conditions définies,
- *L'auto-curation* (*self-healing*) : détection automatique des défauts et leurs corrections,
- *L'auto-protection* (*self-protection*) : identification et protection proactive contre des attaques arbitraires.

A ces quatre grands principes, de nouveaux principes ont été depuis introduit [PST+05, BDK+03, NOR03] :

- Récupération du système des routines de traitement provoquées par des changements environnementaux inattendus,
- Optimisation continue des états du système,
- Augmentation des connaissances sur les composants et le système lui-même (forme d'auto-aprentissage).

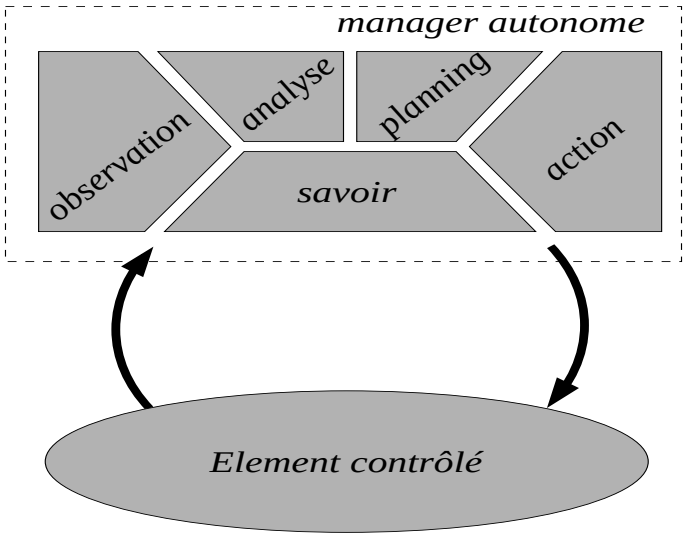


FIG. 1.4 – Élément de base d'un système de calcul autonome - *(autonomic element).*

La brique fondamentale d'un système de calcul autonome est l'*« élément autonome »* (*autonomic element*) dont le schéma synoptique est présenté sur la figure 1.4. L'*élément autonome* est composé d'un ou plusieurs *éléments contrôlés* (*managed element*) couplé(s) avec un *contrôleur autonome* (*autonomic manager*). L'*élément contrôlé* peut être soit une ressource matérielle (par exemple un processeur), soit une ressource logicielle (par exemple une base de donnée) ou bien une combinaison des deux. Le *contrôleur autonome* récolte toutes les informations relatives à l'*élément contrôlé* dans l'environnement, les analyse et planifie les actions appropriées devant être appliquer par *l'élément contrôlé.*

Chaque élément d'un *système de calcul autonome* est responsable de la gestion de son état, de sa structure interne et des interactions avec son milieu environnant et les autres *éléments autonomes* du système. Le comportement d'un *élément autonome* est défini par le concepteur

à différents niveaux d'abstraction de façon précise et transparente par rapport à ses voisins, à travers des règles établies au cours de la conception du système. Chaque *élément autonome* peut assurer seul ou en interaction avec d'autres éléments du système un nombre de « services » selon les règles prédéfinies par le concepteur dans la phase de conception. Un *élément autonome* ne pourra jamais accepter de faire partie d'un service ou d'une action qui enfreindra ses règles de base. En résumé, un *système de calcul autonome* est défini comme une somme d'*éléments autonomes* délivrant des « services » définis initialement par son concepteur.

Les caractéristiques principales et les politiques de gestion (*policy management)* de tels *systèmes de calcul autonome* sont décrites dans [WHW+04, ALL05]. La conception de ces systèmes autonomes, d'inspiration autonomique, pose néanmoins des problèmes de standardisation et il manque les mécanismes d'auto-gestion rendant un système autonome auto-organisé [HMG05].

1.3.2 Systèmes de calcul inspirés organique - *Organic computing*

Une autre approche cherchant à mettre en œuvre le principe d'auto-organisation dans les systèmes apparaît dans les systèmes de calcul inspiré organique (*Organic Computing*) [Sch05, MS04]. Cette approche repose sur l'étude et l'analyse des structures organiques à tous les niveaux d'abstraction (au niveau moléculaire, organisme, cognitif et social), dans le but d'extraire des connaissances qui peuvent contribuer à la conception de systèmes de calcul. L'objectif de cette démarche est de réaliser des systèmes de calcul capables de gérer tout changement de la même manière qu'un « être » organique le réalise.

Un système est appelé *organique* si tous ses composants et sous-systèmes sont coordonnés d'une manière utile et réfléchie [ORG]. Les structures organiques sont des processus hiérarchiquement imbriqués les uns aux autres, structurés de façon à répondre à des changements inattendus des environnements dans lesquels ils évoluent à travers des réactions « but-orientées » (*goal-oriented reactions*). De tels systèmes doivent être réactifs et « conscients » de leur environnement. Cette « conscience environnementale » s'obtient à travers un interfaçage avec de nombreux capteurs fournissant à ces systèmes des informations sur leur « voisinage » et leur environnement. Tandis que leurs réactivités sont reflétées à travers les tâches accomplies par leurs actionneurs agissant uniquement sous ordre du système lui-même. Contrairement aux systèmes classiques de calcul, la notion de calcul organique ne repose pas sur une répartion algorithmique des tâches à réaliser, mais sur des processus d'évolution, de développement, d'auto-organisation, d'adaptation, d'apprentissage et de réactions *but-orientées*. De tels systèmes de calcul inspirés organique doivent « incarner » la vie dans le sens propre du terme [ORG]. Ils doivent être à la fois robuste, flexible, fiable et sûr dans la mesure du possible. D'autre part, ils doivent être capable d'effectuer des « *auto* - * actions » telles que s'auto-organiser, s'auto-optimiser, s'auto-configurer, s'auto-proteger et s'auto-restaurer [MS04, ORG, RPB+].

Les méthodes de conception actuelles des systèmes complexes, qui reposent majoritairement sur des approches de type *top-down*,[4] ne sont pas adaptées à la conception des systèmes de calcul organique [MS04, Sch05], caractérisés par des comportements émergents et auto-organisés. En effet, les approches de conception des systèmes organiques doivent se focaliser principalement sur les relations entre les composants du système dont les interactions définissent le comportement global du système. Actuellement, aucune discipline scientifique ne propose de méthodologie permettant la mise en œuvre du principe d'auto-organisation. Dans ce contexte, l'élaboration d'une approche de conception de systèmes électroniques devant posséder des caractéristiques d'une structure organique, nécessite une interdisciplinarité (biologie, informatique, électronique, etc).

1.3.3 Systèmes multi-agents *(Multi-agents Systems - MAS)*

Afin de réduire la complexité croissante des systèmes informatiques et de les rendre distribués et répartis, le concept de systèmes « multi-agents » a fait son apparition dans les années 70 pour évoluer depuis les années 90 sur les concepts actuels des systèmes multi-agents.

Les systèmes « multi-agents » sont des systèmes constitués d'un ensemble d'entité (« agent ») évoluant dans un même environnement. Le terme « agent » est un terme générique se rapportant à des entités de nature différente (biologique, des robots autonomes, des logiciels informatiques et leurs composants) [FG97]. Un agent présente une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir [Fer95, DC96]. Par ailleurs, un agent est capable de communiquer avec d'autres agents et est doté d'un comportement autonome et flexible. Dans ce contexte, on attend d'un système multi-agent une flexibilité, une réactivité (réponses aux changements environnementaux), une pro-activité (un système qui génère et satisfait des objectifs) et une sociabilité correspondant à sa capacité d'interaction avec d'autres systèmes [Woo99]. La figure 1.5 illustre un schéma synoptique d'un agent.

D'une manière plus concrète, un système multi-agents est constitué d'un ensemble de processus s'exécutant simultanément dans le temps, communicants entre eux et partageant des ressources communes. Du fait d'une communication importante entre les agents, les systèmes multi-agents sont caractérisés par un comportement global émergent. Ainsi, à partir d'un ensemble d'agents ou entités pas forcément intelligents, on élabore des systèmes complexes dont l'intelligence émerge d'un comportement global [Gas92]. De plus, un système multi-agents dans sa globalité doit être modulaire et extensible. La modularité correspond au fait que le système multi-agents est composé de plusieurs sous-systèmes mis en relation et ayant chacun leur

[4] *Approche descendante* - méthode d'élaboration de structures nanométriques, qui consiste à réduire progressivement la taille de matériaux existants, en les découpant ou les sculptant, jusqu'à ce qu'ils possèdent les dimensions et les caractéristiques voulues. [GRD]

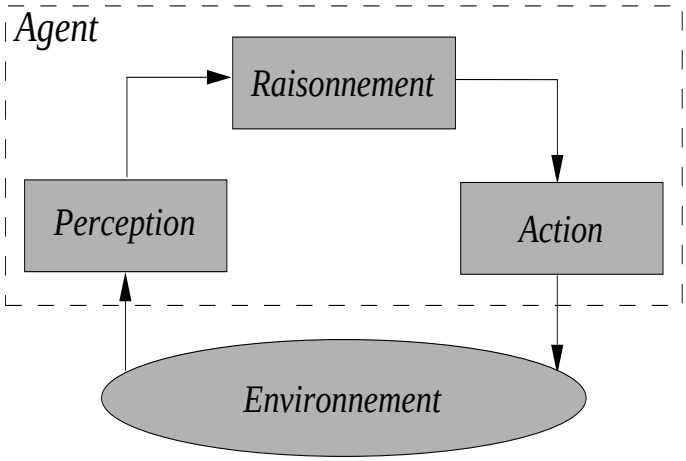


FIG. 1.5 – Le schéma synoptique d'un « agent » d'un système *MAS*.

propre mode de fonctionnement. L'extensibilité se traduit par le fait que le système multi-agents supporte l'ajout et le retrait dynamique d'éléments.

Les systèmes multi-agents ont principalement des applications dans le domaine de l'intelligence artificielle. Dans ce domaine d'applications, ils permettent de réduire la complexité de résolution d'un problème en le divisant en sous-ensembles associés à chaque sous-ensemble un agent intelligent indépendant et coordonnant l'activité des agents [Fer95]. On parle ainsi d'intelligence artificielle distribuée. Il existe d'autres exemples d'applications des systèmes multi-agents tels que : les télécommunications, le commerce électronique sur Internet, la robotique, l'optimisation et la gestion des systèmes de transports, etc. Si on considère, les caractéristiques principales des systèmes multi-agents, nous pouvons constater des propriétés propres et similaires au principe d'auto-organisation.

1.4 Nouvelle approche à base de technologie reconfigurable

Une nouvelle approche de mise en œuvre du principe d'auto-organisation est l'utilisation de technologies matérielles reconfigurables de type *FPGA* par l'exploitation de la reconfiguration dynamique des ces derniers. Par la suite, nous considérons la définition d'une architecture reconfigurable dynamiquement suivante [Bru04] :

« Toute architecture permettant une modification de sa structure de traitement des données à tout moment est une architecture reconfigurable dynamiquement. Ceci quel que soit le degré d'utilisation de cette propriété et quel que soit le temps de reconfiguration. »

Une architecture reconfigurable dynamiquement permet de répondre dans une certaine mesure à l'optimisation des ressources matérielles tout en garantissant une flexibilité et une puissance de calcul suffisante pour des applications temps réels. Comme exemple, on peut citer la

mise en œuvre du calcul reconfigurable par partitionnement temporel d'une matrice logique *FPGA* [Tan01]. Ici, l'optimisation des ressources pour une puissance de calcul donnée est obtenue grâce à une meilleure exploitation de la densité fonctionnelle surfacique [Bru04, Liu08]. L'atout principal d'une telle solution technologique est l'apport d'une flexibilité permettant des traitements adaptés. Cependant, il ne demeure pas moins déterministe dans la mesure ou la conception basée sur cette technologie doit planifier les calculs successifs à mettre en œuvre. Or, dans le cas d'un traitement de flot de données non déterministe, c'est-à-dire le cas où il existe une évolution non précise et non connue des traitements et par conséquent ne pouvant être planifiés antérieurement, une simple solution architecturale reconfigurable ne permettrait pas une auto-organisation répondant à ces traitements évolutifs. Dans ce cas, la conception actuelle sur technologie reconfigurable ne permet pas l'adaptabilité et la flexibilité nécessaire à l'implantation d'un traitement non déterministe. Néanmoins, par rapport aux systèmes existants, dont la mise en œuvre de la propriété d'adaptabilité repose principalement sur des solutions majoritairement logicielles (processeurs), de telles technologies permettent le maintien d'une flexibilité associée à une forte puissance de calcul. En effet, les solutions actuelles présentent des performances insuffisantes dans certains domaines d'application (par exemple applications temps réel). L'utilisation de la reconfiguration dynamique des circuits *FPGA* permet une adaptabilité structurelle de calcul tout en répondant à des performances élevées. Ces reconfigurations permettent ainsi d'assurer la mise en œuvre d'une auto-organisation et d'une émergence.

1.5 Conclusion

Après avoir recensé les différentes définitions liées aux principes d'auto-organisation et d'émergence, nous avons proposé dans ce chapitre une définition synthétique et globale des systèmes auto-organisés et/ou émergents. Ainsi, on définit l'auto-organisation d'un système à la fois :

- Par les principes et les mécanismes qui résident dans sa restructuration et sa gestion sans aucun contrôle externe,
- Par son adaptation dynamique aux changements de l'environnement dans lequel il évolue,
- Par sa capacité à répondre aux changements imprévus par l'intéractivité des éléments qui le consituent,
- Par sa complexité et sa modularité.

Parmi les solutions de mise en œuvre de tels systèmes, nous pouvons citer les *systèmes de calcul inspirés organique* (*organic computing*), les *systèmes de calcul autonome* (*autonomic computing*) et les *systèmes multi-agents* (*multi-agent systems*). Ces systèmes sont caractérisés par une complexité et une taille très élevées. D'une manière générale, ils sont spécifiés par la composition et la structuration de plusieurs entités d'une complexité plus ou moins éle-

vée. Cependant, ces systèmes présentent des performances insuffisantes dans certains domaines d'application nécessitant un fonctionnement temps réel. En effet, de façon plus concrète les systèmes existants reposent principalement sur l'association de processeurs modélisant des entités les constituant et permettant une forte flexibilité.

Nous proposons une autre approche de mise en œuvre de l'auto-organisation de système à base de technologie *FPGA*. Plus précisement, nous proposons une mise en œuvre de la propriété d'auto-organisation et / ou émergence d'un système à travers la mise en place de mécanismes reposant sur l'utilisation de la reconfiguration dynamique des circuits *FPGA*. L'objectif visé par cette approche est d'apporter une adaptabilité structurelle de calcul tout en répondant à des exigences de performances de traitement élevées. Dans ce cadre, des entités de calcul matérielles ou logicielles (*IP*, *Hard* ou *Soft* processeurs, etc.) au sein d'une structure reconfigurable constituent les entités au niveau *micro* d'un système auto-organisé. Des phases de « reconfiguration exécution » partielles ou globales de cette structure permettent de mettre en œuvre les propriétés d'adaptabilité, de flexibilité et de réorganisation de tâches de calcul en vue d'une émergence au niveau *macro* d'un système. Ces phases de reconfiguration permettent d'assurer la mise en œuvre d'une auto-organisation et d'une émergence. C'est l'objet des travaux présentés dans ce manuscrit de thèse dont les chapitres suivants présentent et détaillent les concepts, les mécanismes et les besoins architecturaux et structurels de l'approche proposée.

Chapitre 2

Concepts de l'auto-organisation matérielle à base de technologie reconfigurable

2.1 Introduction

Depuis plusieurs années, la technologie reconfigurable dynamiquement de type *FPGA* est considérée comme une nouvelle alternative pour la réalisation d'architectures de traitement numérique de l'information. En effet, elle permet de mettre en œuvre un nouveau paradigme correspondant au concept de « calcul reconfigurable ». Bien que ce concept soit déjà relativement ancien [EV62], la démocratisation de la technologie *FPGA* a conduit à un accroissement d'intérêt pour la mise en œuvre d'une stratégie de calcul reconfigurable. En pratique, cela s'est traduit initialement par l'association de ces technologies à un processeur pour évoluer de nos jours vers un couplage plus étroit se traduisant par l'intégration d'une matrice *FPGA* et de processeurs sur la même puce [BRM+99, Tan01]. L'objectif est de profiter simultanément des performances d'une technologie permettant une logique câblée tout en gardant une forte flexibilité à travers la disposition d'une structure microprogrammée [CH02].

Avec l'évolution actuelle des systèmes complets intégrant des modules de nature différente sur une même puce (*SoC - System on a chip*), les technologies reconfigurables deviennent primordiales. En effet, ayant à l'origine une flexibilité matérielle très limitée (structure de type ASIC)[5]) , les *SoC* ont besoin à la fois de gérer leur fonctionnement de manière autonome et de s'adapter à des modifications de l'environnement dans lequel ils évoluent. C'est dans ce contexte que la technologie *FPGA* permet une versatilité à travers des phases de reconfiguration dynamique (globales ou partielles) tout en conservant un minimum de performances. Cette versatilité s'exprime à travers des changements spatio / temporels de la structure du circuit per-

[5]ASIC - *Application-specific integrated circuit* - circuit intégré à application spécifique. Circuit intégré conçu et réalisé pour une ou plusieurs applications particulières [GRD]

mettant ainsi une grande souplesse matérielle. La figure 2.1 illustre des phases d'« exécution-reconfiguration » d'une structure reconfigurable dynamiquement.

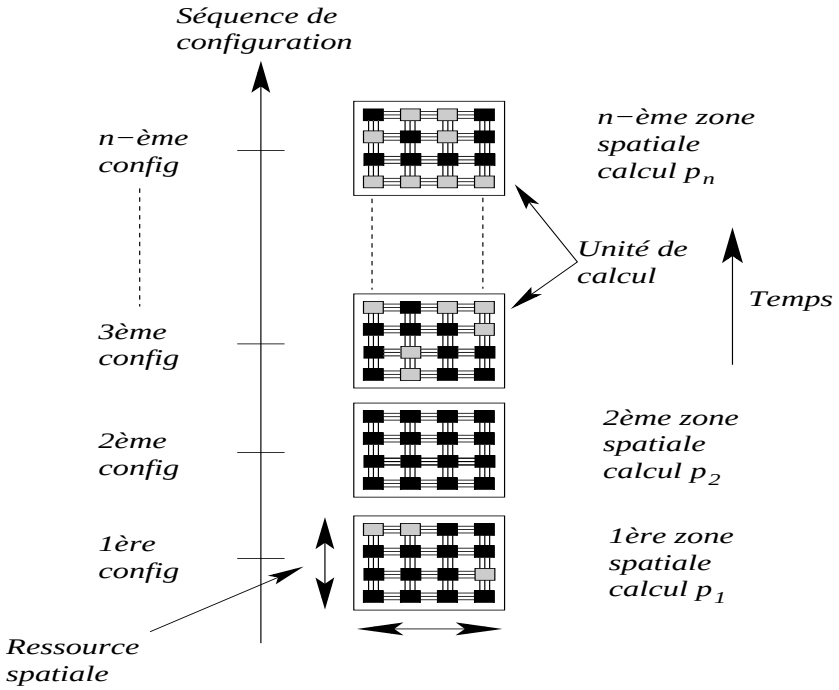


FIG. 2.1 – Illustration de l'évolution spatio-temporelle d'une structure matérielle reconfigurable dynamiquement de type *FPGA*.

Les propriétés de reconfiguration dynamique de circuit *FPGA* sont appropriées à la conception de systèmes auto-organisés. En effet, l'auto-reconfiguration d'une structure matérielle permet la mise en œuvre de l'auto-restructuration matérielle. On parlera alors de systèmes auto-organisés reconfigurables. L'objet de ce chapitre est l'étude d'une conception d'un système auto-organisé reconfigurable. Pour cela, une transposition des propriétés de systèmes auto-organisés vers les systèmes reconfigurables dynamiquement à base de technologie *FPGA* doit être menée.

La suite de ce chapitre est organisée de la manière suivante. Dans une première partie nous rappelons les principales caractéristiques de la reconfiguration dynamique à base de technologie *FPGA*. Ensuite, nous proposons une définition d'un système auto-organisé reconfigurable. Puis, une transposition des caractéristiques d'un système auto-organisé (présentées dans le chapitre précédent) est présentée dans le cadre de la mise en œuvre d'un système reconfigurable. Enfin, nous concluons ce chapitre en recensant les principaux besoins nécessaires à la conception de tels systèmes auto-organisés reconfigurables.

2.2 Systèmes reconfigurables à base de *FPGA*

2.2.1 Introduction

Le choix des technologies reconfigurables est d'apporter une solution intermédiaire entre les circuits spécifiques (*ASIC*) possédant des performances remarquables (dues à une intégration toujours maximale par rapport au progrès technologique du moment), un degré de flexibilité faible pour un temps de développement élevé (*time-to-market*[6]) et les processeurs possédant une forte flexibilité et une puissance de calcul limitée pour un temps de développement court. Ces technologies présentent une solution intermédiaire permettant un compromis entre puissance de calcul (supérieur aux processeurs) et degré de flexibilité (supérieur à celui des *ASIC*). La figure 2.2 schématise le rapport performance / flexibilité pour les principales technologies de calcul disponible actuellement. Outre leur bon compromis puissance / flexibilité, les technologies programmables permettent des phases de développement et de prototypage rapides par rapport à la conception d'un circuit spécifique *ASIC*. Les temps sont comparables à ceux du développement microprogrammé.

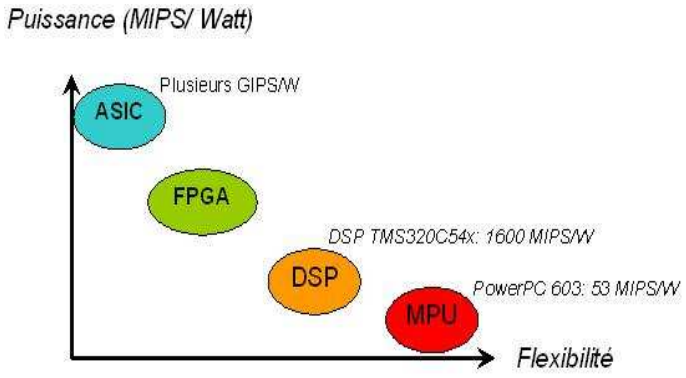


FIG. 2.2 – Rapport performances / flexibilité pour les principales technologies

Un système reconfigurable est composé d'éléments de calcul dont la fonctionnalité est programmable. Ces unités de calcul sont connectées entre elles à travers des canaux de routage configurables. Ainsi, une fonction numérique quelconque peut être implémentée et exécutée dans un système reconfigurable par une configuration des unités de calcul et de leurs connexions. Les systèmes reconfigurables les plus souvent utilisés sont les systèmes à base de *FPGA* (*Field Programmable Gate Array*[7]). Les systèmes reconfigurables à base de *FPGA* sont

[6] *Time-to-market* - temps de mise sur le marché. Temps qui est nécessaire à la fabrication d'un produit, de sa conception à sa mise sur le marché [GRD].

[7] *Field Programmable Gate Array* - réseau prédiffusé programmable par l'utilisateur. Réseau prédiffusé de portes logiques qui est destiné à être programmé sur place, avant d'être utilisé pour une fonction particulière [GRD].

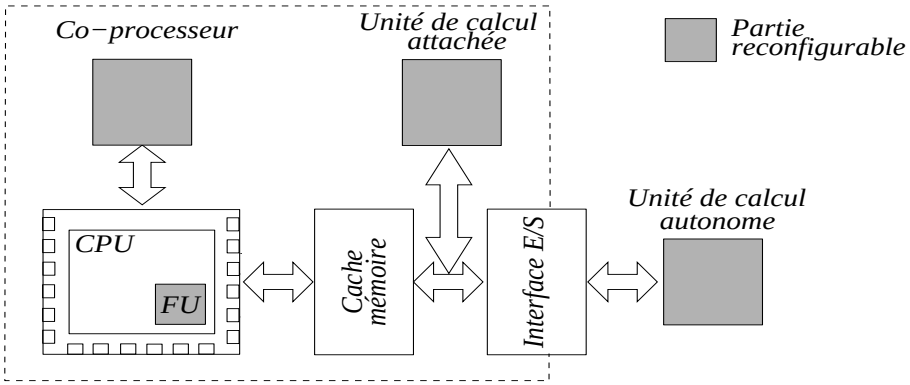


FIG. 2.3 – Système reconfigurable à différents niveaux de couplage

principalement réalisés par des combinaisons de couplage entre des matrices de *FPGA* et des processeurs. On distingue ces systèmes, d'une part par leur taux ou leur nombre de reconfiguration possible au cours d'une application ; d'autre part, par le degré de couplage entre la partie reconfigurable chargée d'accélérer ou d'exécuter des structures de calcul de type « chemin de données »(*data-path*) d'une application. Tandis qu'une unité de calcul généralement de type processeur est chargé d'exécuter principalement les parties contrôles de l'application [CH02, RL97, WC96]. La figure 2.3 illustre un tel système reconfigurable.

2.2.2 Les circuits *FPGA*

Un circuit *FPGA* est généralement composé de réseaux d'éléments de calcul souvent appelés « blocs logiques »(*logic blocks*) dont la fonctionnalité est déterminée par la programmation de bits de configuration (figure 2.4-a). Ces unités de calcul sont connectées à travers des canaux de routage configurables(figure 2.4-b).

La structure classique d'un *FPGA* à mémoire statique (*SRAM*) est présentée sur la figure 2.5 [Xil00]. La plupart des architectures *FPGA* sont organisées de telle sorte que la communication entre les blocs logiques le long des lignes et des colonnes de la logique de routage soit rapide et efficace. Une organisation en une matrice de blocs logiques distincts, reliés par des ressources d'interconnexions, constitue la solution la plus générale. Un bloc ou une cellule logique abrite une ou plusieurs fonctions programmables réalisées par des tables de transcodage (*LUT - Look Up Table*) et des éléments de mémorisation (registres). Suivant les technologies proposées, la granularité de ces cellules logiques varie entre des *LUT* à 2, à 4 ou à 5 entrées [BR96] et des blocs de calcul intégrés plus ou moins complexes. En fonction de la taille et de la complexité des blocs logiques leur structure est classée en fonction de leur granularité fine, moyenne et grosse (*FPGA*, *Systolic Ring* [STG+01], etc.). Plusieurs familles de circuits *FPGA* sont commercialisées (Xilinx, Altera, Atmel, etc.) [Xil00, Atm98, Alt98]. Elles se distinguent

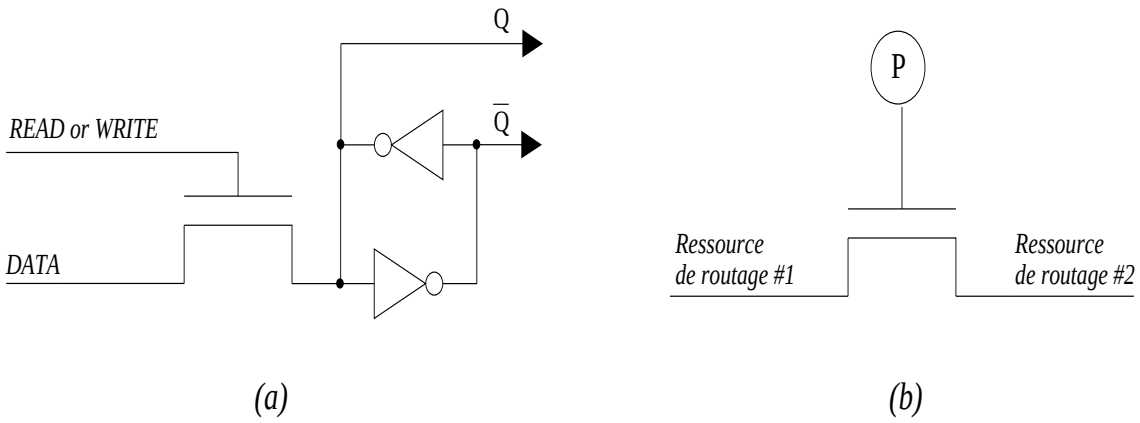


FIG. 2.4 – a) Bit de programmation d'un *FPGA* à mémoire statique *SRAM* [Xil94] b) Connexion de routage programmable.

les unes des autres par leur mode de fonctionnement (fonctionnalités des blocs logiques), leur architecture interne, leurs ressources de routage, etc.

2.2.3 Modes de reconfiguration

En plus du nombre de cellules disponibles (taille du circuit) et la fréquence maximale à laquelle peut fonctionner un circuit, les *FPGA* peuvent être caractérisés par le mode et la vitesse de programmation ou (re)configuration. Cette dernière est liée aux temps d'accès de la mémoire *SRAM* de configuration du circuit. Il s'agit du temps mis pour reconfigurer la fonctionnalité des cellules [Xil00, Atm98, Alt98]. Ces durées de reconfiguration correspondent alors au temps mis pour charger la mémoire *SRAM* définissant la fonctionnalité du circuit. Les temps de programmation dépendent principalement des technologies *FPGA* disponibles et peuvent varier de quelques millisecondes à quelques secondes [Liu08]. Une vitesse de reconfiguration élevée est fondamentale pour réaliser des applications reconfigurables dynamiquement. En fonction du nombre de reconfigurations (principalement lié au degré de couplage existant entre les élé-

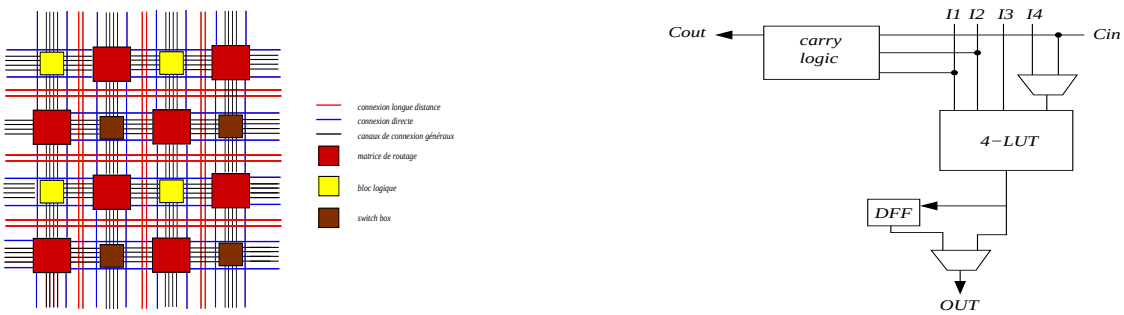


FIG. 2.5 – a) Architecture globale d'un circuit *FPGA*, b) d'un bloc logique.

ments constituant le système reconfigurable), nous pouvons distinguer les systèmes statiques, dynamiques et ultra-dynamiques [Tan01].

La grande majorité des architectures à base de *FPGA* utilise le mode de reconfiguration statique. Les *FPGA* sont configurés une seule fois pour exécuter un traitement unique sur une même série de données. Ces architectures restent actives tout au long des opérations (voir figure 2.6). Parmi les systèmes fonctionnant en reconfiguration statique nous trouvons principalement les accélérateurs reconfigurables dont le rôle est de réaliser des tâches gourmandes en temps de calcul nécessitant une exécution rapide et ne pouvant pas être effectuées sur les systèmes microprogrammés [GHK+90, Ber93, VBR+96, ABD92].

Dans le cas des systèmes reconfigurables dynamiquement, pour une même application le nombre de reconfiguration est au moins égal de deux au cours du traitement [SBB+06]. La mise en œuvre de cette stratégie de fonctionnement se traduit par la configuration du système reconfigurable évoluant tout au long de l'exécution des algorithmes et autant de fois que nécessaire. L'ensemble se comporte comme un processeur dont les ressources matérielles évoluent dans le temps à une fréquence liée à celle des données. La figure 2.7 illustre le procédé. Il existe plusieurs types de reconfiguration dynamique : reconfiguration totale ou partielle (locale et globale). Dans le procédé de la reconfiguration dynamique totale, la globalité du contenu de la mémoire de reconfiguration du *FPGA* est remplacée par de nouvelles données de configuration. La figure 2.8 illustre ce mode de reconfiguration sur un exemple de traitement avec quatre opérateurs A, B, C et D. La reconfiguration globale consiste alors à implanter les opérateurs (selon un ordonnancement opérateurs) en reconfigurant entièrement le système à chaque nouvelle étape. La reconfiguration globale est le mode de reconfiguration dynamique le plus simple à réaliser.

La reconfiguration dynamique partielle désigne la modification d'une partie des données de configuration du circuit reconfigurable [HB06]. En d'autres termes, une partie du circuit est reconfigurée pendant que le reste du circuit exécute des opérations. La figure 2.8 illustre ce mode de reconfiguration sur un exemple de traitement. Ici seul l'opérateur D est reconfiguré en opérateurs A et C tandis que l'opérateur initial B est toujours en fonctionnement. Pen-

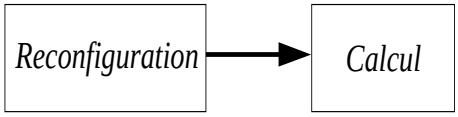


FIG. 2.6 – Reconfiguration statique d'un circuit *FPGA*.

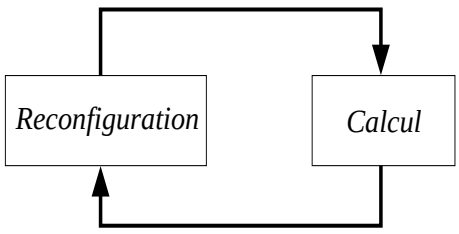


FIG. 2.7 – Mode de reconfiguration dynamique d'un circuit *FPGA*.

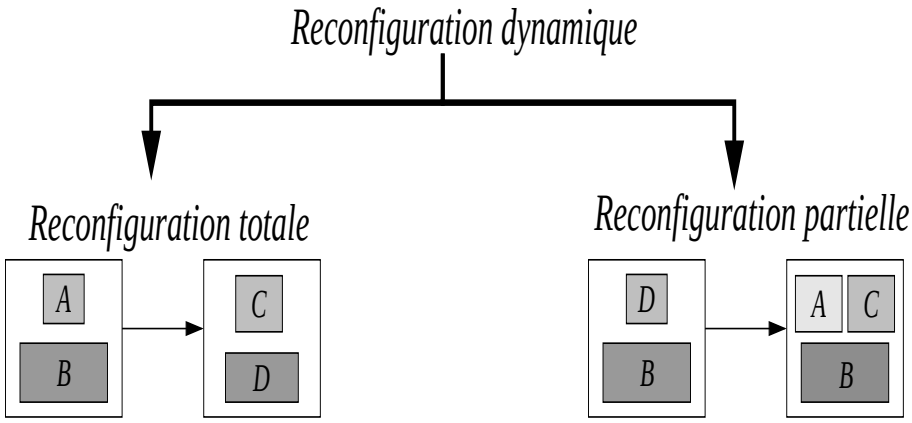


FIG. 2.8 – Mode de reconfiguration dynamique d'un circuit *FPGA*.

dant la reconfiguration, la fonctionnalité du reste du circuit reste donc inchangée. La mise en œuvre du concept de la reconfiguration dynamique partielle nécessite des technologies *FPGA* permettant ces formes de reconfigurations. Parmi les systèmes fonctionnant en reconfiguration dynamique, nous trouvons principalement les coprocesseurs reconfigurables ayant pour but d'accélérer des tâches spécifiques pour lesquelles un processeur n'est pas performant et adapté [DPW99, GBTW97, HSKB06].

La reconfiguration ultra-dynamique d'un système reconfigurable à base de *FPGA* présente un type de reconfiguration dynamique prenant en compte des temps de reconfiguration du circuit. Ce type de reconfiguration dynamique s'utilise notamment dans les applications où la notion du temps réel est fondamentale. Afin d'utiliser le procédé de la reconfiguration ultra-dynamique, les temps perdus pendant les phases de reconfiguration doivent être réduits. Plusieurs solutions ont été proposées mettant en œuvre le masquage des temps de reconfiguration [Ama06, ABT04]. Parmi les systèmes qui aspirent à évoluer vers des systèmes ultra-dynamiques, on trouve les unités reconfigurables. Dans ce cas, la partie reconfigurable est directement placée en tant qu'une unité particulière sur le chemin de données de l'unité de calcul et est généralement intégrée sur la même puce [DG06].

2.3 Systèmes reconfigurables auto-organisés

2.3.1 Introduction

Les technologies reconfigurables de type *FPGA* ont permis de répondre dans une certaine mesure à l'optimisation des ressources matérielles tout en garantissant une puissance de calcul suffisante pour des applications temps réel. L'atout principal d'une telle solution technologique est l'apport d'une flexibilité permettant des traitements adaptés. Cependant, il ne demeure pas

moins déterministe dans la mesure où la conception basée sur cette technologie doit planifier les calculs successifs à mettre en œuvre. Or, dans le cas d'un traitement de flot de données non déterministe où une évolution non précise et non connue des traitements existe, une simple solution architecturale reconfigurable ne permettrait pas une réponse adaptée à ces traitements évolutifs. Néanmoins, la reconfigurabilité dynamique des technologies *FPGA* est sans doute l'aspect le plus prospectif des travaux autour des systèmes adaptatifs et auto organisés. Afin que le concept d'auto-organisation soit utilisé et appliqué dans les systèmes reconfigurables dynamiquement, certains adaptations de ce concept prenant en compte les propriétés des technologies reconfigurables doivent être réalisées.

2.3.2 Définition d'un système auto-organisé reconfigurable

Un système auto-organisé reconfigurable, dont les caractéristiques principales devront être définies et formulées, doit être capable à la fois de :

- Répondre par l'adaptation fonctionnelle à tout changement de paramètres environnementaux d'importance pour lui-même,
- Manifester de la robustesse et de la flexibilité à toute défaillance fonctionnelle de ses constituants,
- Gérer ses ressources d'une manière autonome et indépendante,
- Organiser sa structure interne sans aucune influence de l'extérieur et dans le but d'un fonctionnement optimal par rapport à des critères prédéfinis (comme par exemple *adéquation / algorithme architecture*),
- Anticiper les traitements à venir par une organisation adaptée de ses ressources,
- Faire preuve d'une structure dotée d'une « intelligence distribuée » permettant la prise de décisions d'importance et faire face à des événements susceptibles de survenir,
- Utiliser la technologie reconfigurable pour permettre la mise en œuvre des points cités ci-dessus.

Un système auto-organisé reconfigurable répondant aux critères généraux présentés ci-dessus est illustré sur la figure 2.9. De manière plus détaillée, un tel système :

- Adapte son fonctionnement à des perturbations survenues,
- Distribue, en cas de défaillance d'une de ses entités, les fonctions ou tâches vers les autres entités,
- Remplace ses parties ou entités défaillantes par le biais d'une phase auto-reconfiguration dynamique,
- Détecte le changement de type de données d'entrée à traiter et adapte son fonctionnement global à travers les fonctions possibles de ses entités à ces nouveaux traitements,
- Organise d'une manière autonome ses entités de telle sorte qu'elles soient « prêtes » à anticiper des traitements à venir,

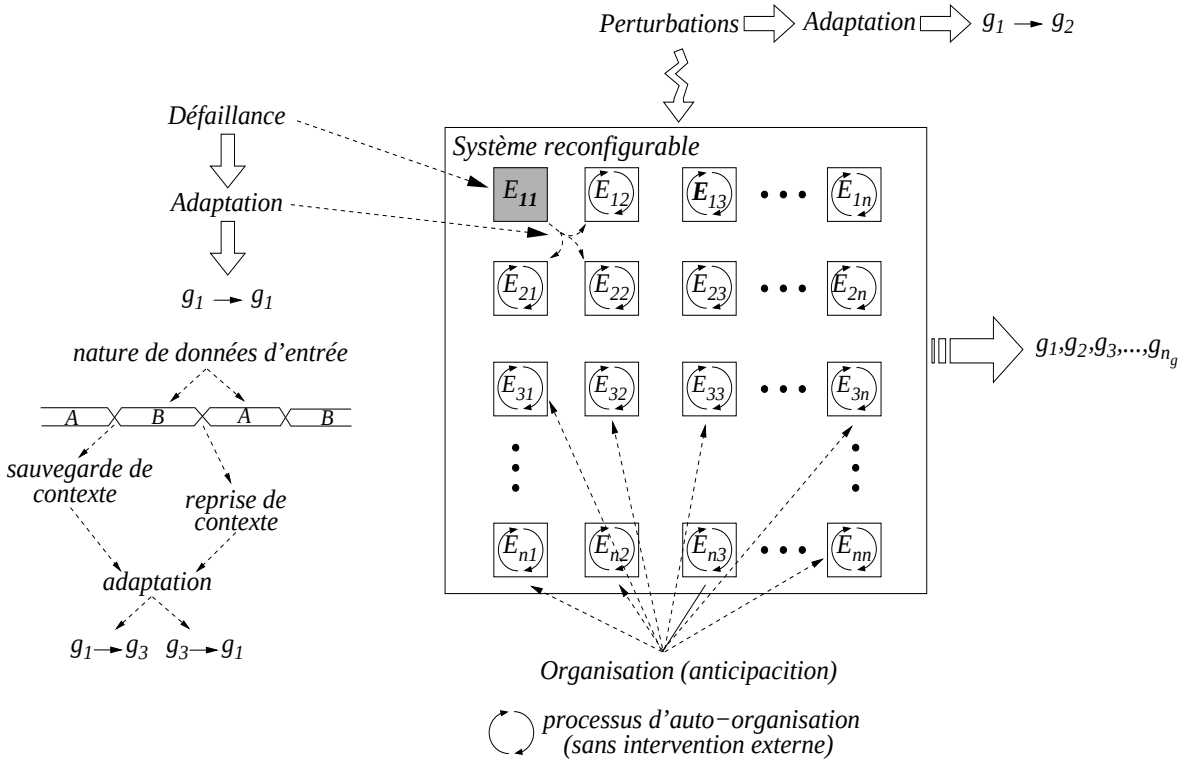


FIG. 2.9 – Illustration des critères d'auto-organisation d'un système reconfigurable.

– Agit de sa propre initiative pour optimiser selon des critères prédéfinis l'utilisation de ses ressources matérielles, etc.

En définissant d'une part, la reconfigurabilité d'un système comme une architecture matérielle capable de changer de *forme / fonction* lors de divers traitements au cours du temps ; d'autre part le terme *dynamique* attribué à la reconfiguration comme la caractéristique d'un système capable à tout moment de changer d'état, on aboutit à la définition d'un système reconfigurable dynamiquement [Bru04]. Par ailleurs, en pointant particulièrement le fait que la reconfiguration dynamique d'un système présente l'aspect le plus prometteur pour la mise en œuvre du principe d'auto-organisation, les relations explicites entre ce principe et la technologie reconfigurable doivent être établies.

A partir des définitions d'un système auto-organisé et d'un système reconfigurable dynamiquement, nous proposons la définition synthétique suivante d'un système auto-organisé reconfigurable dynamiquement :

On définit un système auto-organisé reconfigurable comme tout système ou architecture :

– *permettant sa restructuration et sa gestion fonctionnelle sans contrôle externe,*
– *possédant la capacité de s'adapter à tout changement imprévu de l'environnement où il évolue grâce à l'intéractivité de ses éléments constitutifs et par des moyens de reconfigurabilité de sa structure.*

2.3.3 Modélisation et formalisation d'un système auto-organisé reconfigurable

Soit un système auto-organisé reconfigurable (SAR) composé de N_e entités E_{ij} ($1 \leq i \leq n$, $1 \leq j \leq m$, $n, m \in \mathbb{N}$, $N_e = n \cdot m$), dont une illustration est donnée figure 2.10. Chaque

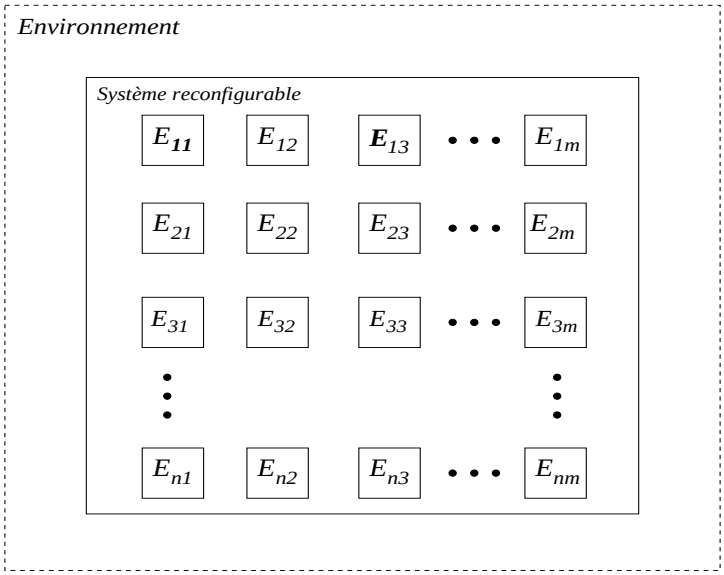
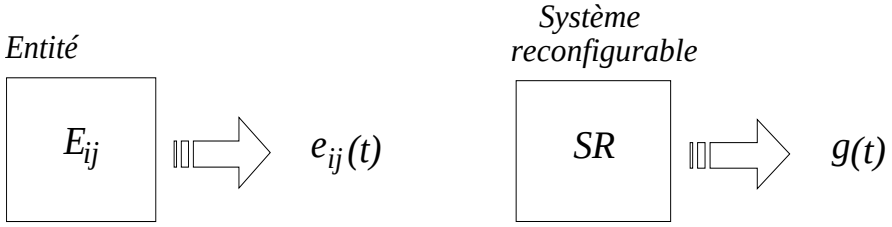


FIG. 2.10 – Schéma synoptique global d'un système reconfigurable modulaire.

entité E_{ij} réalise une fonction $e_{ij}(t)$ parmi l'ensemble S_{ij} des fonctions possibles tel que à un instant donné t, nous avons :

$$e_{ij}(t) \in S_{ij},\ 1 \leq i \leq n, 1 \leq j \leq m,\ n, m \in \mathbb{N}. \tag{2.1}$$

De même, la fonction principale effectuée par le système à un instant donné t, présentée par la fonction globale $g(t)$ est telle que :



(a) Entité E_{ij} - e_{ij} (b) Système reconfigurable - g_{ij}

FIG. 2.11 – Modélisation d'une *Entité / fonction* associée et d'un système *auto-organisé / fonction principale* associé.

$g(t) \in S$, où S représente l'ensemble des fonctions pouvant être implémentées par le système. La figure 2.11 illustre cette modélisation.

A partir de cette modélisation, une formalisation fonctionnelle de l'ensemble des entités du système, peut être spécifiée de la façon suivante :

$$\begin{gathered}
e_{11}(t) = e_{11_{i_{11}}} \in S_{11}\,(0 \leq i_{11} \leq n_{11}),\ S_{11} = \left\{\emptyset, e_{11_1}, e_{11_2}, ..., e_{11_{n_{11}}}\right\}, \\
e_{12}(t) = e_{12_{i_{12}}} \in S_{12}\,(0 \leq i_{12} \leq n_{12}),\ S_{12} = \left\{\emptyset, e_{12_1}, e_{12_2}, ..., e_{12_{n_{12}}}\right\}, \\
\vdots \\
e_{ij}(t) = e_{ij_{i_{ij}}} \in S_{ij}\,(0 \leq i_{ij} \leq n_{ij}),\ S_{ij} = \left\{\emptyset, e_{ij_1}, e_{ij_2}, ..., e_{ij_{n_{ij}}}\right\}, \\
\vdots \\
e_{nm}(t) = e_{nm_{i_{nm}}} \in S_{nm}\,(0 \leq i_{nm} \leq n_{nm}),\ S_{nm} = \left\{\emptyset, e_{nm_1}, e_{nm_2}, ..., e_{nm_{n_{nm}}}\right\}, \\
\text{avec } 1 \leq i \leq n,\ 1 \leq j \leq m;\ n, m \in \mathbb{N}.
\end{gathered} \tag{2.2}$$

où e_{ij_k}, $\emptyset$ et n_{ij} représentent respectivement une fonction spécifique (exécutable), l'inactivité fonctionnelle (contribution nulle à la fonction principale du système) et le nombre maximum de fonctions pouvant être délivrées par une entité E_{ij}. De même, pour la fonction globale du système, nous pouvons la formuler de la manière suivante :

$$g(t) = g_i \in S\,(0 \leq i \leq n_g),\ S = \left\{\emptyset, g_1, g_2, ..., g_{n_g}\right\}. \tag{2.3}$$

où g_i correspond à une fonction principale spécifique et exécutable par le système. L'indice n_g représentent le nombre maximum de fonctions pouvant être délivrées par par le système SAR. On suppose que toutes les fonctions pouvant être réalisées par une entité E_{ij} ($1 \leq i \leq n,\ 1 \leq j \leq m$) sont décomposables en opérateurs ou fonctions de base correspondant à des fonctions non simplifiables ou non-décomposables en sous-fonctions. On considère également d'une part, l'ensemble des opérateurs ou fonctions de base B_{ij} ($1 \leq i \leq n,\ 1 \leq j \leq m$) utilisé pour former les fonctions des entités E_{ij}. D'autre part, le multi-ensemble $B_{ij}^{m_{ij}}$ où m_{ij} est le nombre de répétitions ou la redondance des opérateurs de base de l'ensemble B_{ij} tel que :

$$\begin{gathered}
B_{ij} = \left\{(b_{ij})_1, (b_{ij})_2, \ldots, (b_{ij})_k, \ldots, (b_{ij})_{n_{b_{ij}}}\right\} \text{ et } B_{ij}^{m_{ij}} = \{B_{ij}, m_{ij}\}, \\
m_{ij}(b_{ij_1}) = (nb_{ij})_1,\ m_{ij}(b_{ij_2}) = (nb_{ij})_2,\ \ldots,\ m_{ij}(b_{ij_{n_{b_{ij}}}}) = (nb_{ij})_{n_{b_{ij}}},
\end{gathered} \tag{2.4}$$

$n_{b_{ij}}$ et $(nb_{ij})_{n_{b_{ij}}}$ désignent respectivement le nombre d'opérateurs de base distincts de l'ensemble B_{ij} utilisés par E_{ij} et le nombre de répétitions de l'opérateur $(b_{ij})_{n_{b_{ij}}}$ dans les fonctions de E_{ij} ($1 \leq i \leq n,\ 1 \leq j \leq m$).

Si on considère un ensemble S_g comme le produit cartésien des ensembles S_{11}, S_{12}, ..., S_{nm}, les expressions 2.2 et 2.3 peuvent s'exprimer de la manière suivante :

$$\begin{gathered}
e_{11}(t) : B_{11}^{m_{11}} \rightarrow S_{11},\ S_{11} = \left\{\emptyset, e_{11_1}, e_{11_2}, ..., e_{11_{n_{11}}}\right\}, \\
e_{12}(t) : B_{12}^{m_{12}} \rightarrow S_{12},\ S_{12} = \left\{\emptyset, e_{12_1}, e_{12_2}, ..., e_{12_{n_{12}}}\right\}, \\
\vdots \\
e_{nm}(t) : B_{nm}^{m_{nm}} \rightarrow S_{nm},\ S_{nm} = \left\{\emptyset, e_{nm_1}, e_{nm_2}, ..., e_{nm_{n_{nm}}}\right\},
\end{gathered} \tag{2.5}$$

La formulation de la fonction principale du système correspond alors à l'expression suivante :

$$g(t) : S_g \rightarrow S,\ S_g = S_{11} \times S_{12} \times \ldots \times S_{nm} \text{ et } S = \left\{\emptyset, g_1, g_2, ..., g_{n_g}\right\}. \tag{2.6}$$

En résumé, chaque fonction d'une entité E_{ij} correspond un couple ou une combinaison d'opérateurs de base décrits par l'expression $\sum b_{ij}$. De même, pour chaque fonction principale du système, il existe au moins un couple ou une combinaison de fonctions d'entité décrits par l'expression $\sum e_{ij}$ et constituant le système. Si une entité d'un système ou le système lui-même est inactif, cela est formalisé par l'élément ensemble vide $\emptyset$. Inversement, si une entité ou le système est fonctionnel et actif, leurs fonctions sont décrites par des éléments différents de l'ensemble vide. Ceci est résumé par les expressions suivantes :

$$e_{ij}(t) = \begin{cases} \neq \emptyset, & E_{ij} \text{ en marche} \\ = \emptyset, & E_{ij} \text{ en arrêt} \end{cases} \quad g_k(t) = \begin{cases} \neq \emptyset, & g_k \text{ en marche} \\ = \emptyset, & g_k \text{ en arrêt} \end{cases} \tag{2.7}$$
$$1 \leq i \leq n,\ 1 \leq j \leq m,\ 0 \leq k \leq n_g$$

A partir des expressions énoncées (2.2 - 2.6), on établit le comportement globale du système SAR à partir des fonctions d'entités par la relation suivante :

$$g(t) \geq \sum_{i,j=1}^{N_e} e_{ij}(t), 0 < i \leq n,\ 0 < j \leq m; n, m \in \mathbb{N} \tag{2.8}$$

La condition d'inégalité présentée dans l'équation 2.8 exprime, pour les systèmes à forte interactivité, le comportement global du système. Ce comportement ne peut pas se traduire simplement comme la somme de comportements d'éléments ou entités constituant le système. La signification des termes de la relation 2.8, entre $g(t)$ et de la somme $\sum e_{ij}(t)$ signifie un comportement inattendu, imprévu mais réel dans le système et correspondant à la notion d'émergence. D'une manière générale, pour la plupart des systèmes reconfigurables, cette expression peut être exprimée de la façon suivante :

$$g(t) = \sum_{i,j=1}^{N_e} e_{ij}(t), 0 < i < n,\ 0 < j < m; n, m \in \mathbb{N} \tag{2.9}$$

où le degré d'interactivité présent dans ces systèmes est insuffisant pour faire apparaître les propriétés d'émergence.

2.3.4 Propriétés d'un système auto-organisé reconfigurable

Les définitions et les caractéristiques principales du principe d'auto-organisation dans le contexte des systèmes reconfigurables font l'objet de cette sous-section. Le système reconfigurable défini et formalisé dans les sections précédentes servira de base pour cette étude.

2.3.4.1 Adaptabilité des systèmes reconfigurables

Nous avons défini la propriété d'« adaptabilité » d'un système comme étant sa capacité de maintenir sa fonction principale tout en modifiant son comportement aux perturbations provenant de l'environnement extérieur (voir la section 1.2.3.2). Pour un système reconfigurable, on définit sa capacité d'adaptation de fonctionnement aux changements environnementaux de la manière suivante :
« *Si on considère que pour une perturbation $p \in P$, quelle que soit sa nature, un système reconfigurable SAR exerçant une fonction globale $g(t) \in S$ trouve à chaque instant une réponse d'un point de vue fonctionnelle pour y faire face, on dit qu'il est* ***adaptable*** ».
Ceci est défini par l'expression suivante :

$$\forall p \in P, \exists g_i \in S\ (\neq \emptyset)\ (1 \leq i \leq n_g)\ => SAR \text{ adaptable} \tag{2.10}$$

La figure 2.12 illustre la propriété d'adaptabilité d'un système dans le cas d'une perturbation survenue au niveau système. Supposons que les paramètres d'environnement dans lequel se

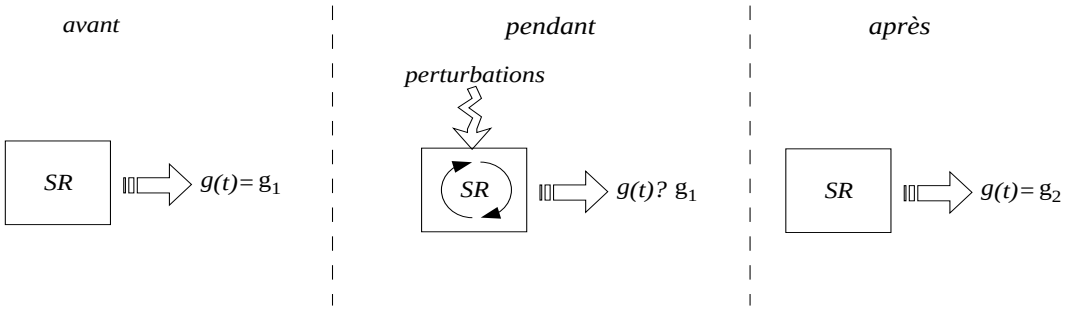


FIG. 2.12 – Illustration de la propriété d'adaptabilité d'un système reconfigurable.

situe un système reconfigurable SAR exerçant une fonction $g(t) = g_1 \in S$ changent à un instant t_1. Dans ce cas, et afin de répondre à ces changements environnementaux, le système SAR rentre dans « un processus d'adaptation ». Ce processus est schématisé par les flèches circulaires dans la figure 2.12. Si la perturbation survenue est connue du système SAR, ce dernier répond à cette perturbation par une adaptation de son comportement. Ainsi, si on considère l'illustration de la figure 2.12, au lieu d'exécuter une fonction globale en cours g_1, le système va évoluer à un moment t_2 ($t_2 > t_1$), après un temps de latence ou d'adaptation, vers une fonction globale g_2 en s'auto-reconfigurant de manière à faire face à ces changements environnementaux. Cette procédure consiste à un changement de fonction globale du système de g_1 vers g_2. Par analogie, ce même processus d'adaptation peur s'effectuer également au niveau d'une entité du système (figure 2.13). Plus précisément, une perturbation ou un changement d'un paramètre environnemental important entraîne le système vers une adaptation fonctionnelle pour y faire face. Le système « renseigne » tous les éléments le constituant à travers des mécanismes d'information

du problème survenu. L'adaptation s'effectue premièrement à un niveau entité et ainsi à partir de toutes ces adaptations locales, résulte d'une nouvelle fonction principale du système adaptée aux exigences environnementales. Dans l'exemple présenté sur la figure 2.13, la fonction globale g_2 découle des adaptations locales au niveau de chaque entité. En résumé, la perturbation a provoqué une demande d'adaptation au niveau de chaque entité dont le résultat est de nouvelles interactions des fonctions des entités aboutissant à la fonction globale g_2.

On caractérise également la notion de degré d'adaptabilité d'un système reconfigurable. Plus précisément, un système ne pouvant s'adapter uniquement à une gamme de perturbations est dit *partiellement adaptable* ou adaptable par rapport à une gamme de perturbations spécifique ou à un spectre fini de perturbations. En général, la plupart des systèmes que l'on rencontre sont partiellement adaptables.

2.3.4.2 Robustesse et flexibilité des systèmes reconfigurables

Les propriétés de robustesse et de flexibilité sont souvent employées dans le cadre d'adaptabilité des systèmes auto-organisés. Nous attribuons à ces deux propriétés les capacités d'un système à maintenir ses performances malgré des changements des conditions de fonctionnement ou la présence d'incertitudes liées à ses paramètres (voir la section 1.2.3.2). De plus, un système robuste et flexible dispose d'une large variété de comportements différents permettant de répondre à des éventuelles perturbations. Dans le contexte d'un système reconfigurable, on

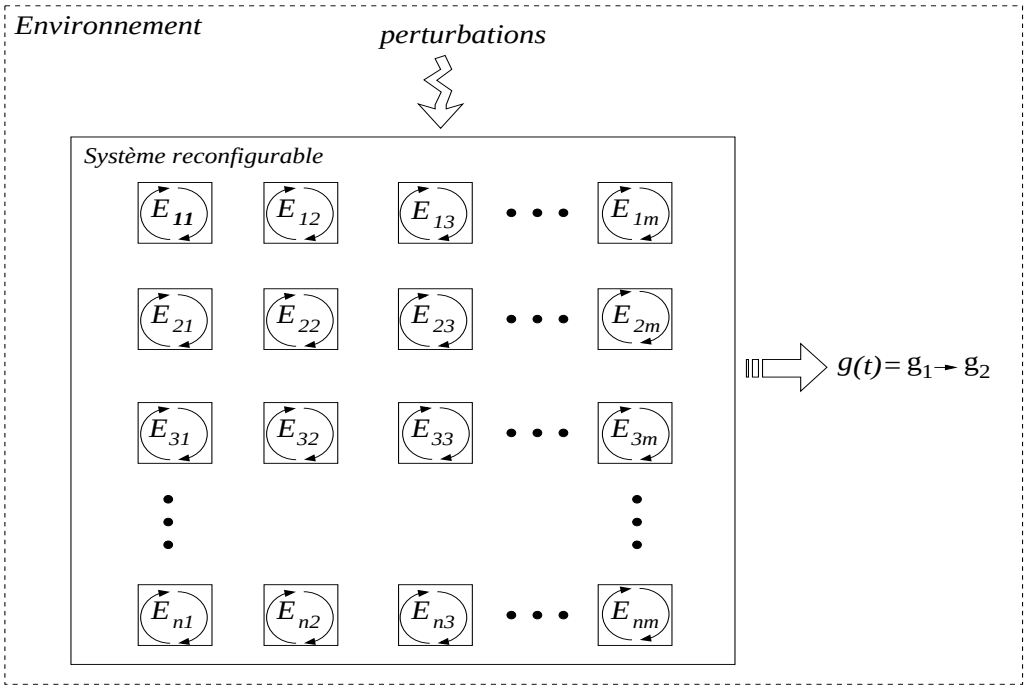


FIG. 2.13 – L'adaptation d'un système reconfigurable aux changements d'environnement.

définit la robustesse de la manière suivante :
« Si on considère que l'ensemble S_{g_i} représente les combinaisons de fonctions d'entités E_{ij} ($1 \leq i \leq n$, $1 \leq j \leq m$, $n, m \in \mathbb{N}$) qui en résultent de la fonction globale g_i, on dit qu'un système reconfigurable est ***robuste et flexible*** *si et seulement si $card(S_{g_i}) > 1$. »*
Autrement dit, pour qu'un système reconfigurable soit considéré robuste à des changements provenant soit de l'extérieur soit de l'intérieur du système, il faut que sa fonction globale puisse être réalisée de façon redondante. C'est à dire, que la mise en œuvre d'une fonction globale par le système doit pouvoir résulter d'au moins deux et plus de combinaisons différentes des fonctions d'entités constituant le système. La robustesse peut être considérée comme un cas particulier d'adaptabilité dont l'objectif final est de conserver la fonction dont l'exécution est en cours. La figure 2.14 illustre la notion de robustesse et de flexibilité dans le cas de défaillance d'une entité du système reconfigurable.

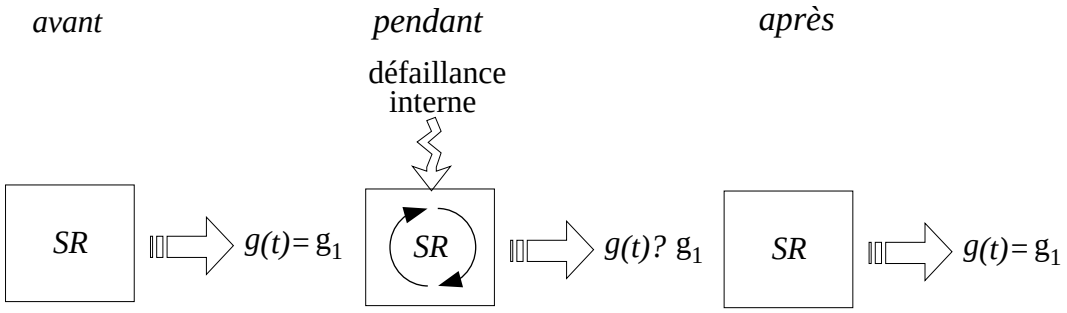


FIG. 2.14 – Robustesse d'un système reconfigurable : une forme d'adaptabilité.

Les concepts les plus souvent utilisés pour rendre un système robuste et flexible sont *la modularité* et *la redondance*. Pour qu'un système soit robuste par rapport à sa capacité d'adaptation, chaque module d'un système exerçant un ensemble de fonction doit être remplaçable d'un point de vue fonctionnel par les autres modules du système tout en maintenant leurs fonctionnalités principales. Cela est assuré par l'intégration de plusieurs fonctions dans les modules ou entités du système.

Pour un système reconfigurable présenté dans la figure 2.10 comme un système modulaire, la robustesse peut s'exprimer analytiquement de la façon suivante :

$$\forall e_{ij} \in S_{ij}, \exists \sum e_{kl} \in \{S_{11} \cup S_{12} \cup \ldots \cup S_{nm}\} \setminus S_{ij} \Rightarrow e_{ij} \subset \sum e_{kl},\ (k, l \neq i, j) \quad (2.11)$$

Chaque fonction d'un module ou d'une entité peut être remplacée soit par une autre fonction soit par la combinaison ou la somme de fonctions d'une autre entité ou de plusieurs entités.

Les retombées directes de la propriété de robustesse sont que les systèmes la possédant sont beaucoup plus complexes et plus difficiles à réaliser. Dans le cas des systèmes reconfigurables où les procédés de la reconfiguration dynamique sont utilisés pour le remplacement éventuel

de leurs parties défaillantes afin d'assurer leur bon fonctionnement, l'intégration de la propriété de robustesse est de moindre complexité. Plus précisément, une fonction d'une entité pourrait être distribuée ou répartie dans les entités avoisinantes, dans le cas où une défaillance apparaît dans l'une d'entre elles, afin d'assurer provisoirement cette fonction jusqu'à ce que l'entité défaillante soit corrigée à travers une étape de remplacement ou de modification par reconfiguration dynamique.

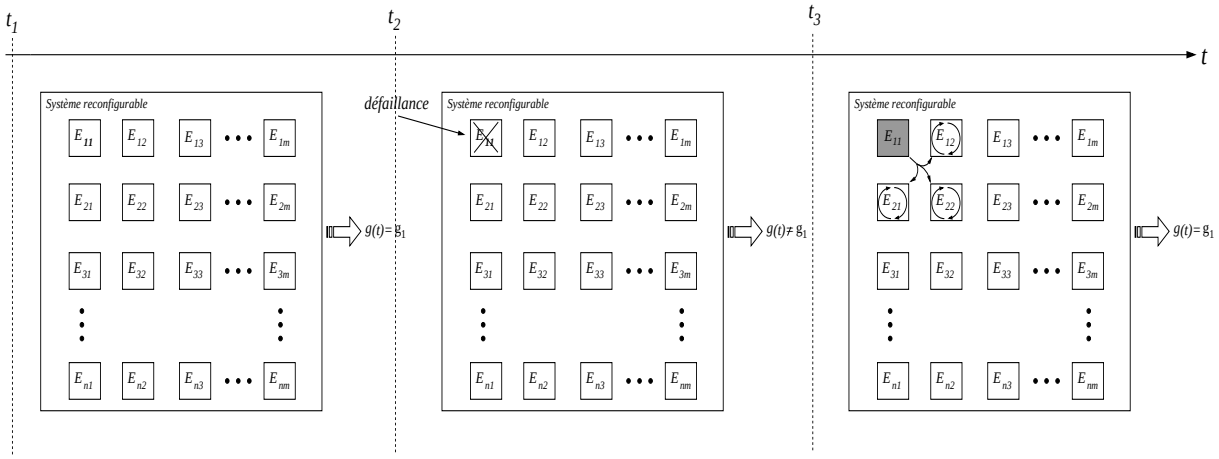


FIG. 2.15 – Illustration de la propriété de robustesse d'un système auto-reconfigurable.

La figure 2.15 illustre un exemple de la propriété de robustesse d'un système reconfigurable dans le cas de défaillance d'une de ses entités. Par exemple, l'entité E_{11}, subit une défaillance au cours du fonctionnement du système. Après avoir détecté le problème, les entités voisines de l'entité défaillante assurent la substitution de la fonction défaillante par la décomposition fonctionnelle et la distribution mutuelle. Cette robustesse peut être à plusieurs niveaux, dans la mesure où une défaillance d'entités peut ne pas permettre d'assurer pleinement la fonctionalité globale du système mais éventuellement un fonctionnement avec une dégradation de performances (mode de fonctionnement dégradé).

2.3.4.3 « Augmentation de l'ordre » des systèmes reconfigurables

Une des caractéristiques importantes d'un système auto-organisé est la propriété d'« augmentation de l'ordre ». Cette propriété décrit un système auto-organisé manifestant une organisation, une structuration et un arrangement de ses entités de manière à permettre l'accomplissement de fonctions globales possibles du système. L'« augmentation de l'ordre » d'un système implique qu'il débute par un état initial (aléatoire ou non) et évolue par changements d'états (au cours de son fonctionnement) vers un état de fonctionnement du système mieux adapté. On dit qu'un état ou un comportement du système est plus conforme à un autre, par rapport à des critères préalablement définis, si le comportement est caractérisé par « plus d'ordre » par rapport

à des états (comportements) précédents.

Dans le domaine des systèmes reconfigurables, on interprète cette propriété comme un phénomène menant le système d'un état initial vers des états mieux adaptés aux traitements en cours ou à réaliser. Par cette évolution, le système cherche de manière plus efficace et plus commode à répondre à tous changements de fonctionnement du système ou à toutes perturbations possibles. Afin d'exprimer un comportement auto-organisé par l'augmentation de l'ordre, un système reconfigurable doit disposer d'une « intelligence ». Cette dernière doit être soit incorporée dans les entités constituant le système, soit émerger des interactions entre ces entités.

Si on considère, un système reconfigurable SAR robuste (la figure 2.10), défini par les expressions 2.2 à 2.9 et correspondant à la mise en place de mécanisme de redondance de chaque fonction globale dans au moins deux entités différentes du système, *on dit que le système présente une auto-organisation (augmentation de l'ordre) si au cours de son fonctionnement et sans aucun stimuli extérieur ou intérieur, il agit de sa propre initiative de sorte que ses modifications structurelles ou ses modes de fonctionnements vont vers un état de fonctionnement « optimal » selon des critères préalablement prédéfinis.* Ces derniers peuvent s'exprimer sous formes de contraintes telles que les contraintes de surface, de consommation, de vitesse, etc. De manière plus formalisée, l'augmentation de l'ordre peut être décrite par les expressions suivantes :

$$\begin{gathered} \forall g_i \in S\,, g_i = \textstyle\sum_l e_{ij},\ , 0 \leq Ordre(\textstyle\sum_l e_{ij}) \leq 1,\, p = \emptyset, \\ \exists \textstyle\sum_k e_{ij} = g_m, k \neq l, Ordre(\textstyle\sum_l e_{ij}) < Ordre(\textstyle\sum_k e_{ij}), \\ 0 \leq i, m \leq n_g,\, 1 \leq i \leq n,\, 1 \leq j \leq m,\, 1 < k, l < N_{11} \cdot N_{12} \cdot \ldots \cdot N_{mn}. \end{gathered} \tag{2.12}$$

où $p = \emptyset$ modélise l'absence de perturbations ou de contrôle externe, tandis que la fonction $Ordre()$ représente la valeur quantitative de l'ordre du système selon des critères prédéfinis sous forme de contraintes. Les termes 0 et 1 expriment respectivement l'absence absolue de l'ordre et l'ordre parfait.

La condition nécessaire pour qu'un système reconfigurable exprime de l'« augmentation de l'ordre » dans son comportement est la robustesse. Cette condition n'est pas réciproque. En effet, il n'y a pas d'auto-organisation dans les systèmes reconfigurables sans robustesse, cependant un système robuste n'est pas systématiquement auto-organisé.

2.3.4.4 Anticipabilité des systèmes reconfigurables

Un système possédant la propriété d'anticipabilité est défini comme un système capable de prévoir les changements et les perturbations en vue d'adapter son comportement global dans un délai raisonnable (voir chapitre 1). L'anticipabilité peut être considérée comme un cas particulier des propriétés d'« augmentation de l'ordre » ou d'adaptabilité. En effet, les réarrangements produits par l'anticipation d'un évènement peuvent être considérés soit comme une

pré-adaptation avant qu'un évènement se produise soit comme une organisation interne préparant le système à des événements susceptibles.

En résumé, l'anticipabilité correspond à la propriété d'« augmentation de l'ordre » d'un point de vue fonctionnel. Plus précisement, l'anticipabilité d'un système peut être exprimée comme un nouvel arrangement interne structurel dans le but d'anticiper de façon appropriée des traitements futurs. Ainsi, la structure interne du système converge vers une forme « commune » entre les traitements en cours et futurs tout en maintenant ses performances aux traitements. Par exemple, considérons la propriété d'anticipabilité d'un système reconfigurable sur un exemple concret dans le domaine d'un système de traitement audio et / ou vidéo. Soit un système reconfigurable composé de N entités (figure 2.16) dont la fonction principale est le traitement de données mixtes de type audio / vidéo. L'arrivée des données d'entrées est aléatoire ou non-déterministe. Par exemple, un flot de données audio peut être interrompu par plusieurs trames de données vidéo dont les traitements associés et à mettre en œuvre ne sont pas prédéfinis. Le système reconfigurable dispose de modules ou entités destinés soit aux traitements de données type audio, vidéo ou aux deux types. Pendant que le système traite une trame d'un type

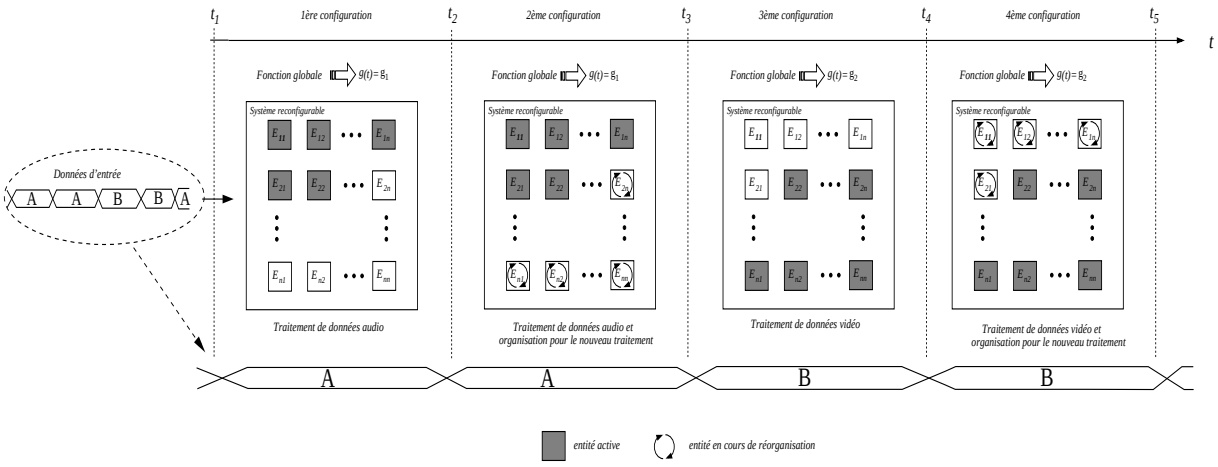


FIG. 2.16 – L'anticipabilité d'un système reconfigurable à l'exemple d'un traitement audio / vidéo.

de données (audio ou vidéo), le reste du système anticipe les traitements futurs correspondant à l'autre type de données. Puisque ces systèmes peuvent exécuter une variété de traitements possibles, la principale difficulté réside dans la détermination des traitements futurs à mettre en œuvre par le système. Cette anticipation nécessite une forme d'apprentissage par le système acquise à travers ses expériences de traitements antérieurs. Autrement dit, afin d'assurer une mise en œuvre efficace et rapide de nouveaux traitements susceptibles à venir, le système doit considérer ses expériences et connaissances acquises au cours des traitements précédents dans le but de choisir les traitements appropriés aux événements futurs.

2.3.4.5 Autonomie des systèmes reconfigurables

Afin qu'un système puisse s'organiser spontanément et sans aucun contrôle externe dans le but de répondre soit à des exigences imposées par des changements environnementaux soit aux processus de son auto-organisation interne, il doit être autonome (voir section 1.2.3.2). L'autonomie est un pilier fondamental de tout système auto-organisé. Un système autonome n'attend pas qu'une intervention extérieure, généralement sous forme de signaux de contrôle, survienne pour réagir à d'éventuelles perturbations ou évolution de fonctionnement. En effet, un système autonome fait preuve d'initiative et d'indépendance. Dans ce cadre, il initie l'échange entre ses éléments constitutifs, le monde extérieur et « décide » de son fonctionnement.

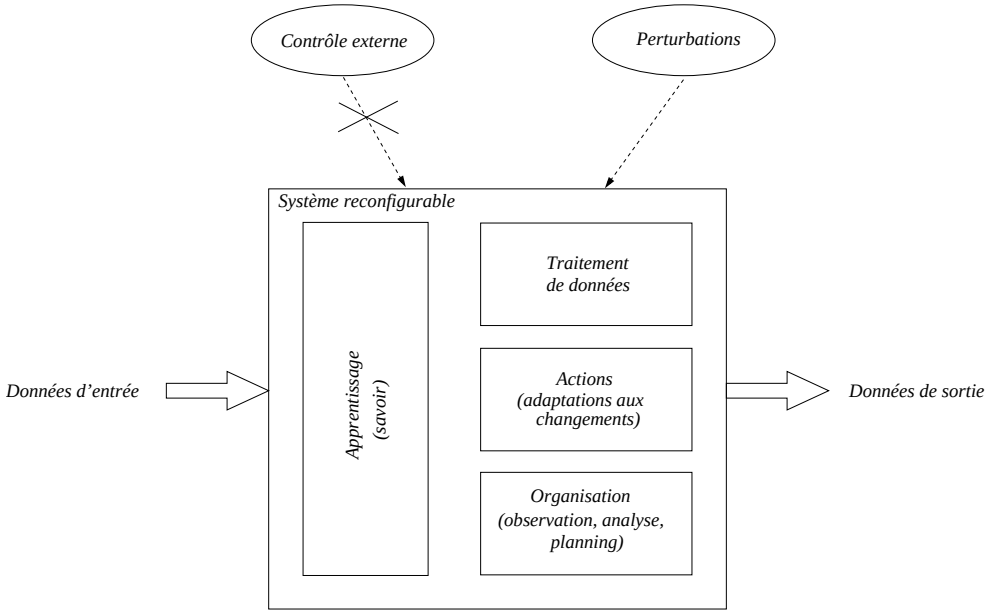


FIG. 2.17 – Illustration de la propriété d'autonomie d'un système reconfigurable.

Pour qu'un système reconfigurable manifeste de l'autonomie, l'accent principal doit être mis sur l'approche architecturale employée pour sa conception. Comme nous l'avons évoqué dans la section précédente, pour extérioriser l'auto-organisation, un système reconfigurable doit avoir de l'« intelligence » enfouie dans ses éléments constitutifs. Cette « intelligence » doit permettre au système à la fois de gérer ses ressources, d'analyser et de contrôler le fonctionnement de ses éléments, d'observer les changements environnementaux, et cela de manière independante (voir la figure 2.17). C'est l'intégration de ces mécanismes au sein du système qui permettrait à ce dernier de faire preuve d'une grande autonomie par rapport à son environnement.

2.3.4.6 Contrôle décentralisé pour systèmes reconfigurables

La propriété de « contrôle décentralisé » est une caractéristique associée à l'émergence. La totalité des éléments contrôleurs d'un système est répartie et distribuée dans ses entités le constituant. Ce type de gestion de système repose sur les mécanismes locaux. Aucune partie du système ne se charge explicitement du contrôle global du système. Le contrôle décentralisé est une des propriétés la plus difficile à mettre en œuvre. La figure 2.18 schématise le concept d'une décentralisation de contrôle d'un système reconfigurable.

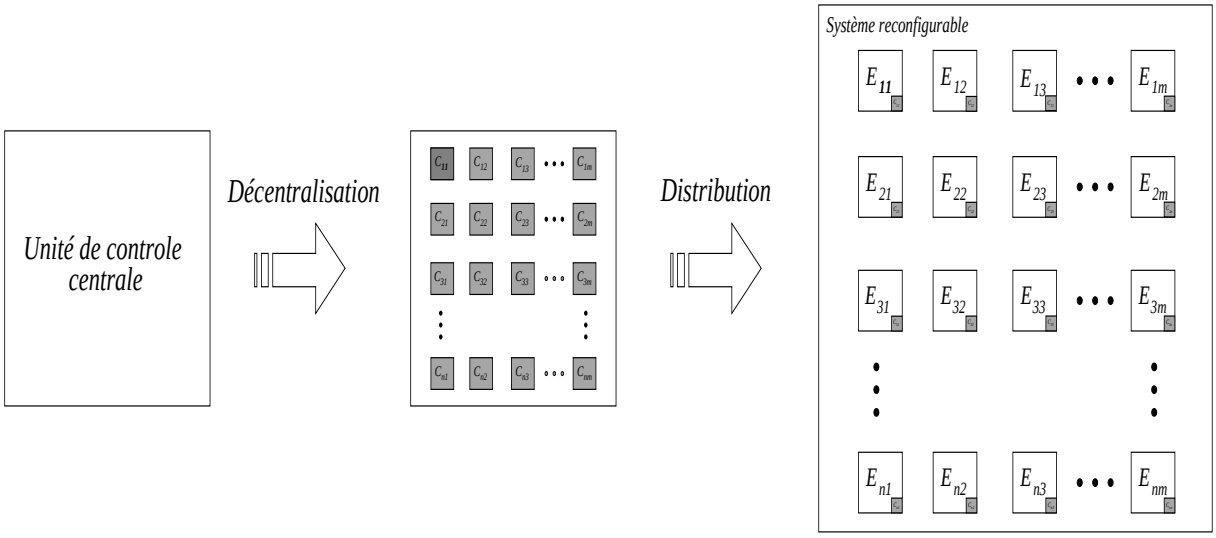


FIG. 2.18 – Décentralisation de contrôle d'un système reconfigurable.

Chaque entité du système prend en charge une partie du « contrôle » du système. Ainsi la gestion du système est assurée par les entités elles-mêmes. Un contrôle reparti impose au système qu'il soit doté de moyens de communication adaptés permettant non seulement l'échange de données entre les entités mais également l'échange de données de contrôle entre les contrôleurs locaux intégrés dans chaque entité du système.

Si on considère que l'ensemble C représente le contrôle d'un système reconfigurable, ses éléments les fonctions de contrôle locales des entités du système, on décrit la décentralisation de la façon suivante :

$$\begin{gathered} C = \{C_{11}, C_{12}, C_{13}, ..., C_{nm}\}, \\ C_{ij} \notin S_{ij}, C_{ij} \cap S_{ij} = \emptyset,\ 1 \leq i \leq n,\ 1 \leq j \leq m,\ n,\ m \in \mathbb{N}. \end{gathered} \tag{2.13}$$

Une entité E_{ij} ($1 \leq i \leq n$, $1 \leq j \leq m$, n, $m \in \mathbb{N}$) effectuant la fonction $e_{ij}(t)$ ($e_{ij}(t) \in S_{ij}$) est responsable non seulement de sa fonction principale e_{ij} mais également de l'exécution d'une partie du contrôle système C_{ij}. Ce contrôle local est independant de la fonction de l'entité dont il fait partie et il s'exécute en parallèle à cette fonction locale.

Le besoin pour un contrôle décentralisé découle également des propriétés de robustesse et de flexibilité. Si l'unité de gestion d'un système était centralisée, la défaillance éventuelle de cette dernière mettrait le système complet en danger. Les unités de contrôle, généralement d'une grande complexité, ne peuvent pas être simplement remplacées sans affecter le comportement global du système. En décentralisant la gestion d'un système, les unités de contrôles perdent en complexité et deviennent plus facilement remplaçables. De plus, en distribuant le contrôle d'un système reconfigurable, le degré d'interactivité des éléments le constituant augmente considérablement favorisant l'apparition des propriétés d'émergence.

2.3.4.7 Interactivité et autres propriétés émergentes des systèmes reconfigurables

La relation 2.8 définit la relation entre le comportement global d'un système reconfigurable et ses entités constitutives tout en faisant apparaître les propriétés possibles d'émergence du système. En effet, cette relation peut s'exprimer d'une manière plus explicite concernant la notion d'émergence :

$$g(t) = \sum_{i,j=1}^{N_e} e_{ij}(t) + I(t), 0 < i \leq n,\ 0 < j \leq m,\ N_e = n \cdot m,\ n,\ m \in \mathbb{N} \tag{2.14}$$

où $I(t)$ représente l'émergence d'un système dans son comportement global. La contribution $I(t)$ dans les systèmes est la raison pour laquelle ces systèmes, étant caractérisés par un fort degré d'interactivité présent dans les relations entre leurs éléments constitutifs, donnent l'impression que leur comportement global dépasse la simple somme des comportements de tous leurs constituants. Si le système est caractérisé par un degré d'interactivité très faible, l'expression 2.14 se réduit à la relation 2.9.

Les propriétés d'émergence m'apparaissant pas uniquement au niveau système comme illustré en figure 2.19, mais peuvent également être appliquées sur une entité. Autrement dit, un

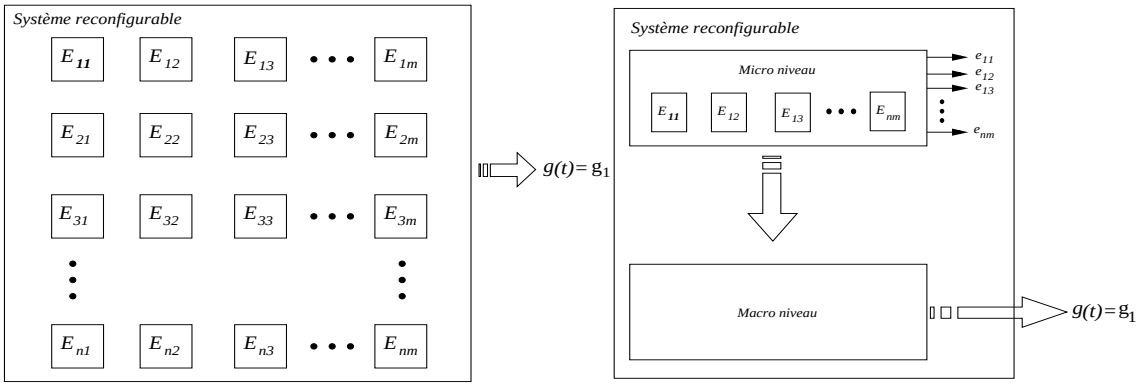


FIG. 2.19 – Illustration de l'émergence d'un système reconfigurable.

système caractérisé par des propriétés d'émergence et par un ensemble des fonctions globales peut être considéré comme une entité sous-hiérarchique d'un autre système dont le comportement globale émerge également des interactions de ses propres entités. D'une manière générale, l'émergence d'un système, ne peut être obtenue si ses entités sont isolées et déconnectées les unes aux autres ou au reste du système. Ces entités doivent « participer » activement à l'activité du système par interactions les unes avec les autres. Les résultats finaux (les données de sortie) d'une entité dépendent à la fois de l'entité qui les fournit mais également d'autres entités du système. La possibilité de faire apparaître l'émergence (« micro-macro effet », « nouveauté radicale », etc.) est plus grande si les résultats finaux dépendent d'autres entités et si le système est complexe (nombre d'entités important).

2.4 Un système reconfigurable auto-organisé : besoins

Pour qu'un système reconfigurable soit conforme aux propriétés et aux exigences imposées par le concept d'auto-organisation et / ou émergence, des travaux doivent être entrepris sur le développement de la méthodologie de conception de tels systèmes. Nous recensons ci-dessous les principaux besoins nécessaires à la conception d'un système reconfigurable auto-organisé :

- *un moyen de communication* robuste, distribué et scalable permettant au système et à ses modules de communiquer et d'échanger des informations à débits élevés. L'utilisation d'un tel moyen de communication se traduit par l'augmentation des niveaux d'interactions entre les modules du système en favorisant ainsi l'apparition des propriétés d'émergence. Ce moyen de communication doit également être adapté à l'utilisation de la technologie reconfigurable et aux changements dynamiques de structure du système supportés par cette technologie.
- *un mécanisme de contrôle* distribué et délocalisé dans le système permettant une gestion globale grâce à des mécanismes locaux déployés dans chaque module. Afin d'assurer une bonne coordination entre tous les modules de contrôle locaux, le mécanisme de contrôle du système doit s'appuyer essentiellement sur le moyen de communication utilisé. La coordination des modules du système peut être assurée par un protocole de communication définissant clairement la manière dont les échanges entre modules s'effectuent.
- *un mécanisme cognitif d'apprentissage* permettant au système d'« apprendre » de ces expériences précédentes et d'utiliser « ses acquis » pour la résolution de nouvelles situations. Cet apprentissage permet au système de réagir à toute situation déjà connue de façon plus efficace et rapide et permet également de renforcer ses réponses comportementales. De plus, les techniques d'apprentissage employées doivent encore permettre au système d'organiser son comportement de manière à ce que ce dernier converge vers une solution (comportement) optimale (selon certaines conditions).

– *des procédés de reconfiguration dynamique* permettant de changer la structure globale ou partielle du système de manière efficace et sûre. Les techniques de reconfiguration dynamique partielle présentent un aspect intéressant dans la conception de tels systèmes. En effet, la possibilité de modifier au cours de fonctionnement une partie matérielle du système sans dégrader le fonctionnement du reste du système permet une auto-restructuration du système fournissant ainsi une solution réelle d'auto-adaptation.

2.5 Conclusion

Après avoir présenté les aspects les plus importants de la technologie reconfigurable, nous avons analysé le principe d'auto-organisation dans le contexte de cette technologie. Nous avons défini et proposé un schéma synoptique d'un système reconfigurable auto-organisé. Ainsi, nous définissons un système auto-organisé reconfigurable comme un système permettant d'une part, sa restructuration et sa gestion fonctionnelle sans aucun contrôle externe ; d'autre part, le système possédant la capacité de s'adapter à tout changement imprévu de l'environnement où il évolue grâce à l'interactivité de ses éléments constitutifs et par des moyens de reconfigurabilité matérielle de sa structure. Nous avons également donné les définitions des caractéristiques principales de l'auto-organisation dans le contexte des systèmes reconfigurables dynamiquement. Chacune de ces caractéristiques a été formalisée et illustrée sur l'exemple d'un système reconfigurable auto-organisé dont le schéma synoptique a été détaillé. Ensuite, à partir de l'analyse des caractéristiques d'auto-organisation dans le contexte des systèmes reconfigurables, nous avons établi les besoins nécessaires pour la conception d'un système reconfigurable auto-organisé. Nous avons identifié et recensé les principaux aspects architecturaux permettant d'aboutir à un système auto-organisé reconfigurable. Ainsi, les aspects de communication interne entre les éléments constituant le système joue un rôle fondamental dans la conception de tels systèmes. En effet, un moyen de communication robuste, scalable et adapté à la reconfigurabilité d'un système doit être développé. Le deuxième aspect à considérer dans la conception des systèmes auto-organisés reconfigurables est l'aspect approche architecturale. Cet aspect comprend les mécanismes de distribution de contrôle d'un système, la coordination de ses modules et la manière dont les décisions niveau système se prennent à travers les mécanismes locaux utilisés. Enfin, le dernier aspect à prendre en considération lors d'une telle conception est la notion de l'auto-apprentissage à travers ses traitements ou calculs antérieurs.

La deuxième partie de cette thèse, traite les deux premiers besoins pour la conception d'un système auto-organisé reconfigurable : l'approche architecturale générale d'un tel système et la structure de communication interne de type *NoC* reconfigurable. Dans le chapitre 3 suivant nous proposons une solution de mise en œuvre d'une approche architecturale permettant la décentralisation de contrôle d'un système et sa gestion d'une manière autonome et independante. Le

chapitre 4 propose un *NoC* scalable adapté pour les systèmes auto-organisés reconfigurables.

Chapitre 3

Proposition architecturale d'un système auto-organisé reconfigurable

3.1 Introduction

La plupart des systèmes que l'on rencontre dans la littérature sont caractérisés soit par un *contrôle centralisé*, où les prises de décision s'effectuent à des niveaux plus élevés dans leur hiérarchie organisationnelle ; soit par un *contrôle décentralisé*, où les prises de décisions s'effectuent à des niveaux hiérarchiques inférieurs. Dans le premier type de contrôle, toutes les connaissances et informations système sont concentrées au plus haut niveau hiérarchique correspondant au niveau système. Puis, ces décisions sont diffusées par une approche descendante vers les niveaux plus bas représentés par les modules constituant le système. Dans le type de *contrôle décentralisé*, les informations et connaissances système circulent en sens inverse, du bas niveau vers le haut niveau de l'organisation. C'est-à-dire, elles circulent depuis les modules constituant le système considéré vers l'ensemble des modules du système. Ces deux types de contrôle sont illustrés dans la figure 3.1.

Les fortes variabilités environnementales dans lesquelles évoluent les systèmes, font que les systèmes à *contrôle centralisé* ne permettent pas de répondre de manière adéquate et efficace à ces évolutions dynamiques. En effet, les systèmes à *contrôle centralisé* présentent de mauvaises performances en termes de rapidité de réaction, de flexibilité, de robustesse et de reconfigurabilité puisqu'ils sont basés sur des structures de contrôle rigides et peu adaptables. Cependant, ces systèmes à *contrôle centralisé* présentent de bonnes caractéristiques en termes de productivité, dû essentiellement à leurs optimisations internes. En terme de robustesse, si l'unité de contrôle d'un système à *contrôle centralisé* a un dysfonctionnement au cours de son fonctionnement, le système considéré dans sa globalité, ne sera plus opérationnel. C'est la raison principale aujourd'hui pour laquelle les approches traditionnelles de conception initialement à base de

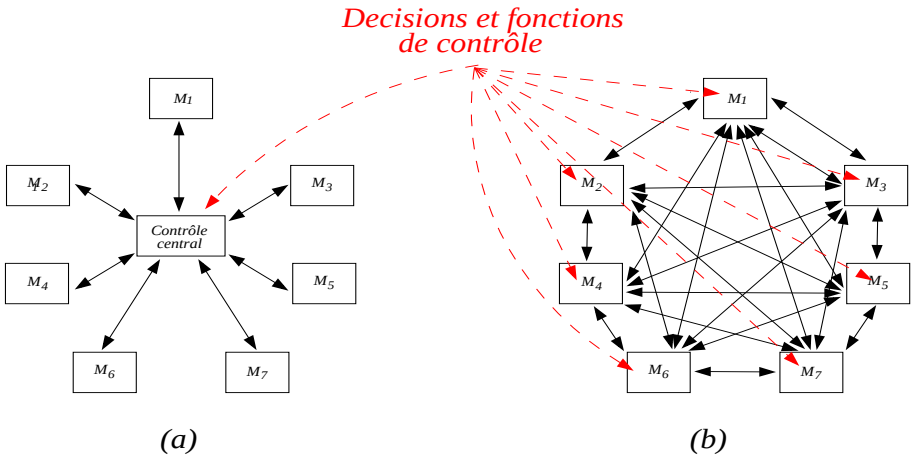


FIG. 3.1 – Types de contrôle : a) *centralisé* et b) *décentralisé*.

contrôle centralisé s'orientent vers des approches à *contrôle décentralisé*. Dans cette approche décentralisée, la gestion globale du système est distribuée dans tous ses modules constitutifs et est assurée par leurs interactions mutuelles. Le tableau 3.1 illustre un résumé des principales différences entre ces deux types d'intégration de contrôle.

Parmi les systèmes dont la gestion est assurée principalement par une approche de décentralisation de contrôle, on trouve les systèmes *multi-agents* (voir la section 1.3.3). Des nombreux exemples d'applications de ces systèmes peuvent être trouvés dans l'industrie [SF89, Par96, LR06]. Ces systèmes reposent essentiellement sur des solutions à base de processeurs (solution logicielle). Il existe peu d'exemples de systèmes à *contrôle décentralisé* implantés dans des structures matérielles de type *FPGA*. En général, les systèmes à base de *FPGA* sont principalement basés sur des structures de contrôle rigides et peu adaptables [CZF+07]. Afin de répondre à des exigences et besoins imposés par le principe d'auto-organisation dans les systèmes reconfigurables, une nouvelle approche architecturale basée sur le concept de *contrôle décentralisé* doit être développée. C'est l'objet de ce chapitre.

	contrôle centralisé	**contrôle décentralisé**
architecture	rigide et statique	flexible, programmable et dynamique
relation	système↔module	module↔module
approche	*top-down*	*bottom-up*
communication	un vers plusieurs	plusieurs vers plusieurs
réponse aux perturbations	faible	élevé

TAB. 3.1 – Comparaison des deux types de contrôle.

3.2 Une approche architecturale matérielle pour les systèmes auto-organisés

3.2.1 Formulation du principe d'auto-organisation matérielle

L'approche proposée repose sur le concept d'auto-organisation défini dans le chapitre 2 et dont le schéma synoptique est présenté dans la figure 3.2. On considère un système composé de N_e entités E_{ij} ($1 \leq i \leq n$, $1 \leq j \leq m$; $n, m \in \mathbb{N}$, $N_e = n \cdot m$). Chaque entité E_{ij} exécute à un instant t donné une fonction $e_{ij}(t)$ parmi l'ensemble de fonction S_{ij} tel que :

$$e_{ij}(t) = e_{ij_k} \in S_{ij}\,(0 \leq k \leq n_{ij}),\ S_{ij} = \left\{\emptyset, e_{ij_1}, e_{ij_2}, ..., e_{ij_{n_{ij}}}\right\}, \tag{3.1}$$

L'ensemble des fonctions en cours et est exécuté par chaque entité contribue à la mise en œuvre d'une fonction principale du système $g(t)$ ($g(t) \in S,\ S = \left\{\emptyset, g_1, g_2, ..., g_{n_g}\right\}$) tel que :

$$g(t) = \sum_{i,j=1}^{N_e} e_{ij}(t), 0 < i \leq n,\ 0 < j \leq m,\ N_e = n \cdot m,\ n,\ m \in \mathbb{N}. \tag{3.2}$$

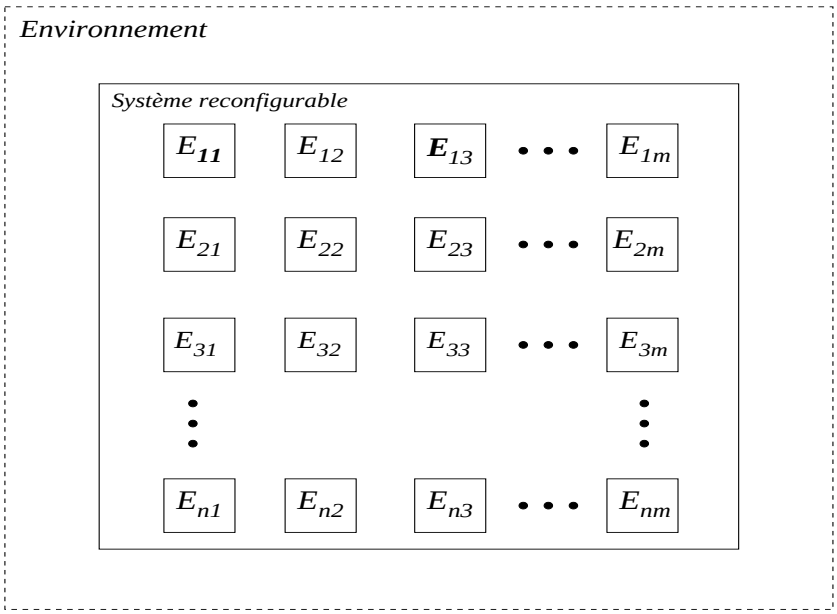


FIG. 3.2 – Schéma synoptique global d'un système auto-organisé.

On considère également que pour chaque fonction e_{ij} d'une entité E_{ij} la relation 2.11 est satisfaite. Plus précisément, pour chaque fonction e_{ij} d'une entité E_{ij}, il existe au moins une combinaison de fonctions e_{kl} des autres entités du système permettant sa substitution (ou remplacement) sans nuire aux fonctionalités (par exécution des fonctions propres) des entités mettant en œuvre cette substitution. Cette substitution s'exprime par la condition suivante :

$$\forall e_{ij} \in S_{ij}, \exists \sum e_{kl} \in \{S_{11} \cup S_{12} \cup \ldots \cup S_{nm}\} \setminus S_{ij} \Rightarrow e_{ij} \subset \sum e_{kl},\ (k,l \neq i,j)$$

L'expression $\sum e_{kl}$ présente la combinaison de fonctions des entités du système permettant la substitution par remplacement de la fonction e_{ij}.

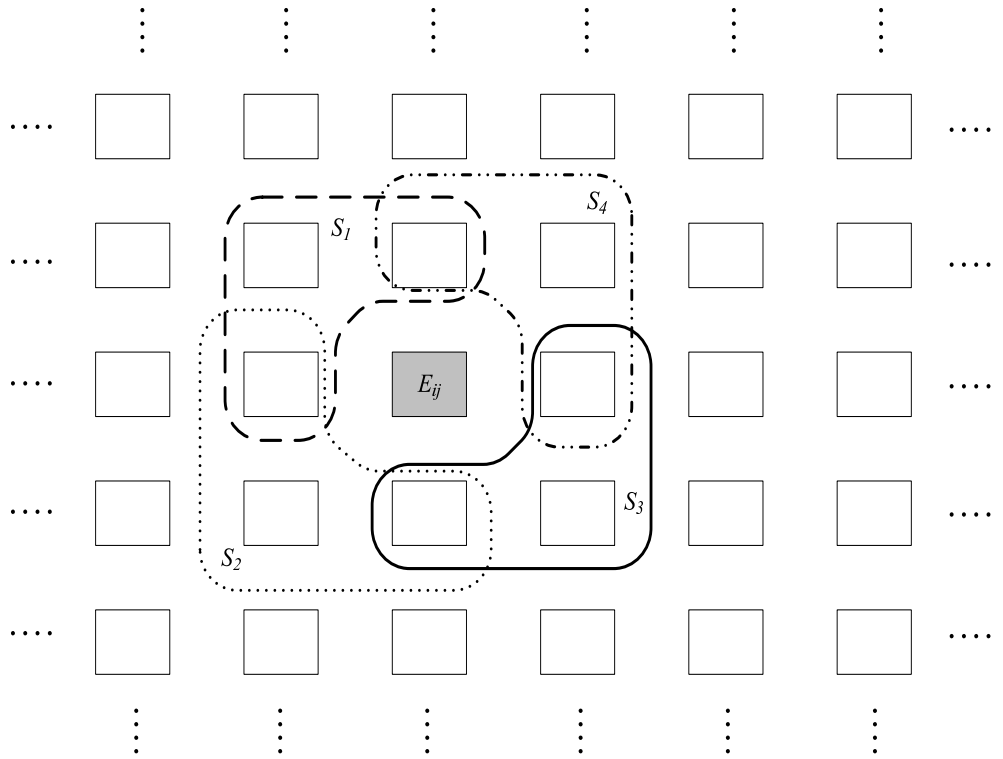


FIG. 3.3 – Règle locale de substitution d'une entité E_{ij} par les entités avoisinantes directes.

Considérant la structure topologique matricielle des technologies reconfigurables actuelles (*FPGA*), nous adoptons comme structure topologique d'un système auto-organisé, une structure de type maillé (*mesh*). Dans un soucis de simplification de la conception et d'une formalisation réelle d'un système auto-organisé, nous adoptons une stratégie de substitution de la fonction d'une entité système, par une seule combinaison de fonctions d'un des ensembles de trois entités voisines directes de l'entité considérée. Ainsi, malgré le fait qu'une entité puisse appartenir à plusieurs groupes d'entités voisines directes, pour chaque fonction d'une entité il existe un seul et unique groupe de trois entités voisines dont les fonctions peuvent assurer sa substitution. Nous adoptons donc une structuration qui limite la redondance fonctionnelle à 1, puisque la fonction d'une entité ne peut pas être substituée par plus d'une combinaison fonctionnelle des ensembles possibles d'entités voisines d'une entité. Bien évidemment, cette stratégie de substitution fonctionnelle considérée limite la robustesse aux éventuelles défaillances simultanées de plusieurs entités mais simplifie notre conception d'un système auto-organisé reconfigurable.

La figure 3.3 illustre la stratégie de substitution fonctionnelle adoptée d'une entité E_{ij}. Dans cet exemple, compte tenu de la position de l'entité E_{ij} considérée, sa fonctionalité (ou ses fonctions) peut être remplacée par l'un des groupes des entités voisines S_1, S_2, S_3 ou S_4. L'attribution du rôle de substitution à l'un de ces groupes est prédéfinie à la conception ou par les groupes eux mêmes en fonction des combinaisons possibles des fonctions des entités les constituant. L'essentiel est que chacune fonctionalité d'une entité puisse être assurée par la combinaison des fonctions de trois entités voisines.

3.2.2 Concept d'une structure informationnelle de perception environnementale : notion de « flux »

Afin que les entités d'un système prennent en considération les activités des entités avoisinantes dans le but de coordonner leurs mécanismes de contrôle locaux, nous introduisons une structure informationnelle dynamique locale que l'on nomme « *flux* » [JTW08b]. L'objectif principal de cette structure est de véhiculer toutes les informations relatives au fonctionnement d'une entité (par exemple états actuels et futurs) pour les transmettre aux entités avoisinantes dans le but de leur donner une perception du fonctionnement environnemental et local au sein du système. Pour chaque quadruple entité du système, un *flux* est associé dans le but de diffuser une information locale de manière décentralisée au système global. Cette stratégie contribue à la mise en place d'une intelligence décentralisée.

Ce *flux informationnel* circule à travers chaque entité du système, rassemblant les informations relatives à leur fonctionnement, à leurs modes de travail et leurs états de fonctionnement en cours ou à venir. Un *flux* n'est pas une structure de contrôle au sens propre du terme, car il n'impose aucune décision aux entités du système qui lui sont associées. Son rôle se limite uniquement à une aide aux entités du système pour des prises de décision locales en leur transmettant les informations vitales pour le maintien ou une émergence fonctionnelle du système global.

Dans notre conception de mise en œuvre d'auto-organisation, aucune entité du système n'effectue une prise de décision sans informations préalables aux autres entités avoisinantes. Chaque décision se prend en « accords mutuelles » des entités associées à un *flux*. Par conséquent, une entité ne peut pas modifier individuellement son mode de fonctionnement et ainsi « mettre en danger » les fonctionalités en mode commun de ses entités avoisinantes ou du système global. Au cours d'un changement de contexte ou environnemental du système, les entités pouvant détecter ces changements informent les entités avoisinantes à travers les *flux*. En fonction de leurs états, les entités concernées peuvent proposer des actions adaptées à ces changements de contexte. En fonction des réponses des entités avoisinantes à travers le *flux informationnel* les actions décidées sont ou non réalisées. Dans le cas où des entités sont autorisées à mettre en

œuvre des solutions répondant aux changements survenus, elles informent les entités avoisinantes au travers les *flux*. Ainsi, une mise à jour des *flux* en fonction des états ou actions en cours ou futures des entités est réalisée durant toute la durée de fonctionnement du système. D'une manière générale, un *flux* permet au système de « résoudre » localement tous les « problèmes » qui surviennent au cours du temps. Dans le cas où une solution locale n'émerge pas, une diffusion informationnelle à d'autres groupes d'entités de proche en proche à travers les *flux* locaux est réalisée en vue d'entendre une recherche de solution ou de résolution aux changements survenus.

La figure 3.4 illustre un exemple de propagation informationnelle au sein du système entier pour résolution de problèmes locaux. Comme illustré dans la figure 3.4, une entité E_{ij} en analysant les paramètres environnementaux découvre que son fonctionnement actuel ne permet pas de fournir une réponse adéquate à ces paramètres, mais faute de solutions à sa disposition elle n'arrive pas à s'organiser afin d'y faire face. A travers une propagation par des *flux informationnels* locaux (propagation par *flux* présenté sur la figure 3.4 par des cercles fléchés), elle informe ses entités voisines de cette situation problématique. Dans le cas où un *flux informationnel local* couvre un groupe de quatre entités, alors au maximum 8 différentes entités voisines du système peuvent être informées de la situation survenue. Ces entités sont représentées en gris clair dans

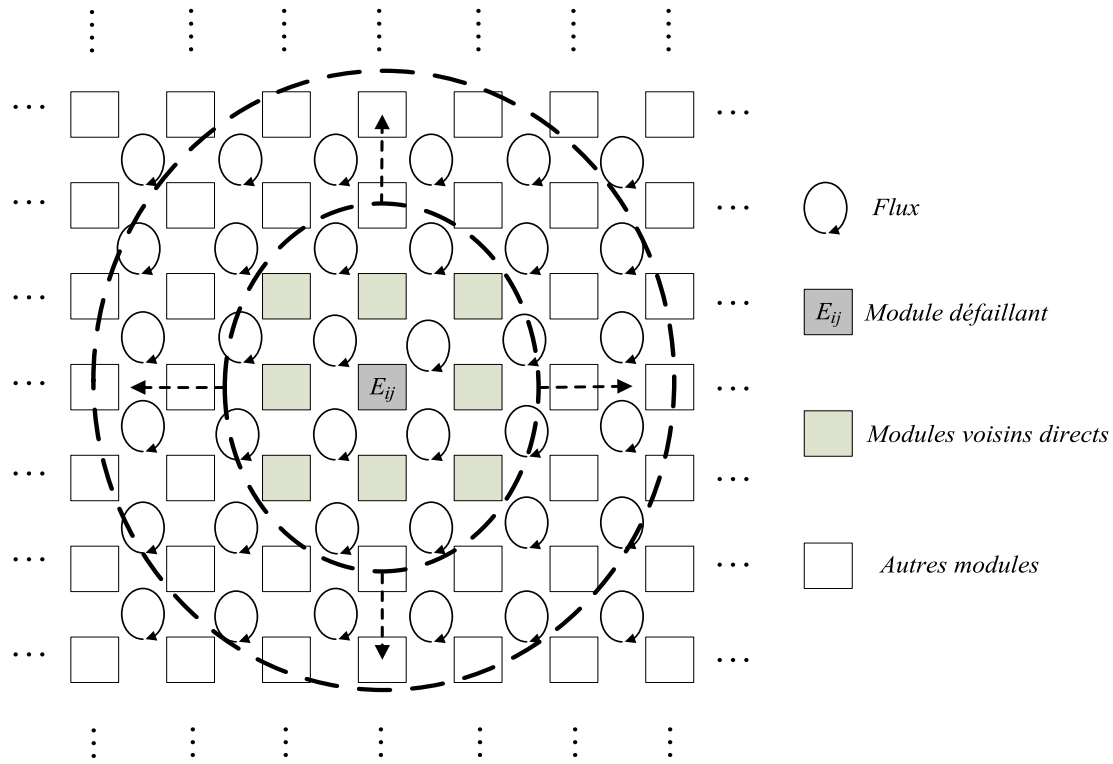


FIG. 3.4 – Propagation d'une information de défaillance d'une entité locale E_{ij} par un mécanisme de *flux informationnel local* au sein d'un système auto-organisé proposé.

la figure 3.4. Dans le cas critique où les entités voisines du module défaillant ne trouvent pas une solution locale de résolution, une solution plus globale parmi les autres entités du système est recherchée. Cette recherche s'effectue au sein du système à travers une propagation progressive de l'information de défaillance dans les entités du système de proche en proche. Cette propagation s'effectue au sein du système à travers une diffusion informationnelle des *flux* couvrant principalement les entités voisines de l'entité concernée. Puis, elle s'étend successivement aux autres entités voisines tant qu'une solution de résolution n'est pas proposée par les entités informées au fur et à mesure de la propagation informationnelle. Dans la figure 3.4 les cercles pointillés illustrent cette propagation informationnelle d'une entité locale vers la globalité du système.

L'approche proposée qui repose sur un concept de *flux* informationnel présente une similitude avec une approche architecturale d'un système de calcul autonome (voir section 1.3.1, chapitre 1). La figure 3.5 illustre cette similitude. En effet, un *contrôleur autonome* (*manager autonome*) récolte des informations environnementales par ses *éléments contrôlés* associés et évoluant dans l'environnement considéré. Ensuite, il analyse et planifie les actions appropriées à mettre en œuvre pas ses *éléments contrôlés* dans le but de répondre à tout changement d'environnement dans lequel le système autonome évolue. En plus d'une décentralisation de contrôle dans les entités du système, notre stratégie repose sur une structure informationnelle circulant entre les entités du système. En effet, notre approche architecturale dispose d'une unité de traitement de *flux informationnel* et d'analyse d'états pour chaque entité du système (correspondant au bloc « observation, analyse et planning » dans la figure 3.5b). Ce bloc suit en temps réel l'évolution des paramètres d'importance du mode de fonctionnement de l'entité considérée et par la même du système. A travers un *flux informationnel* ce bloc renseigne tous les change-

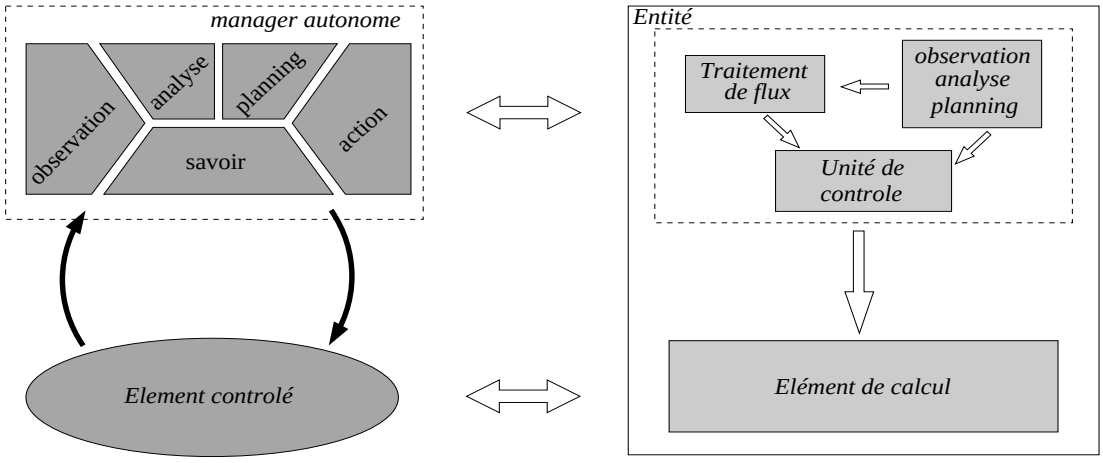


FIG. 3.5 – Similitude d'un système autonome et d'un système auto-organisé proposé contrôlé par le mécanisme de *flux informationnel*.

ments éventuels d'une entité à ses voisines directes. Un *élément autonome* délivre seul ou en interactions avec d'autres *éléments autonomes* du système un certain nombre de services qui sont en accord avec des règles de fonctionnement prédéfinies par le concepteur dans la phase de conception du système. De même, dans notre approche, chaque entité effectue un certain nombre de fonctions également prédéfinies dans la phase de conception du système. De plus, un *élément autonome* n'accepte jamais de faire partie d'un service ou d'une action qui enfreint ses règles de base ou celles de ses voisins. Dans notre approche, les règles de fonctionnement des entités ne sont enfreint grâce au *flux informationnel* qui assure que les entités ne prennent pas de décisions contradictoires avec les modes de fonctionnement des entités du système qu'il renseigne. Ceci est la conséquence d'un *monitoring* et d'un contrôle continu des entités du système à travers un *flux informationnel*.

3.2.3 La structure du *flux local*

La figure 3.6 illustre la composition d'un *flux* pour un système constitué de 4 entités. Un *flux* est composé de plusieurs champs tels que les champs de début « *BEGIN* » et de fin « *END* ». Le critère fondamental de la constitution d'un *flux* est qu'il dispose d'un ou plusieurs champs permettant son identification en tant que *flux*, dans le but de le distinguer d'autres données circulant au sein du système. Hormis les champs de début et de fin, le nombre de champs dans un *flux informationnel* est proportionnel au nombre d'entités qu'il informe ou renseigne dans son parcours entre les entités et au sein du système. Ce parcours peut être prédéfini au cours de la conception. Le nombre de champs peut également être étendu dynamiquement en fonc-

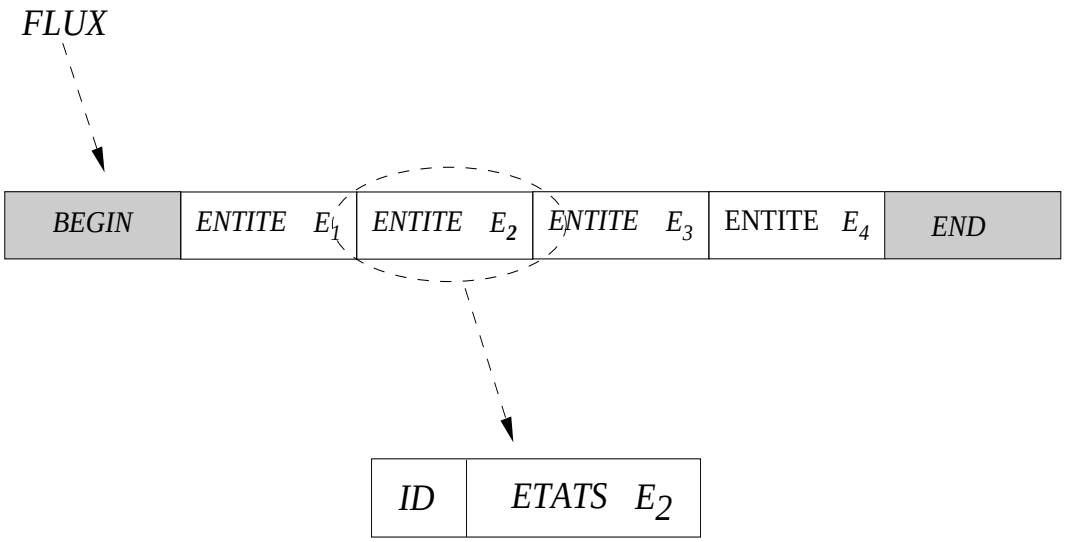


FIG. 3.6 – Structuration en champs d'un *flux informationnel* couvrant un groupement de quatre entités d'un système auto-organisé proposé.

tion de nouveau champ informationnel nécessaire au cours du temps et du nombre d'entités associées au *flux*. A chaque entité du système correspond un champ dans le *flux informationnel*. Plus précisément, un champ du *flux informationnel* contient le numéro d'identification (*ID*) de l'entité associée à ce champ et plusieurs sous-champs. Ces sous-champs sont réservés pour contenir des informations sur l'état global de fonctionnement de l'entité considérée. Le nombre de sous-champs dans un champ du *flux informationnel* n'est pas limité et dépend principalement du nombre d'états possibles qu'une entité peut prendre. Un état correspond soit à un mode opératoire d'une entité à un instant t donné (par exemple, une entité E_i au cours de fonctionnement et d'attente de données de l'entité Ej, entité en arrêt, etc.), soit à une fonction en cours d'exécution. Les sous-champs d'un *flux informationnel* peuvent contenir des informations sur la fonctionnalité d'une entité si un module de vérification de fonctionnalité est implémentée est implanté dans l'entité considérée.

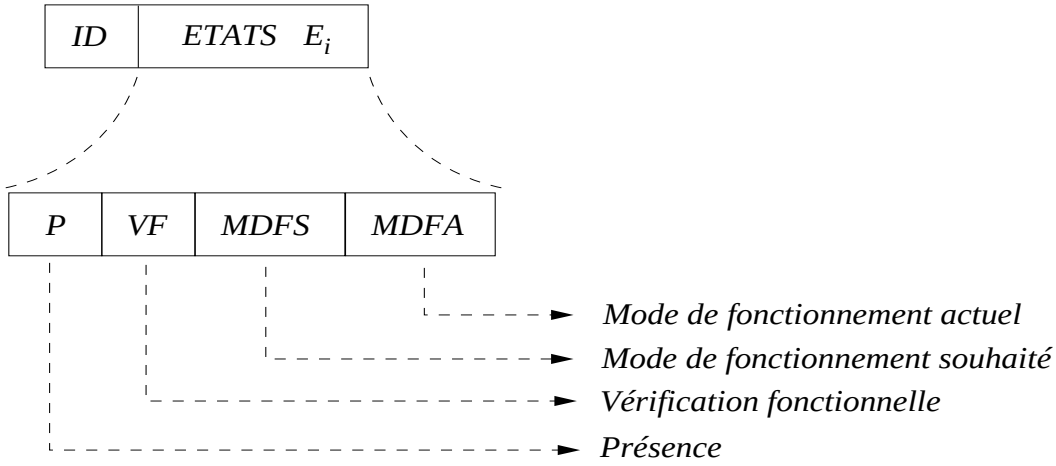


FIG. 3.7 – Structuration en sous-champs d'un *flux informationnel* couvrant un groupement de quatre entités d'un système auto-organisé proposé.

La figure 3.7 illustre un exemple de structuration d'un champ du *flux informationnel* couvrant un groupement de quatre entités d'un système auto-organisé. Dans ce cas d'exemple, chaque champ du *flux informationnel*, hormis les champs de *DEBUT* (*BEGIN*) et de *FIN* (*END*), sont constitués de 4 sous-champs. Le premier sous-champ, nommé *PRESENCE*, indique la présence ou non physique d'une entité dans le système. Ainsi, si ce sous-champ contient la valeur nulle, cela signifie pour les autres entités couvertes par le même *flux informationnel* que cette entité sera supprimée physiquement du système et substituée par une autre entité ou module de calcul. Dans le cas d'une valeur égale à '1' de ce champ, cela signifie que l'entité associée à ce sous-champ au sein du système reste sur puce. Dans le cas où ce sous-champ est de valeur nulle, cette situation présente le cas le plus exigeant en terme d'auto-organisation. En effet, dans ce cas de figure, les entités voisines de l'entité considérée doivent se répartir sa fonction de calcul

au moment de sa désinstallation du système. Un autre exemple de sous-champs présenté dans la figure 3.7 est le sous-champ analyse fonctionnelle *VF*. Ce sous-champ indique aux entités voisines que l'entité associée à ce sous-champ, fonctionne correctement. Plus précisément, afin que les sous-champs d'un *flux informationnel* puissent contenir les informations relatives à l'exactitude des données fournies par une entité, chaque entité doit posséder un module de *vérification fonctionnelle* en relation avec le sous-champ *VF*. De même que pour l'option *PRESENCE*, la *vérification fonctionnelle* présente également un cas d'exigence d'un point de vue organisationnel du système. En effet, si le sous-champ *VF* indique à travers le *flux informationnel* que les données fournies par une entité sont erronées, l'entité considérée et associée à ce sous-champ doit être remplacée. En pratique, la mise en œuvre de telle procédure nécessite l'implantation de structures fonctionnelles redondantes. L'ensemble de ces sous-champs présentés dans la figure 3.7 sont considérés optionnels et leur emploi est décidé par le concepteur selon les spécifications du système auto-organisé à mettre en œuvre (définition de sous-champs complémentaires dans le *flux* correspondant à des optionalités à intégrer dans le système).

Les deux derniers sous-champs présentés dans la figure 3.7 sont des sous-champs relatifs aux modes de fonctionnement d'une entité du système. Ainsi, le sous-champ *MDFA* contient les informations sur le mode de fonctionnement en cours d'une entité, tandis que le sous-champ

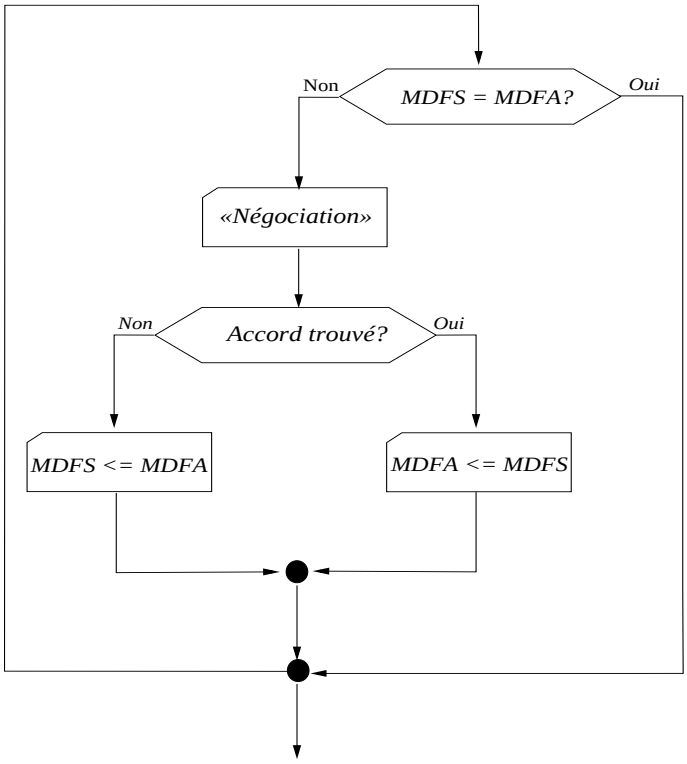


FIG. 3.8 – Un extrait du graphe de contrôle du *flux informationnel* relatif aux requêtes de changement de modes de fonctionnement d'une entité d'un système auto-organisé proposé.

MDFS contient les informations sur le mode de fonctionnement futur souhaité. En général, au cours du fonctionnement d'une entité, ces deux sous-champs contiennent des valeurs identiques, excepté dans la phase de fonctionnement de l'entité correspondant à une requête de changement de mode de fonctionnement. Cette requête est signalée par une entité à ses entités voisines par une écriture dans son sous-champ *MDFS* du *flux informationnel*. Ensuite, une phase de « négociation » entre l'entité considérée et ses entités voisines s'enclenche. En fonction du résultat de cette négociation (favorable ou non), la requête de changement de mode de fonctionnement d'une entité devient ou non effective. Dans le cas où cette négociation a échoué, l'entité initiatrice conserve son mode de fonctionnement initial. La figure 3.8 illustre ce procédé de demande et de négociation de requête de changement de mode de fonctionnement d'une entité d'un système auto-organisé.

Bien qu'en apparence l'exemple du *flux informationnel* présenté par la figure 3.6 présente une structure statique, le concept du *flux informationnel* proposé est une structure dynamique dont la taille évolue en fonction du nombre d'entités associées et au cours du fonctionnement du système auto-organisé. Dans le cas de figure illustré précédemment, le *flux informationnel* présenté par la figure 3.6 correspond à un *flux* couvrant quatre entités d'une structuration de système, chaque entité étant caractérisée par des champs spécifiques. Plus précisément, la taille du *flux informationnel* proposé change et varie particulièrement, dans les situations où une entité doit être substituée fonctionnellement par ses entités voisines. Il s'agit de la situation de fonctionnement dans laquelle les sous-champs *PRESENCE* ou *VF* ont une valeur nulle. L'interprétation et l'évolution d'un *flux informationnel* s'appuient sur des blocs locaux dans chaque entité du système nommé « descripteur » et permettant de les caractériser.

3.2.4 *Descripteur* d'une entité système

Le bon déroulement de la procédure de communication et de « négociation » entre les entités d'un système auto-organisé à travers un *flux informationnel* repose sur les blocs « descripteurs » des entités constituant le système [JTW08b]. Ces blocs correspondent à des structures statiques et locales spécifiques à chaque entité dont les contenus sont établis lors de la phase de conception des entités. Chaque structure décrit de manière précise les spécificités et les modes de fonctionnement de son entité du système associé. Chaque *descripteur* contient donc les informations relatives aux modes de fonctionnement d'une entité, au nombre de services qu'elle peut délivrer, à la procédure de substitution des fonctions de l'entité spécifiée ainsi que les informations utiles aux entités voisines en cas de défaillance.

Un *descripteur* établi lors de la phase de conception d'une entité ne peut être ni modifié ni mis à jour au cours de fonctionnement de son entité, excepté dans le cas où une entité « change » de structure lors, par exemple d'une reconfiguration de l'une de ses parties (reconfiguration partielle d'une entité). Ainsi, l'ensemble des services qu'une entité peut délivrer est

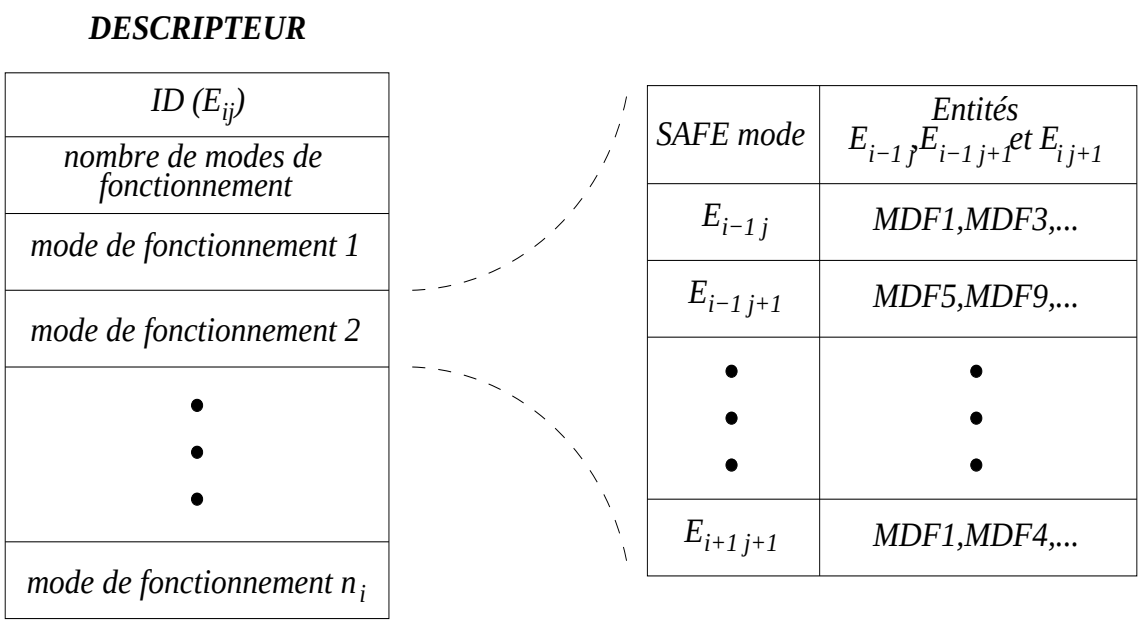
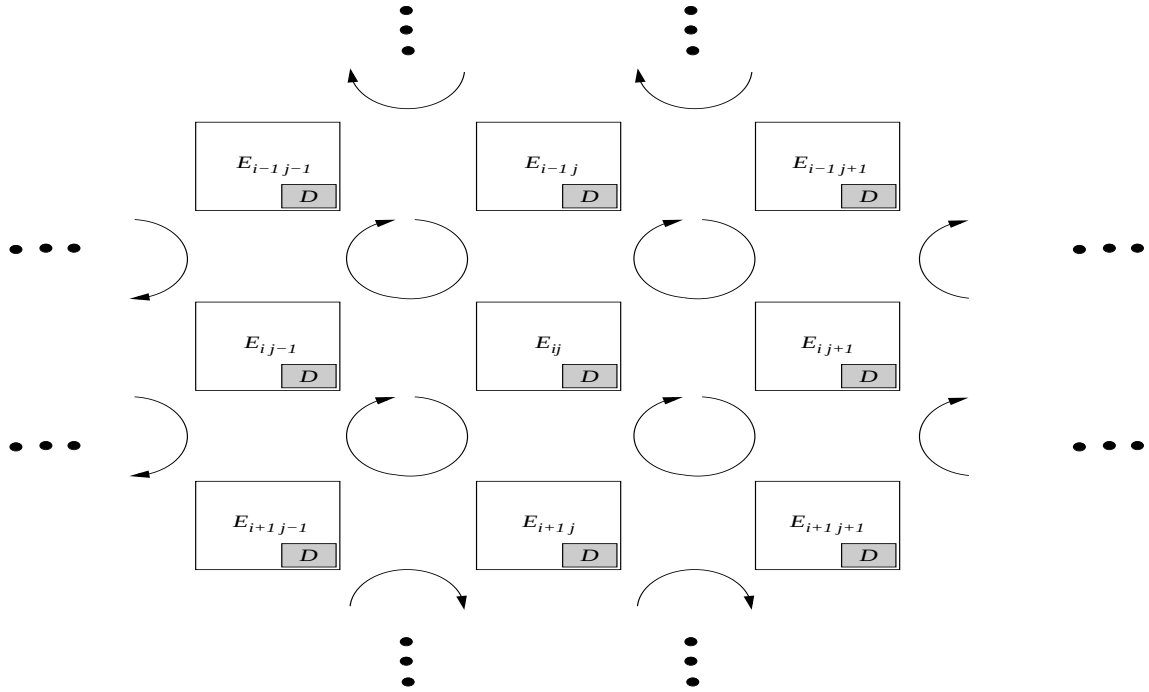


FIG. 3.9 – Exemple de structure d'un *descripteur* d'une entité E_{ij}.

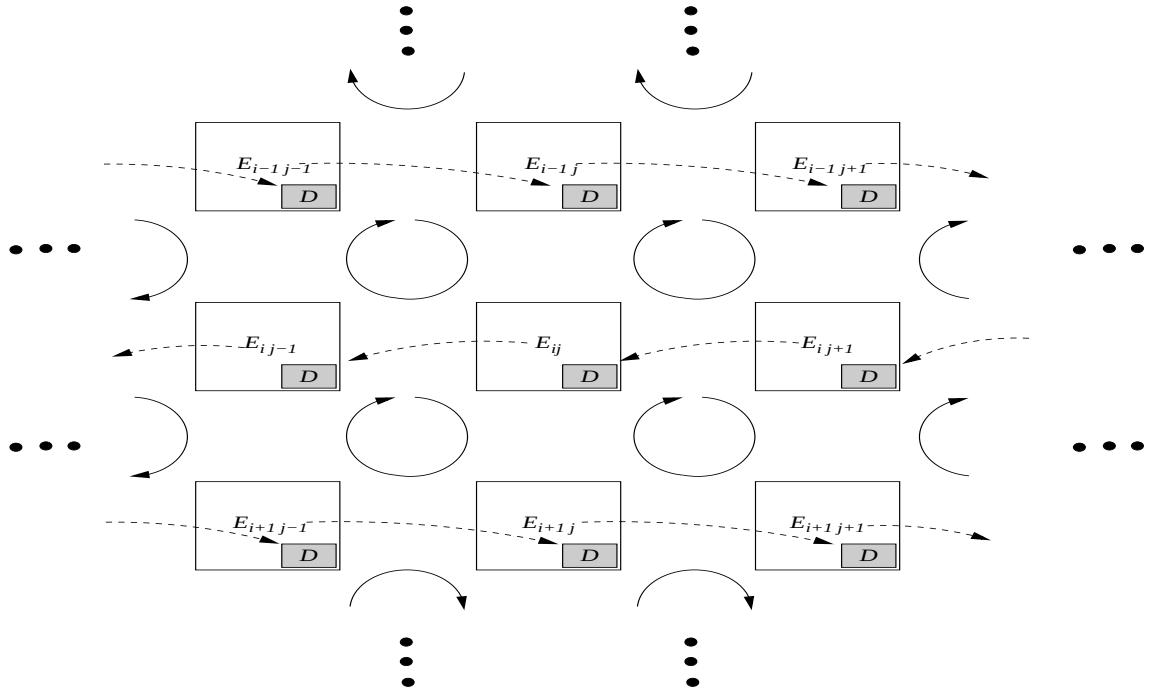
modifié et par conséquent son *descripteur* est mis à jour.

La figure 3.9 illustre un *descripteur* d'une entité E_{ij}. Ce *descripteur* contient à la fois tous les modes de fonctionnement de l'entité considérée, les procédures de substitution de fonctions à partir des fonctions d'autres entités du système, les « compatibilités » entre les fonctions des entités couvertes par un ou plusieurs *flux informationnels*, etc. Par exemple, la figure 3.9 présente un *descripteur* dont le « mode de fonctionnement 2 » de l'entité E_{ij} peut être remplacé par les modules E_{i-1j}, E_{i-1j+1} et E_{ij+1}. Cette possibilité de substitution est spécifiée par le champ clé *SAFE mode* dans un *descripteur*. De plus, cet exemple présente que le « mode de fonctionnement 2 » de l'entité E_{ij} est « compatible » avec les fonctions *MDF1, MDF3, ...* de l'entité E_{i-1j}, *MDF5, MDF9, ...* de l'entité E_{i-1j+1}, etc. Dans un *descripteur*, les informations sur les passages d'un mode de fonctionnement à un autre sont également incluses ainsi que les informations jugées utiles par le concepteur lors de la conception d'une entité système. Ainsi, un *descripteur* d'entité permet aux entités voisines à déterminer des solutions les mieux adaptées à d'éventuels changements environnementaux du système ou défaillances d'une ou plusieurs entités système au cours de fonctionnement.

Un *descripteur* peut être intégré de deux manières différentes au sein d'une entité : soit sans duplication, c'est-à-dire qu'un seul descripteur est intégré dans une entité qui lui est associée ; soit de façon redondante où un descripteur est intégré simultanément dans une entité qui lui est associée mais également dans l'une de ses entités voisines situées dans le sens de circulation de l'un des *flux informationnels* couvrant les deux entités (voir figure 3.10). Dans les deux cas, chaque entité contient son propre *descripteur*. La principale différence entre ces deux



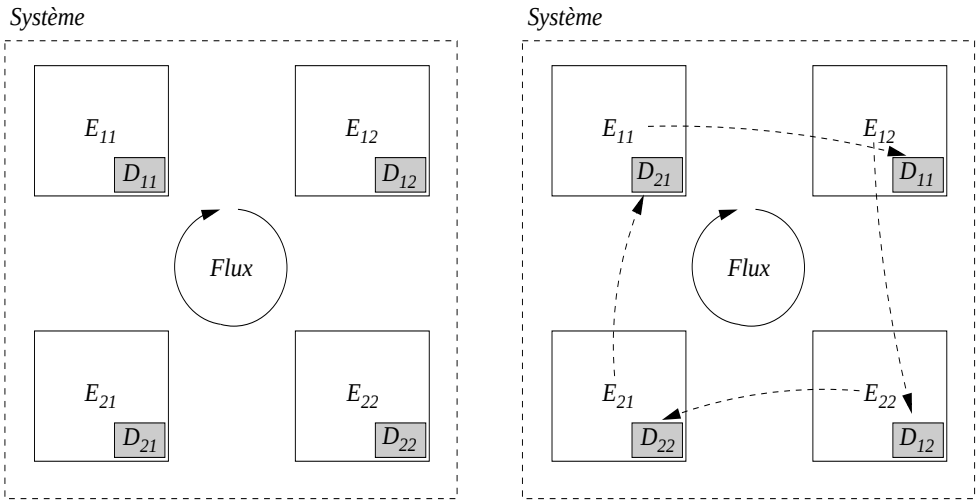
(a) Intégration de blocs *descripteurs* dans les entités.



(b) Intégration redondante de blocs *descripteurs* dans les entités voisines selon le sens de circulation des *flux informationnels*.

FIG. 3.10 – Mode d'intégration de blocs *descripteurs* dans les entités d'un système auto-organisé proposé.

modes d'intégration de blocs *descripteurs* réside dans le module de *vérification de flux*. Dans le premier mode, on considère que le module de *vérification de flux* est infaillible et ce quel que soit l'état de l'entité (défaillant ou non). Dans ce cas de figure théorique, un *descripteur* délivrera à tout moment et sans défaillances possibles les informations relatives de son entité associée. Dans le deuxième cas de figure, plus pragmatique, aucune supposition sur l'infaillibilité de fonctionnement du module de *vérification de flux* n'est tolérée. En pratique, dans le cas d'une intégration non redondante et lors d'une défaillance complète d'une entité, ses entités voisines couvertes par un des *flux informationnels* ne peuvent accéder aux informations relatives au fonctionnement de l'entité défaillante. Par contre, pour le cas d'une intégration redondante correspondant pour chaque entité à l'intégration simultanée de deux blocs *descripteurs* (le *descripteur* propre à l'entité et un *descripteur* de redondance d'une des entités voisines selon le sens de circulation du *flux informationnel* les associant), et en cas de défaillance complète de l'entité, cette redondance permet à une (ou plusieurs) entité voisine de restituer les informations relatives à l'entité défaillante. Ainsi, la substitution fonctionnelle de l'entité considérée peut être entreprise par les autres éléments constituant le système global. La figure 3.11 illustre l'intégration de blocs *descripteurs* dans un système auto-organisé composé de quatre entités.



(a) Intégration de blocs *descripteurs* dans les entités.

(b) Intégration de blocs *descripteurs* dans les entités voisines selon le sens de circulation du *flux informationnel.*

FIG. 3.11 – Exemple de règle d'intégration de *descripteurs* dans un système auto-organisé à quatre modules.

3.2.5 Mise en œuvre du concept d'auto-organisation par vérification de *flux informationnel*

Chaque entité d'un système possède un module de *vérification de flux*. Ce module reçoit le(s) *flux informationnel(s)* pour analyse et traitement des informations relatives aux entités associées à ce *flux*. Cette procédure de vérification s'effectue de la manière suivante : dans une première étape, chaque entité vérifie les états des entités voisines couvertes par un même *flux informationnel*. Ensuite, chaque entité met à jour les champs du *flux informationnel* qui lui sont associés tout en prenant en considération les états des entités voisines. Par exemple, dans le cas d'un système auto-organisé constitué de quatre entités, chaque entité vérifie les états des trois autres entités. La figure 3.12 illustre ce mécanisme de *vérification de flux* sur un système auto-organisé constitué de quatre entités. Les modules $V_{11},\ V_{12},\ V_{21}$ et V_{22} représentent respectivement les modules de *vérification de flux* des entités $E_{11},\ E_{12},\ E_{21}$ et E_{22}.

La figure 3.13 illustre le graphe de contrôle d'un module de *vérification de flux* couvrant ses quatre entités système. Le mécanisme de *vérification de flux* est illustré par le bloc nommé *VERIFICATION*. Chaque sous-champ du *flux informationnel* reçu est donc analysé. Si on considère une structure de *flux informationnel* telle que présentée dans la figure 3.7, alors dans la phase de *vérification de flux*, l'ensemble des sous-champs (*P*, *VF*, *MDFS* et *MDFA*) sont vérifiés. Ainsi,

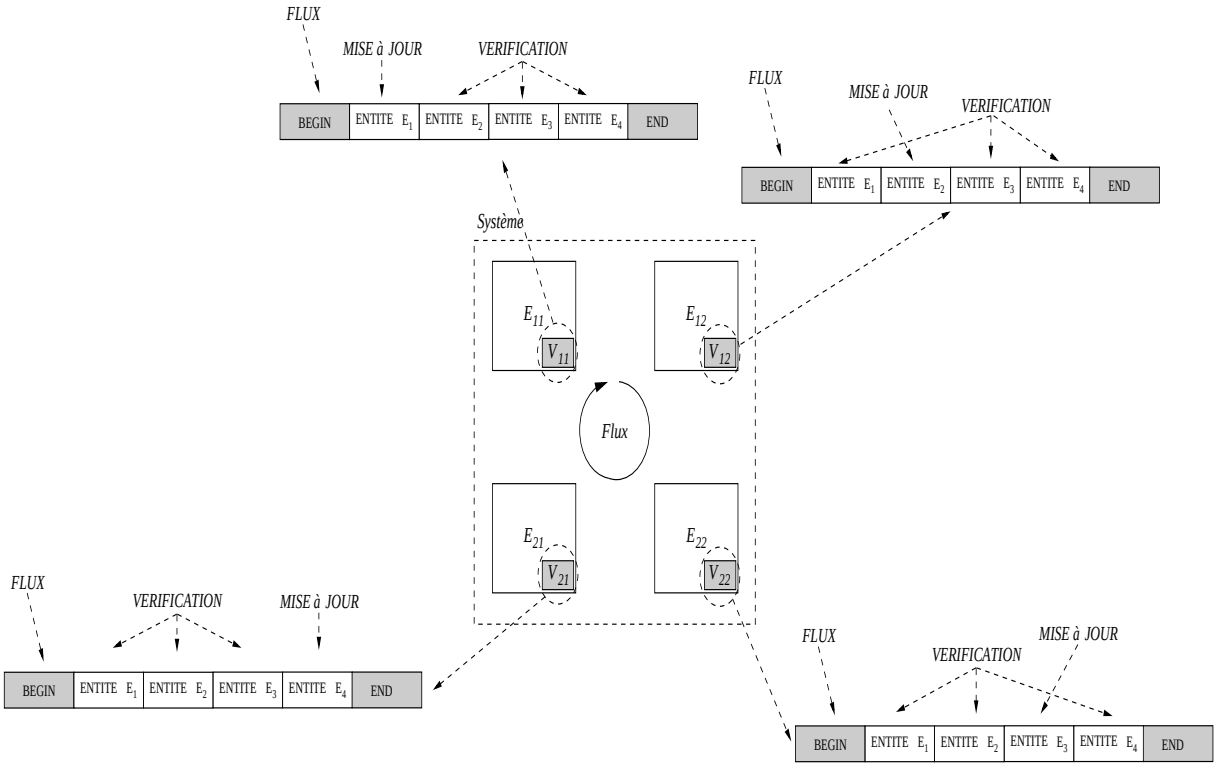


FIG. 3.12 – Vérification de *flux informationnel* dans un système composé de quatre modules.

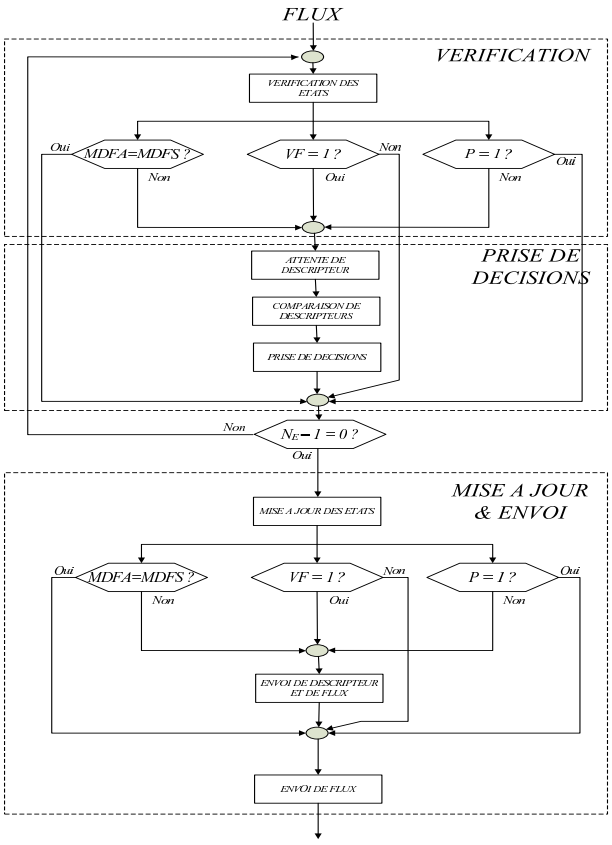


FIG. 3.13 – Graphe flot de contrôle de module de vérification de *flux*.

par exemple, si le sous-champ *VF* indique que les résultats d'une entité sont corrompus, l'entité effectuant l'analyse en cours du *flux informationnel* à travers son module de *vérification de flux* décide ou non de proposer une réponse de résolution de la défaillance détectée. Dans le cas où l'entité analysant et traitant le *flux informationnel* ne dispose pas d'informations suffisantes sur le fonctionnement de l'entité détectée comme défaillante, elle attend que d'autres entités voisines, ou que l'entité défaillante elle-même transmet les informations relatives à l'entité défaillante et contenues dans son *descripteur*. Dans le cas où l'entité analysant le *flux informationnel* dispose de la redondance du bloc *descripteur* de l'entité défaillante, elle transmet ces informations aux autres entités voisines couvertes par le même *flux informationnel*. La figure 3.14 illustre le processus d'insertion des informations utiles du *descripteur* de l'entité E_1 correspondant à son mode de fonctionnement en cours lors de la procédure de vérification de *flux informationnel* par l'entité E_2. Ce processus s'effectue au cours d'une défaillance de l'entité E_1 signalée à l'entité E_2 à travers le *flux informationnel*. Cette procédure d'insertion partielle du *descripteur* de l'entité E_1 par l'entité E_2 correspond au cas d'intégration des *descripteurs* de manière redondante dans les entités du système. Dans le cas d'exemple présenté dans la figure 3.14, l'entité E_2 contient le *descripteur* de l'entité voisine E_1 et est responsable de l'envoi du

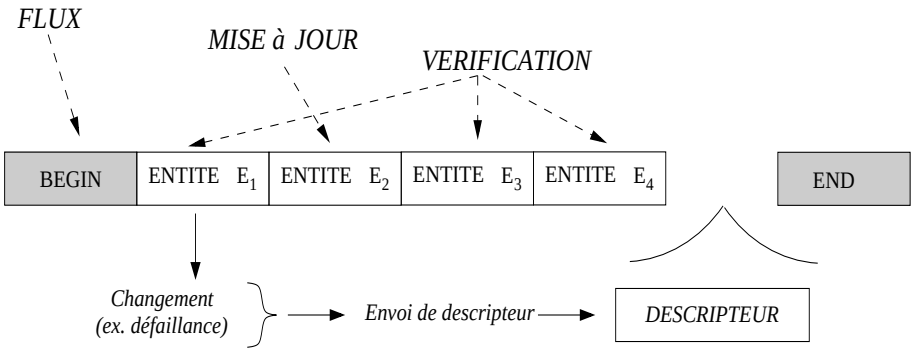


FIG. 3.14 – Illustration d'insertion d'une partie d'un *descripteur* dans un *flux informationnel*.

descripteur considéré aux autres entités couvertes par le même *flux informationnel*. Cette redondance servira en cas de besoin lors d'une défaillance complète nécessitant la substitution d'une entité. Une fois que les informations *déscripteurs* sont reçues, l'entité passe à la phase *PRISE DE DECISIONS*. Dans cette phase, l'entité traitant du *flux informationnel*, choisit éventuellement un mode de fonctionnement approprié contribuant à la résolution de la défaillance détectée à travers la mise en œuvre d'une substitution partielle du fonctionnement de l'entité défaillante tout en assurant son mode de fonctionnement en cours (exécution de ses fonctions en plus de fonctions de substitution). Après l'analyse des états de ses entités voisines et éventuellement la mise en œuvre d'une prise des décisions suite à cette analyse, l'entité considérée passe à une phase *MISE A JOUR*. Dans cette phase, l'entité met à jour ses états dans les champs du *flux informationnel* qui lui sont associés. Cette mise à jour d'états intègre les prises de décisions des phases précédentes. Une fois mis à jour, le *flux informationnel* peut être transmis vers les autres entités couvertes par le même *flux informationnel*.

Chaque entité du système effectue la même procédure de *vérification de flux*. Si, lors d'un passage d'un *flux informationnel*, un changement d'états d'une des entités couvertes par ce *flux* se produit, toutes les entités sont informées de ce changement à l'issu du passage suivant du *flux informationnel*. De plus, si une des entités doit être supprimée et remplacée ultérieurement par une autre entité, l'entité en question ne doit plus être fréquentée par le(s) *flux informationnel(s)* qui la couvraient lors de son fonctionnement.

L'approche conceptuelle matérielle d'auto-organisation présentée est opérationnelle pour les situations où au maximum une entité d'un groupement de quatre entités couvertes par un *flux informationnel*, subit une défaillance au cours de son fonctionnement. Des extensions de cette approche dans le but d'apporter une plus importante flexibilité par rapport à des défaillances éventuelles complètes d'une des entités du système sont possibles mais au détriment de ressources matérielles supplémentaires pour chaque entité du système et donc du système auto-organisé global.

3.3 Conclusion

Après avoir présenté brièvement les méthodes de répartition des aspects « contrôle » généralement utilisées dans la conception des systèmes, nous avons proposé une approche matérielle de contrôle permettant la mise en œuvre du concept d'auto-organisation des systèmes. La stratégie adoptée repose sur deux structures : une structure dynamique au sein du système que l'on nomme *flux informationnel* circulant à travers les entités constituant le système auto-organisé et une structure statique locale (intégrée dans chaque entité) nommée *descripteur*. On définit le *flux informationnel* comme une structure variable dont l'objectif principal est de recenser toutes les informations relatives aux fonctionnements des entités qui lui sont associées. L'objectif est d'établir des liens de communication avec les entités voisines dans le but de les renseigner du contexte environnemental local (perception de chaque entité) pour la mise en œuvre d'une émergence éventuelle par prise de décision individuelle possible par chaque entité constituant le système proposé. Dans le concept proposé, le *flux informationnel* n'est pas une structure de contrôle imposant des décisions aux entités du système. Cette structure dynamique assure le renseignement sur les modes de fonctionnement ou les états d'une entité en permettant des communications informationnelles entre les entités du système. L'originalité de cette approche est qu'elle ne nécessite pas un contrôle centralisé à travers une structure de contrôle spécifique et figée des communications, tout en intégrant les aspects d'auto-adaptabilité et d'auto-organisation structurels des entités constituant le système. Cette approche permet donc de mettre en œuvre une propriété fondamentale des systèmes auto-organisés qu'est le *contrôle décentralisé*. Le *flux informationnel* permet donc une éventuelle prise de décisions dans le cas de changements environnementaux ou de défaillances fonctionnelles des éléments constituant le système. Ces changements susceptibles d'intervenir correspondent à des changements des paramètres d'environnement dans lesquels les entités évoluent. Associé au concept de *flux informationnel*, la notion locale informationnelle au niveau des entités du système est la définition d'un *descripteur* correspondant à une structure statique. Cette dernière est définie dans la phase de conception de chaque entité potentiellement intégrée dans le système au cours de son fonctionnement. D'une manière générale, un *descripteur* est une structure de données embarquée caractérisant sont entité qui lui est associée. Plus précisément, un *descripteur* contient toutes informations sur les modes de fonctionnement possibles d'une entité ainsi que ces configurations fonctionnelles permettant à l'entité considérée de substituer des fonctions d'autres entités du système considérées comme défaillantes ou bien devant exécuter d'autres fonctions prioritaires selon l'évolution environnementale du système. En résumé, l'approche proposée basée sur ces deux structures informationnelles et de données correspond à un procédé de délocalisation des mécanismes de contrôle dans les entités du système tout en assurant une auto-coordination à travers des phases de négociations entre les entités du système dans le but de faire émerger une

auto-organisation.

L'approche d'auto-organisation basée sur un *flux informationnel* et des blocs *descripteurs* locaux aux niveaux des entités a été présentée et détaillée dans le cas d'un système auto-organisé constitué de quatre entités couvertes par un même *flux*. Plus précisément, la constitution d'un *flux informationnel* pour le cas de défaillance d'une entité a été considérée et présentée. De même, le mécanisme d'auto-surveillance du système global par l'ensemble des entités constitutives, à travers les différents *flux informationnels* couvrant l'ensemble de ces entités associées, a également été présenté et détaillé.

La solution de mise en œuvre du concept d'auto-organisation proposé, nécessite le développement d'une structure de communication sur puce adaptée à la fois aux architectures reconfigurables mais également aux *flux informationnels* et de données présentés dans ce chapitre. C'est l'objet du chapitre suivant, qui consiste à la proposition de solutions de mise en œuvre de moyens de communication sur puce (*NoC*) permettant à la fois l'échange d'informations et de données entre les entités d'un système tout en permettant une auto-adaptabilité structurelle de communication répondant aux exigences d'une architecture sur puce reconfigurable auto-organisée.

Chapitre 4

Réseau de communication sur puce pour un système reconfigurable auto-organisé

4.1 Introduction

La communication et l'interconnexion entre les modules d'un système électronique ont été toujours des aspects primordiaux dans sa conception. L'évolution des systèmes électroniques vers des systèmes complets entièrement intégrés (*System on Chip - SoC*) a conduit également à une évolution des communications des données de ces systèmes. Nous sommes passés de l'interconnexion au niveau composant élémentaire dans les années quatre-vingts, à actuellement l'interconnexion au niveau d'une puce (*intra-chip communication*). D'une manière générale, les problèmes d'interconnexion ont simplement changé d'échelle. Dans le cas du développement de système microélectronique de type *MPSoC reconfigurable auto-organisé,* permettant une auto-adaptabilité à travers une flexibilité de la structure matérielle constituant de tels systèmes, de nouvelles structures de communication entre les entités sur puce doivent être élaborées. Ce chapitre se limite donc aux communications internes sur puce des systèmes électroniques embarqués.

Nous distinguons plusieurs types de communication entre les modules (*processeurs, IP, mémoires, etc.*) d'un système sur puce :

- communication point à point,
- communication par un bus partagé,
- communication par un réseau sur puce.

Ces structures de communication correspondent aux besoins apparus au cours de l'évolution des systèmes microélectroniques (*Système sur puce - SoC* vers des *Systèmes sur puce Multi-Processeurs - MPSoC*) et s'orientent aujourd'hui à travers nos travaux vers des *MPSoC reconfigurables et auto-organisés* dans le but de disposer d'une forte flexibilité et adaptabi-

lité. Nous décrivons brièvement ci-dessous les principales caractéristiques de ces structures de communication :

- ***La communication point à point*** entre des modules d'un système s'effectue à travers les connexions ou fils dédiés uniquement à l'échange de données entre ces modules [HDM02]. Ce type d'interconnexion entre les modules d'un système est simple de réalisation et permet des échanges de données très rapides selon des protocoles d'échanges mais limite toute flexibilité et extensibilité lors de la conception de systèmes sur puce. Cette structure de communication est utilisée principalement dans le cas où des débits de transfert de données très élevés sont nécessaires entre un nombre de modules du système relativement faible au détriment d'une rigidité du système.
- ***La communication par un bus partagé ou hiérarchique*** propose un moyen de communication inter-modules majoritairement utilisé jusqu'à ce jour [Rab00, PD08]. Nous distinguons trois types de bus partagé les plus souvent utilisés : les bus *backplane*, les bus *Processeur-Mémoire* et les bus *Entrée-Sortie*. Les bus *backplane* permettent la communication entre les processeurs, les mémoires et les blocs entrée / sortie, tandis que les bus *Processeur-Mémoire* disposant d'une grande bande passante sont couramment utilisés dans la communication entre les processeurs et les mémoires. Les bus *Entrée-Sortie* sont caractérisés par une bande passante moins élevée que les deux précédents bus mais pouvant être connectés à deux premiers. Ce type de bus est généralement utilisé dans l'industrie. Des exemples d'architectures utilisant ce type d'interconnexion nous trouvons dans la littérature [Ini94, Son02, JS00, Mic00].

  Dans ce type d'interconnexion, tous les modules d'un système partagent un même bus de données. Un bus partagé ou hiérarchique est composé de lignes de données, de contrôle et d'un module d'arbitrage. Les lignes de données transportent les informations entre un module source et un module destinataire. Les lignes de contrôle portent les informations sur les signaux de contrôle (signal de requête, de validation, d'acquittement, etc). Le module d'arbitrage gère toutes les opérations demandées sur un bus et l'attribution du bus en autorisant les connexions entre une source (désignée comme un maître ayant droit d'utiliser les services du bus) et une cible (désignée comme un esclave). Avec ce type de structure de communications, une seule opération de communication entre des modules connectés est possible. Néanmoins, ils sont caractérisés par une mise en œuvre simple permettant de répondre aux besoins de conception de systèmes sur puce tout en offrant un faible coût de réalisation. Par contre, cette structure de communication présente d'une part de fortes limitations de performances (goulot d'étranglement des données) dès que le nombre de modules interconnectés augmente significativement (supérieur à une vingtaine d'entités interconnectées). D'autre part, elle limite la flexibilité et la scalabilité de la structure matérielle du système sur puce puisqu'elle introduit des contraintes d'intégra-

	P à P	Bus partagé	NoC
Parallélisme	très bon	mauvais	très bon
Scalabilité	très mauvais	bon	très bon
Réutilisation	très mauvais	très bon	très bon
Consommation	bon	mauvais	très bon

TAB. 4.1 – Comparaison des interconnexions.

tion et de placement des modules du système sur la même puce. En effet, le nombre de modules connectés à un bus est limité car les effets parasites (effets capacitifs et résistifs, couplage entre lignes - *crosstalk*) et la consommation d'énergie, associés principalement à la longueur des lignes et au nombre de modules connectés, deviennent significatifs. Ceci augmente considérablement les retards de propagation sur le bus, qui peuvent avoisiner les valeurs de fréquences de fonctionnement du bus pouvant entraîner alors un dysfonctionnement du système.

– ***La communication par réseau sur puce (Network on Chip - NoC)*** propose une alternative au bus partagé en ne manifestant pas ses principaux inconvénients. En effet, avec la complexité grandissante des systèmes sur puce vers des systèmes sur puce *Multi-Processeurs*, une interconnexion par bus partagé montre des limites significatives proportionnelles au nombre de modules ou entités (*IP, processeurs, etc.*) constituant ces systèmes [BDM01, DT01, BDM02, GG00]. Ces réseaux sur puce sont caractérisés par un parallélisme explicite, une architecture distribuée, un degré de modularité important et une bande passante élevée [AG03, MBV+02, KND02, PGIS03].

Considérant les avantages et les inconvénients des structures de communication interne pour les systèmes intégrés, l'approche de conception des inter-communications dans les systèmes sur puce (*SoC* ou *MPSoC*) a évolué des architectures de bus partagés ou hiérarchiques vers des architectures qui reposent actuellement principalement sur le développement de structure de type *réseau sur puce - NoC*. Le tableau 4.1 récapitule les trois types d'interconnexions présentés et les propriétés qui leurs sont associées.

L'objet de ce chapitre est de proposer une nouvelle structure de communication sur puce adaptée à la conception d'un *système reconfigurable auto-organisé*. La suite de ce chapitre est organisée de la manière suivante. Dans un premier temps, nous rappelons les principales caractéristiques des réseaux sur puce. Nous justifions la raison pour laquelle les *NoC classiques* ne sont pas adaptés à des architectures *MPSoC reconfigurables* et embarquées dans les circuits *FPGA* et la nécessité de développer des architectures *NoC reconfigurables* (*CuNoC* et *QNoC*) correspondant à une évolution de nos recherches sur la conception d'architectures adaptées pour les systèmes reconfigurables. Nous donnons leurs principaux avantages et inconvénients

et faisons une comparaison entre ces deux architectures. Enfin, un algorithme de routage spécifiquement destiné à être utilisé pour les deux architectures *NoC* reconfigurables proposées est présenté et détaillé. Une conclusion résumant l'apport de nos travaux et des perspectives à effectuer dans la conception de structures *NoC reconfigurables* pour technologies *FPGA* termine ce chapitre.

4.2 Réseaux sur puce

Les réseaux sur puce (NoC) apparaissent comme une alternative pour connecter des modules intégrés sur une même puce vis-à-vis d'une approche d'interconnexion basée sur une structure de communication de type bus partagé ou hiérarchique. Ils représentent une solution de transposition et de réduction d'échelle (au niveau d'un même substrat) des concepts des grands réseaux de communications (*large-scale networking*) vers le domaine des systèmes sur puce embarqués (*SoC, MPSoC*). Les *NoC* reposent sur une paquetisation de données pour transporter les données d'une source vers une cible à travers un réseau d'aiguilleurs (*routeurs*) et de canaux d'interconnexion (*interconnection link*). Un exemple d'une structure de *NoC* couramment utilisée est illustré dans la figure 4.1. Il s'agit d'une structure *NoC* de type maillé *2D* (*mesh*), composée

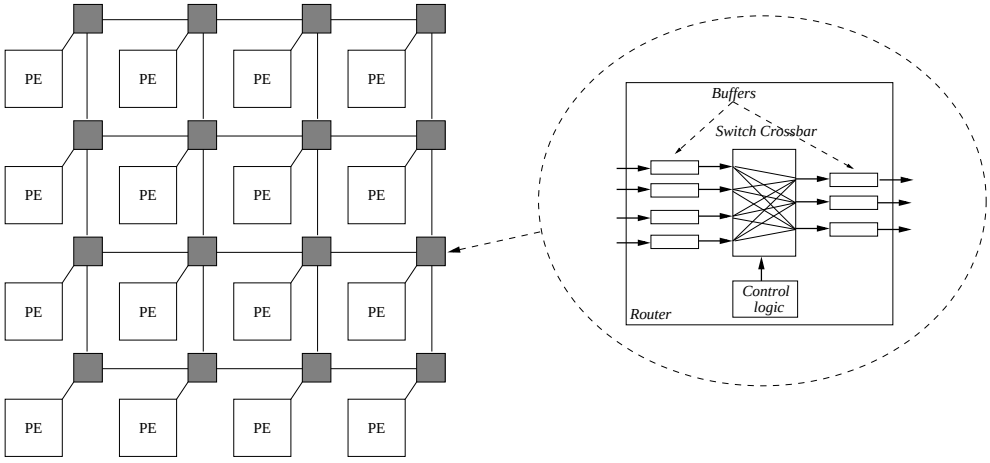


FIG. 4.1 – Exemple d'une structure de *NoC* de type mesh.

de plusieurs éléments de calcul *PE* (*Processing Element*) connectés les uns aux autres via les routeurs et les fils d'interconnexion d'une taille fixe. Un élément de calcul *PE* (souvent appelé un nœud), peut être soit un *processeur*, une fonction spécifique de calcul (*IP*), soit un bloc de mémorisation ou une combinaison de tous ces composants reliés ensemble. Un *PE* est généralement connecté à un routeur via un module d'interface *NI* (*Network Interface*) dont la fonction principale est la mise en paquets de toutes les données générées et à transmettre par un *PE* vers un autre *PE* à travers le réseau de communication. Un routeur est composé de *buffers d'entrée*,

de *sortie* ou d'*entrée-sortie* dont le rôle est d'accepter temporairement les paquets à transmettre de la part du *PE* qui lui est associé ou de ses routeurs voisins dans le but de faire transiter vers une direction des paquets que ne lui sont pas destinés. Pour cela, un routeur dispose généralement d'un module *crossbar* utilisé pour l'aiguillage des paquets des ports d'entrée vers les ports de sorties du routeur selon l'adresse de destination contenue souvent dans le premier paquet (*paquet d'en-tête - header packet*). Afin d'éviter les colisions des paquets arrivant simultanément sur les entrées d'un même routeur et ayant pour destination la même sortie du routeur, ce dernier intègre un module de contrôle ou d'arbitrage (*control logic*). En résumé, un message constitué de plusieurs paquets de données traverse un ou plusieurs routeurs à travers les canaux d'interconnexion avant d'arriver à un *PE* destinataire final. Cet exemple illustre le fonctionnement d'un réseau sur puce particulier de type maillé *2D*. Néanmoins, il illustre les principales différences d'une interconnexion de type réseau sur puce par rapport à une interconnexion de type bus partagé ou hiérarchique et la manière générale dont la plupart des architectures de type *NoC* fonctionnent. Par la suite, nous rappelons les principales topologies des réseaux sur puce et les techniques d'aiguillage couramment employées ainsi que les algorithmes de routage généralement utilisés dans ces réseaux sur puce.

4.2.1 Réseau sur puce selon le modèle de référence *OSI*

Un réseau sur puce est conçu sur un modèle de référence OSI (*Open Systems Interconnection*[8]) comportant 3 grandes couches :

- La couche physique,
- La couche architecture (liaison de données, réseau et transport) et
- La couche application (session, présentation et application).

La figure 4.2 détaille ces différentes couches. Les couches *basses* (les couches *physique* et *architecture*) sont nécessaires à l'acheminement des informations entre les entités concernées et dépendent du support physique. La couche *haute* (la couche *application*) est responsable du traitement de l'information relative à la gestion des échanges entre les modules d'un système ou entre les systèmes. Les couches 1 à 3 (voir figure 4.2) interviennent entre nœuds voisins et pas entre des nœuds d'extrémité pouvant être séparées et distant de plusieurs routeurs. Par contre, les couches 4 à 7 sont les couches qui interviennent entre des nœuds distants.

La couche *physique* est la couche de plus bas niveau d'un *NoC*. Cette couche assure la transmission des bits de données sur un canal de communication. Cette couche doit également garantir la parfaite transmission des données (un bit 1 envoyé doit bien être reçu comme bit valant 1). Généralement, les paquets de données traversent ces lignes de transmission en un cycle

[8]OSI - *Open Systems Interconnection - modèle de référence OSI* - Cadre de référence pour l'organisation des réseaux locaux, qui décompose la gestion du transfert des données en sept couches superposées réalisant une interface entre l'application locale et le matériel utilisé pour la transmission des données [GRD].

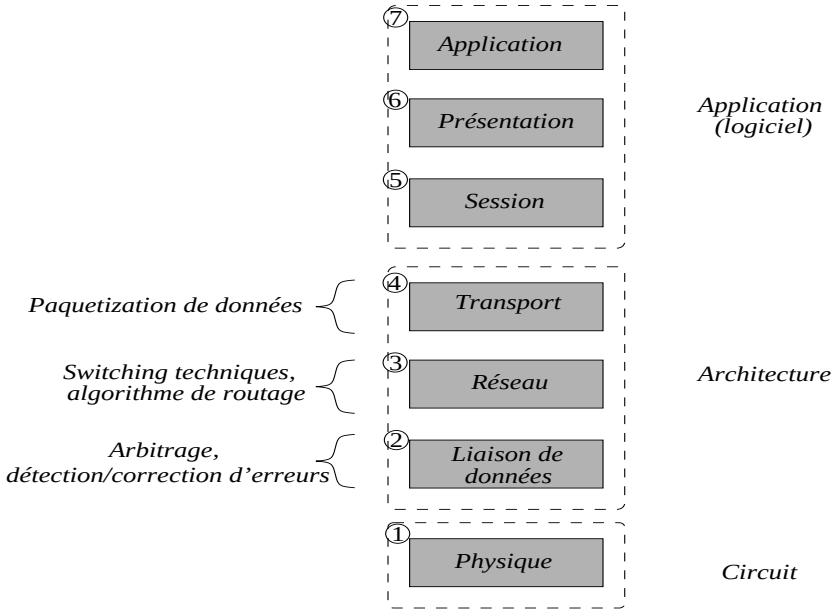


FIG. 4.2 – Réseau sur puce dans le contexte de modèle de référence *OSI*.

d'horloge, et sont stockés dans les buffers d'entrée ou de sortie des routeurs avant d'être définitivement acheminés vers d'autres routeurs. Pour minimiser les effets parasites (effets capacitifs, résistifs, propagation, etc.), ces liaisons physiques sont courtes. Ainsi, ces structures de communication s'affranchissent de tous les problèmes associés à la longueur des lignes et caractérisant les structures d'interconnexion de type bus partagé ou hiérarchique, dès les premières phases de leur conception de manière structurée et précise. La couche *physique* doit également « normaliser » les caractéristiques électriques et mécaniques d'un *NoC* (niveau de tensions des valeurs logiques, forme des connecteurs, de la topologie, etc.), ainsi que les caractéristiques fonctionnelles des circuits de données avec les procédures d'établissement, de maintien et de libération de ces circuits de données. L'unité d'information typique de cette couche est le bit.

La couche *architecture* gère la communication entre les routeurs. Chacune de ses sous-couches (voir figure 4.2) réalise un ensemble de fonctions bien définies. La sous-couche *liaison de données* (*data link*) assure le fractionnement des données d'entrée du nœud émetteur en trames, la transmission de ces trames en séquence et la gestion des trames d'acquittement renvoyées par le nœud cible. Cette sous-couche *liaison de données* doit être capable de renvoyer une trame lorsqu'elle détecte un problème sur la ligne de transmission. D'une manière générale, un rôle important de cette sous-couche est la détection et la correction d'erreurs intervenues sur la couche physique à travers l'emploi de techniques de détection et de correction d'erreurs. Cette sous-couche intègre également une fonction d'arbitrage pour éviter les collisions entre les paquets arrivant simultanément et l'engorgement du routeur récepteur. L'unité d'information de la sous-couche *liaison de données* est la trame de données.

La sous-couche *réseau* gère la manière dont les paquets sont aiguillés et acheminés vers d'autres routeurs. Les techniques d'aiguillage (*switching technique*) sont utilisées pour établir les connexions entre les routeurs, tandis que les algorithmes de routage sont nécessaires pour déterminer le « chemin » parmi l'ensemble des routeurs par lequel un paquet reçu va être acheminé pour atteindre sa destination finale. Ces deux aspects seront détaillés dans la section suivante. Enfin, le rôle principal de la sous-couche *transport* est de découper les messages en plus petites unités (paquets) du côté émetteur (nœud source) et de les réassembler pour reconstituer les messages initiaux au niveau du récepteur (nœud cible). La granularité des messages à transmettre présente un aspect important dans la conception des réseaux sur puce. En effet, un mauvais choix (généralement une taille de messages trop importante) peut affecter considérablement les performances du réseau (débit de transfert, consommation, ressources matérielles occupées, etc.)

La couche *application* est la couche de plus haut niveau. Ses sous-couches « logicielles » gèrent les communications entre les modules du système (*cœurs de processeurs, les IP*, etc.). Les protocoles de communication (*OCP-IP, VCI,* etc.) et les adaptateurs d'interfaçage au réseau (*wrap-*

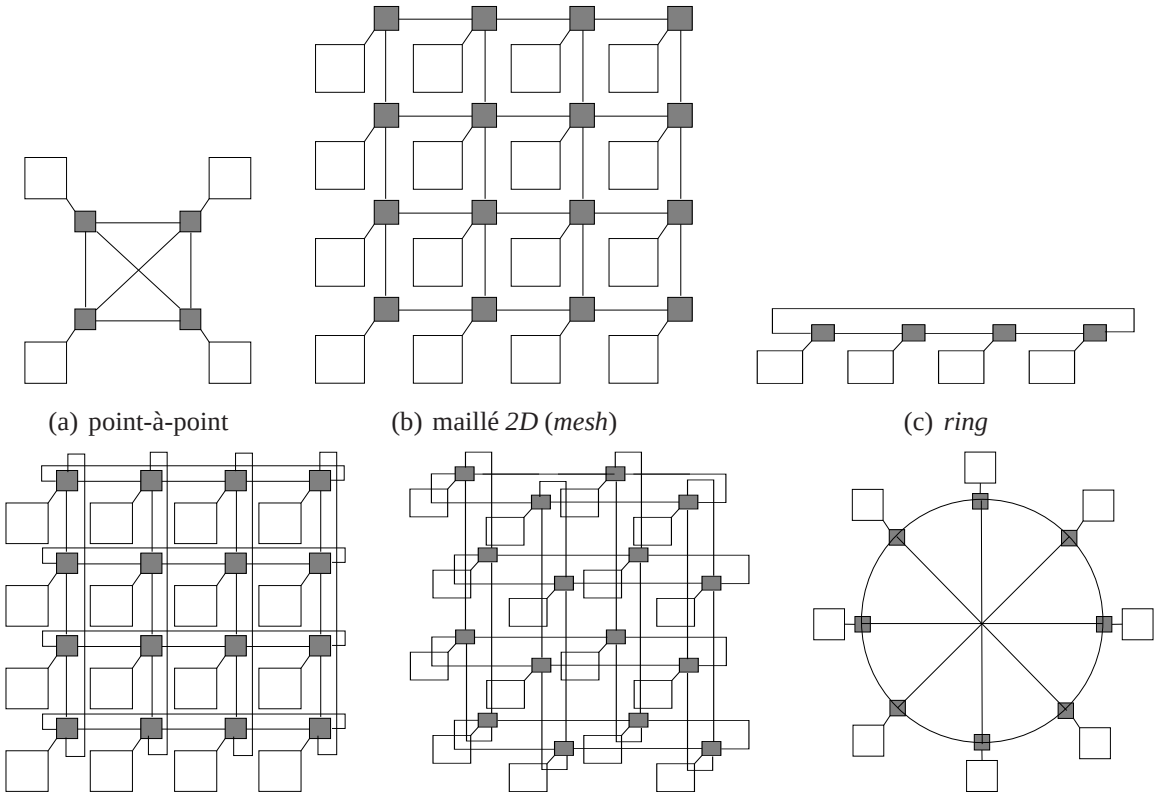


FIG. 4.3 – Exemples de topologies des réseaux sur puce de type *direct*.

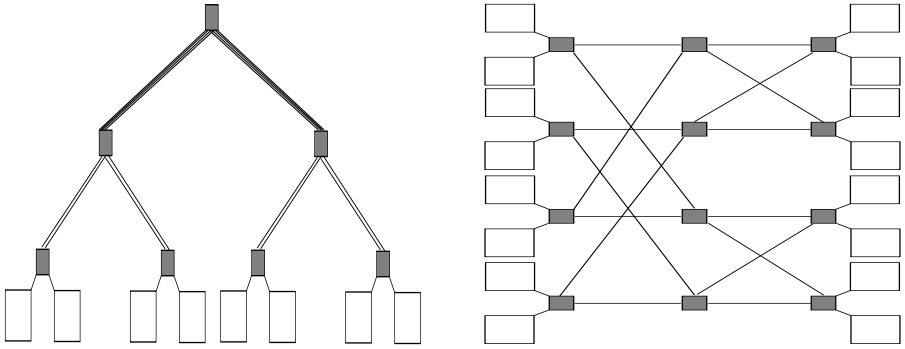
pers) sont définis et fixés à ce niveau du modèle. Les sous-couches de la couche *application* organisent et synchronisent les échanges entre tâches exécutées par des nœuds distants dans le réseau et réalisent également des liaisons entre des programmes d'application devant coopérer.

4.2.2 Topologies des réseaux sur puce

De nombreuses topologies des réseaux sur puce sont actuellement disponibles. Une topologie définit la manière dont les nœuds, les routeurs et les liaisons de données sont connectés les uns aux autres. Les topologies des réseaux sur puce peuvent être classées en 3 grands groupes [DYN02] : les réseaux *directs, indirects* et les réseaux *irréguliers*.

Dans les réseaux *directs*, chaque nœud a des liaisons point-à-point avec un ensemble de *noœuds voisins* [DYN02]. Les nœuds contiennent les blocs de calcul et / ou de mémorisation, les modules d'interfaçage avec le routeur (les blocs *NI - Network interface*). Dans ces réseaux *directs*, un routeur est connecté à d'autres routeurs voisins à travers les canaux de communication. Ces réseaux sont caractérisés par l'augmentation de la bande passante globale du réseau de manière proportionnelle au nombre de nœuds connectés. Souvent, lors de la conception de tels réseaux, des compromis doivent être faits entre le nombre de connectivités et le coût de conception. En effet, avec une meilleure « connectivité » du réseau résulte de meilleures performances du réseau au détriment des fortes consommation d'énergie et de ressources d'implantation matérielles. Les réseaux sur puce de type *direct* plus couramment utilisés sont présentés dans la figure 4.3. La plupart de topologie de type *direct* ont une structure orthogonale, où les nœuds sont disposés de telle manière que chaque canal produit un déplacement dans une seule direction. Le routage dans ce type de réseaux est relativement simple et peut être matériellement facilement implanté. Le réseaux sur puce *directs* le plus souvent utilisés sont les réseaux maillés *2D mesh, torus, folded torus* et *octagon* [MBV$^+$02, DT01, KND02, KNDR01]. La topologie matricielle *mesh* semble être la topologie la plus répandue dans le domaine des réseaux sur puce. Ceci est dû à la même longueur de tous les canaux dans le réseau et à la scalabilité de sa structure qui croît linéairement avec le nombre de nœuds connectés.

Dans les réseaux sur puce de type *indirect*, chaque nœud est connecté à un routeur qui est connecté à d'autres routeurs par des liaisons point-à-point. Le réseau sur puce de type *indirect* le plus simple est le réseau *crossbar*, où chaque élément de calcul *PE* est connecté à d'autres *PE* via un seul routeur. Cependant, le principal inconvénient de ce type de réseau *indirect crossbar* est son manque d'extensibilité et son coût relativement important en terme de connectivité proportionnel au nombre de *PE*. C'est la raison pour laquelle, ce réseau est rarement utilisé. La figure 4.4 illustre les réseaux sur puce de type *indirect* les plus souvent utilisés. Dans la topologie de type *tree*, structure de type « arbre », les éléments de calcul sont connectés uniquement aux extrémités de la structure topologique que l'on nomme « feuilles » de l'arbre. Ce type de structure présente des gros inconvénients en terme d'embouteillage de trafic des paquets de



(a) *fat tree* (b) *butterfly*

FIG. 4.4 – Exemples de topologies des réseaux sur puce de type *indirect*.

données. En effet, le routeur le plus distant des éléments de calcul, que l'on appelle souvent le routeur *root*, est généralement surchargé de trafic et limite donc fortement les performances de tel type de réseau en comparaison à une structure *crossbar* mais plus coûteuse. Afin de réduire les encombrements et les embouteillages qui puissent se produire dans ce type de réseau, on utilise une topologie de type *fat tree*, où le nombre des canaux entre les routeurs adjacents augmente en se rapprochant du routeur *root*. Une variation de *fat tree* topologie est la structure *papillon* (*butterfly*). Ces solutions permettent d'améliorer le débit de ces réseaux tout en restant moins coûteux en terme de connectivité vis-à-vis d'une structure *crossbar*. De nombreux autres exemples des *NoC indirects* peuvent être trouvés dans la littérature [PGIS03, GG00, AG03]

La dernière grande famille de topologies des réseaux sur puces sont les topologies *irrégulières*. Ce sont des structures d'interconnexion qui représentent généralement une combinaison de bus partagé ou hiérarchique avec des topologies de type *direct* et *indirects*. Ce genre de topologie est souvent associé à des applications spécifiques. Un exemple d'une topologie *irrégulière* est présenté dans la figure 4.5. Dans cet exemple on trouve une topologie de type *mesh* où les routeurs non utilisés sont simplement enlevés du réseau. Parmi ce type de topologie de réseau sur puce *irréguliers* nous pouvons citer les réseaux *Xpipes* [DBG+03] et Æthereal [GDR05].

4.2.3 Techniques d'aiguillage - *Switching techniques*

Les techniques d'aiguillage (*switching techniques*) déterminent la manière dont les paquets de données traversent les routeurs à travers le réseau [DYN02]. Ces techniques définissent également la granularité du transfert de données. Les nœuds (*PE*) génèrent les messages que l'on fractionne en plusieurs *paquets* de donnée possibles. Chaque *paquet* est ensuite partitionné en plus petites unités de contrôle que l'on appelle *flits (flow control unit)*. Un *flit* représente un paquet élémentaire qui peut être également subdivisé en *phits* (*physical units*). Généralement, un *phit* est une unité de données qui peut être transférée à travers un canal de données entre deux

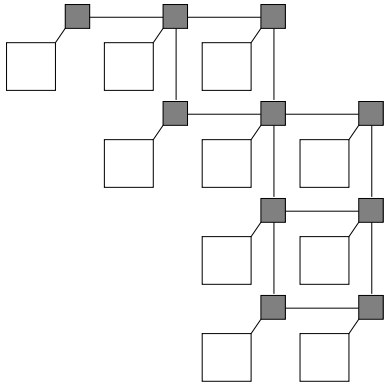


FIG. 4.5 – Exemples d'une topologie *irrégulière* des réseaux sur puce basée sur la structure de type *mesh 2D*.

routeurs en un seul cycle d'horloge. La taille d'un *phit* est souvent la taille du canal de données en bits reliant les routeurs du réseau. La figure 4.6 illustre la structuration et la décomposition d'un message en *paquets*, *flits* et *phits*. Les différentes architectures *NoC* utilisent différents types de *phits*, de *flits*, de *paquets* et de messages. En effet, le choix judicieux de la taille de ces unités a un impact considérable sur les performances, le coût et la consommation lors de la conception d'une architecture *NoC*.

Les deux principales techniques d'aiguillage dans un *NoC* sont les techniques dites de « *circuit switching* » et « *packet switching* ». La technique d'aiguillage *circuit switching* repose sur deux étapes. Dans une première étape, un chemin entre la source et la destination est établi. C'est le rôle de l'*entête (header) flit* de « réserver » au cours de son cheminement ce chemin de transmission. Si le *header flit* arrive jusqu'à sa destination sans avoir des collisions avec d'autres demandes en cours, un acquittement positif est envoyé à la source. Une fois que cet acquittement est reçu, la deuxième étape de transmission débute. Dans cette seconde étape, les messages sont entièrement transmis vers la cible selon le cheminement réservé au cours de l'étape précédente. Par contre, si le *header flit* n'arrive pas jusqu'à la destination dû à d'autres demandes de transmission en cours, un acquittement négatif est envoyé vers la source pour lui indiquer que

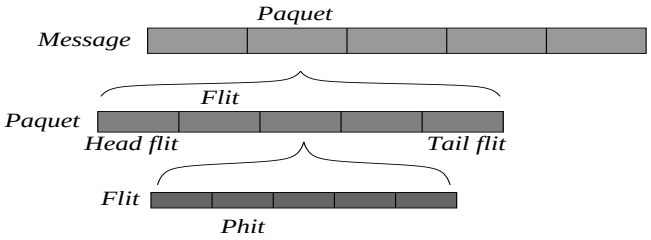


FIG. 4.6 – Structure de message, paquets, *flits* et *phits*.

la transmission ne peut s'effectuer à cet instant et donc n'aura pas lieu. Les principaux avantages de cette technique d'aiguillage sont une large bande passante et une faible latence entre les sources et destinations lorsque le chemin entre elles est établi. Un exemple de *NoC* utilisant cette technique d'aiguillage est l'architecture *NoC SOCBUS* [WSRS05]. Le principal inconvénient de ce type de technique d'aiguillage est qu'il n'est pas très « extensible ». En effet, en augmentant la taille du réseau, la durée d'occupation des canaux de transmission devient trop importante. En particulier au cours de la phase de réservation du chemin de transmission. Des solutions pour y remédier à cet inconvénient ont été proposées. Une solution consiste à extension par des canaux de transmission supplémentaires en parallèle à ceux de base. Ces canaux supplémentaires sont appelés *canaux virtuels* (*virtual channels*). L'architecture *NoC* utilisant ce type de technique avec un seul *buffer* par un canal virtuel est l'architecture *NoC MANGO* [Bje05]. Une autre variante de la technique *circuit switching* à canaux virtuels est l'utilisation d'un seul *buffer* par un canal physique [GDR05, KJS+02].

Contrairement à la technique d'aiguillage *packet switching*, dans la technique *circuit switching* il n'y a pas de réservation de chemin entre un nœud source et un nœud destinataire du réseau sur puce. Les paquets, transmis dans le réseau par une source, établissent leur chemin vers la cible au fur et à mesure de transmission à travers le réseau. Cette technique ne nécessite pas une durée d'établissement de connexion entre les communicants (*start-up time*). Cependant, elle est caractérisée par un temps de latence plus élevé par rapport à une technique d'aiguillage *circuit switching*. En effet, les situations d'encombrements de paquets dans un routeur arrivent plus fréquemment et augmente donc la latence de transmission des paquets.

Nous distinguons trois types de technique d'aiguillage de type *packets switching* : *store-and-forward (**SAF**)* , *virtual cut trough (**VCT**)* et *wormhole switching (**WS**)* . Dans la technique ***SAF***, un paquet est transmis d'un routeur à autre (routeur voisin) seulement si ce dernier a la capacité de stocker entièrement un paquet à transmettre [KJS+02]. Dans ce type d'aiguillage, la taille des *buffers* des routeurs utilisés doit être au moins égale à la taille d'un paquet de données. Il s'agit alors de définir un dimensionnement judicieux des tailles des *buffers routeurs*. En effet, l'utilisation des *buffers* de grande taille présente un grand inconvénient de cette technique en terme de consommation de ressources matérielles. La technique ***VCT*** est caractérisée par un temps de latence plus faible que la technique ***SAF***. Dans cette technique, un paquet peut ne pas être transmis dans son intégrité d'un routeur à un autre. La transmission des paquets d'un routeur à un autre repose sur la capacité de stockage par les routeurs des *flits de paquets*. Ainsi, dès qu'un routeur voisin peut stocker un *flit*, ce dernier est transmis vers ce routeur. Les autres *flits* suivent le premier sans délais supplémentaires. Par contre, dans le cas où le premier *flit* ne peut pas être transféré vers le routeur voisin (espace de stockage du buffer routeur voisin insuffisant), le *flit* demeure en attente avec d'autres flits dans le routeur initial, le temps de libérer l'espace buffer du routeur voisin. Comme pour la technique d'aiguillage ***SAF***, la technique ***VCT***

a également l'inconvénient de nécessité des routeurs utilisant des *buffers* de grande taille.

La technique d'aiguillage la plus couramment utilisée dans les *NoC* est la technique *wormhole switching (**WS**)*. Pour cette technique d'aiguillage, les besoins en terme de taille de buffers des routeurs se réduisent au stockage d'un seul *flit*. Le *flit* est transféré depuis un routeur courant vers un routeur voisin si ce dernier dispose d'un espace suffisant pour stocker le *flit*. Dans le cas contraire, ce dernier demeure dans le routeur courant. Ainsi, un paquet peut être distribué à travers ses *flits* dans plusieurs routeurs avant d'être finalement complètement acheminé vers la cible. Étant cette distribution autorisée dans cette technique, les situations d'embouteillage ou de blocage (*deadlock*) apparaissent plus fréquemment par rapport aux techniques ***SAF*** et ***VCT***. Parmi les architectures *NoC* utilisant la technique ***WS***, nous pouvons citer l'architecture *NoC SPIN* [AG03]. D'autres architectures *NoC* reposent sur une combinaison des techniques ***WS*** et ***VCF*** [Bje05, GDR05].

4.2.4 Algorithme de routage

Les algorithmes de routage déterminent le chemin dans le réseau que sera pris par un paquet entre un nœud source et un nœud destinataire. Le choix d'un algorithme de routage dépend de plusieurs critères tels que la basse consommation de routage, la complexité logique de routage (plus ou moins simple), les meilleures performances de transmission, la tolérance aux fautes du réseau, la meilleure distribution de trafic, etc. Les algorithmes de routage peuvent être classés en plusieurs catégories telles que le routage *statique* ou *dynamique*, *distribué* ou *source*, *minimal* ou *non-minimal*, *tolérant aux fautes* ou non, etc [DYN02].

Si on considère la classification des algorithmes de routage statique ou dynamique, on caractérise les algorithmes de routage statiques (*static routing*) comme les algorithmes définissant des chemins fixes et identiques pour transférer les paquets de données d'un nœud source vers une nœud destinataire au sein d'un réseau. Ce type de routage ne prend pas en compte les conditions du réseau telles que sa charge actuelle (*network load*), l'occupation des canaux de transmission, la distribution du trafic, etc. Ce type de routage est simple à mettre en œuvre et nécessite de faibles ressources logiques. Parmi les exemples des algorithmes de routage statiques nous trouvons les algorithmes de routage ***XY***, ***turn model*** (*west-first, north-last, negative-first* [KN06], etc.). Par contre, dans les algorithmes de routage dynamiques (*dynamic routing*), toutes les décisions d'acheminement des paquets de données sont prises en considérant les conditions en cours de la transmission dans le réseau. Dans ce type de routage, le chemin entre un nœud source et un nœud destinataire évolue au cours du temps en fonction des conditions du réseau. Généralement, le cheminement des paquets entre deux nœuds du réseau est rarement le même. Ces techniques de routage nécessitent davantage de ressources matérielles (logique plus complexe) pour leur réalisation en comparaison à la réalisation de technique de routage statique. Parmi les exemples d'algorithmes de routage dynamiques nous pouvons citer les algorithmes

fully-adaptive [Dal04], ***odd-even*** [HM04], ***congestion look-ahead*** [KPT+05], etc.

Les algorithmes de routage statique et dynamique peuvent être également classés selon la façon dont les informations de routage sont stockées et l'endroit où les décisions de routage sont prises (au niveau routeur local ou routeur source). Dans l'algorithme de routage *distribué*, les décisions de routage s'effectuent au niveau de chaque routeur. Lorsqu'un routeur reçoit un paquet, il consulte soit ses tables de routage, soit sa fonction de routage pour calculer la destination du paquet. Ceci implique que pour chaque routeur du réseau, les fonctions ou tables de routage doivent être implantées. Dans le cas d'un algorithme de routage *source*, avant l'envoi d'un paquet à un nœud destinataire présumé, le nœud source intègre la « feuille de route » du paquet à transmettre. Cette dernière permet à tous les routeurs dans lesquels le paquets transite de l'acheminer selon son plan de route indiqué. L'inconvénient de ce type d'algorithme de routage est la nécessité d'intégrer des informations additionnelles au sein des paquets de données augmentant ainsi leurs tailles.

Une autre façon de distinguer les algorithmes de routage est leur classification selon la distance parcourue par un paquet à travers le réseau entre son nœud source et nœud destinataire. On distingue ainsi les algorithmes de routage *minimaux* et *non-minimaux*. Dans le cas d'un algorithme *minimal*, le chemin de routage d'un paquet entre deux nœuds est le chemin le plus court. L'algorithme de routage *XY* est un exemple de routage minimal. Par contre, les algorithmes de routage *non-minimaux* n'ont pas de contraintes de distance pour acheminer des paquets au sein d'un réseau. Ainsi, le nombre de chemins alternatifs entre deux nœuds est plus élevé. L'avantage de tels algorithmes est qu'ils permettent d'éviter des situations d'embouteillages tout en distribuant de façon plus homogène le trafic dans le réseau. Cependant, ce type de routage nécessite généralement des ressources additionnelles pour son implantation comparé à des algorithmes *minimaux*.

Enfin, les algorithmes de routage peuvent être considérés comme tolérants aux fautes ou non. Un algorithme de routage est tolérant aux fautes, si malgré la défaillance d'un des routeurs du réseau, l'acheminement des paquets s'effectue néanmoins au sein du réseau entre deux nœuds. La majorité des algorithmes de routage déjà mentionnés ne sont pas tolérants aux fautes. Généralement, les défaillances éventuelles d'un des routeurs du réseau, changent la structure et la topologie du réseau et les rendent irrégulières, compliquant ainsi et rendant impossible le routage de paquets. Plusieurs exemples d'algorithmes de routage *tolérants aux fautes* peuvent être trouvés dans la littérature [GN93, Wu03, BC95, CC98b].

4.3 Réseaux sur puce reconfigurables

4.3.1 Introduction

Une classification des *NoC* en fonction des changements possibles de leurs paramètres au cours de fonctionnement peut être établie comme soit des *NoC statiques* ou des *NoC dynamiques*. Les réseaux sur puce dont les topologies, le placement d'éléments de calcul *PE*, les fonctions de routage et d'aiguillage et les routeurs sont fixes et inchangeables dans le temps sont considérés comme *statiques*. Dans cette classe de *NoC* nous trouvons les architectures *NoC* mentionnées dans la section précédente [GG00, AG03, KJS+02, MBV+02, KND02, PGIS03]. Dans le cas où, les réseaux sur puce dont les paramètres mentionnés ci-dessous peuvent changer au cours du fonctionnement sont considérés comme *réseaux dynamiques*. Les *NoC* dynamiques sont souvent confondus et associés avec les *NoC reconfigurables*. Par la suite, nous donnons un résumé des travaux existants dans la littérature sur les *NoC reconfigurables* et nous proposons une définition précise de la notion de « *reconfigurable* » associée aux *NoC*.

Une méthodologie de développement des processus dynamiques reconfigurables dans le cadre des *NoC* a été proposée dans la littérature [LPD03]. Ces travaux définissent la notion de *reconfigurable* ou *reconfiguration* au sein d'une structure *NoC* comme le changement de sa fonction de routage au cours du fonctionnement tout en conservant le réseau fonctionnel et opérationnel pendant le processus de reconfiguration. De même, au cours du fonctionnement d'un *NoC*, ses spécificités de conception initiales tels que la topologie du réseau ou l'algorithme de routage, peuvent être modifiées au cours du fonctionnement du *NoC* de façon soit involontairement suite à des défaillances des parties du réseau, soit volontairement à cause de suppressions ou d'ajouts de nouveaux modules au sein du réseau. En pratique, ces situations de changement mènent souvent vers des blocages des messages ou paquets de données dans le réseau. En général, pour y remédier, l'algorithme de routage utilisé doit être modifiable dans le but de rétablir les connexions entre tous les nœuds ou routeurs du réseau associées à des modules de calcul PE. Une approche théorique dans ce sens, permettant à un *NoC*, de continuer de fonctionner sans interruptions tout en assurant les contraintes définies initialement lors de sa conception et une *qualité de service* (*QoS - Quality of Service*[9]), a été également proposée dans [LPD03]. L'approche proposée considère que la topologie du réseau reste inchangée au cours de ce processus de reconfiguration.

Une autre approche, basée également sur l'hypothèse d'une conservation de la topologie du réseau au cours d'un processus de reconfiguration est détaillée dans [HG07]. Dans cette approche, la reconfigurabilité du *NoC* est assurée par des modules appelés *maître de configuration* (*configuration master*), correspondant à des processeurs ou CPU typiques. Ces modules peuvent

[9] *QoS* - Aptitude d'un service à répondre adéquatement à des exigences, exprimées ou implicites, qui visent à satisfaire ses usagers. [GRD]

accéder, au cours du fonctionnement et à travers l'infrastructure de communication existante du réseau, aux interfaces de réseau *NI* (*Network Interface*) dans le but de modifier ou de mettre en œuvre la « (re)configuration » du réseau.

Une autre approche reposant sur un outil de modélisation, de simulation et d'implantation de *NoC* reconfigurable a été également proposée dans [CSV05]. Plus précisément, à partir d'une description haut niveau du réseau et d'un environnement de simulation variable permettant d'appliquer des stimulis correspondant à divers applications, une configuration optimale du réseau peut être déterminée et transcrite en langage *VHDL* pour implantation. Cette approche permet la conception de *NoC* (re)configurables et statiques, car les critères et paramètres pour une application donnée ne sont pas imposés par l'environnement de simulation durant l'exécution mais dans la phase de modélisation du réseau *NoC*.

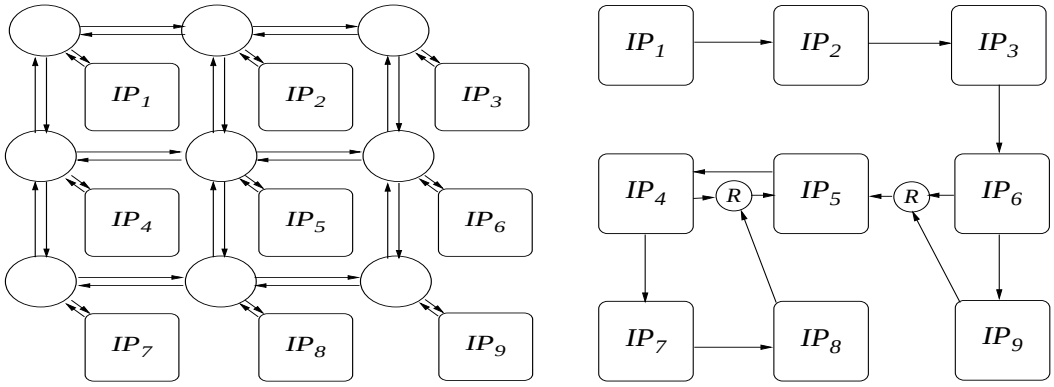
Une architecture *NoC* reconfigurable dynamiquement et destinée pour les *MPSoC*[10] reconfigurables est présentée dans [AEK06]. Cette architecture permet la (re)configuration dynamique (au cours du temps d'exécution du réseau), de certains paramètres du réseau tels que l'algorithme de routage, la technique d'aiguillage, la taille des messages à transmettre. Dans cette approche, toutes les ressources du réseau utilisées doivent être allouées statiquement dans la phase de conception. Les *PE* sont également reconfigurables dynamiquement. L'idée maîtresse de cette approche est que différents *PE* connectés à un *NoC* ont des différents besoins en communication et que donc à chaque type de *PE*, il faut adapter et (re)configurer le réseau *NoC* associé. Ce type de *NoC* assure la reconfigurabilité de ses nœuds grâce à un noyau (*kernel*) sous forme d'un modèle de référence *OSI* modifié et nommé *SNS* (*Smart Network Stack*). En fonction des données à transférer, le *SNS* décide quel type d'algorithme de routage, de technique d'aiguillage et des tailles de données à choisir. Ces informations sont placées dans l'entête des paquets considérés, qui seront analysés par les routeurs du réseau. Ainsi, par exemple, dans le cas où des besoins importants en communication sont sollicités, une taille de paquet plus grande sera choisie, tandis qu'une technique d'aiguillage initiale par défaut de type *packet switchning*, sera remplacée par la technique de type *circuit switching*.

Une approche similaire, destinée également pour la conception de *MPSoC* est présentée dans [KHHC07]. La principale différence est le nombre de paramètres du réseau (*NoC* de type Æthereal [GDR05]) pouvant être reconfigurés au cours de son temps d'exécution et la possibilité de reconfiguration des *PE* (*Silicon Hives processing cores* [SH09]) associés au réseau.

Une approche, nommé *ReNoC* pour *Reconfigurable NoC*, permettant la reconfigurabilité de la topologie d'un réseau *NoC* par combinaison des techniques d'aiguillage *circuit* et *packet switching* est proposée dans [SS08]. Dans ce type de réseau ReNoC, deux types de couches de

[10] *Multiprocessor System-on-Chip* - Système sur puce multiprocesseur. Circuit intégré sur une seule puce, qui comporte plus d'un processeur dans sa structure. Certains auteurs emploient parfois les termes *système multicœur* et *architecture multicœur* pour décrire ce type de composant électronique [GRD].

topologie sont distingués : une couche *physique* correspondant à la couche physique réelle du réseau et une couche *logique* correspondant au réseau perçu par l'application et qu'elle utilise pour ses besoins en communication. La couche *logique* est configurée dans la phase d'initialisation en fonction des demandes en communication. Pour des besoins en communication élevés et pour une consommation plus faible, la technique d'aiguillage *circuit switching* est privilégiée par le *ReNoC* par rapport à la technique d'aiguillage *packet switching* . Ainsi, la principale caractéristique de l'approche *ReNoC* permettant une reconfiguration topologique est que les routeurs d'un tel réseau sont composés d'une association d'un routeur classique et d'un *aiguilleur topologique* (*topology switch*) permettant le passage d'une technique d'aiguillage à une autre. Par conséquent, la topologie du réseau peut être perçue par une application comme une structure de réseau irrégulière alors que le *NoC* physique est physiquement régulier. La figure 4.7 illustre un réseau *ReNoC*.



(a) topologie physique (b) topologie logique

FIG. 4.7 – Illustration de reconfiguration de topologie dans un réseau *ReNoC*.

Une approche *NoC*, nommée *NOVA* et destinée pour les systèmes reconfigurables à base de réseau sur puce, est présentée dans [MVJS07]. Le réseau *NOVA* combine les aspects de communication de type *NoC* classique à topologie *star* et de type *point-à-point*. Le réseau *NOVA* est composé de *cluster*[11] correspondant à des zones au sein d'un circuit *FPGA*. Ces zones sont, soit reconfigurables pour implanter des *PE*, soit fixes pour implanter un routeur du réseau. Chaque *cluster* est organisé comme une matrice de taille 3×3, contenant un seul routeur *tile switch* à la position centrale de la matrice et des *PE* sur les autres positions de la matrice. Tous les *PE* voisins (au sein d'une même matrice) peuvent communiquer directement via des liaisons point-à-point, tandis que la communication entre des *PE* non-voisins (PE dans différents *clusters*) s'effectue via des routeurs que l'on nomme *cluster switch*. La figure 4.8

[11] *cluster* - grappe. Ensemble d'appareils de même type (terminaux, ordinateurs, etc.) rattachés à une même unité de contrôle. [GRD]

donne une illustration d'un réseau *NOVA* [MVJS07]. Aucune information n'est communiquée dans le cas d'une connexion d'un nombre de *clusters* supérieur à 8.

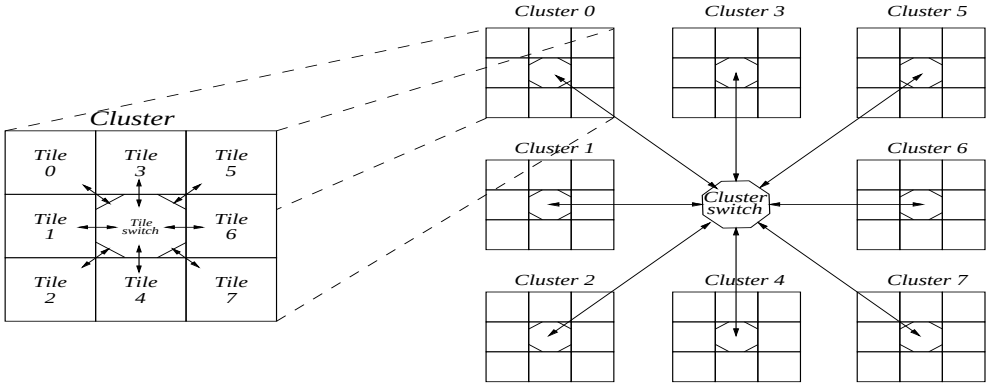


FIG. 4.8 – Structure d'un réseau *NoC* de type *NOVA*.

Dans les systèmes reconfigurables dynamiquement à base de *FPGA*, les contraintes tels que le placement de *PE* et les performances nécessaires pour une application donnée sont variables au cours du fonctionnement de ces systèmes. C'est pourquoi, les systèmes reconfigurables qui reposent sur une structure de communication *NoC* nécessitent des réseaux dont la topologie est modifiable. C'est dans ce cadre que l'architecture *CoNoChi*[12] a été proposé [PKA06]. Cette architecture *NoC* permet un changement et une adaptation dynamique de la structure du réseau en fonction du nombre de *PE* associés au réseau à un instant donné. La solution architecturale proposée consiste à adapter le nombre de routeurs au nombre de *PE* placés statiquement ou dynamiquement sur un circuit *FPGA* en utilisant le procédé de reconfiguration dynamique partielle des circuits *FPGA* utilisés (technologie *Virtex* de *Xilinx*). En adaptant le réseau de communication pour un nombre de *PE* donné, le nombre de routeurs nécessaires reste toujours minimal. Ceci réduit considérablement la latence moyenne du réseau et les ressources nécessaires pour son implantation. L'approche *CoNoChi* a été validé expérimentalement sur une plateforme *Xilinx Virtex 2 Pro* [PKA06].

Une approche similaire aux structures *CoNoChi* et *ReNoC* est présentée dans [RAS+08]. L'approche repose sur un circuit *FPGA* divisé en deux parties. Une première partie statique (non-reconfigurable), contenant des *PE* (cœurs logiciels ou matériels de processeurs *soft-core* - *MicroBlaze* ou / et *hard-core* - *PowerPC*), des mémoires et des interfaces réseaux permettant de connecter les *PE* entre eux via un réseau *NoC*. Une seconde partie reconfigurable, contenant un réseau *NoC* et des éventuelles connexions de type *point-à-point*. Cette deuxième partie peut également contenir des *PE*. Le routage au sein du réseau est géré par des tables de routage, stockées dans les *BRAM* (mémoire sur puce du *FPGA*), dont le contenu peut être changé dy-

[12] *CoNoChi* - Configurable Network-on-Chip. Le réseau sur puce configurable.

namiquement au cours du fonctionnement. Cette reconfiguration dynamique est assurée par un contrôleur de reconfiguration connecté à une mémoire externe contenant les fichiers de configuration (*bitstream*[13]). Ce contrôleur peut, en cours d'exécution, reconfigurer les parties reconfigurables du *FPGA* en chargeant les bits de configuration stockés dans la mémoire externe via l'interface *ICAP*[14] vers les zones reconfigurables. Ainsi, grâce à une reconfiguration partielle du circuit, le réseau peut être adapté aux besoins d'une application donnée (voir figure 4.9 [RAS+08]). Les résultats d'implantation et expérimentaux de cette approche montrent que des gains considérables en terme de consommation et de temps de latence des transmissions de paquets de données peuvent être obtenus. Cependant, cette approche nécessite des durées des phases de reconfiguration des zones reconfigurables relativement élevées de l'ordre de dizaines, voir centaines de *ms* pour les zones reconfigurables de taille de plusieurs *frames* [RAS+08].

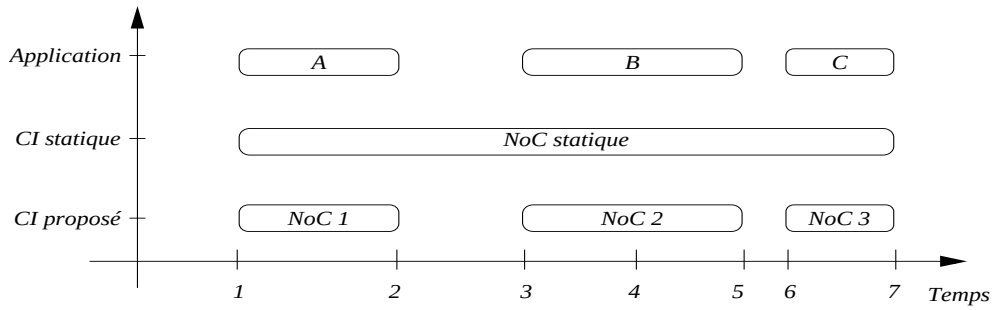


FIG. 4.9 – Illustration de changement de topologie de *NoC* dans l'approche [RAS+08] pour différentes applications.

L'approche *DyNoC* (*Dynamic NoC*) permet un placement dynamique de modules (*PE*) au cours du temps d'exécution d'un réseau *NoC* [BAM+05, MBAT05]. Dans cette approche, un réseau *NoC* classique est initialement créé, où à chaque routeur du réseau un *PE* est associé occupant une surface (*tile*). Au cours du fonctionnement, de nouveaux *PE* peuvent être placés ou intégrés au sein du réseau. Ces nouveaux *PE* peuvent être de taille supérieure que la taille initiale destinée pour un *PE* (*tile*). Dans ce cas, le nouveau *PE* dynamiquement placé dans le réseau couvre une surface du réseau correspondant à certains *PE* et routeurs placés initialement à cet endroit du réseau. Les routeurs qui sont couverts par un *PE* placé dynamiquement seront désactivés et ne pourront plus être utilisés pour le routage de messages au sein du réseau. Par contre, ces routeurs peuvent être utilisés comme de la logique supplémentaire pour le nouveau *PE*. Les routeurs désactivés seront réactivés de nouveau, une fois le *PE* placé dynamiquement a accompli sa tâche et le système revient à la configuration initiale. Ce processus d'activation et de désactivation de routeurs rend le réseau dynamique. En plaçant un module dynamiquement dans

[13] *bitstream* - Séquence de données constituée de données strictement numériques [GRD].
[14] *ICAP - Internal Configuration Access Port*

le réseau, ce dernier devient d'une structure hétérogène et nécessite un algorithme de routage adapté pour acheminement des messages vers toutes les destinations du réseau sans blocages et / ou difficultés. Le réseau *DyNoC* repose sur un algorithme de routage *XY* modifié et adapté aux placements dynamiques de *PE* au sein du réseau (Algorithme *S-XY - Surrounding XY* - le *XY* « contournant »). Parmi les principaux inconvénients de *DyNoC*, nous citons les ressources importantes nécessaires pour son implantation et un taux *PE* / NE[15] assez élevé.

La plupart des approches *NoC* reconfigurables présentées dans la littérature définissent la reconfigurabilité des *NoC* comme un moyen de modifier leurs configurations au cours de leur fonctionnement. Ces configurations de réseau correspondent des changements de paramètres de réseau tels que l'algorithme de routage, les techniques d'aiguillage, la *Qualité de Service - QoS*, adaptés à une application donnée et pouvant être changés au cours du fonctionnement du réseau. Cependant, dans le cadre de la conception de système reconfigurable à base de technologie *FPGA*, nous considérons une architecture reconfigurable comme une architecture permettant une modification de sa structure architecturale ou de traitement de données à tout moment (voir le chapitre 2). Dans ce contexte, nous considérons uniquement les travaux de la littérature relatifs aux approches de conception de réseau de type ou similaires aux structures *ReNoC* et *CoNoChi* [PKA06, SS08, RAS+08] comme des *NoC reconfigurables*. En effet, hormis ces approches, les réseaux présentés dans la littérature possèdent des structures topologiques fixes au cours de leur fonctionnement. Nous les classifions donc par la suite comme des *NoC paramétrables* aux côtés de *NoC reconfigurables*. Nous notons également, que ces *NoC reconfigurables* distinguent le réseau utilisé pour la communication des *PE*, possédant une structure homogène et composée uniquement de routeurs, au reste du système (voir la figure 4.10). Dans ces approches, pour une application donnée, soit les *PE* sont reconfigurés en d'autres *PE*, soit on reconfigure le réseau *NoC* pour l'adapter à l'ensemble de *PE* utilisés pour l'application considérée.

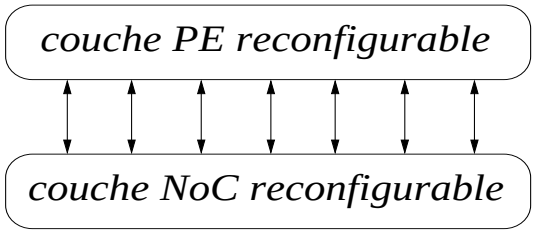


FIG. 4.10 – Illustration de deux couches distinctes (*NoC* et *PE*) dans les approches *NoC* reconfigurables.

Dans le cadre des systèmes reconfigurables à base de *FPGA*, où des *PE* peuvent apparaître, disparaître ou être délocalisés au sein du réseau au cours du temps, il apparaît plus « natu-

[15] PE / NE - le taux *Processing Element* / *Network Element*

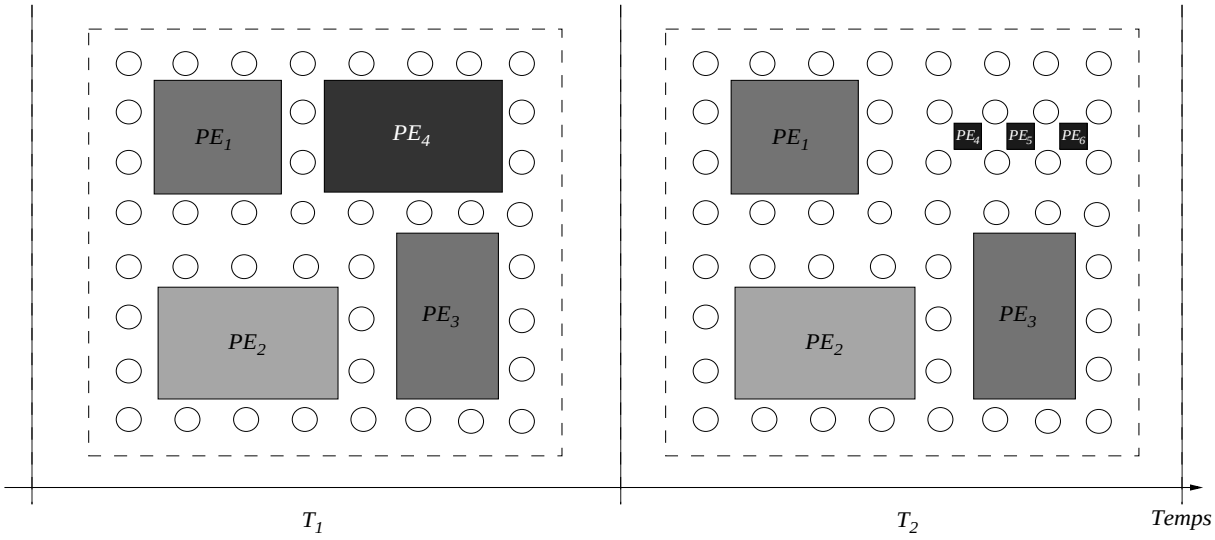


FIG. 4.11 – L'approche *DyNoC* [BAM+05] : Illustration de l'évolution d'un réseau dynamique reconfigurable dans le temps.

rel », d'un point de vue reconfiguration, que le réseau de communication utilisé soit « omniprésent » et que les *PE* apparaissent et disparaissent dans le réseau de manière dynamique tout en adaptant sa topologie et sa façon d'acheminer les messages dans le réseau. L'approche *DyNoC* en est un exemple (voir la figure 4.11).

C'est dans ce cadre de considérations que ce situent nos travaux de développement d'un réseau de communication sur puce pour la conception d'un système reconfigurable auto-organisé. La suite de ce chapitre présente en détails deux approches *NoC* de ce type en vue de développer un tel système auto-organisé.

4.3.2 Architecture *CuNoC*

Nous proposons une nouvelle approche de réseaux sur puce destinée à la conception de systèmes sur puce reconfigurables à base de *FPGA* [JTWB07a, JTWB07b, JTBW09]. Cette architecture réseau que nous nommons *CuNoC* permet la communication entre des modules placés dynamiquement sur puce au cours de leur fonctionnement. Il correspond à un réseau sur puce de type *packet switching* composé de routeurs intelligents appelés *CU* (*Communication Unit* - Unité de communication). Le rôle principal de ces routeurs est le routage des paquets, à partir de *PE* sources vers des *PE* destinations selon des adresses contenues dans les paquets de données à transmettre, au sein du réseau dont la structure peut évoluer au cours du temps. Un *CU* est caractérisé à la fois par une technique d'arbitrage basée sur un principe de priorité à droite, par un nouvel algorithme de routage, par sa spécificité architecturale et la manière dont il est connecté avec les éléments de calcul *PE* (*Processing elements*).

4.3.2.1 Structure de message dans le *CuNoC*

Les éléments de calcul associés au réseau *CuNoC* communiquent à travers des messages paquetisés. Un message est composé d'un nombre fixe de paquets. Le format d'un paquet est illustré dans la figure 4.12 [JTWB07b].

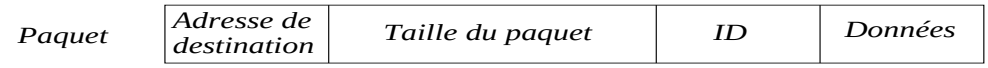


FIG. 4.12 – Le format d'un paquet utilisé dans le *CuNoC*.

Le premier champ représente l'adresse de destination dont la taille est proportionnelle au nombre total d'unité de communication (*CU*) constituant un réseau *CuNoC*. En pratique, il correspond au nombre de *CU* pouvant être implantés dans la zone reconfigurable d'un *FPGA*. Le second champ contient les informations sur la taille du paquet, tandis que le troisième champ contient le numéro d'identification (identifiant - *ID*) du paquet. On considère que la taille du champ *ID* reste inférieure ou égale à la taille du message et est jamais nulle. Le dernier champ contient les données à transmettre.

4.3.2.2 Le routeur *Unité de communication* (*CU - Communication Unit)*

L'élément de base de *CuNoC* est le routeur *Unité de communication - CU*. La fonction principale d'un *CU* est le routage de paquets reçus de la part d'autres *CU* voisins vers la cible des paquets dont ils contiennent l'adresse physique. Un routeur *CU* peut être utilisé d'une part comme une unité de communication indépendante pour acheminer des messages entre 4 éléments de calcul au maximum. D'autre part, il peut être utilisé comme un routeur faisant partie d'un plus grand réseau de routeurs pour les besoins de communication plus importants. L'architecture d'une unité de communication *CU* est présentée dans la figure 4.13 [JTWB07b, JTBW09].

Comparé à la plupart des routeurs présentés dans la littérature possédant plusieurs buffers en entrées / sorties, le routeur *CU* est caractérisée par un unique buffer. C'est une des originalités de *CuNoC*. Par conséquent, les besoins en ressources pour un seul routeur (de même pour un réseau entier *CuNoC*) diminuent considérablement par rapport aux routeurs ou réseaux existants. Afin de gérer l'aiguillage de plusieurs paquets arrivant simultanément au routeur (jusqu'à 4 paquets maximum correspondant aux entrées des quatre directions d'un même routeur) et d'éviter les situations d'embouteillage au niveau d'un seul routeur, un *CU* utilise une politique d'arbitrage basée sur le principe d'une priorité à droite (par analogie avec les priorités du code de la route dans un carrefour). Ainsi, les paquets arrivant au niveau d'un routeur ne sont pas prioritaires par rapport aux paquets arrivant simultanément au même routeur et situés à la droite de sa direction.

Lorsque les paquets arrivent sur les ports d'entrée d'un *CU*, ce dernier les reçoit simultanément, et les traite à tour de rôle selon la règle de priorité à droite. Ainsi, le paquet arrivant d'une

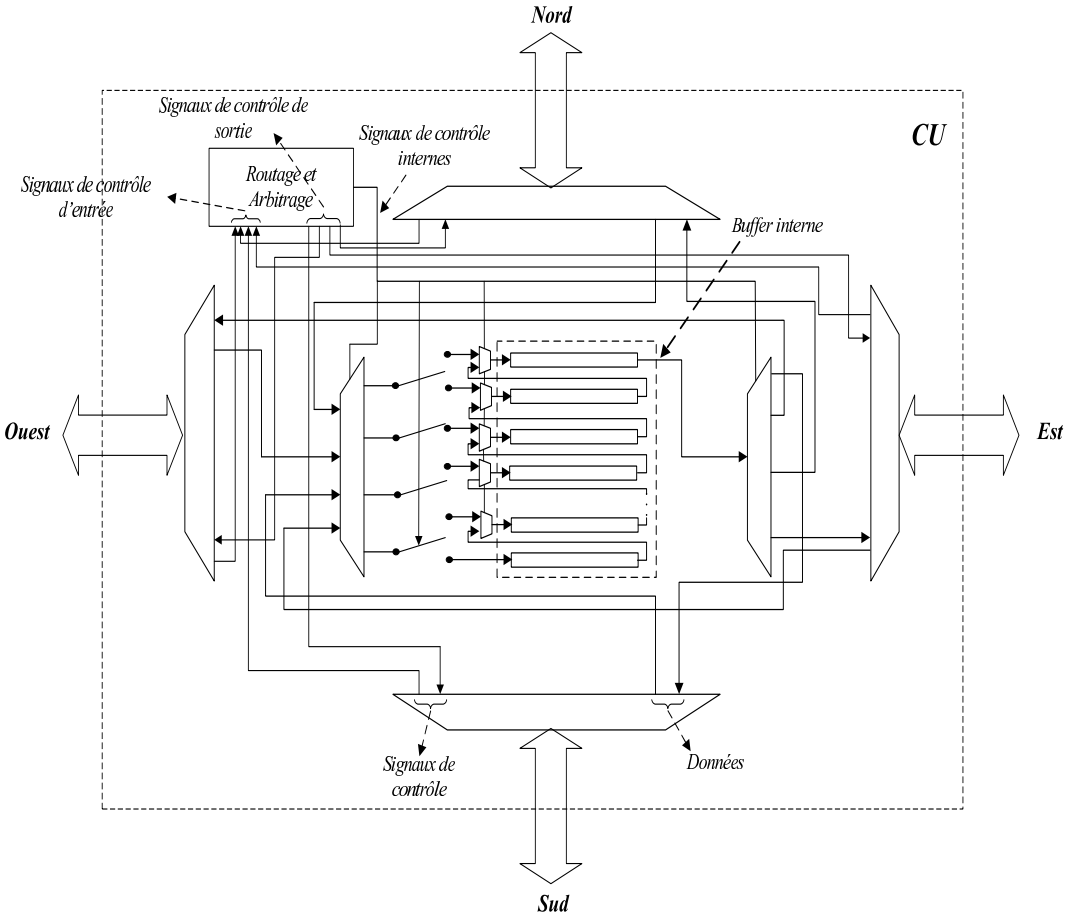


FIG. 4.13 – L'architecture d'une unité de communication *CU*.

direction et n'ayant aucun paquet à sa droite au moment d'arrivée, aura la priorité de passage la plus élevée et sera placé temporairement dans le premier registre du buffer avant d'être transféré le premier vers sa direction de sortie du routeur. Plus précisément, le paquet à priorité la plus élevée sera placé dans le premier registre interne du buffer et traité en premier. Tandis que, le paquet à priorité de passage moins élevée sera placé dans le registre interne suivant du buffer et sera traité en deuxième, etc. Dans le cas où 4 paquets arrivent simultanément, la priorité de passage la plus élevée est préalablement définie dans la phase de conception du réseau, tandis que les priorités de passage moins élevées sont déterminées selon la règle de priorité à droite.

Un routeur *CU* utilise la technique d'aiguillage *store-and-forward*. Cela signifie qu'un paquet ne peut pas être transféré vers un autre routeur tant que ce dernier ne peut le recevoir entièrement. Dû au fait qu'un paquet n'est pas composé de *flits* et qu'il pourra être stocké dans un routeur en un coup d'horloge, cette technique d'aiguillage ne nécessite pas de ressources supplémentaires dans le cas de *CuNoC*. Le seul inconvénient est qu'elle introduit une latence supplémentaire par *CU*. Cette latence est de 2 cycles d'horloge par paquet reçu. Dans le cas de 3 paquets arrivant simultanément, le « traitement » des paquets par le routeur et leur transfert

vers les *CU* voisins nécessitent 6 cycles d'horloge (voir 4.14). Pendant ce temps de traitement, un *CU* ne peut pas recevoir d'autre paquets. Une fois le temps de traitement a écoulé, le *CU* est de nouveau disponible pour la réception de nouveaux paquets.

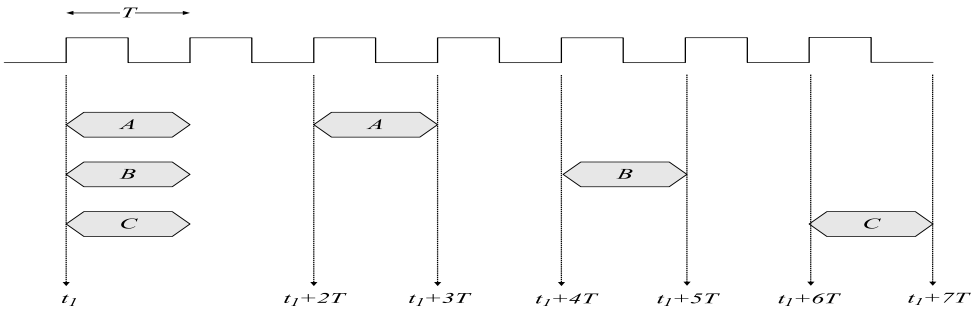


FIG. 4.14 – Illustration de transfert de 3 paquets à travers un *CU* et de la latence introduite.

Les paquets véhiculés au sein d'un réseau *CuNoC* ne peuvent pas être routés et acheminés en utilisant simplement l'algorithme de routage *XY* [DYN02]. En effet, le algorithme de routage *XY* pour des réseaux *2D*, achemine premièrement un paquet selon l'axe *X*, puis selon l'axe *Y*. Ce type d'algorithme de routage n'est donc pas adapté pour les réseaux sur puce dont la structure évolue et change au cours du temps. En effet, dans le contexte considéré des systèmes reconfigurables auto-organisés, de nouveaux modules de calcul (*PE*) peuvent être placés au sein du réseau en modifiant alors sa structure régulière et homogène initiale, et en rendant les communications entre les modules de calcul plus complexe (voire parfois impossible). La figure 4.15 illustre des cas d'inadaptation de l'algorithme de routage *XY* pour des structures de réseau reconfigurable tel que *CuNoC*. En plaçant dynamiquement un module dans le réseau comme le montre la figure 4.15, le routage de paquets entre plusieurs nœuds devient alors impossible.

Pour remédier à ce genre de situations, l'approche *DyNoC* repose sur l'algorithme de routage *S-XY* (*surrounding XY*) [BA05]. Avec cet algorithme, lorsque un paquet sur son chemin atteint un obstacle (un module placé dynamiquement), il choisit une direction parmi deux possibles. Si le paquet arrive de la direction *X*, il choisit une des directions selon l'axe *Y*. C'est-à-dire, une direction vers le « haut » ou vers le « bas » du réseau. Réciproquement, si le paquet arrive de la direction *Y*, il va choisir une des directions selon l'axe *X* (vers la « droite » ou vers la « gauche » du réseau). Afin d'éviter l'effet *ping-pong* pouvant survenir dans les cas où une des coordonnées du paquet est égale à une des coordonnées de la cible située après l'obstacle (du côté opposé), chaque routeur *DyNoC* « appose » sa signature dans le paquet pour indiquer à d'autres routeurs le chemin que le paquet a déjà emprunté. En pratique, pour chaque routeur, les directions à prendre en cas d'un obstacle sont prédéfinies à l'avance, ce qui peut provoquer des très longs chemins de contournement d'un obstacle à suivre par les paquets.

L'algorithme de routage utilisé dans le *CuNoC* repose également sur l'algorithme *XY* modifié. Plus précisément, un routeur *CU*, après avoir reçu un paquet, compare l'adresse contenue

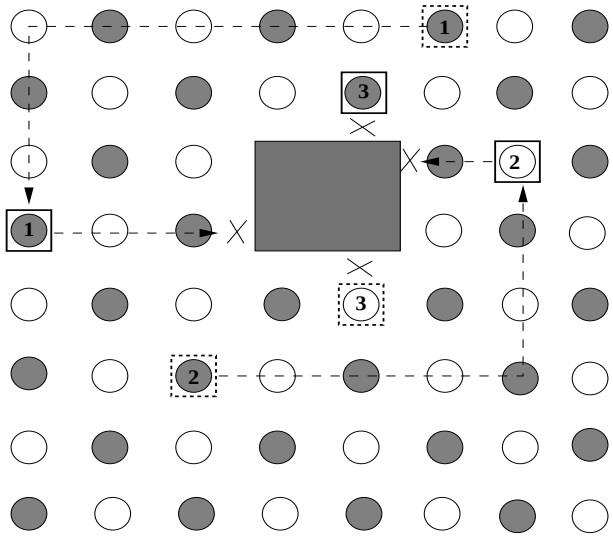


FIG. 4.15 – Illustration de l'inadaptation des algorithmes de routage classiques tel que *XY* pour le *CuNoC*.

dans le paquet avec la sienne et calcule la direction de sortie du paquet en prenant en compte les conditions du trafic et les éventuels obstacles (modules dynamiquement placés pouvant couvrir des routeurs et donc non actifs pour le réseau). Un *CU* prend également en considération la direction d'arrivée d'un paquet pour le calcul de la direction de sortie en évitant ainsi l'effet *ping-pong*, mentionné précédemment. L'algorithme de routage proposé et utilisé ne permet pas forcément les chemins d'acheminement des paquets les plus courts entre un nœud destinataire et un nœud source. Ceci est dû au fait qu'il prend en compte non seulement les conditions du trafic mais également les modules placés dynamiquement. Cela signifie que dans certains cas le chemin emprunté par le paquet entre une source et une destination lorsqu'il n'y a pas d'obstacles n'est pas forcement le plus court. L'aspect algorithme de routage est traité dans la dernière partie de ce chapitre.

Comparé à la plupart des architectures de routeurs présentés dans la littérature, une autre caractéristique d'un routeur *CU* est qu'il ne possède pas d'interface spécifique de connexion à un élément de calcul. C'est-à-dire qu'il ne possède pas d'entrée / sortie dédiées uniquement à un *PE* pouvant leur être associé. Ainsi, un routeur *CU* peut être connectée via un ou plusieurs de ses ports d'entrée / sortie soit à un autre routeur *CU* soit à un élément de calcul (*PE*). Afin d'éviter les erreurs de livraison de paquets, dans la phase de calcul de direction qu'un paquet va emprunter, un *CU* prend également en considération les signaux venant de ses routeurs voisins directs. A titre d'exemple, selon les conditions de trafic, les signaux de directions et d'occupation d'un routeur indiquent qu'une direction est libre et peut être prise par un paquet. Ce dernier ne pourra prendre cette direction si l'adresse de ce routeur correspond à un élément de calcul dont l'adresse n'est pas contenue dans le champ *adresse* du paquet. Ceci est plus détaillé par la

suite dans la section relative au placement de modules dans le *CuNoC*.

4.3.2.3 Interconnexions entre *CU*

La figure 4.16 illustre l'interface de connexions entre deux routeurs *Unités de communication* (*CU*) [JTBW09]. Cette interface est composée de plusieurs signaux d'entrée, de sortie et de ports de données bidirectionnels. Chaque *CU* a pour chaque port identifié selon sa direction (direction *Nord, Sud, Est* et *Ouest* respectivement *N, S, E* et *W*), des signaux de contrôle et de données. Lorsqu'un *CU* envoie un paquet à son routeur voisin, il positionne un signal de contrôle d_out à la valeur logique '1'. Ainsi, le routeur voisin est informé de l'envoi d'un paquet. Ce signal de contrôle n'est jamais positionné à '1' si le *CU* voisin est dans un état « occupé » (signalé par son signal CU_occ_in). Ainsi, avant de générer les signaux de contrôle de sortie, l'unité de contrôle d'un *CU* prend en considération les signaux d'entrées relatifs à l'occupation de toutes les routeurs *CU* avoisinants. Si le signal CU_occ_in d'un *CU* est positionné à '0' à un de ses ports, cela signifie que le *CU* voisin connecté à ce port est disponible et peut recevoir un paquet. De même, à travers son signal de contrôle CU_occ_out, un *CU* indique à ses voisins son état de disponibilité ou non.

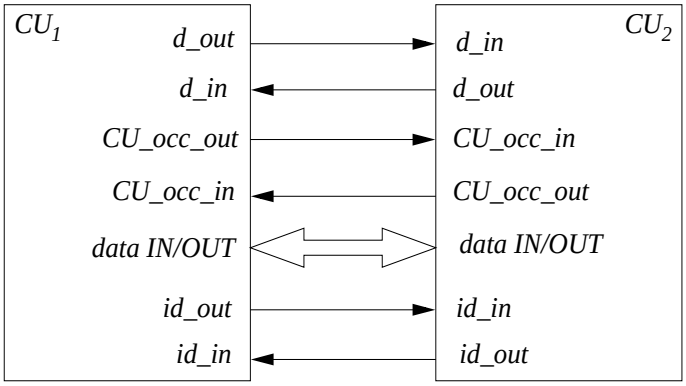


FIG. 4.16 – Interface physique entre deux *Unités de Communication (CU)*.

Le signal de contrôle d_in indique à un *CU* qu'il va recevoir un paquet. Dans ce cas, la logique d'arbitrage du *CU* décide dans quel de ses registres internes de son buffer le paquet sera stocké temporairement avant d'être transféré vers un autre routeur. Cela dépend du nombre de paquets arrivant simultanément sur ses autres ports. Dans le cas, où un *CU* n'a pas d'autres paquets sur ses autres ports, le paquet sera placé dans le premier registre interne de son buffer. Dans le cas contraire, la politique d'arbitrage basée sur la priorité à droite sera appliquée pour déterminer l'ordre de passage des paquets et leur stockage temporaire dans le buffer. De plus, la logique de contrôle d'un *CU* veille à ce qu'aucuns autres paquets ne lui sont transmis tant qu'il se trouve dans un « état occupé ». Ces signaux de contrôle et états de disponibilité ou non des *CU* permettent de ne pas perdre des paquets dans le réseau.

L'interface d'un *CU* dispose également de deux signaux d'identification d'entrée et de sortie (id_in et id_out). En effet, comme mentionnée précédemment, un *CU* peut être connectée soit à un autre *CU* soit à un élément de calcul. Afin de distinguer le type de blocs voisins connectés à un *CU*, ces signaux d'identification ont été introduits. Dans le cas d'une connexion avec un autre routeur *CU*, ces deux signaux sont positionnés à l'état '0'. Par contre dans le cas d'une connexion avec un élément de calcul (*PE*), ils sont positionnés à l'état '1'.

Pour l'échange de données un routeur *CU* dispose, pour ses quatre directions (*N, S, E* ou *W*), d'un port d'entrée / sortie bidirectionnel. Cela signifie que deux *CU* voisins ne peuvent pas s'envoyer simultanément des paquets. Ainsi, le bus de données entre deux *CU* voisins est partagé dans le temps. Un seul routeur *CU* peut envoyer des paquets de données à un instant donné. La figure 4.17 illustre un cas d'établissement de 4 connexions entre des ports entrées et sorties à travers un routeur *CU*. Chaque paire de connexion entre deux ports reliant deux *CU* ou deux éléments de calcul à travers un routeur *CU* doit attendre son tour pour pouvoir établir la connexion et échanger des données. La politique d'arbitrage basée sur la règle de priorité à droite détermine l'ordre d'établissement de connexions.

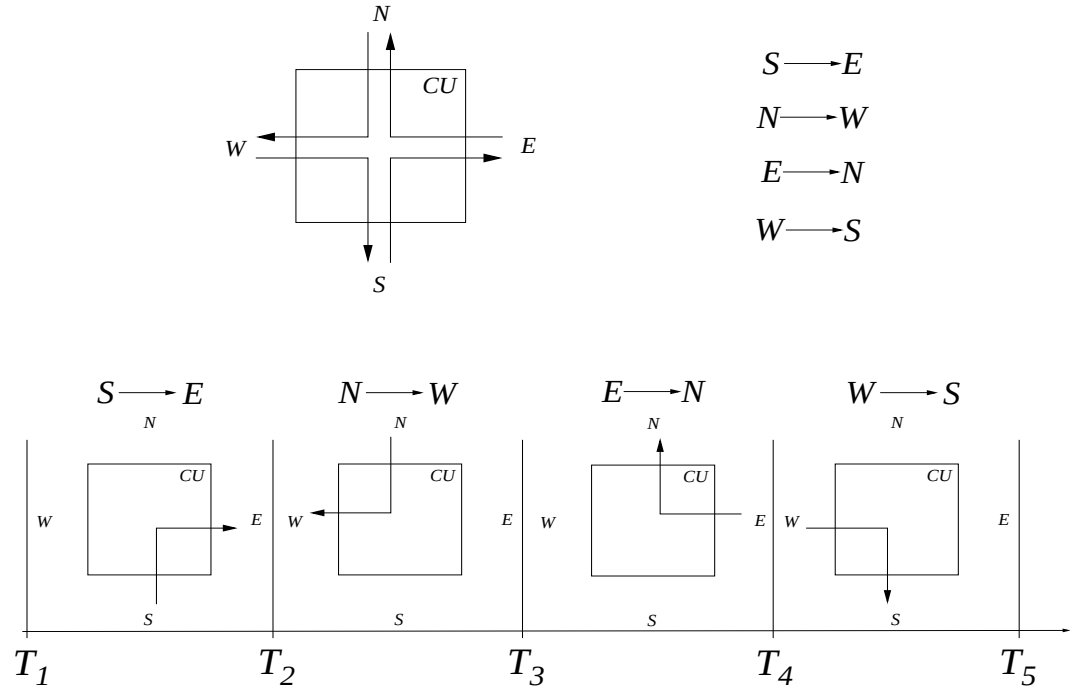


FIG. 4.17 – Exemple d'établissement de connexion entre des modules à travers une *CU* via les ports de données bidirectionnels partagés dans le temps.

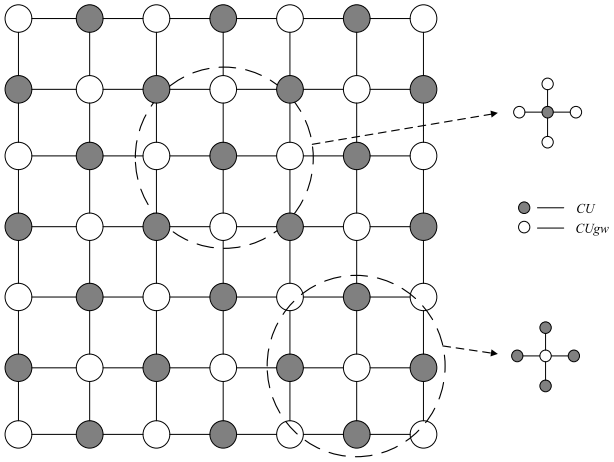


FIG. 4.18 – Placement valide d'unités de communication *CU* dans le *CuNoC*

4.3.2.4 Types d'*Unité de communication*

Le figure 4.18 illustre un placement valide de routeurs *CU* dans le réseau *CuNoC*. Nous distinguons deux types de routeurs *Unité de communication* : un routeur *CU* « classique » (*classic CU*) et un routeur *CU* « qui-cède-le-passage » (*to-give-way CU - CUgw*) [JTWB07a, JTWB07b, JTBW09]. Entre ces deux types de *CU*, il n'y pas de différences dans les conditions normales de communication dans le réseau. C'est-à-dire, dans les conditions où des situations d'embouteillage n'apparaissent pas au sein du réseau. La différence entre ces deux types de routeurs se situe au niveau de leurs modules de contrôles. Les graphes flot de contrôle de chacun de ces deux *CU* sont présentés respectivement dans les figures 4.19 et 4.20. Pour le transfert d'un paquet de données, quel que soit son type, chaque routeur *CU* emploie la procédure suivante :

1. A un instant donnée, un *CU* (classique ou « qui-cède-le-passage ») reçoit les paquets de toutes les directions (au maximum 4) et se met dans un mode « occupé »,
2. Ensuite, le routeur *CU* détermine l'ordre de transfert de chaque paquet selon la politique d'arbitrage basée sur la priorité à droite,
3. Puis, le routeur analyse les adresses contenues dans le champ « adresse de destination » de chaque paquet, les compare avec son adresse et positionne les signaux de direction pour acheminer le paquet considéré : *west*, *east*, *south*, *north*, *stay-x* et *stay-y*,
4. Le routeur poursuit une analyse pour chaque paquet reçu, à travers les signaux d'interconnexion, les états de ses routeurs *CU* voisins. En particulier, les états des routeurs voisins désignés par les signaux de direction déterminés préalablement. Pour un paquet, plusieurs directions de transfert sont possibles. Lors de la détermination des directions possibles pour un paquet, une liste ordonnée de directions possibles est créée. Si un *CU* voisin, désigné premier dans la liste de directions possibles est disponible, le routeur transfère alors le paquet vers sa direction. Dans le cas contraire, il effectue la même procédure

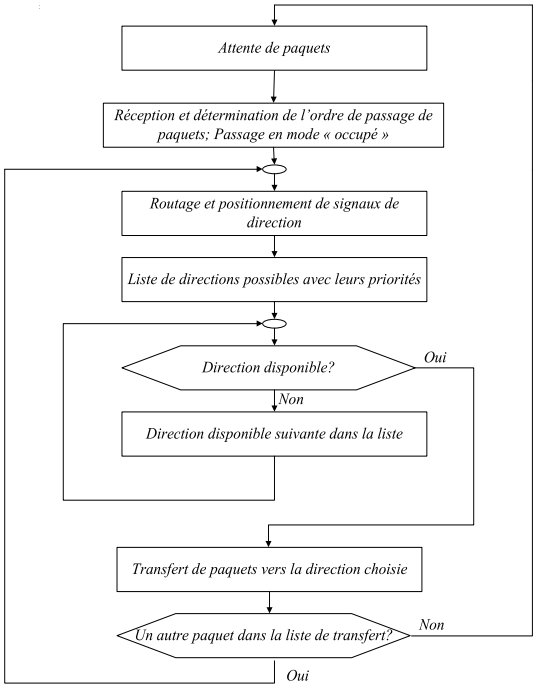


FIG. 4.19 – Graphe flot de contrôle d'un *CU* classique.

avec le *CU* suivant de la liste de directions possibles. Dans le cas où tous les routeurs de la liste de directions possibles sont occupés, le paquet est alors en attente de transfert jusqu'à la disponibilité d'un des *CU* voisins faisant partie de la liste des directions possibles. Cette situation peut entraîner une « situation d'embouteillage ». C'est pour éviter ces situations de blocage qu'ont été définis les deux types de *CU*. Plus précisément, lorsqu'une situation d'embouteillage survient, un *CU* « qui-cède-le-passage » initialement en mode « occupé » passe volontairement en mode « libre » ou disponible et reçoit alors les paquets de ses *CU* classiques voisins. La procédure de fonctionnement est illustrée dans le flot de contrôle d'un *CUgw* par le bloc encadré et pointillé dans la figure 4.20. Une fois que les paquets sont reçus par le routeur *CUgw* concerné, il peut ensuite transférer ses propres paquets à travers ses *CU* classiques voisins libérés de leurs paquets par la procédure précédente. Cette procédure permet ainsi de dissoudre des situations locales d'embouteillage. Cette procédure d'échange de deux paquets entre deux *CU* dans une situation d'embouteillage est illustrée par la figure 4.21.

5. Pour tous les paquets reçus, un *CU* effectue les mêmes opérations décrites dans l'étape 3 jusqu'à la fin du graphe flot de contrôle.

6. Enfin, après avoir transféré tous les paquets reçus au cours de la première phase vers ses *CU* voisins, le routeur se remet en mode « libre » et est de nouveau disponible pour la

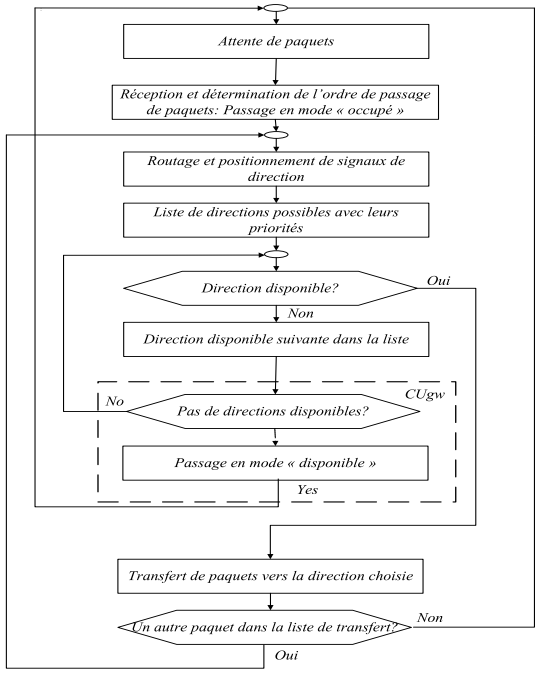


FIG. 4.20 – Graphe flot de contrôle d'un *CU* « qui-cède-le-passage ».

réception de nouveaux paquets.

La complémentarité entre les deux types de *CU* présentée dans les étapes de fonctionnement précédentes impose une structuration d'un réseau *CuNoC* dans laquelle chaque *CU* d'un type est entouré par des *CU* d'un autre type. Ainsi, un *CU* classique est entourée par des *CU* « qui-cédent-le-passage » et réciproquement un *CUgw* est entouré de *CU* classiques (figure 4.18).

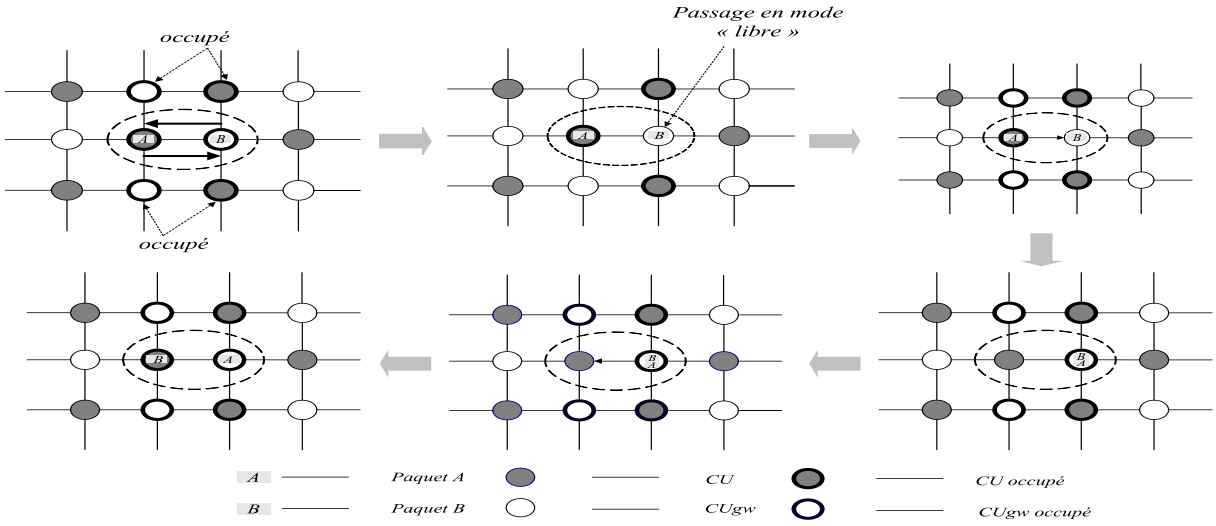


FIG. 4.21 – Illustration d'une situation d'embouteillage dans un réseau *CuNoC* et sa résolution sur l'exemple d'échange de 2 paquets

4.3.2.5 Placement et conditions de placement des modules dynamiques dans un réseau *CuNoC* à topologie maillée (*mesh*)

Le réseau *CuNoC* est adapté pour le placement dynamique de modules, grâce au calcul de chemin entre un nœud source et un nœud destinataire au cours du fonctionnement du réseau (*run-time*) et grâce à son algorithme de routage. Les conditions de transmission des paquets de données au sein d'un réseau évoluent au cours du temps puisque d'une part, placement dynamique de modules peut avoir lieu à chaque instant, d'autre part selon le trafic des routeurs qui sont soit en mode « libre » ou en mode « occupée ». Chaque paquet doit donc « se débrouiller » pour atteindre sa destination finale à travers les algorithme de routage et technique d'aiguillage. Néanmoins, en plaçant un module dynamiquement au centre d'un réseau *CuNoC*, les transferts de paquets entre des modules ayant un échange de communication important avant ce placement dynamique, ne doivent pas être interrompus ou « détériorés ». Pour cela, les paquets doivent considérer le module placé dynamiquement comme un obstacle et alors chercher à le contourner en trouvant d'autres chemins menant vers destinations finales. C'est pourquoi, dans le but d'assurer une communication viable et des transfert de paquets efficace entre tous les modules statiques ou dynamiques placés au sein d'un réseau *CuNoC*, un certain nombre de règles de placement doit être respecté [JTWB07b] :

Règle N^o1 : Chaque module (élément de calcul - PE) doit avoir au moins un accès au réseau via un des ports parmi ses routeurs avoisinants ou l'entourant.

Règle N^o2 : Tous les modules *PE* communiquent à travers les routeurs *CU*.

Règle N^o3 : Les routeurs *Unités de communication* sont connectées, soit à d'autres routeurs, soit à des *PE*. Chaque *CU* est connecté au moins à un routeur de type différent à lui même (CU classique à un *CU* « qui-cède-le-passage » et vice versa, voir la figure 4.18).

Règle N^o4 : Chaque module *PE* est entouré au maximum dans ses quatre directions part trois modules *PE*.

Règle N^o5 : Tous les modules, placés dynamiquement ou statiquement, étant entourés dans leurs quatre directions uniquement par les routeurs *CU* sont toujours joignables (règle *DyNoC* [BAM$^+$05]).

Règle N^o6 : Entre tous les modules *PE*, il doit exister au moins un chemin leur permettant des échanges de paquets de données.

La figure 4.22 illustre deux exemples de placement de modules dans un réseau *CuNoC* [JTWB07b]. La figure 4.22a présente un placement invalide des modules au sein de ce réseau. En effet, la plupart des règles de placement ne sont pas respectées, en particulier la règle N^o6. Le module A placé au milieu du réseau n'est pas joignable par certains modules (les modules B et C), car il est « entouré » pour chacune de ses directions par d'autres modules. Par conséquent, ce placement ne permet pas une communication entre le module A et les modules B et C (voir la figure 4.22b). Par contre, la figure 4.22b présente un placement valide des modules

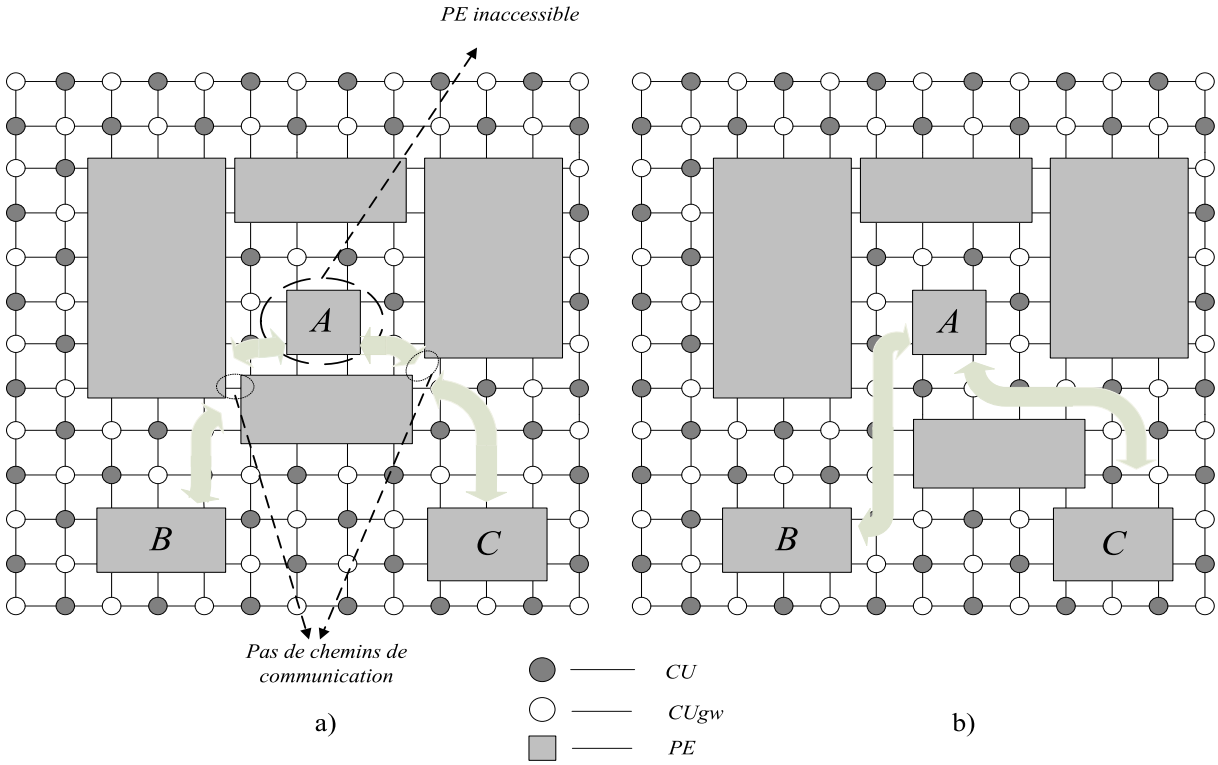


FIG. 4.22 – Placement de modules dans le réseau *CuNoC* : a) invalide b) valide.

dans un réseau *CuNoC*. Cet exemple de placement valide présente qu'une de nombreuses situations possibles de placement de modules au sein d'un réseau *CuNoC* respectant les règles de placement prédéfinies. Ces règles de placement reposent sur une adaptation pour un réseau *CuNoC* des règles de placement au sein d'un réseau *DyNoC*. Par ailleurs, toutes les contraintes de placement de modules présentées dans le réseau *DyNoC* sont applicables à un réseau *CuNoC* [BAM+05, BA05]. Cependant, les contraintes de placement de modules du réseau *CuNoC* sont moins contraignantes que celles établies pour un réseau *DyNoC*. Par exemple, dans un réseau *CuNoC*, il doit avoir au moins un chemin entre tous les modules (la règle N^o6). Par contre, contrairement à un réseau *DyNoC*, tous les modules placés dans un réseau *CuNoC* ne doivent pas être obligatoirement entourés dans toutes leurs directions par des routeurs (soit des *CU* dans le cas d'un réseau *CuNoC*).

Un routeur *CU* possède 4 ports d'entrées / sorties et ne dispose pas de connexion spécifique locale pour être connecté avec un élément de calcul *PE*. Cette structure permet de combiner ou d'associer directement des routeurs *CU* avec des modules *PE* au sein d'un réseau *CuNoC*. Les figures 4.23 et 4.24 illustrent quelques exemples de ces structures hétérogènes [JTWB07b, JTBW09].

La figure 4.23 présente une façon de connecter un nombre maximum de modules pouvant communiquer dans un réseau *CuNoC* d'une taille m×n. Ce nombre maximum de modules est

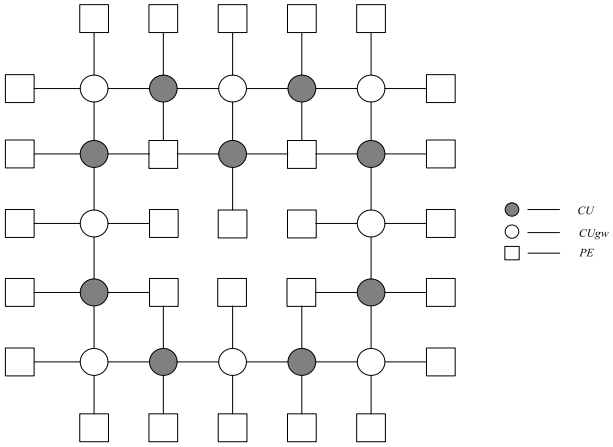


FIG. 4.23 – Illustration de placement d'un nombre maximum de modules pour un réseau *CuNoC* m×n.

déterminé en supposant que la taille maximum de chaque module ne dépasse pas la taille d'un *CU*. On considère un réseau homogène *CuNoC*, composé uniquement de routeurs *CU*. Ensuite, on place des modules de calcul *PE* autour de ce réseau par des connexions avec les routeurs périphériques du réseau à travers leurs ports d'entrée / sortie. Puis, on remplace les *CU* dans le réseau avec les modules de calcul tout en respectant les règles de placement prédéfinies. On obtient alors une structure de réseau comme présenté par la figure 4.23. On en déduit alors, le nombre maximum de modules de taille minimale (taille d'un *CU*), qui peuvent être connectés à un réseau *CuNoC* d'une taille initiale de m×n, par les expressions suivantes :

$$N_{mod_{max}} = 3m + 4n - 8 + trunc\left[\frac{(n-4)(m-3)}{2}\right] \tag{4.1a}$$

$$N_{CU} = m * n - N_{mod_{max}} \tag{4.1b}$$

où N_{CU} représente le nombre d'unités de communication *CU* qui reste dans un réseau d'une taille initiale m×n. L'expression $trunc\left[\frac{(n-4)(m-3)}{2}\right]$ est égale à 0 pour les valeurs de n et $m \leq 3$.

La structure de connexion des modules telle que présentée dans la figure 4.23 est réalisable, mais pas souhaitable car dans ce type de structure de connexion un très grand nombre de modules communiquent à travers un nombre restreint de routeurs *CU* aboutissant à une structure de réseau de performances médiocres dans des situations de transfert de paquets élevé (besoins en communication importants).

Une structure de réseau préférable de connexion des modules et de routeurs est présentée dans la figure 4.24. Chaque module est connecté à au moins un *CU* lui-même connectée à travers d'autres ports d'entrées / sorties uniquement à d'autres routeurs *CU*. Dans ce cas de figure, le nombre maximum de modules pouvant être connectés à un réseau *CuNoC* d'une taille initiale de m×n est défini de la manière suivante :

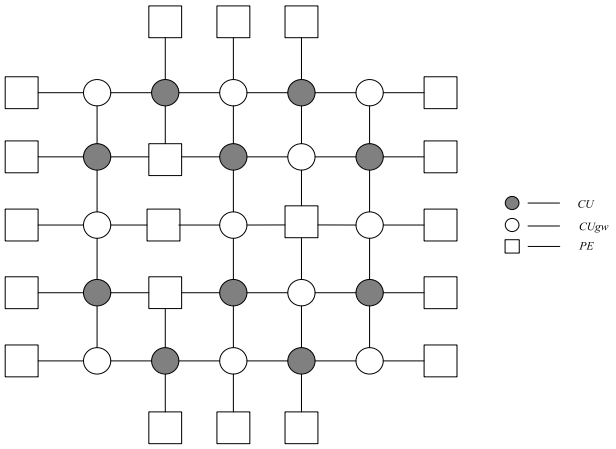


FIG. 4.24 – Placement préférable de modules dans le réseau *CuNoC*.

$$N_{mod_{max}} = 2(m+n-2) + trunc\left[\frac{(m-2)(n-2)}{2}\right] \tag{4.2a}$$

$$N_{CU} = m * n - N_{mod_{max}} \tag{4.2b}$$

où l'expression $trunc\left[\frac{(m-2)(n-2)}{2}\right]$ est égale à 0 pour n et $m \leq 2$.

4.3.2.6 Placement dynamique des modules dans un *CuNoC*

La figure 4.25 illustre 3 phases successives de construction d'un réseau *CuNoC* maillé et de placement des modules de calcul dans ce dernier dans une surface reconfigurable [JTWB07b, JTBW09]. On considère initialement un placement valide des *CU* dans la zone reconfigurable. Après le placement des routeurs *Unités de communication*, la deuxième étape consiste à placer les modules de calcul dans ce réseau. Pour un module de taille inférieure ou égale à la taille d'un *CU*, ce module remplace uniquement un *CU* du réseau. Pour un module de taille supérieure à celle d'un *CU*, un groupe de plusieurs *CU* dont la somme des tailles est égale ou supérieure à celle du module à placer dynamiquement sont substitués ou remplacés par ce module. Au moment du placement dans le réseau, le module qui remplace un ou plusieurs *CU*, hérite de toutes leurs adresses. Cela signifie qu'un module substituant 4 *CU* va posséder 4 adresses différentes et au moins 4 ports d'entrées / sorties différents correspondant aux différentes directions vers le réseau.

Le réseau *CuNoC* est particulièrement adapté au placement dynamique des modules de calcul au cours de son fonctionnement. Supposons la situation suivante : un réseau *CuNoC* 7×7 comprenant 5 modules statiques de calcul dans sa phase de construction. Chaque module de calcul placé effectue une fonction ou un ensemble de fonctions précises. Dans le cadre d'exécution de ses fonctions, des échanges de données avec d'autres modules statiques du système ont lieu

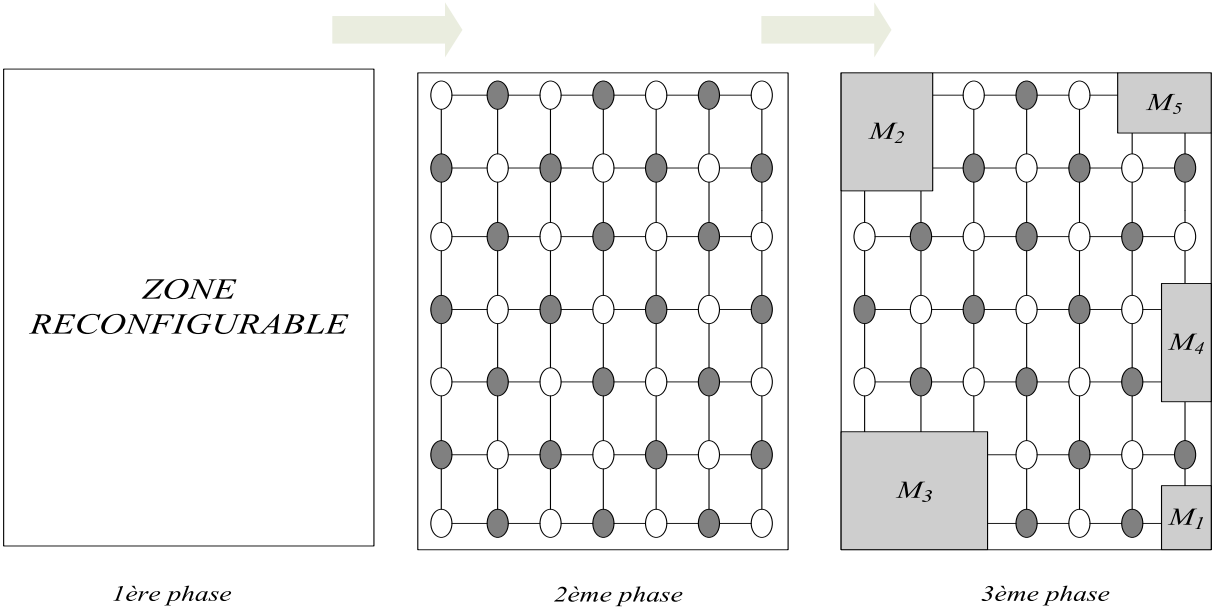


FIG. 4.25 – Trois phases de construction d'un réseau *CuNoC*.

et aboutissant au bout d'un certain temps à des communications importantes entre ces modules du système (illustration figure 4.26a). A un instant donné, une nouvelle demande fonctionnelle du système, ne pouvant pas être assurée par les modules statiques apparaît. Dans ce cas de figure, un nouveau module de calcul permettant l'exécution de cette requête fonctionnelle est

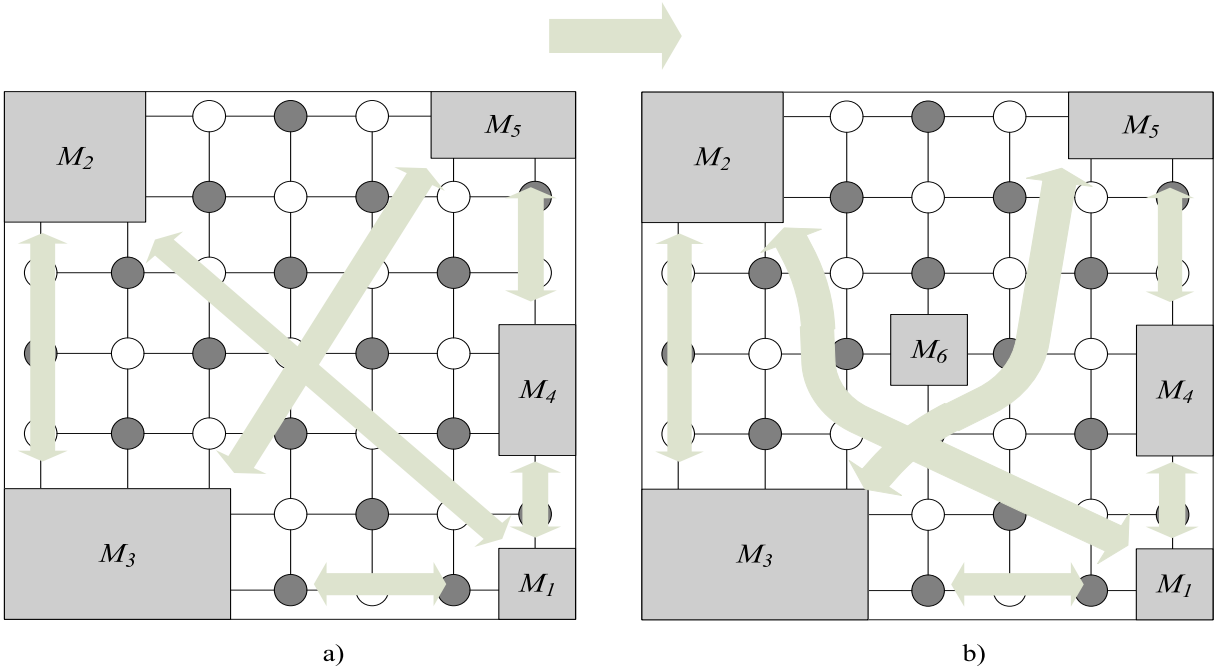


FIG. 4.26 – Placement dynamique des modules dans un réseau *CuNoC*.

alors placé dynamiquement dans le réseau. Le contrôleur de la zone reconfigurable, chargé des placements des modules de calcul dans le réseau reconfigurable, analyse l'espace disponible de la zone reconfigurable et décide du placement du nouveau module, tout en respectant les règles prédéfinies de placement. La zone géographique du réseau désigné par ce contrôleur pour accueillir le module dynamique substitue ces *CU* par le nouveau module de calcul lors de sa phase d'insertion dans le réseau.

Pour ce faire, le contrôleur du placement de modules renseigne tous les *CU* dans la zone d'insertion de cette substitution afin que les routeurs concernés vident leurs buffers avant d'être remplacés. A l'issu de cette opération de vidage, les *CU* désignés par le contrôleur se mettent alors dans un mode « module » en indiquant ainsi aux *CU* voisins de leurs nouveaux statuts. Ainsi, les routeurs restant (pas concernés par l'insertion du module dynamique), considéreront par la suite, le module placé dynamiquement, comme un obstacle. Ils intégreront alors cet obstacle lors de leurs phases de routage et d'aiguillage des paquets de données suivants à transmettre. Cette situation est illustrée par la figure 4.26b.

4.3.2.7 Résultats de simulation

Afin de démontrer la validation fonctionnelle d'un réseau *CuNoC*, plusieurs simulations dans des différents conditions ont été réalisées. Nous considérons un réseau *CuNoC* de dimension 4×4 pour plusieurs dispositions de modules de calcul. Le premier cas de simulation, considère 16 modules de calcul statiques connectés dans ce réseau. Chaque module de calcul communique avec un autre module du réseau à travers l'envoi et la réception simultanée et mutuelle de paquets de données. Dans un deuxième cas de simulation, l'insertion d'un module de calcul dynamique dans le réseau 4×4 reliant les 16 modules de calcul statiques est simulée.

La figure 4.27a illustre la topologie du *CuNoC* utilisée pour le premier cas de simulation. Le réseau est homogène et composé uniquement de routeurs *CU*. Dans cet exemple, tous les modules de calcul statiques sont disposés en périphérie du réseau considéré. Un extrait des résultats de simulation de ce premier cas de simulation est présenté dans la figure 4.28. Ces résultats de simulation montrent que tous les modules de calcul transmettent simultanément et reçoivent des paquets de données après une durée de latence correspondant aux temps de transfert dans le réseau. Le temps de latence pour chaque paquet est différent. En effet, certains paquets sont retardés plus que des autres à cause de leur ordre de passage dans les *CU* du réseau selon la politique d'arbitrage adoptée par le réseau. Dans les cas de condition de transfert chargé du réseau, les paquets envoyés par des modules *PE* n'arrivent pas à destination dans l'ordre d'envoi. Cet inconvénient peut être réduit, voire éviter, si les valeurs de taux d'injections de paquets dans le réseau par des modules *PE* ne sont pas élevées.

Le deuxième cas de simulation présente l'insertion dynamique d'un module de calcul dans le réseau identique dans le premier cas de simulation. La structure des routeurs *Unités de com-*

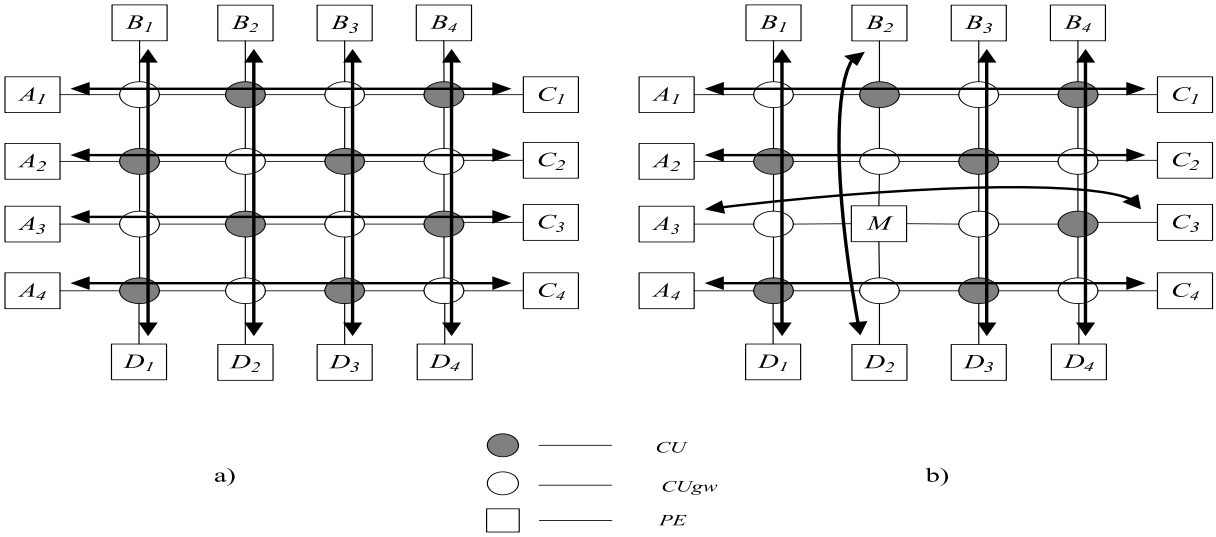


FIG. 4.27 – Cas de simulation : a) Communication entre 16 modules via un *CuNoC* 4×4 b) Insertion d'un module de calcul dans un réseau *CuNoC* de dimension 4×4 reliant 16 modules de calcul.

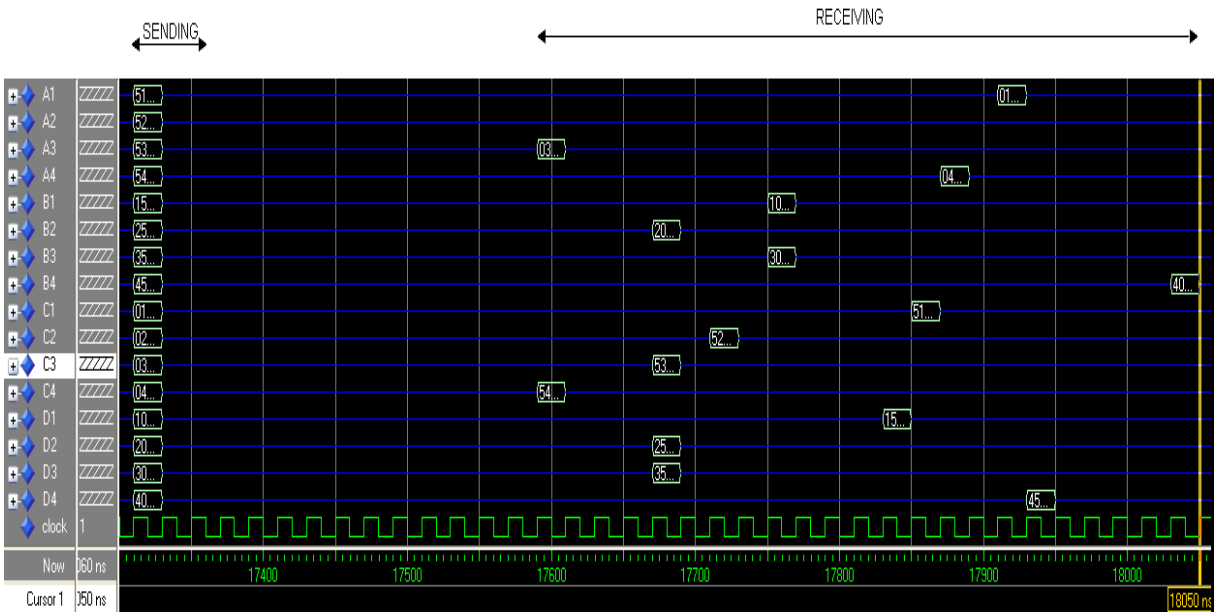


FIG. 4.28 – Le premier cas de simulation : communication entre 16 modules de calcul via un réseau 4×4.

munication utilisée pour ce cas de simulation et la disposition des modules de calcul sont présentées dans la figure 4.27b. L'extrait des résultats de simulation pour ce cas de simulation est illustré dans la figure 4.29. Premièrement, les communications entre ces 16 modules de calcul sont établies comme illustré dans la figure 4.28. A un instant donné, une requête d'insertion

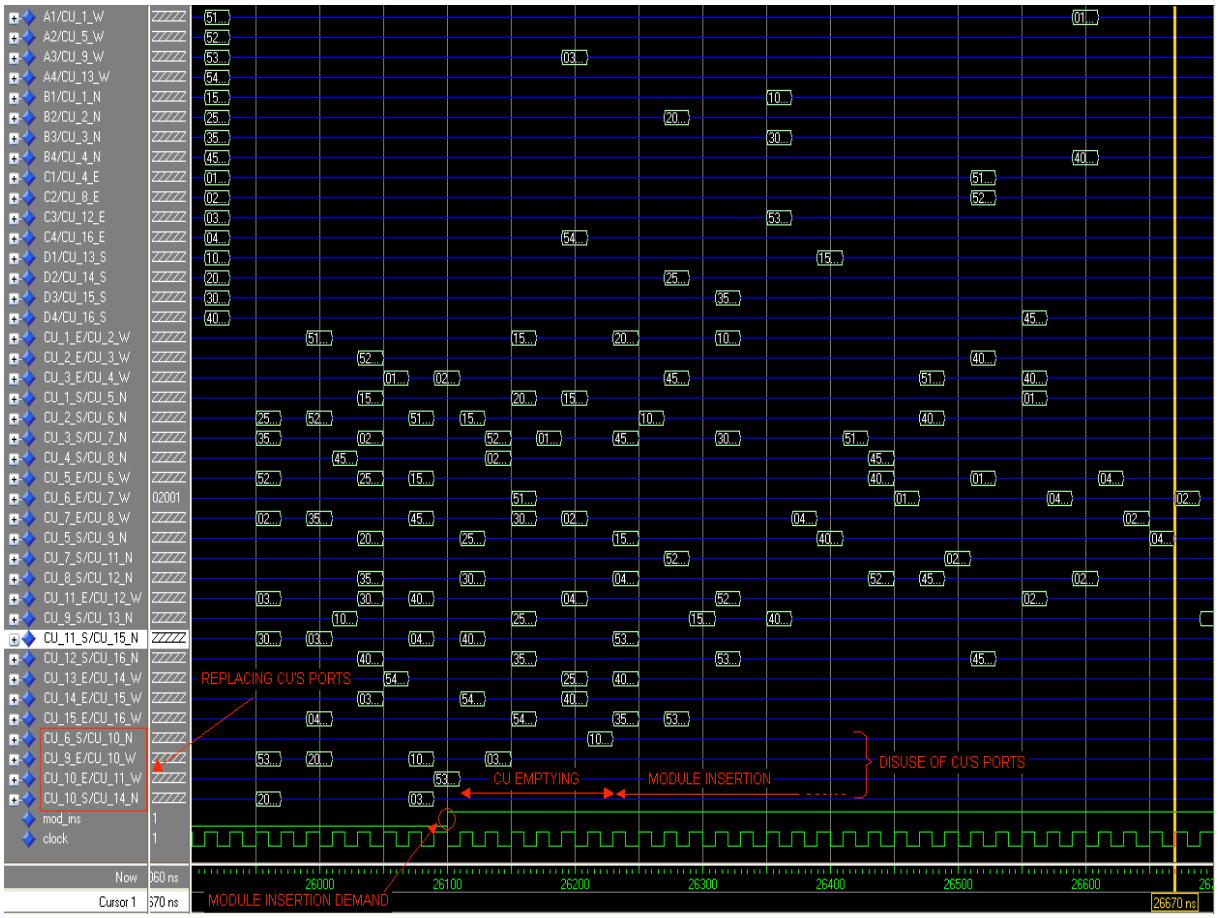


FIG. 4.29 – Le deuxième cas de simulation : l'insertion d'un module de calcul dans le réseau 4×4 reliant 16 autres modules de calcul.

d'un module de calcul dynamique se produit. Cette demande est illustrée dans la figure 4.29 à travers un signal *mod_ins*, positionné à '1'. Le routeur *CU* qui sera remplacé par le module de calcul (CU_{10} dans le cas de la figure 4.27b) change le statut en passant d'un statut « CU » en statut « module » et vide son buffer interne. Le vidage du buffer prend plusieurs cycles d'horloge comme illustré dans la figure 4.29. Après avoir changé son statut en « module », le *CU* de substitution devient indisponible pour participer au routage de paquets provenant de ses *CU* voisins. Ceci est illustré dans la figure 4.29 par l'absence de paquets dans les ports d'entrées / sorties du routeur de substitution CU_{10}. Ainsi, tous les paquets envoyés par des modules de calcul et passant par cette zone du réseau correspondant à la zone de localisation du routeur de substitution CU_{10}, ne sont plus acheminés vers ce routeur par ses *CU* voisins. En résumé, les chemins de communication entre les modules de calcul, après vidage du buffer interne de CU_{10}, ne considèrent plus le routeur CU_{10}. Ainsi, des contournements s'effectuent autour de ce routeur comme illustré dans la figure 4.30b.

Une analyse des chemins pris par les paquets envoyés par des modules de calcul vers leurs

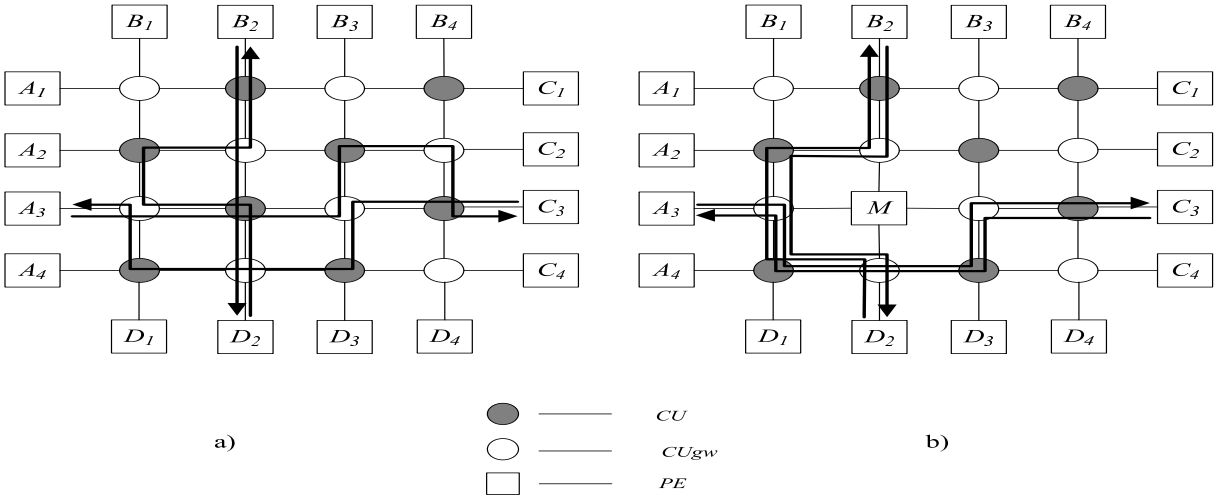


FIG. 4.30 – a) Illustration de communication entre 4 modules de calcul dans un réseau *CuNoC* 4×4 utilisé pour la communication de 16 modules de calcul avant b) et après l'insertion du module de calcul.

destinations, avant et après l'insertion du module dynamique de calcul a été réalisée. Ces résultats sont présentés dans la figure 4.30. Avant l'insertion, les chemins pris par les paquets entre deux modules de calcul ne sont pas forcément optimaux (chemin tout droit et minimal). Ceci s'explique par le fait que le cheminement des paquets est déterminé au cours du fonctionnement du réseau et en fonction des conditions de trafic dans le réseau. Nous observons particulièrement les chemins de communication entre les modules de calcul contenant le routeur CU_{10}, qui sera ultérieurement désigné comme routeur de substitution par le contrôleur de la zone reconfigurable. Plus précisément, dans les résultats de simulation, les cas de communication entre les modules de calcul A_3 et C_3, et entre B_2 et D_2 ont été considérés. Avant l'insertion du module dynamique de calcul, certains paquets envoyés par ces modules de calcul transitent à travers le routeur CU_{10}. Par contre, après l'insertion du module dynamique, plus aucune transmission à travers ce routeur s'effectue. Tous les paquets envoyés par ces modules, choisissent un cheminement de contournement de l'unité de communication CU_{10}. Ceci est illustré dans la figure 4.30b.

Ces résultats de simulation démontrent clairement la propriété principale de placement dynamique du réseau *CuNoC* au cours de son fonctionnement. De plus, ces résultats montrent que le placement dynamique des modules de calcul dans un réseau *CuNoC* n'interrompt pas et ne détériore pas les communications établies entre les modules statiques initialement placés. Cette propriété de placement dynamique dans un réseau *CuNoC* peut très bien être associée avec la propriété de la reconfiguration dynamique partielle des circuits *FPGA*. Un réseau *CuNoC* implanté dans une zone reconfigurable d'un circuit *FPGA*, maintient les communications entre des modules de calcul malgré qu'une partie du réseau devient intactive en subissant une

reconfiguration dynamique partielle [MG00].

4.3.2.8 Résultats d'implantation d'un réseau *CuNoC*

Une modélisation structurelle et des implantations de routeurs *Unités de communication* (*CU*) de différente taille de données pour la technologie *FPGA Xilinx Virtex II* et *IV* ont été réalisées. Les résultats de synthèse sont données dans le tableau 4.2 en termes de ressources logiques consommées (blocs logiques *CLB*) et de performances en fréquence (fréquence maximale de fonctionnement). Pour un routeur de taille de données de 8 bits, les ressources nécessaires sont 50 *CLB* et la fréquence maximale de fonctionnement sur un *FPGA Virtex IV* est 549.3 MHz.

TAB. 4.2 – Résultats de synthèse de routeurs *CU* de différents formats de données sur *FPGA Xilinx Virtex II* et *Virtex IV*.

	CU classique			CU « qui-cède-le-passage »		
format de données	Virtex II f [MHz]	CLB Slices	Virtex IV f [MHz]	Virtex II f [MHz]	CLB Slices	Virtex IV f [MHz]
8 bit	320.7	49	549.3	302.7	50	549.3
16 bit	320.7	84	549.3	279.6	74	549.3
24 bit	272.1	112	549.3	279.2	98	549.3
32 bit	272.1	140	549.3	279.2	122	549.3
48 bit	272.1	196	549.3	279.2	170	549.3
64 bit	272.1	252	549.3	279.2	218	549.3
128 bit	250.0	476	549.3	250.0	410	549.3
256 bit	279.2	924	549.3	279.2	820	549.3
512 bit	244.3	1820	549.3	250.4	1564	549.3

Une implantation et une synthèse de réseaux *CuNoC* de différente taille sur les technologies *FPGA Xilinx Virtex II* et *Virtex IV* ont également été réalisées. Ces résultats sont présentés dans le tableau 4.3 en termes des blocs logiques (*CLB*) pour la surface *FPGA* occupée et en termes de fréquence maximale de fonctionnement du réseau considéré. Pour un réseau *CuNoC* de dimension 4×4 utilisant un format de données 32 bits, son implantation nécessite des ressources environ 2100 blocs logiques et possède une fréquence maximale de fonctionnement avoisinant 250 MHz et 550 MHz, respectivement pour les technologies *FPGA Xilinx Virtex II* et *Virtex IV*. Ces résultats de synthèse sont obtenus sous l'environnement de l'outil de synthèse *Leonardo Spectrum* [LEO].

4.3.2.9 Évaluations des performances du *CuNoC*

La bande passante (*throughput*) maximale d'un routeur *CU* de format de données de n bits, fonctionnant à une fréquence f et permettant la communication par transfert de paquets au

TAB. 4.3 – Résultats de synthèse sur les *FPGA Xilinx Virtex II* et *Virtex IV*, de réseaux *CuNoC* pour différentes dimensions et différents formats de données.

	CuNoC 2x2			*CuNoC* 3x3			*CuNoC* 4x4		
data width	Virtex II f [MHz]	CLB Slices	Virtex IV f [MHz]	Virtex II f [MHz]	CLB Slices	Virtex IV f [MHz]	Virtex II f [MHz]	CLB Slices	Virtex IV f [MHz]
8 bit	259.1	197	549.3	241.7	443	549.3	279.6	788	549.3
16 bit	241.6	293	549.3	279.6	659	549.3	213.0	1172	549.3
24 bit	241.6	418	549.3	228.8	948	549.3	259.4	1672	549.3
32 bit	241.6	522	549.3	228.8	1184	549.3	250.4	2088	549.3
48 bit	228.8	730	549.3	250.4	1656	549.3	279.2	2920	549.3
64 bit	252.9	938	549.3	250.4	2128	549.3	272.1	549.3	549.3

maximum de 4 modules de calcul est déterminée par les expressions suivantes :

$$Throughput_{max}[\,Gbps\,] = n * f/2 \tag{4.3}$$

$$Throughput_{max}[\,paquet\,/\,clk\,] = 1/2 \tag{4.4}$$

L'expression 4.3 est divisée par un facteur 2, dû à la fois à la latence maximale introduite par un routeur *CU* et au fait qu'un *CU* dès la réception de paquets se met en mode « occupé » dans le but d'interdire la réception de nouveaux paquets jusqu'à sa nouvelle disponibilité (après transfert de derniers paquets reçus). Ainsi, dans le cas de 4 modules de calcul connectés à un routeur *CU*, et de la réception de 4 paquets de données simultanément, cette latence atteint 8 cycles d'horloge. Par exemple, dans le cas d'un *CU* de format de données 8 bits, à laquelle 4 modules de calcul sont connectés et ayant une fréquence de fonctionnement de 100 MHz, la bande passante maximale avoisine 400 Mbps. Pour d'autres formats de données (jusqu'à 512 bits), les résultats d'évaluation de la bande passante maximale d'un *CU* sont illustrés dans la figure 4.31. Ces résultats, prenant en compte les fréquences maximales de fonctionnement pour différents formats de données, sont obtenus pour les technologies *FPGA Virtex II* et *IV* et sont présentées dans le tableau 4.2. Sur les courbes de la figure 4.31, on constate que pour un format de données 64 bits avec la technologie *FPGA* Virtex IV à une fréquence maximale de 549.3 MHz, la bande passante d'un *CU* est de l'ordre de 20 Gbps. On notera qu'un routeur de format de données 64 bits consomme peu de ressources logiques (252 blocs logiques CLB).

Une des raisons, autre que la latence d'un *CU*, qui mène également à la décroissance des performances du réseau, notamment en termes de la bande passante, est la liaison physique bidirectionnelle entre deux *CU* qui est partagé dans le temps. La figure 4.32 illustre les performances d'un routeur *CU* en termes de latence moyenne et maximale pour deux types de distribution de trafic : un trafic *aléatoire* dans lequel chaque *PE* envoie un paquet vers un autre *PE* de manière aléatoire ; un trafic *défini*, où chaque *PE* connecté à un routeur *CU* envoie un

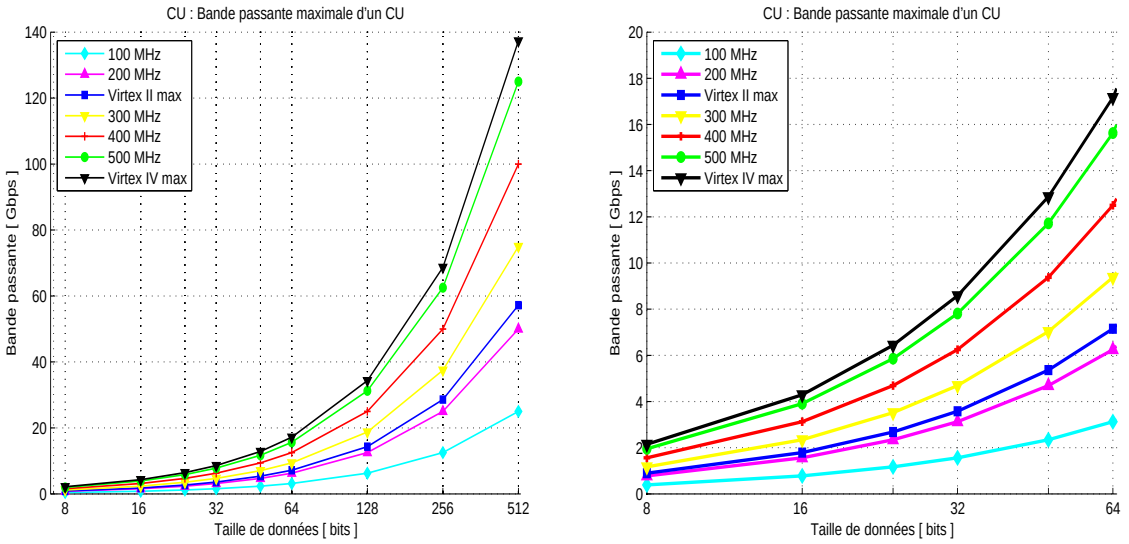


FIG. 4.31 – Bande passante d'un *CU* pour différents formats de données : a) de 4 à 512 bits (gauche) b) de 4 à 64 bit (droite).

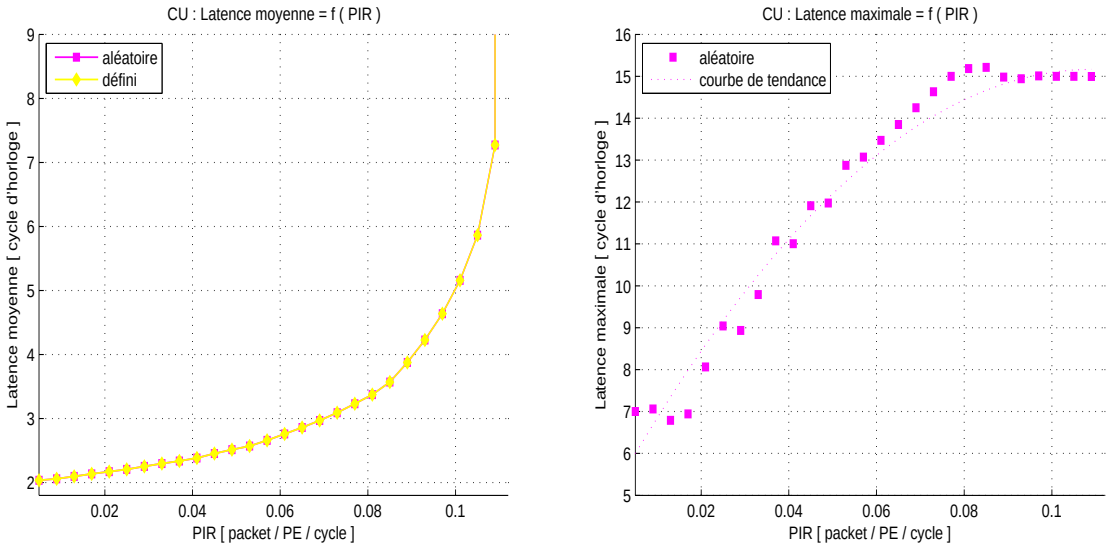


FIG. 4.32 – Latences moyenne et maximale d'un routeur *CU*.

paquet vers un *PE* situé à la direction opposée du routeur *CU*. En analysant les courbes présentées par la figure 4.32, on remarque que le taux d'injection du nombre de paquets maximal (*Packet Injection Rate - PIR*) d'un *PE* connecté à un routeur *CU*, pour les deux types de trafic est approximativement de l'ordre de 0.12 paquets par cycle d'horloge. Cela signifie que dans les conditions d'un réseau très chargé, un *PE* peut envoyer au maximum 12 paquets différents en 100 cycles d'horloge de durée de fonctionnement du réseau. De plus, la bande passante d'un

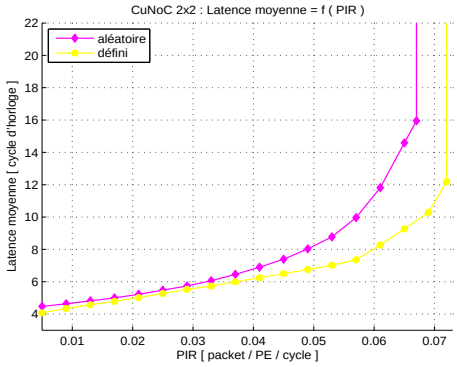
CU, calculée comme le produit entre le nombre de *PE* connectés à ce routeur et son taux d'injection de paquets *PIR*, correspond approximativement à sa valeur maximale de bande passante pour les deux types de trafic ($4 * 0.12 = 0.48 \approx Throughput_{max}[\,paquet\,/\,clk\,]$). On remarque également que la latence moyenne d'un *CU* peut varier de 2 à 8 cycles d'horloge principalement en fonction de la valeur de *PIR*.

Dans un réseau *CuNoC* de dimension n×n, la bande passante maximale ne peut pas être déterminée de manière classique comme pour des réseaux directs. C'est-à-dire, par le produit du nombre de routeurs (et de *PE* associés) du réseau par la bande passante maximale d'un routeur. Néanmoins, un réseau *CuNoC* de dimension plus élevée permet de faire communiquer un plus grand nombre de modules de calcul au sein de son réseau. Cependant, la détermination de sa bande passante globale n'obéit pas à l'expression mentionnée précédemment. Les principales difficultés d'exprimer la bande passante d'un réseau *CuNoC* est que d'une part, chaque routeur n'est pas forcement associé un élément de calcul *PE*. D'autre part, les routeurs *CU* sont connectés à travers des liaisons bidirectionnelles. Bien évidemment, lors d'échange de données entre deux routeurs *CU*, des retards supplémentaires peuvent survenir. D'une manière générale, en augmentant la taille du réseau, la bande passante maximale ne croît pas de façon linéaire.

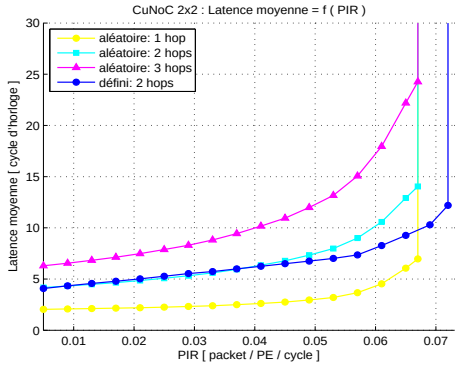
Une évaluation de la latence moyenne pour les différentes dimensions de réseau : 2×2, 3×3 et 4×4 a été réalisée. Pour réaliser cette évaluation, des générateurs de trafic (*TG*) ont été modélisés en langage C. L'ensemble réseau et générateurs de trafic ont été simulés sous environnement de co-simulation *C-VHDL* avec l'outil *ModelSim* [MSI]. Chaque *TG* envoie des paquets vers d'autres *TG* connectés au réseau selon le type de distribution de trafic sélectionné (*aléatoire* ou *défini*). D'autre part, il reçoit des paquets envoyés par d'autres *TG* du réseau. Les *TG* étant modélisés en C, toutes les informations relatives au départ et à l'arrivée d'un paquet sont disponibles, permettant ainsi le calcul des latences du réseau. Dans chaque cas considéré, chaque *TG* envoie 125 000 de paquets différents.

Les figures 4.33 - 4.37 illustrent les résultats d'évaluation de performances de réseaux *CuNoC* de taille 2×2, 3×3 et 4×4 respectivement pour les types de distribution de trafic *aléatoire* et *défini*. Pour toutes ces évaluations de performances, deux structures du *CuNoC* sont considérées. Une première structure, similaire à celle présentée dans la figure 4.27a, dans laquelle tous les *TG* sont placés en périphérie du réseau, aboutissant ainsi à une disposition homogène de routeurs au sein du réseau. Une seconde structure, similaire à celle présentée dans la figure 4.27b, où nous trouvons également des *TG* en périphérie du réseau mais également un *TG* supplémentaire au centre du réseau par substitution d'un routeur. Cette deuxième structure est donc une structure plus hétérogène que la précédente.

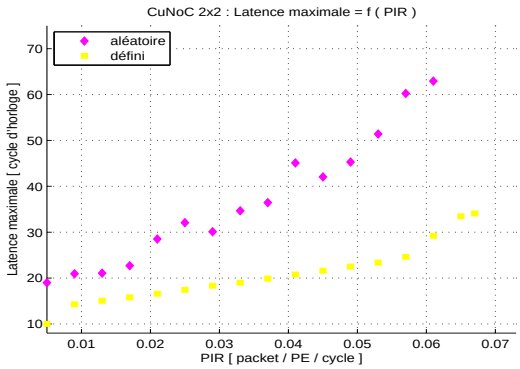
Les résultats de simulation pour ces deux cas de figures sont présentés dans les figures 4.33 - 4.37. Ils montrent les performances d'un réseau *CuNoC* de différente taille en termes de latence moyenne du réseau, latence maximale du réseau, latence moyenne de différentes



(a) latence moyenne



(b) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le réseau

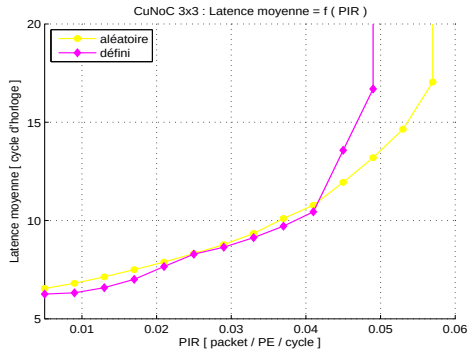


(c) latence maximale

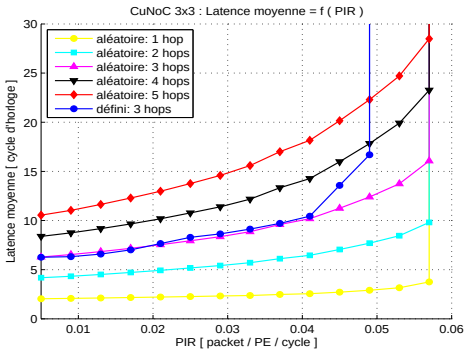
FIG. 4.33 – Résultats d'évaluation de performances d'un réseau *CuNoC* 2x2.

distances (*hops*) possibles entre les *TG* dans le réseau et distribution du trafic pour les réseaux de taille 3×3 et 4×4. On remarque également qu'en augmentant la taille du réseau, le *PIR* maximale d'un *PE* ne dépasse pas la valeur maximale de *PIR* d'un routeur *CU*. Cela confirme le fait, qu'une augmentation de la taille du réseau ainsi que le nombre de *PE* connectés, les conditions dans le réseau changent et deviennent plus difficiles en engendrant ainsi des délais supplémentaires.

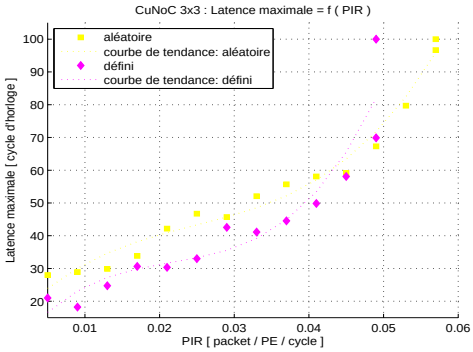
Pour les réseaux *CuNoC* de taille 3×3 et 4×4, une évaluation de performances dans le cas où le réseau de routeurs n'est pas homogène entre les *TG* périphériques a été réalisée. Plus précisément, nous avons substitué un *CU* par un *TG*. Pour l'introduction d'un *TG* au sein du réseau, nous avons considéré trois cas. Le premier cas, nommé « obstacle », dans lequel le *TG* placé n'effectue aucune communication avec les *TG* périphériques du réseau (aucune émission ou réception de paquets de données vers ou depuis les autres *TG*). Le deuxième cas, appelé « réception », correspond à un cas où le module de simulation dynamique *TG* réalise des communications limitées à la réception de paquets de données des autres *TG*. Aucune émission de



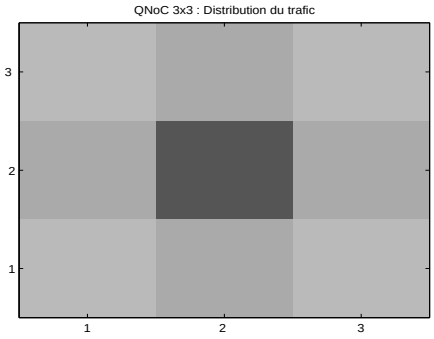
(a) latence moyenne



(b) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le réseau



(c) latence maximale

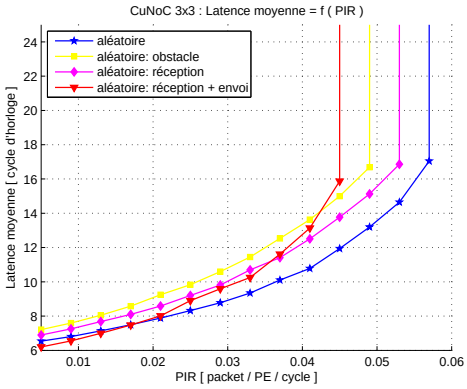


(d) distribution de trafic dans le réseau

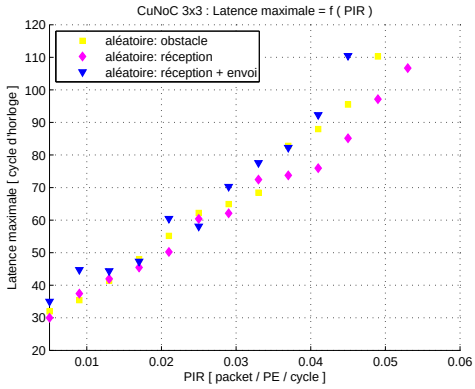
FIG. 4.34 – Résultats d'évaluation de performances d'un réseau *CuNoC* 3×3.

paquets n'est réalisée par le *TG* dynamique vers les autres *TG*. Le troisième cas, appelé « réception + envoi », présente le cas où le *TG* réalise des communications avec les autres *TG* (émission / réception de paquets vers ou depuis les autres *TG*). Pour les deux réseaux *CuNoC* de dimension 3×3 et 4×4, les routeurs situés respectivement aux positions $(2, 2)$ et $(2, 3)$ de ces réseaux sont remplacés par un *TG* (*TG* dynamique). En remplaçant un routeur *CU* par un *TG* et rendant ainsi ces deux réseaux hétérogènes, les performances de ces réseaux se dégradent. Les résultats de simulation pour ces trois cas sont présentés par les figures 4.35 et 4.37 respectivement pour les réseaux *CuNoC* 3×3 et 4×4. On remarque que pour ces deux réseaux *CuNoC*, le cas « réception + envoi » montre les résultats les plus faibles. Ceci est dû au trafic supplémentaire généré par le *TG* aux la position $(2, 2)$ et $(2, 3)$ respectivement pour les *CuNoC* de taille 3×3 et 4×4. D'autre part, le cas « réseau homogène », où entre tous les *TG* connectés au réseau il n'y a pas d'obstacles générant ainsi les chemins et délais supplémentaires, montre les meilleures performances.

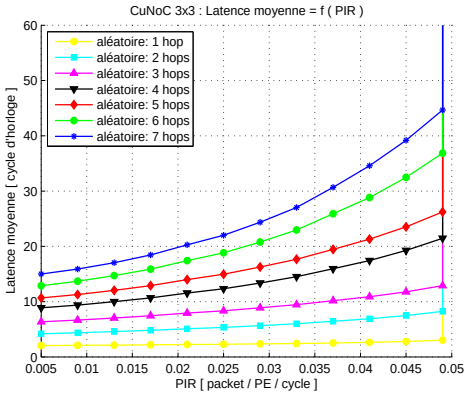
Une évaluation de la distribution du trafic dans les deux réseaux *CuNoC* de taille 3×3 et 4×4 a également été réalisée. La figure 4.34d montre la distribution du trafic dans le cas d'un



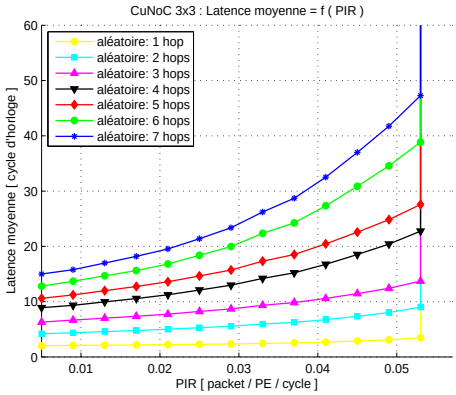
(a) latence moyenne



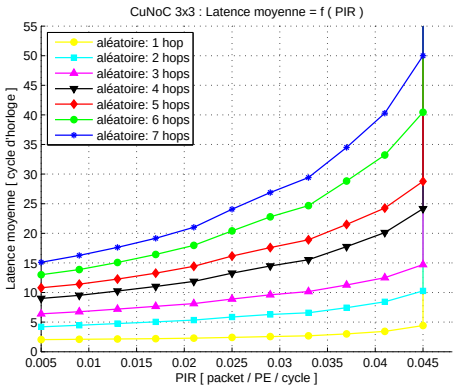
(b) latence maximale



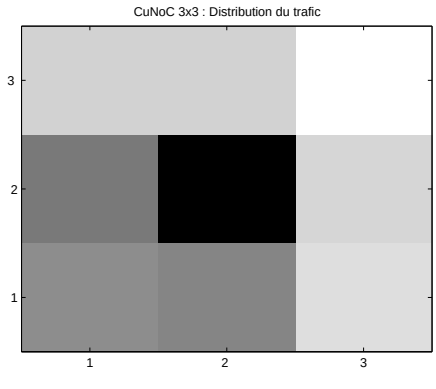
(c) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « obstacle »



(d) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception »

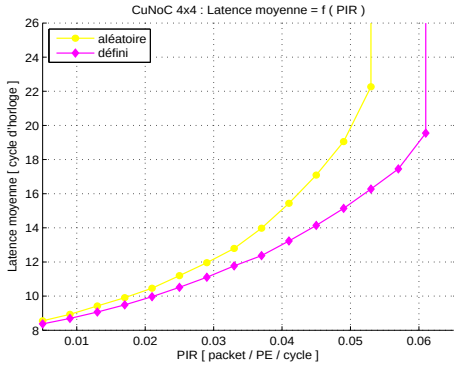


(e) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception + envoi »

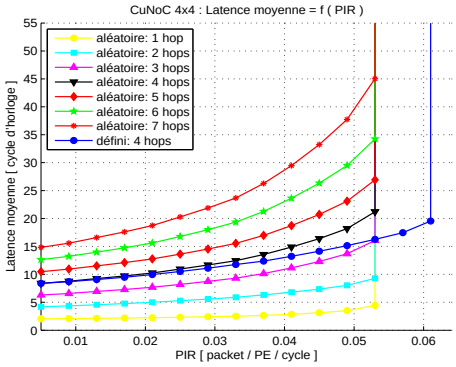


(f) distribution de trafic dans le réseau

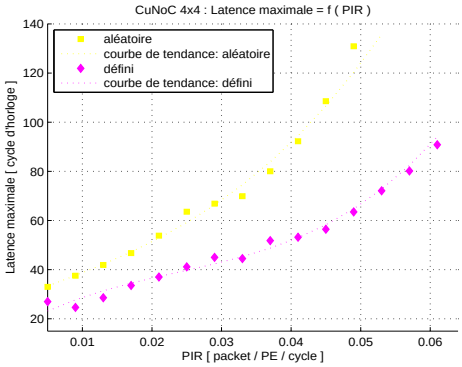
FIG. 4.35 – Résultats d'évaluation de performances d'un réseau *CuNoC* 3×3 et distribution du trafic dans le cas d'un *TG* dynamique à la position $(2, 2)$.



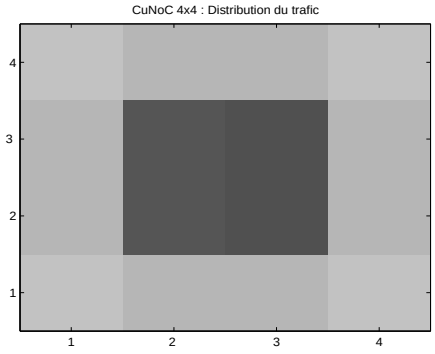
(a) latence moyenne



(b) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le réseau



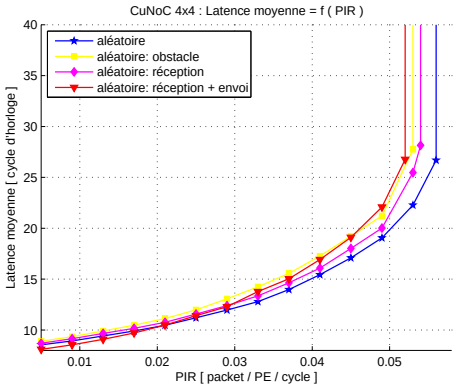
(c) latence maximale



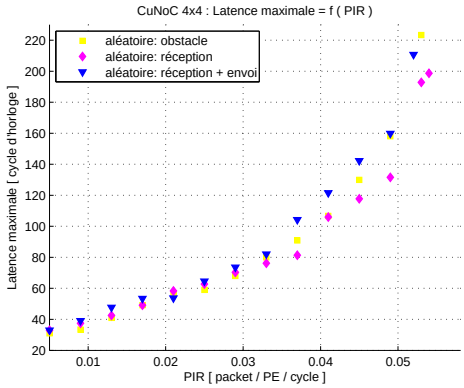
(d) distribution du trafic

FIG. 4.36 – Résultats d'évaluation de performances d'un réseau *CuNoC* 4×4.

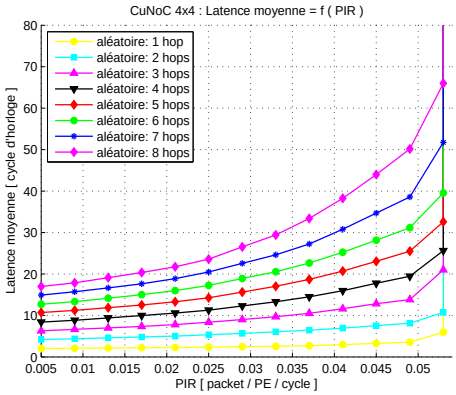
réseau 3×3 homogène. L'intensité du trafic est illustrée selon le niveau de clarté des différentes zones du réseau. Les zones les plus claires correspondent à un trafic de forte intensité ou chargé. Cette intensité se réduit au fur et à mesure que les zones s'assombrissent. Les zones les plus foncées correspondent donc aux zones de trafic moins denses. On remarque que les routeurs situés dans les extrémités ou « coins » du réseau subissent un trafic plus chargé. En effet, pour ces routeurs locaux, des paires de *TG* ont été associés. D'autre part, le routeur situé au centre du réseau, ne possédant pas un *TG* associé, subit un trafic le moins dense du réseau. Concernant les autres routeurs, la distribution du trafic est uniforme. En remplaçant le routeur à la position $(2, 2)$ par un *TG*, la distribution du trafic dans le réseau est modifiée. Le routeur à la position $(2, 2)$ n'étant plus disponible pour l'acheminement de paquets, influent considérablement le trafic au sein du réseau. On constate que le routeur de position $(3, 3)$ devient le plus chargé du réseau. En effet, l'indisponibilité du routeur $(2, 2)$ entraîne un contournement du *TG* dynamique pour l'acheminement de la majorité des paquets à travers le routeur $(3, 3)$. Ce contournement est réalisé grâce à l'algorithme de routage spécifique à un réseau *CuNoC*. Plus de détails concernant



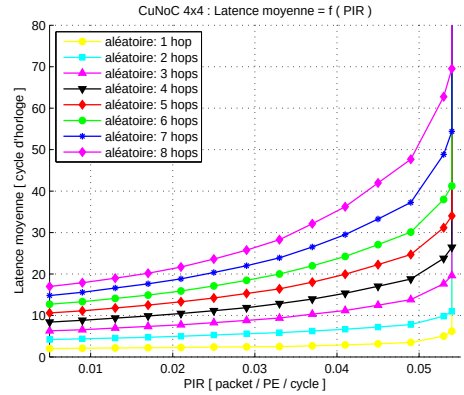
(a) latence moyenne



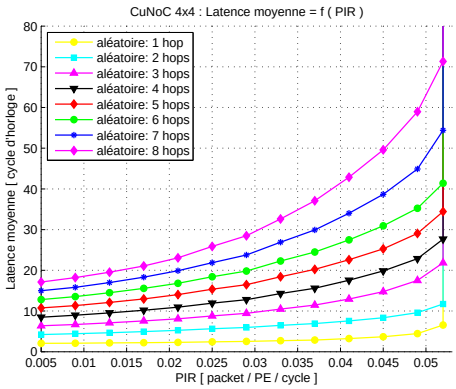
(b) latence maximale



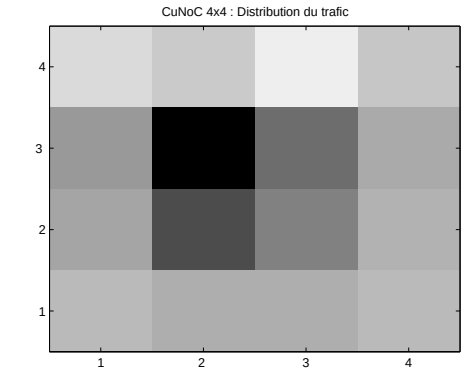
(c) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « obstacle »



(d) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception »



(e) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception + envoi »



(f) distribution du trafic

FIG. 4.37 – Résultats d'évaluation de performances d'un réseau *CuNoC* 4x4 et distribution du trafic dans le cas d'un *TG* dynamique.

cet algorithme sont donnés à la fin de ce chapitre.

Pour le réseau *CuNoC* de taille 4×4, la distribution du trafic est similaire à celle du réseau 3×3. En effet, de même, les routeurs situés aux « coins » du réseau sont connectés à des paires de *TG*, subissant alors un trafic le plus dense du réseau. Les routeurs centraux du réseau (formant un petit carré au milieu du réseau) sont les moins sollicités pour participer au trafic du réseau. En remplaçant le *CU* de position $(2, 2)$ par un *TG*, la distribution du trafic change, notamment à proximité du *TG* dynamique. Comme dans le réseau *CuNoC* de taille 3×3, le routeur situé en haut à droite du *TG* dynamique est le plus sollicité. La densité du trafic du reste du réseau ne subit pas trop de changement.

Nous avons évalué la latence moyenne pour les différentes tailles de réseau en fonction du nombre de *TG* connectés. Pour chaque taille de réseau, le nombre de *TG* connectés varie d'un nombre minimal à un nombre maximal de 16 modules de calcul pour le réseau de dimension 4×4. Ces résultats sont présentés dans la figure 4.38. On remarque que la latence moyenne varie et croît quasi linéairement en fonction de la taille du réseau et du nombre de *TG* connectés.

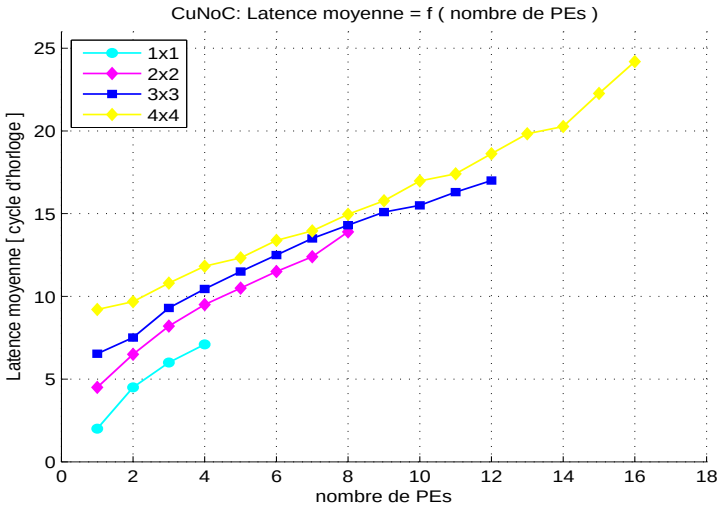


FIG. 4.38 – Latence moyenne en fonction du nombre de modules de calcul (*TG*) connectés au réseau et de sa taille.

Un routeur *CU* utilise la technique d'aiguillage *store-and-forward* (SAF) qui introduit une latence additionnelle par *CU*. Cette latence est de l'ordre de 2 à 8 cycles d'horloge en fonction du nombre de paquets reçus à un instant donné. Pour un réseau *CuNoC* de taille n×n, la latence minimale d'un paquet envoyé d'une source vers une destination est définie par l'expression suivante :

$$latence_{min} = N_{CU} * latence_{CU_{min}} \tag{4.5}$$

où N_{CU} représente le nombre de *CU* parcourus entre la source et la destination, $latence_{CU\,min}$ est la latence minimale d'un *CU* dépendant du nombre de paquets reçus à un instant donné

évaluée par la condition suivante $2 * T_{CLK} < latence_{CU} < 8 * T_{CLK}$, où T_{CLK} est la période d'horloge.

Le tableau 4.4 présente les valeurs minimales et maximales de latence pour tous les cas de simulation considérés. Les valeurs présentées s'expriment en nombre de cycles d'horloge. Par exemple, pour un réseau de dimension 4×4 dans lequel 16 *TG* sont connectés, la latence minimale est de 8 cycles d'horloge, tandis que la latence maximale est de 39 cycles d'horloge.

TAB. 4.4 – Évaluation de la latence de 125 000 paquets envoyés de manière aléatoire par un *TG* pour 4 différentes taille d'un réseau *CuNoC*.

CuNoC	4 x 4	3 x 3	2 x 2	1 x 1
Moyenne	8.55 - 29.76	6.54 - 23.04	4.47 - 15.60	2.04 - 8.5
Minimum	2 - 8	2 - 6	2 - 4	2
Maximum	33 - 192	28 - 125	19 - 84	7 - 15

Les chiffres dans le tableau sont exprimés en cycles d'horloge

4.3.3 *QNoC*

4.3.3.1 Introduction

Un réseau *CuNoC* est caractérisé par un taux faible de ressources nécessaires pour son implantation, par une politique d'arbitrage reposant sur une règle de priorité à droite, par une connexion spécifique entre ses routeurs *CU* et les modules de calcul *PE* associés au réseau et par un algorithme de routage permettant l'acheminement des paquets et des messages même dans des conditions de reconfiguration dynamique du réseau. Cependant, un réseau *CuNoC* montre des résultats médiocres pour des conditions de communication ou de transfert de paquets nécessitant une une bande passante élevée entre plusieurs modules de calcul *PE* associés au réseau. De plus, la conséquence d'un trafic très chargé au sein d'un réseau *CuNoC* est son besoin de réordonner l'acheminement des paquets vers leurs destinations augmentant alors la latence de transmission.

Dans le but de réduire ces inconvénients, des améliorations de l'architecture d'un réseau *CuNoC* ont été proposées et ont permis de développer une approche améliorée d'un réseau sur puce pour système reconfigurable baptisé *QNoC* [JTW08a].

Un réseau *QNoC* hérite de la plupart des propriétés d'un réseau *CuNoC*. En effet, le format de messages utilisé dans un réseau *QNoC* est identique à celui utilisé dans un réseau *CuNoC* (voir figure 4.12). De même, les modules de calcul associés à un réseau *QNoC* communiquent

également à travers des messages composés d'un nombre fixe de paquets de données de même format que dans un réseau *CuNoC* [JTW08a]. Chaque paquet contient l'adresse de destination, les informations sur la taille du paquet considéré, son identifiant (*ID*) et les données à transmettre. Un réseau *QNoC* a également hérité à la fois des aspects du réseau *CuNoC* permettant le placement dynamique de modules de calcul et de la procédure de construction d'un réseau d'une taille précise. La différence majeure entre ces deux réseaux de communication *NoC* est la structure de leurs éléments de base, l'architecture des routeurs qui les composent.

4.3.3.2 Architecture d'un routeur du réseau *QNoC - Q-switch*

L'élément de base d'un réseau *QNoC* est le routeur *Q-switch*. Son architecture est illustrée dans la figure 4.39 [JTW08a]. Un routeur *Q-switch* peut être utilisé soit comme une unité de

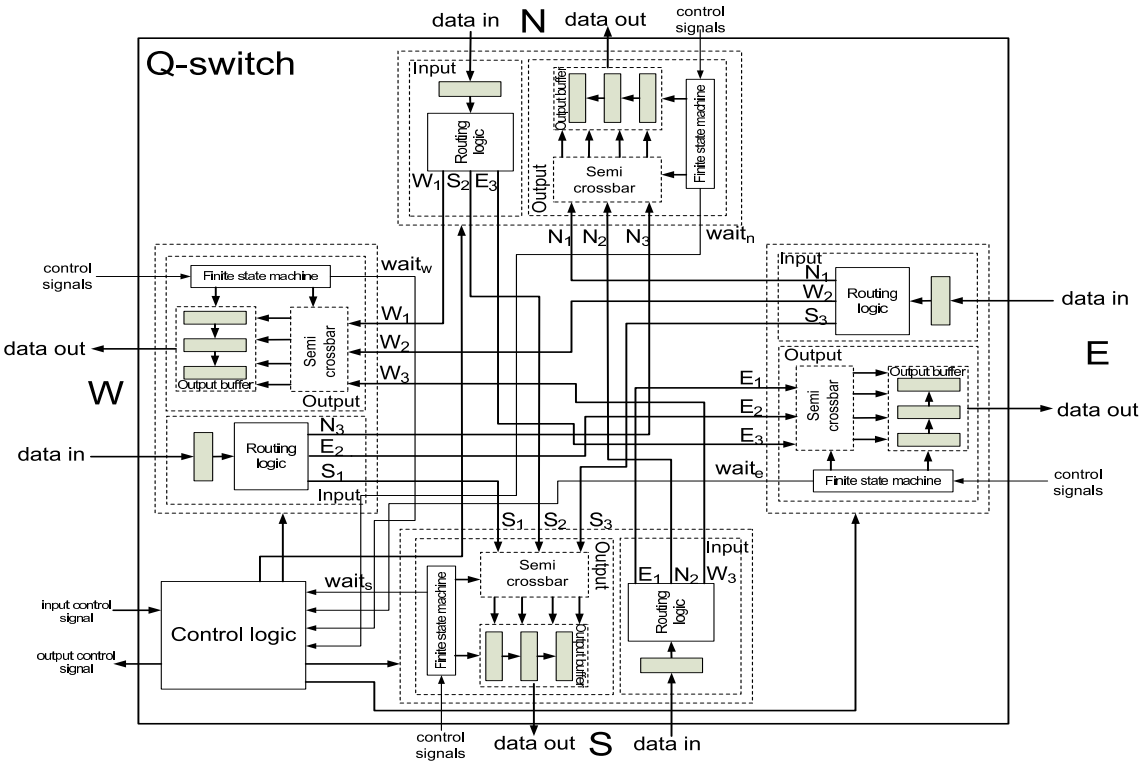


FIG. 4.39 – Architecture du routeur *Q-switch* d'un réseau *QNoC*.

communication indépendante pour l'acheminement des messages à transmettre entre maximum 4 modules de calcul ; soit comme un routeur faisant partie d'un réseau de routeurs de plus grande dimension pour des besoins de communication élevés. Un routeur *Q-switch* possède un registre à chaque port d'entrée. Tous les paquets de données arrivant à un routeur *Q-switch* sont initialement chargés dans ces registres d'entrée. Une fois les paquets reçus et stockés dans ces registres d'entrée, des blocs de calcul appelés « Logique de routage » (*Routing logic*) déterminent leurs directions suivantes. Selon les directions calculées par les blocs « Logique de routage », les

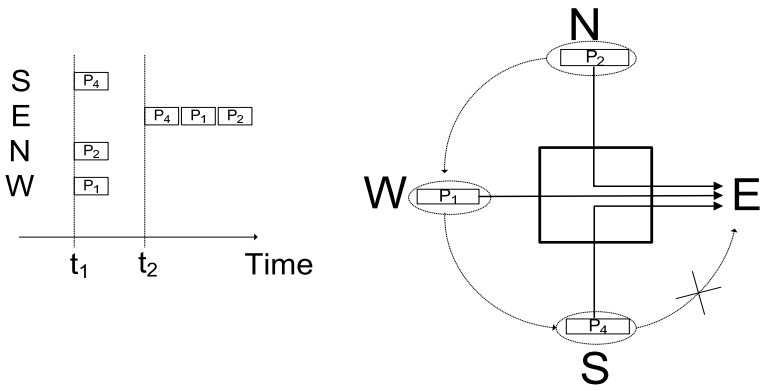


FIG. 4.40 – Politique d'arbitrage basée sur la règle de priorité à droite.

paquets sont acheminés vers des blocs situés aux directions concernées et appelés « Logique de sortie » (*Output logic*). Ces blocs de sortie arbitrent l'ordre de passage de tous les paquets arrivant au même port de sortie. Par exemple, lorsque 3 paquets arrivent simultanément de 3 différentes directions pour être acheminés vers la même sortie, ces 3 paquets doivent donc partager la même sortie. C'est le rôle du bloc « Logique de sortie » d'assurer le bon déroulement de passage de tous les paquets ayant été acheminés vers la sortie considérée. La bloc « Logique de sortie » repose sur la même politique d'arbitrage que celle utilisée dans un réseau *CuNoC*. L'arbitrage est donc basé sur la règle de priorité à droite. Ce mécanisme d'arbitrage dans le cas de 3 paquets arrivant simultanément à un port de sortie est illustré dans la figure 4.40 [JTW08a]. Chaque port de sortie applique l'arbitrage basé sur la priorité à droite mais adapté à sa position au sein du routeur. Par exemple, le port situé à la direction *East* peut prendre uniquement les paquets arrivant des directions *North*, *West* et *South*. Si on applique la règle de priorité à droite sur ces entrées, on a comme résultat que les paquets venant de la direction *South* ont la priorité la plus élevée, ensuite ceux venant de la direction *West* et enfin les paquets venant de la direction *North*. La même procédure est appliquée pour chaque port d'un routeur *Q-switch*. Les priorités ainsi définies sont utilisées dans les blocs de « Logique de sortie » des ports du routeur.

Comme pour un routeur *CU* d'un réseau *CuNoC*, un routeur *Q-switch* utilise la technique d'aiguillage *store-and-forward* [NM93]. Cette technique d'aiguillage introduit une latence additionnelle par routeur *Q-switch* de 2 cycles d'horloge. La figure 4.41 illustre la latence introduite par un *Q-switch* dans 2 situations [JTW08a]. La première situation correspond à un routeur *Q-switch* recevant à l'instant $(t + T)$ 4 paquets provenant de 4 directions différentes et chacun possédant différente adresse de destination. A l'issu de la latence introduite par le routeur et équivalente de 2 cycles d'horloge, tous les paquets sont transféré vers des routeurs voisins à l'instant $(t + 3T)$. La deuxième situation illustre la réception de 4 paquets parmi lesquels 3 possédant la même adresse de destination. Ces paquets sont reçus à un instant $(t + 5T)$ et sont transmis à partir de l'instant $(t + 7T)$. La séquence de transmission de paquets est la suivante : le paquet de priorité de passage la plus élevée passe à l'instant $(t + 7T)$, puis les deux paquets

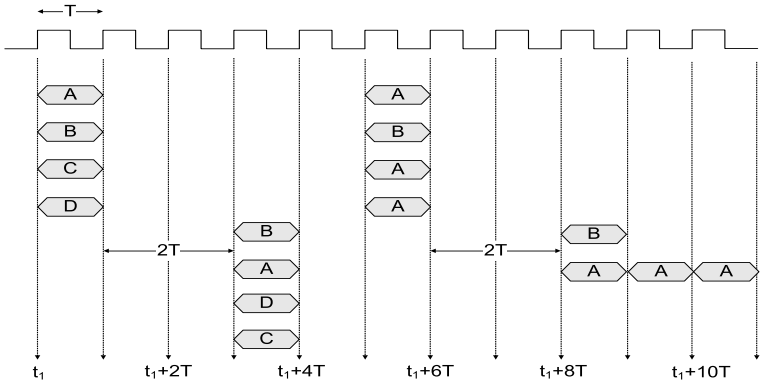


FIG. 4.41 – Illustration de la latence introduite par un routeur *Q-switch*.

suivants sont transmis successivement après une latence dans le routeur d'un cycle d'horloge supplémentaire. Ces deux paquets, quittent donc le routeur dans les 2 cycles d'horloge suivant. L'approche d'aiguillage basée sur la technique *store-and-forward* est la mieux adaptée pour un réseau *QNoC*. En effet, la principale raison pour laquelle un réseau *QNoC* ne repose pas sur la technique d'aiguillage *wormhole routing* (couramment utilisée dans les réseaux sur puce) est dû au fait que la structure des paquets de données à transmettre avec cette technique est composée de *flits*. Ces derniers peuvent « occuper » plusieurs routeurs et entraîner des blocages avant même qu'un paquet de données ne soit pas transféré dans son intégrité vers sa destination. Cette approche n'est donc pas adaptée pour un réseau reconfigurable tel que *QNoC*, car tous les routeurs doivent être disponibles à tout moment ou dans un délai court pour être remplacés par des modules de calcul dynamiques par insertion au sein d'un réseau au cours de son fonctionnement.

4.3.3.3 Architecture structurelle d'un bloc « Logique de sortie »

Un bloc de « Logique de sortie » d'un port de routeur *Q-switch* est composée d'un *semi-crossbar*, d'un buffer de sortie et d'une machine d'états. La figure 4.42 illustre son architecture [JTW08a]. Le rôle du *semi-crossbar* est d'aiguiller 3 entrées issues des trois autres directions vers 4 sorties. Les entrées sont organisées selon les priorités. L'entrée étant numérotée par le chiffre 1 a la priorité la plus élevée, l'entrée 2 a la priorité moins élevée et l'entrée 3 la priorité la moins élevée. Le premier port d'entrée du *semi-crossbar* peut être aiguillé uniquement vers deux premières sorties. La première sortie du *crossbar* est, si le routeur voisin n'est pas occupé, directement reliée à la sortie du routeur. La deuxième sortie du *crossbar* est reliée au premier registre du buffer de sortie, la troisième sortie est reliée au second registre du buffer et la quatrième au dernier registre du buffer de sortie. La deuxième entrée du *crossbar* peut être aiguillée vers trois premières sorties du *crossbar*, tandis que la troisième entrée peut être aiguillée vers toutes les sorties.

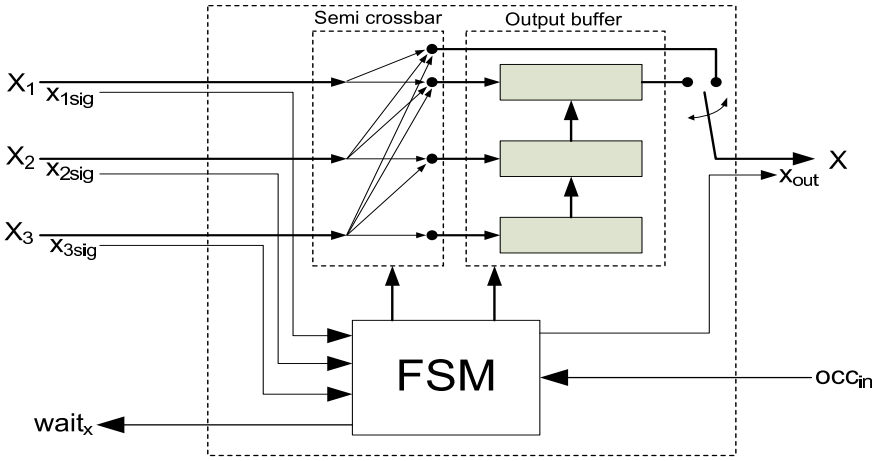


FIG. 4.42 – Bloc « Logique de sortie » d'un routeur *Q-switch*.

Le buffer de sortie est composé de trois registres de taille identique et égale à la taille des paquets. Les registres sont utilisés pour stocker temporairement les paquets avant leur transfert. Dans les situations où plusieurs paquets arrivent aux entrées d'un bloc « Logique de sortie », le paquet qui a la priorité la plus élevée sera directement transféré vers la sortie du routeur si le routeur voisin est disponible au moment du transfert. Les disponibilités des routeurs voisins sont indiquées au routeur considéré à travers les signaux de contrôle occ_{in}. Les autres paquets de priorités moins élevées sont stockés temporairement dans le buffer de sortie pour ensuite être transférés individuellement et successivement après le transfert du premier paquet. Le buffer de sortie est également utilisé pour stocker provisoirement des paquets (au maximum 3) arrivant depuis une même direction et ne pouvant pas être directement transférés pour cause d'indisponibilité du routeur voisin de réception.

L'ensemble des fonctions d'un bloc « Logique de sortie » est pilotée par une machine d'état. Cette machine génère tous les signaux de contrôle et assure une gestion du routeur permettant d'éviter toute sorte de collision entre les paquets. Cette machine d'état est reliée à la logique de contrôle centrale du routeur à travers un signal de contrôle (signal $wait$) pour l'informer des indisponibilités éventuelles des routeurs voisins (voir figure 4.39).

4.3.3.4 Logique de contrôle centrale

Les sorties du *Q-switch* sont pilotées par les logiques de contrôle des sorties. Ces blocs de logique de contrôle sont suffisants pour organiser les transferts des paquets, mais seulement dans les situations où les routeurs voisins ne sont pas occupés. Dans le cas contraire, un bloc de contrôle supplémentaire est nécessaire pour gérer ce genre de situations.

Le *Q-switch* a à chaque port des signaux de contrôle d'entrée qui lui donnent les informations sur l'occupation de ses voisins directs, et de sortie avec lesquels il indique à ses routeurs voisins son état d'occupation. Les situations d'occupation se produisent dans les cas où plu-

sieurs paquets arrivent simultanément aux entrées d'une logique de sortie. Dans ces situations, le transfert de plusieurs paquets n'est pas possible dans un cycle d'horloge. La logique de contrôle centrale gère ces signaux d'occupation de sortie. En effet, elle positionne les signaux d'occupation de sortie à '1' vers les routeurs voisins depuis lesquels les paquets sont arrivés, pour les informer que le *Q-switch* considéré est temporairement indisponible et qu'il ne peut plus accepter d'autres paquets venant de ces directions. Après avoir transféré tous les paquets, la logique de contrôle centrale remet à zéro ces signaux d'occupation de sortie et le *Q-switch* est de nouveau disponible pour recevoir des paquets de ces directions.

La logique de contrôle centrale gère également le stockage des paquets qui, faute d'occupation des routeurs voisins, restent bloqués dans le routeur. Ces paquets sont, soit stockés dans les registres d'entrée, soit dans les buffers de sortie. Dans les deux cas, la logique de contrôle centrale positionne à '1' les signaux d'occupation de sortie vers les routeurs voisins correspondant et leur indique l'indisponibilité provisoire. Une fois tous les paquets sont transférés du routeur vers ses voisins, le routeur redevient disponible pour la réception d'autres paquets de ces directions. Ceci est indiqué par la remise à zéro des signaux d'occupation de sortie.

4.3.3.5 Algorithme de routage

Un réseau *QNoC* utilise un algorithme de routage basé sur un algorithme *XY* modifié. En effet, l'algorithme *XY* ne peut pas être utilisé pour le routage des paquets dans sa version initiale pour un réseau reconfigurable, car il n'est pas adapté à des structures irrégulières du réseau qui peuvent se produire lors d'un placement dynamique des modules de calcul au sein du réseau. En effet, pour certaine structure irrégulière des routeurs d'un réseau reconfigurable, le routage de paquets de données basé sur l'algorithme *XY* est impossible.

Comme pour l'algorithme *XY* classique [DYN02], avec l'algorithme de routage développé pour un réseau *CuNoC* et *QNoC*, les paquets de données sont premièrement acheminés selon l'axe *X* de direction du réseau (réseau maillé *2D*). Ensuite, ces paquets sont acheminés vers leurs destinations finales selon l'axe *Y* du réseau. Si au cours du processus de routage selon les axes *X* et *Y* du réseau, un paquet rencontre des modules dynamiques de calcul obstruant son cheminement dans le réseau, l'algorithme de routage utilisé permet leurs contournements. Pour cela, une *logique de contrôle locale* à chaque routeur permet la distinction du type d'entité (routeur ou module de calcul) connectée à un routeur. Ainsi, si un routeur est connecté directement à un module de calcul ou à un autre routeur *Q-switch*, le routeur *Q-switch* considéré analyse ses signaux de contrôle d'entrée permettant de lui indiquer la nature de ses entités voisines (routeur ou module). Ces signaux de contrôle contribuent et facilitent le routage au sein d'un réseau *QNoC*. En fonction des résultats de cette analyse locale de chaque routeur, l'algorithme proposé permet le contournement de module dynamique dans le réseau à partir d'une approche de routage de type *XY*. De plus, pour éviter des situations de blocage pouvant arriver dans les réseaux *NoC*,

certaines conditions de direction de transmission de paquets à partir d'une direction d'entrée dans un routeur (sens direction - *turn*) ne sont pas autorisées (algorithme de type *turn*). Contrairement à l'algorithme développé pour le réseau *CuNoC*, l'algorithme de routage proposé pour le réseau *QNoC* résout également l'inconvénient de l'ordre d'arrivée des paquets à un nœud du réseau. En effet, si au cours du fonctionnement du réseau, il n'y a pas de placements dynamiques entre deux modules de calcul, les chemins empruntés par les paquets envoyés à partir d'un module vers un autre module destinataire, seront identiques et de même longueur (même situation que l'algorithme *XY*), conservant alors l'ordre d'émission et de réception des paquets transmis. Par contre, les paquets d'un message envoyés par d'un module de calcul, peuvent être imbriqués à la destination avec les paquets d'autres messages envoyés par d'autres modules de calcul du réseau.

4.3.3.6 Conditions de placement des modules dynamiques de calcul dans un réseau *QNoC*

La construction d'un réseau *QNoC* est identique à celle d'un réseau *CuNoC*, présenté précédemment. Les éléments du réseau sont instanciés dans une surface reconfigurable. Dans une première étape, les routeurs du réseau *QNoC* sont placés dans la zone reconfigurable. Ensuite, des modules de calcul sont localement placés au sein du réseau par substitution de routeurs initialement placés. Si le placement d'un module nécessite la substitution de plusieurs routeurs du réseau, alors le module nouvellement placé hérite de toutes les adresses des routeurs substitués. Les règles de placement définies au sein d'un réseau *CuNoC* sont conservées pour un réseau *QNoC*.

La procédure de placement dynamique des modules de calcul dans un réseau *QNoC* est également identique à celle décrite dans un réseau *CuNoC* (voir section 4.3.2.5). Ainsi, avant le placement dynamique d'un module de calcul, le ou les routeurs *Q-switch* de substitution vident leurs buffers et se mettent en mode « module », afin de ne plus accepter de paquets de données par leurs routeurs voisins. D'autre part, ces derniers prennent en compte le nouveau *statut* du ou des routeurs de substitution pour le routage de leurs paquets de données suivants en considérant la zone des routeurs substitués comme une zone obstacle à contourner dans le réseau.

Un réseau *QNoC* permet également une structuration hétérogène de son réseau à travers les placement multiples et possibles de modules de calcul au sein du réseau. Les règles de placement de modules ainsi que le nombre maximal de modules de calcul pouvant être placés dans un réseau *QNoC* sont similaires à celles d'un réseau *CuNoC*.

4.3.3.7 Implantation et résultats de synthèse d'un réseau *QNoC*

Une synthèse et implantation des routeurs *Q-switch* pour de différents formats de paquets de données dans les technologies *FPGA Xilinx Virtex II, IV* et *V* ont été réalisées. Ces résultats sont

donnés dans le tableau 4.5 en termes des ressources logiques (blocs logiques *CLB*) et de fréquence de fonctionnement (fréquence maximale en MHz). Les résultats d'implantation obtenus s'expriment uniquement en termes de blocs *CLB* utilisés car les ressources annexes et disponibles dans un *FPGA* (blocs mémoires *BRAM*, etc) ne sont pas exploitées lors de la synthèse. L'implantation d'un routeur *Q-switch* de taille de données de 8 bits nécessite des ressources logiques de 309 *CLB* et permet une fréquence de fonctionnement maximale sur un *FPGA Virtex IV* de 257.2 MHz. Par comparaison avec un routeur *CU* d'un réseau *CuNoC* pour la même taille de format de données, un routeur *Q-switch* nécessite 6 fois plus de ressources logiques (309/50) et une fréquence maximale de fonctionnement deux fois inférieure (549.3/257.2). Une comparaison plus détaillée en terme de performance entre un réseau *CuNoC* et *QNoC* de même dimension est présentée dans les sections suivantes.

TAB. 4.5 – Résultats de synthèse de routeurs *Q-switch* de formats de données différents sur les technologies *FPGA Xilinx Virtex II, Virtex IV* et *Virtex V*.

	Q-switch					
format de données	Virtex II		Virtex IV		Virtex V	
	f [MHz]	CLB Slices	f [MHz]	CLB Slices	f [MHz]	CLB Slices
8 bit	227.4	299	257.2	309	304.3	129
16 bit	207.8	394	234.9	473	294.1	203
24 bit	204.0	512	233.3	640	293.1	276
32 bit	200.8	638	232.0	808	261.1	349

Une synthèse et implantation de réseaux *QNoC* de différente taille sur les technologies *FPGA Xilinx Virtex IV* et *Virtex V* ont également été réalisées. Ces résultats sont présentés dans le tableau 4.6 pour chaque taille de réseau en termes de blocs logiques (*CLB*) et de performance en termes de fréquences maximales de fonctionnement. Pour un réseau *QNoC* de dimension 3×3 utilisant un format de données de 32 bits, son implantation nécessite des ressources logiques *FPGA* d'environ 3119 *CLB* et à une fréquence maximale de fonctionnement de l'ordre de 260 MHz pour un *FPGA* de la famille *Virtex V*. L'ensemble des résultats de synthèse et d'implantation présentés ont été obtenus à l'aide de l'outil *Precision RTL Synthesis* [PRE].

TAB. 4.6 – Résultats de synthèse de réseaux *QNoC* de taille et de formats de données différents sur les technologies *FPGA Xilinx Virtex II, Virtex IV* et *Virtex V*

	QNoC 2x2						QNoC 3x3					
data width	Virtex II		Virtex IV		Virtex V		Virtex II		Virtex IV		Virtex V	
	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB
8 bit	182.0	1217	218.1	1243	270.6	496	182.4	2747	218.7	2833	244.0	1128
16 bit	177.9	1544	196.9	1904	261.2	796	181.1	3472	212.3	4310	255.0	1796
24 bit	156.7	1953	203.9	2579	260.2	1087	178.3	4392	203.9	5824	254.3	2461
32 bit	155.5	2384	199.2	3241	255.8	1311	173.6	5362	205.2	7317	258.7	3119

4.3.3.8 Évaluation des performances d'un réseau *QNoC*

La bande passante (*throughput*) maximale d'un routeur *Q-switch* de format de données de n bits, fonctionnant à une fréquence f et permettant une communication simultanée d'un nombre de modules de calcul maximal (4 modules situés à chaque direction d'un routeur *Q-switch*) est définie par les expressions suivantes :

$$Throughput_{max}[\,Gbps\,] = 4 * n * f \tag{4.6}$$

$$Throughput_{max}[\,paquet\,/\,clk\,] = 4 \tag{4.7}$$

Par exemple, dans le cas d'un routeur *Q-switch* de format de données 8 bits, connecté à 4 modules de calcul et ayant une fréquence de fonctionnement de 100 MHz, la bande passante du routeur est de 3.2 Gbps.

Pour d'autres formats de données (jusqu'à 32 bits), les résultats d'évaluation de la bande passante d'un routeur *Q-switch* sont présentés dans la figure 4.43. Ces résultats sont obtenus en prenant en compte les fréquences de fonctionnement maximales dans les technologies *FPGA Virtex II*, *IV* et *V*, des routeurs pour différents formats de données (voir tableau 4.5). Ainsi, l'implantation d'un routeur de format de données 32 bits dans un *FPGA Virtex V* de fréquence de fonctionnement maximale de 261.1 MHz, fournit une bande passante évaluée à 33 Gbps.

La figure 4.44 illustre les performances d'un routeur *Q-switch* pour plusieurs conditions de trafic de paquets de données. Ces performances sont données à la fois en termes de latences moyenne et maximale, et de bande passante maximale. Ces conditions d'évaluation sont identiques à celles d'un réseau *CuNoC* : un trafic *aléatoire*, où chaque *PE* envoie aléatoirement un paquet de données vers un autre *PE* du réseau *QNoC* ; un trafic *défini*, où chaque *PE* connecté

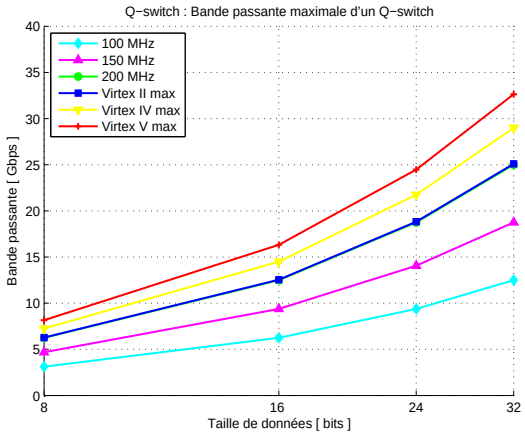
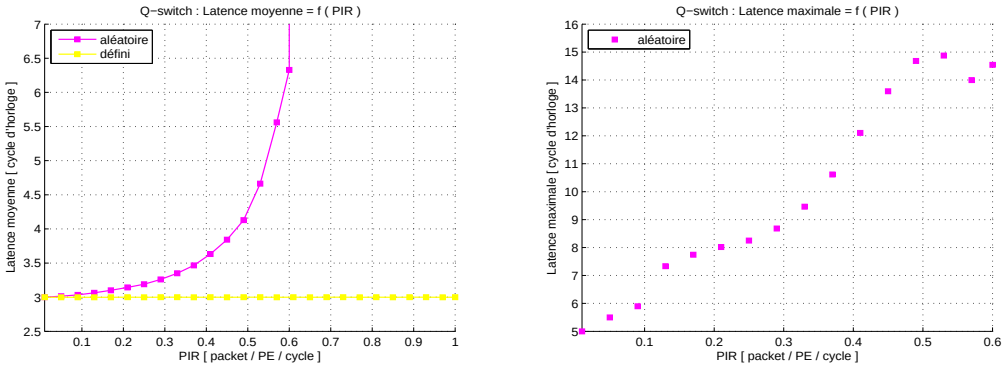


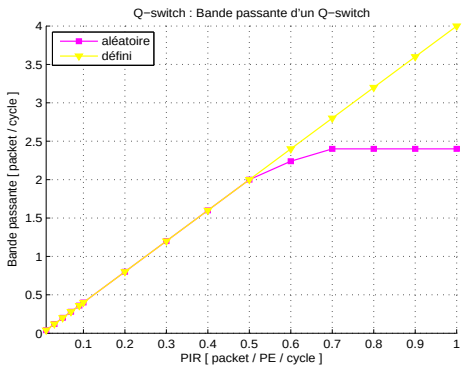
FIG. 4.43 – Bande passante d'un routeur *Q-switch* pour différents formats de données de 4 à 32 bits.

à un routeur *Q-switch* envoie un paquet de données vers un *PE* voisin connecté à la direction opposée dans le réseau. Une analyse des résultats présentés dans la figure 4.44 indique que le taux d'injection de paquets maximal (*PIR*) d'un routeur *Q-switch* est de l'ordre de 0.6 et 1.0 respectivement pour les trafic *aléatoire* et *défini*. On remarque que pour le type de trafic *défini*, le *PIR* atteint sa valeur maximale, principale conséquence de l'absence de chemins communs empruntés par les paquets pour ce type de trafic. On note également que la latence moyenne d'un routeur *Q-switch* varie de 3 à 8 cycles d'horloge en fonction principalement de la valeur de *PIR*.

Une évaluation des performances de réseaux *QNoC* de différente taille : 2×2, 3×3, 4×4 et 5×5 a été réalisée. Ces résultats sont présentés dans les figures 4.45 - 4.51. Cette procédure d'évaluation est identique à celle d'un réseau *CuNoC* (voir section 4.3.2.9). L'évaluation repose sur des générateurs de trafic (*TG*) émulant des envois ou réceptions de modules de calcul et modélisés en langage *C*. L'ensemble routeur-réseau et générateur de trafic modélisant des modules de calcul dynamiques ou non est simulé sous environnement de co-simulation *VHDL-C*. Dans une simulation, chaque *TG* envoie des paquets à d'autres *TG* connectés au réseau selon un type

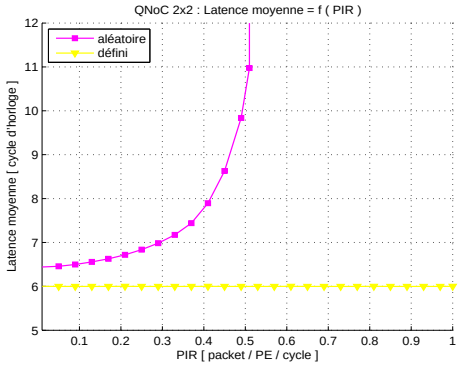


(a) latence moyenne (b) latence maximale

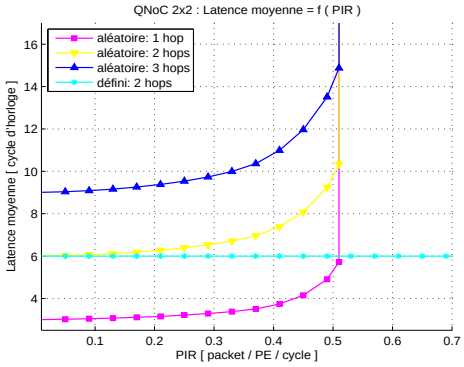


(c) bande passante

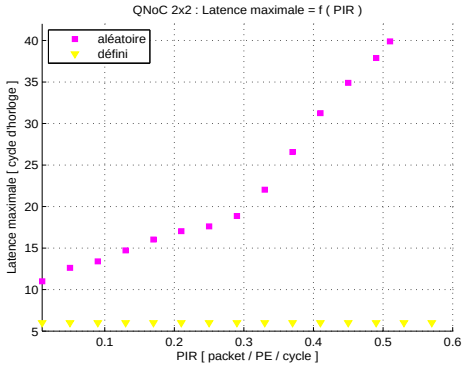
FIG. 4.44 – Performances d'un routeur *Q-switch*.



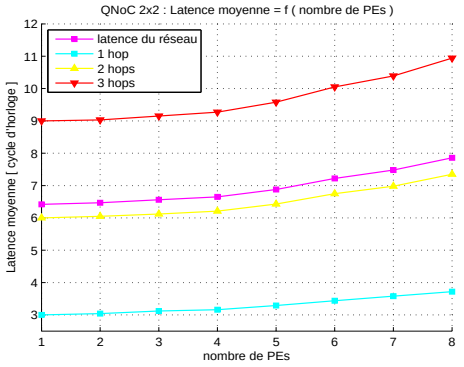
(a) latence moyenne = f (*PIR*)



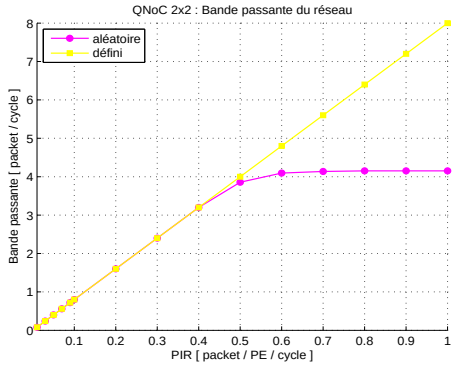
(b) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le réseau



(c) latence maximale = f (*PIR*)



(d) latence moyenne = f (*PE*)



(e) bande passante = f (*PIR*)

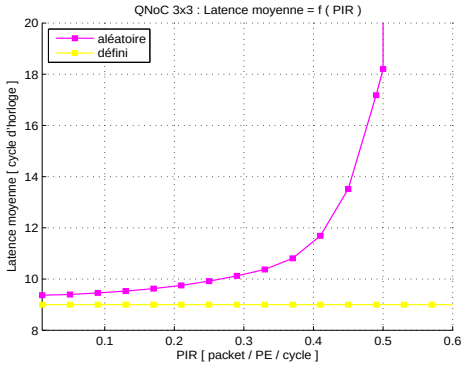
FIG. 4.45 – Résultats de l'évaluation de performances d'un réseau *QNoC* 2x2.

de trafic choisi (*aléatoire* ou *défini*). Le nombre total de paquets de données envoyé par chaque *TG* pour tous les cas considérés est de 125 000 paquets. Comme pour l'évaluation d'un réseau *CuNoC*, pour toutes les évaluations de performances d'un réseau *QNoC*, deux structures topologiques d'un réseau *QNoC* sont considérées : une structure *homogène* dans laquelle tous les *TG*

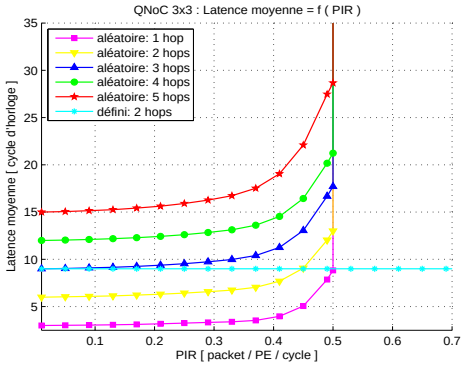
sont disposés en périphérie du réseau ; une structure *hétérogène* correspondant à l'instanciation d'un module dynamique (un générateur *TG* substituant un routeur *Q-switch* dans le réseau) au sein d'un réseau *QNoC* de structure *homogène*.

Les résultats de cette évaluation sont présentés dans les figures 4.45 - 4.51. Ces figures montrent les performances d'un réseau *QNoC* de différente taille à la fois en termes de latence moyenne d'un réseau en fonction du *PIR* et du nombre de générateurs *TG* ; la latence maximale d'un réseau en fonction du *PIR* ; la latence moyenne de différentes distances (*hops*) possibles entre les générateurs *TG* dans un réseau en fonction du *PIR* ; la bande passante d'un réseau en fonction du *PIR* ; ainsi que la distribution du trafic pour les réseaux *QNoC* de taille 3×3, 4×4 et 5×5. Ces résultats de simulation montrent clairement que pour augmentation de la taille d'un réseau, le *PIR* maximal d'un module de calcul *PE* (ou d'un générateur *TG*) ne dépasse pas la valeur maximale de *PIR* d'un routeur *Q-switch* (voir figure 4.44). En effet, l'augmentation de la taille d'un réseau et du nombre de *PE* (*TG*) connectés au réseau engendre un accroissement des délais de transmission des paquets dû aux changements des conditions qui deviennent plus « difficiles » pour la transmission de paquets dans le réseau. La valeur maximale de *PIR* d'un *PE* pour toutes les tailles de réseaux *QNoC homogènes* simulés est située dans une plage de 0.4 à 0.5 paquets par cycle d'horloge. L'augmentation de la taille d'un réseau montre que la bande passante totale du réseau croît selon la règle définie pour des réseaux sur puce directs. Plus précisément, la bande passante croît de manière linéaire et s'exprime comme la multiplication de nombre de module *PE* connectés au réseau par leur *PIR* respectifs. Ainsi, les bandes passantes maximales sont de l'ordre de 4, 6, 8 et 10 paquets par cycle d'horloge pour les tailles de réseaux respectives 2×2, 3×3, 4×4 et 5×5.

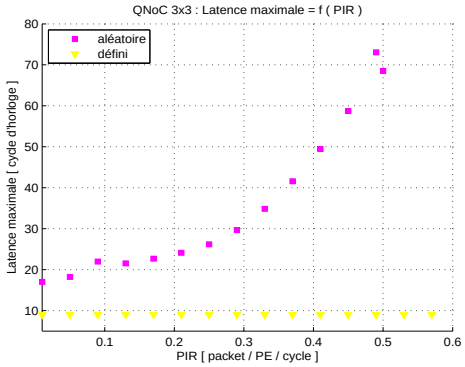
Une évaluation des performances de réseaux *QNoC hétérogènes* de taille 3×3, 4×4 et 5×5 a également été réalisée. Ces réseaux hétérogènes résultent de la substitution d'un routeur central des réseaux par un générateur *TG*. Comme pour l'évaluation d'un réseau *hétérogène CuNoC*, trois cas d'études sont considérés : un cas nommé « obstacle » dans lequel le module dynamique *TG* placé dans le réseau se comporte comme un module obstacle ne recevant et ne transmettant aucun paquet de données vers les autres modules *TG* du réseau ; un cas nommé « réception » dans lequel le module dynamique *TG* placé peut uniquement recevoir des paquets des autres modules statiques *TG* du réseau (le module dynamique n'effectue aucun envoi ou transfert de paquets) ;et un cas d'étude nommé « réception + envoi » dans lequel le module dynamique *TG* introduit dans le réseau peut à la fois transmettre et recevoir des paquets de données vers ou depuis des autres *TG* statiques du réseau. Dans ces trois cas de simulation, le routeur *Q-switch* substitué par un *TG* dynamique dans les réseaux homogènes *QNoC* considérés de taille 3×3, 4×4 et 5×5, est situé respectivement dans les positions $(2,2)$, $(3,3)$ et $(4,4)$. Les résultats de simulation de ces réseaux hétérogènes, montrent que pour la substitution d'un routeur *Q-switch* par un *TG*, émulant le comportement d'un réseau *QNoC* lors de l'instanciation



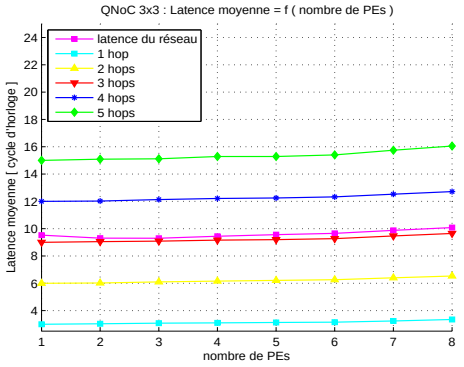
(a) latence moyenne = f (*PIR*)



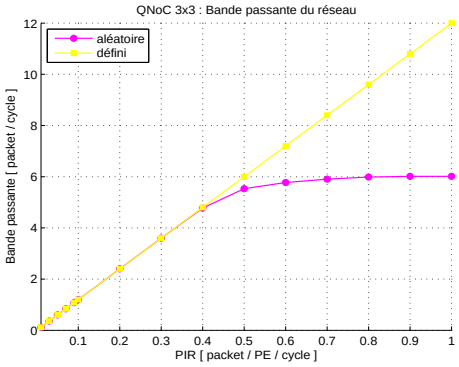
(b) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le réseau



(c) latence maximale = f (*PIR*)



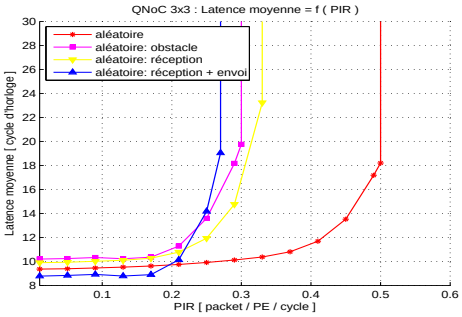
(d) latence moyenne = f (*PE*)



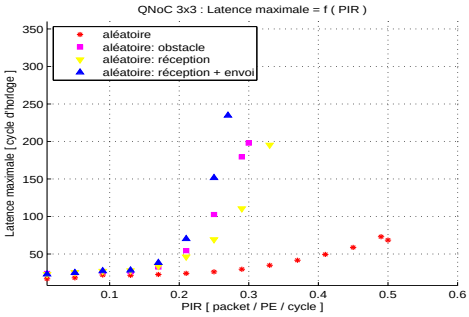
(e) bande passante = f (*PIR*)

FIG. 4.46 – Résultats de l'évaluation de performances d'un réseau *QNoC* 3x3.

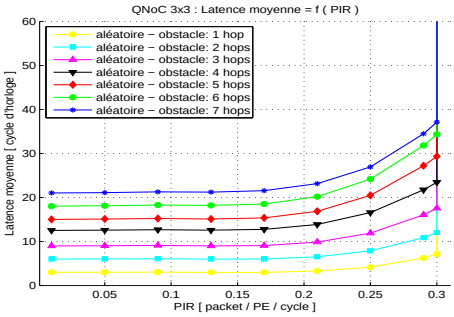
d'un module dynamique, les performances des réseaux se dégradent. Ces résultats sont présentés respectivement dans les figures 4.47, 4.49 et 4.51 pour les réseaux *QNoC* 3×3, 4×4 et 5×5. Comme lors de l'évaluation d'un réseau *CuNoC*, le cas d'étude « réception + envoi » pour toutes les trois dimensions de réseau *QNoC* montre les plus faibles performances étant donné le



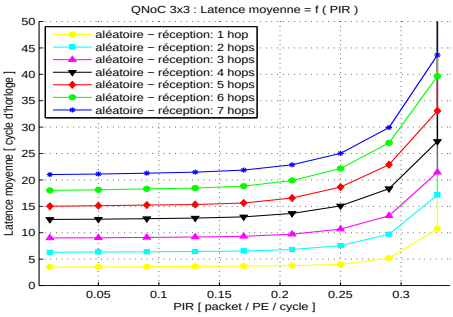
(a) latence moyenne = f (*PIR*)



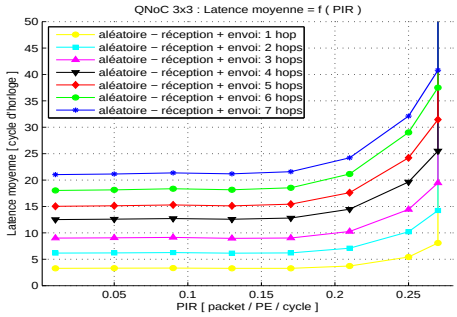
(b) latence maximale = f (*PIR*)



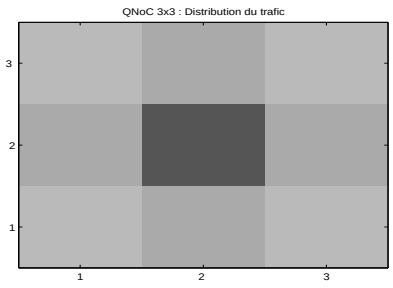
(c) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « obstacle »



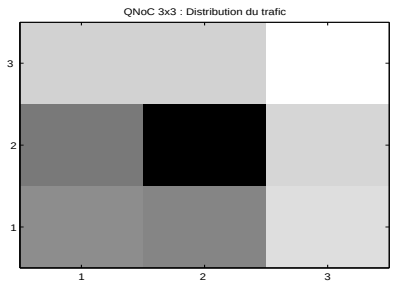
(d) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception »



(e) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception + envoi »

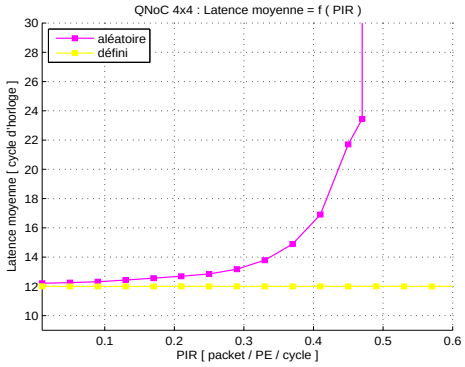


(f) « homogène »

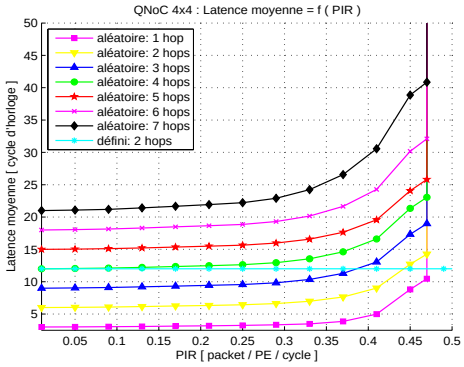


(g) « hétérogène »

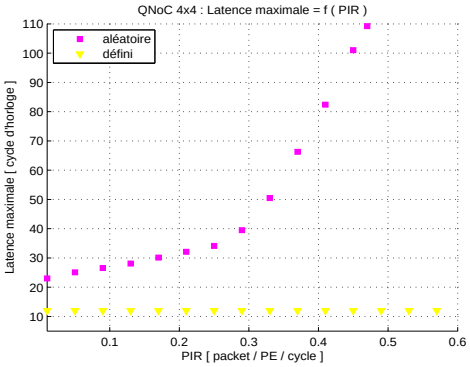
FIG. 4.47 – Résultats de l'évaluation de performances d'un réseau *hétérogène QNoC* 3x3 et distributions de trafic avec ou sans module dynamique *TG* placé.



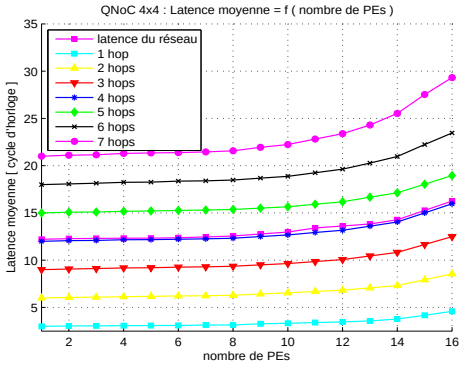
(a) latence moyenne = f (*PIR*)



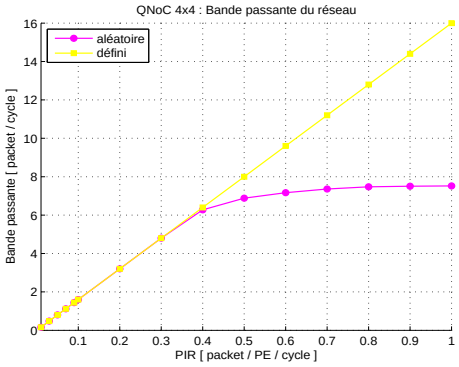
(b) latence moyenne de toutes les distances (*hop*) entre les *TG* dans le réseau



(c) latence maximale = f (*PIR*)



(d) latence moyenne = f (*PE*)

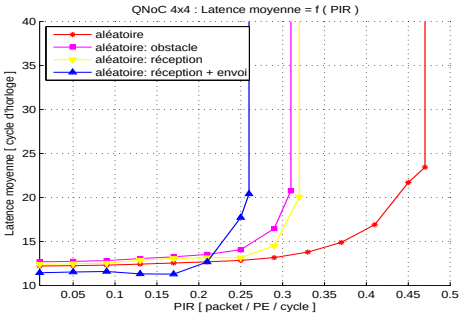


(e) bande passante

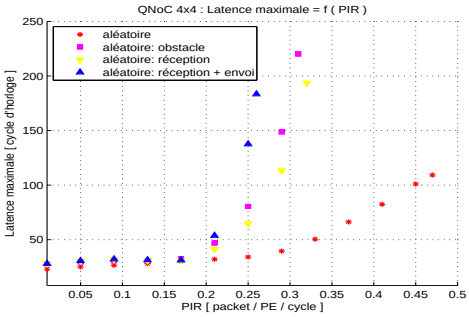
FIG. 4.48 – Résultats de l'évaluation de performances d'un réseau *QNoC* 4x4.

trafic supplémentaire généré par le module dynamique *TG* placé au centre du réseau.

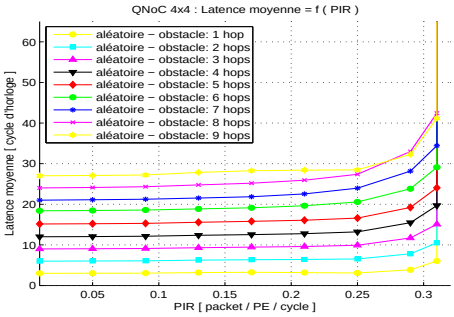
L'ensemble de ces résultats d'évaluation montre également les meilleures performances pour les réseaux *QNoC* homogènes. En effet, ces performances sont principalement dû à l'absence d'obstacles dans l'acheminement des paquets de données dans les réseaux homogènes



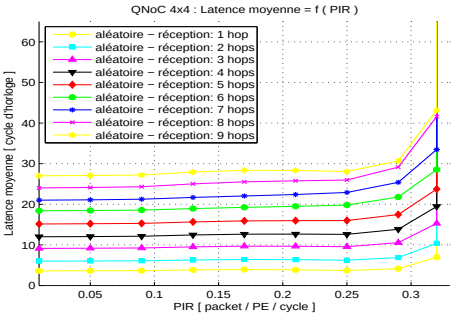
(a) latence moyenne = f (*PIR*)



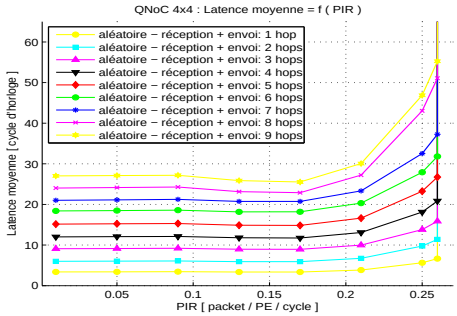
(b) latence maximale = f (*PIR*)



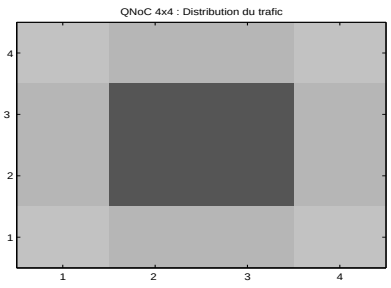
(c) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « obstacle »



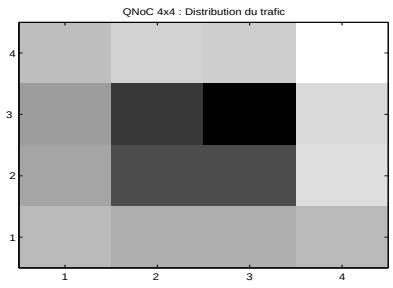
(d) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception »



(e) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception + envoi »

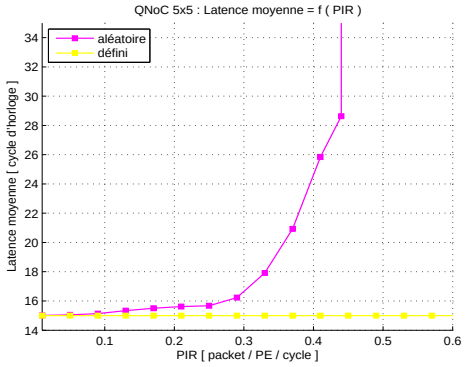


(f) « homogène »

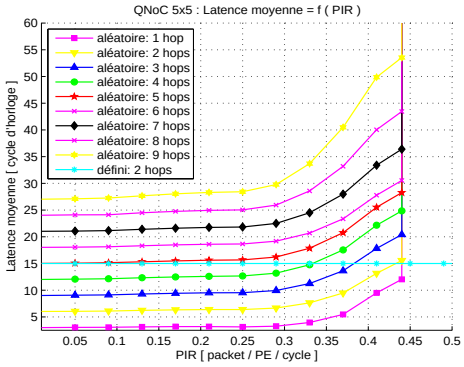


(g) « hétérogène »

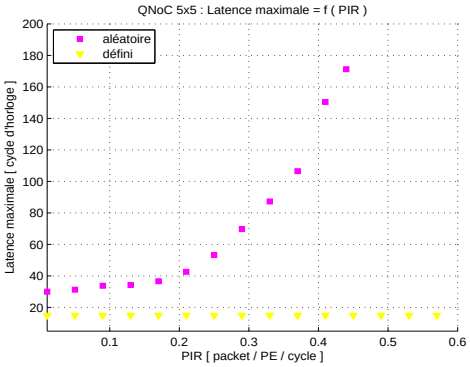
FIG. 4.49 – Résultats de l'évaluation de performances d'un réseau *hétérogène QNoC* 4x4 et distributions de trafic avec ou sans module dynamique *TG* placé.



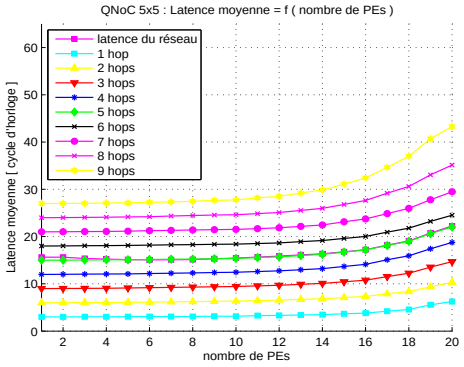
(a) latence moyenne = f (*PIR*)



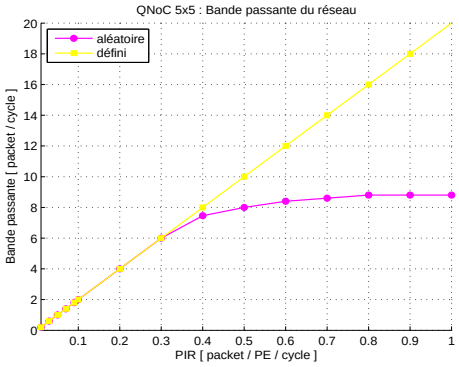
(b) latence moyenne de toutes les distances (*hop*) entre les *TG* dans le réseau



(c) latence maximale = f (*PIR*)



(d) latence moyenne = f (*PE*)

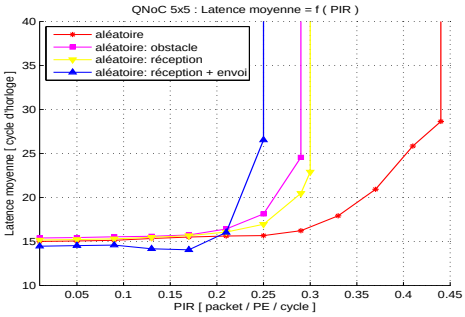


(e) bande passante

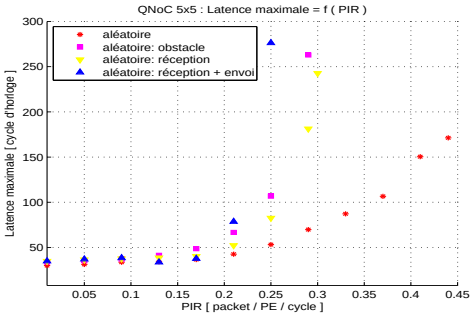
FIG. 4.50 – Résultats de l'évaluation de performances d'un réseau *QNoC* 5x5.

entre les modules statiques *TG* connectés en périphérie de ces réseaux. Des résultats comparatifs entre les réseaux *homogènes* et *hétérogènes* pour les tailles de réseau 3×3, 4×4 et 5×5 sont présentés respectivement dans les figures 4.47a, 4.49a et 4.51a.

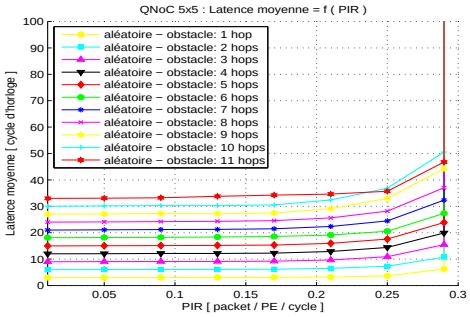
Une évaluation de la distribution de trafic dans les réseaux *QNoC* de taille 3×3, 4×4 et



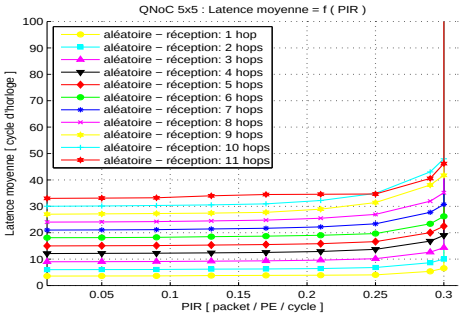
(a) latence moyenne = f (*PIR*)



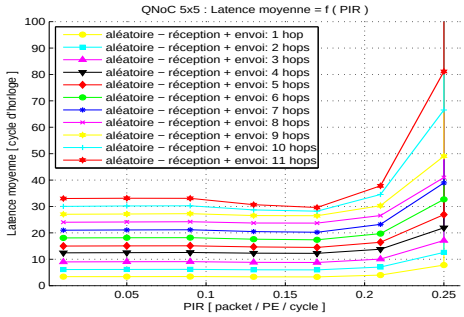
(b) latence maximale = f (*PIR*)



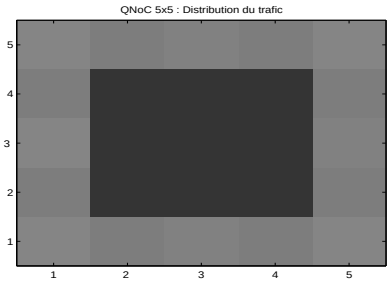
(c) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « obstacle »



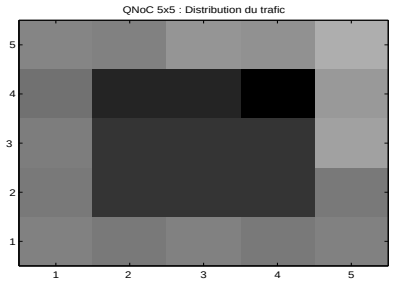
(d) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception »



(e) latence moyenne de toutes les distances (*hops*) entre les *TG* dans le cas « réception + envoi »



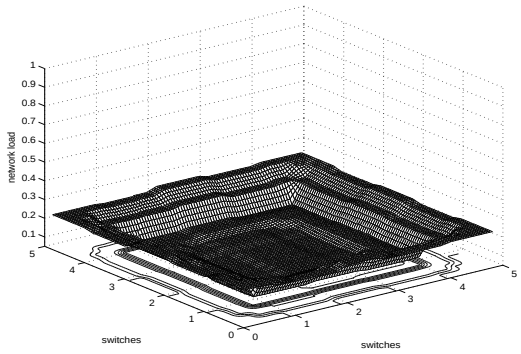
(f) « homogène »



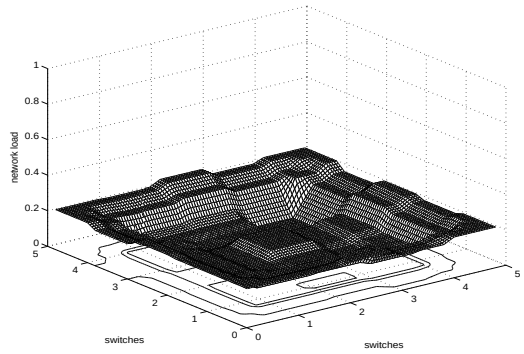
(g) « hétérogène »

FIG. 4.51 – Résultats de l'évaluation de performances d'un réseau *hétérogène QNoC* 5x5 et distributions de trafic avec ou sans module dynamique *TG* placé.

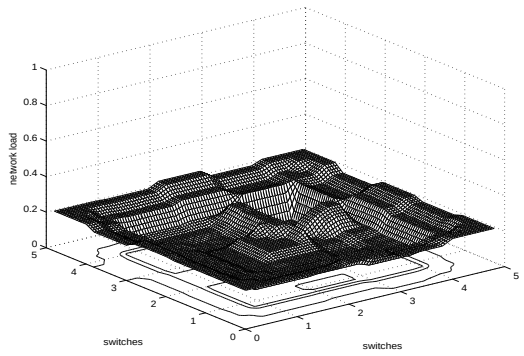
5×5 à la fois pour les cas avec ou sans module dynamique *TG* placé dans ces réseaux, a été également réalisée. Les résultats de simulation de cette évaluation pour les réseaux de taille 3×3, 4×4 et 5×5 sont respectivement présentés dans les figures 4.47f-g, 4.49f-g et 4.51f-g. Les zones claires de ces distributions correspondent à un trafic chargé dans le réseau, tandis que les zones foncées dans le réseau correspondent à un trafic moins dense. Par exemple, la figure 4.47f montre la distribution du trafic d'un réseau *homogène QNoC* 3×3. De ces résultats, il peut être conclut que pour toutes les tailles de réseaux homogènes, les routeurs situés dans chaques « coin » d'un réseau ont des trafics les plus chargés parmi l'ensemble des routeurs des réseaux. En effet, la disposition de ces routeurs dans ces réseaux fait qu'ils sont associés ou connectés à une paire de modules statiques *TG*. Réciproquement, les routeurs centraux des réseaux homogènes, n'étant pas associés à des modules *TG*, possèdent les trafic les moins chargés. Hormis



(a) distribution du trafic pour un trafic aléatoire dans le cas de 20 *TG* connectés au réseau



(b) distribution du trafic pour un trafic aléatoire dans le cas de 21 *TG* connectés au réseau, cas « réception »



(c) distribution du trafic pour un trafic aléatoire dans le cas de 21 *TG* connectés au réseau, cas « réception + envoi »

FIG. 4.52 – Distribution du trafic d'un réseau *QNoC* 5×5.

ces routeurs « coins » et « centraux », les autres routeurs de ces réseaux homogènes possèdent une distribution du trafic uniforme. Dans le cas de réseau hétérogène 3×3, en remplaçant le routeur de position $(2, 2)$ par un module dynamique *TG*, la distribution de trafic au sein du réseau change. Le routeur de position $(2, 2)$ indisponible pour l'acheminement des paquets de données, influence considérablement sur le trafic dans le réseau. En effet, dans ce cas de figure, le routeur de position $(3, 3)$ devient plus chargé que les autres routeurs, car la plupart des paquets de données transitent à travers ce routeur dans le but de contourner le module dynamique *TG*. Comme dans le cas d'un réseau hétérogène *CuNoC*, ce trafic résultant est essentiellement dû au même algorithme de routage utilisé dans un réseau *QNoC*.

Pour les réseaux *hétérogènes QNoC* 4×4 et 5×5, les distributions de trafic sont quasi-similaires à celle d'un réseau hétérogène 3×3. En effet, les routeurs situés dans les « coins » des réseaux sont connectés à des paires de modules statiques *TG*. Ces routeurs subissent donc un trafic de paquets de données dense, tandis que les routeurs situés au centre du réseau sont beaucoup moins solicités (trafic moins dense). En remplaçant les routeurs *Q-switch* des réseaux de taille 4×4 et 5×5 situés respectivement aux positions $(3, 3)$ et $(4, 4)$ par un module dynamique *TG*, la distribution de trafic change, et plus particulièrement aux environs du module dynamique *TG* placé. Comme pour le cas d'un réseau *QNoC* de taille 3×3, le routeur situé en haut et à droite du module dynamique est le plus sollicité pour l'acheminement des paquets face à l'obstacle correspondant au module dynamique. L'algorithme de routage employé du réseau *QNoC* entraîne plus particulièrement la sollicitation du routeur considéré. La distribution de trafic pour les autres routeurs du réseau ne subit pas trop de changement comparée à un réseau homogène.

Pour un réseau *hétérogène QNoC* de taille 5×5, la distribution du trafic pour le module dynamique *TG* placé pour les trois cas « obstacle », « réception » et « réception + envoi » est présentée en trois dimensions dans la figure 4.52. Les axes *X* et *Y* représentent les routeurs dans la topologie utilisée, tandis que l'axe *Z* représente le nombre de passage de paquets par un routeur divisé par le nombre total de paquets envoyés dans le réseau en fonction du *PIR*.

Le tableau 4.7 contient les valeurs des latences moyenne, minimale et maximale pour chaque taille de réseaux *QNoC* considérées. Les valeurs présentées sont données en terme de nombre de cycles d'horloge. A titre d'exemple, pour un réseau *QNoC* de taille 4×4 connecté à 16 modules statiques *TG*, la latence moyenne varie de 12.21 à 24.73 cycles d'horloge. La latence minimale est de l'ordre de 3 cycles d'horloge, tandis que la latence maximale se situe entre 23 à 121 cycles d'horloge.

4.3.4 Analyse comparative entre un réseau *CuNoC* et un réseau *QNoC*

Une comparaison entre les réseaux *CuNoC* et *QNoC* en termes de performances et de ressources nécessaires pour leur implantation a été réalisée. Ces résultats sont présentés dans la figure 4.53 et dans les tableaux 4.8 et 4.9.

TAB. 4.7 – Évaluation des latences moyenne, minimale et maximale pour un trafic total de 125 000 paquets envoyés de manière aléatoire par un *TG* pour différentes tailles d'un réseau *QNoC*

CuNoC	5 x 5	4 x 4	3 x 3	2 x 2	1 x 1
Moyenne	15.0 - 30.0	12.2 - 24.7	9.37 - 19.68	6.44 - 14.84	3.00 - 8.1
Minimum	3 - 15	3 - 12	3 - 9	3- 6	3
Maximum	30 - 223	23 - 121	17 - 74	11 - 45	5 - 16

Les chiffres dans le tableau sont exprimés en cycles d'horloge

Les résultats de comparaison en termes de performances des réseaux *CuNoC* et *QNoC* de taille différente sont présentés dans la figure 4.53. Ces résultats montrent clairement que pour différentes tailles de réseau, un réseau *QNoC* présente de meilleures performances face à un réseau *CuNoC*. Un routeur *Q-switch* possède un *PIR* maximal pour un type de trafic « aléatoire » de 0.6 paquets par cycle d'horloge, tandis que la valeur maximale de *PIR* d'un routeur *CU* pour ce même type de trafic est approximativement de 0.12 paquets par cycle d'horloge. Un routeur *QNoC* présente donc une amélioration en terme de performance cinq fois supérieure comparé à un routeur *CuNoC*. De plus, pour un type de trafic « défini », un routeur *Q-switch* atteint une valeur maximale de *PIR* de 1.0 paquet par cycle d'horloge, tandis qu'un routeur *CU* a une valeur maximale de *PIR* pour ce même type de trafic inchangée. Le rapport de performances entre ces deux routeurs est supérieur à 8 en faveur d'un routeur *Q-switch*. Concernant la latence moyenne, on remarque que pour les valeurs de *PIR* inférieure à 0.08 paquets par cycle d'horloge, un routeur *CU* montre une latence moyenne inférieure à celle d'un routeur *Q-switch*. Ceci est dû au fait que la latence minimale d'un routeur *CU* est de 2 cycles d'horloge tandis que celle d'un routeur *Q-switch* est 3 cycles d'horloge. La figure 4.53b donne une comparaison des bandes passantes des routeurs *CU* et *Q-switch*. Un routeur *CU* atteint au mieux une bande passante maximale de 0.48 paquets par cycle d'horloge pour deux types de trafic. Par contre, les valeurs des bandes passantes maximales d'un routeur *Q-switch* pour les types de trafic *aléatoire* et *défini* sont respectivement 2.4 et 4 paquets par cycle d'horloge.

De même, pour des réseaux de taille plus grande. En effet, en augmentant la taille d'un réseau, les valeurs des latences moyennes pour les deux types de réseau augmentent, tandis que les valeurs maximales de *PIR* diminuent. Ainsi, pour les réseaux *CuNoC* de taille 2×2, 3×3 et 4×4 nous avons des valeurs maximales de *PIR* pour les 2 types de trafic respectivement de 0.07, 0.06 et 0.05 paquets par cycle d'horloge. Ces valeurs sont inférieures à la valeur de *PIR* d'un routeur *CU*. Pour les réseaux *QNoC* de taille 2×2, 3×3 et 4×4, ces valeurs avoisinent respectivement 0.52, 0.50 et 0.47 paquets par cycle d'horloge pour le trafic *aléatoire* (inférieur

TAB. 4.8 – Résultats de comparaison en terme de ressources nécessaires pour l'implantation des routeurs *CU* et *Q-switch* de différents formats de données sur *FPGA Xilinx Virtex IV*

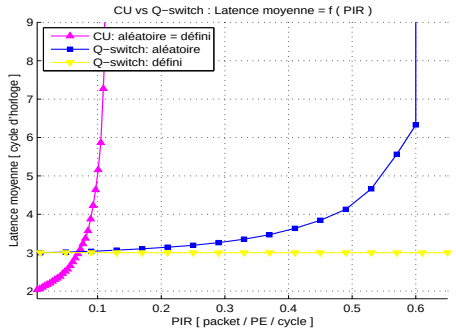
	CU		*Q-switch*		rapport	
format de données	f [MHz]	CLB Slices	f [MHz]	CLB Slices	f (*CuNoC*/QNoC)	ressources (QNoC/*CuNoC*)
8 bit	549.3	49	257.2	309	2.14	6.31
16 bit	549.3	84	234.9	473	2.34	5.63
24 bit	549.3	112	233.3	640	2.35	5.71
32 bit	549.3	140	232.0	808	2.38	5.77

TAB. 4.9 – Résultats de comparaison en terme de ressources nécessaires pour l'implantation des réseaux *CuNoC* et *QNoC* de différente taille et pour différents formats de données sur *FPGA Xilinx Virtex IV*

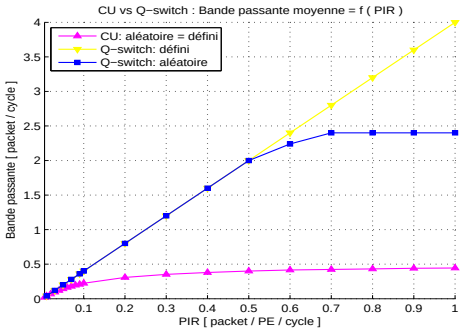
	CuNoC 2x2		QNoC 2x2		*CuNoC* 3x3		QNoC 3x3	
format de données	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB	f [MHz]	CLB
8 bit	549.3	197	218.1	1243	549.3	443	218.7	2833
16 bit	549.3	293	196.9	1904	549.3	659	212.3	4310
24 bit	549.3	418	203.9	2579	549.3	948	203.9	5824
32 bit	549.3	522	199.2	3241	549.3	1184	205.2	7317

à la valeur de *PIR* d'un routeur *Q-switch*) et la valeur de 1.0 paquets par cycle d'horloge pour le trafic *défini*. On remarque également que quelle que soit la taille d'un réseau *QNoC*, les valeurs de *PIR* et de la bande passante arrivent à atteindre les valeurs maximales théoriques pour un trafic *défini*. Ceci n'est le cas pour une approche réseau sur puce de type *CuNoC*.

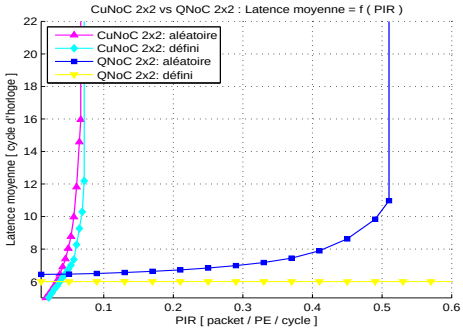
Une comparaison en termes de ressources logiques nécessaires à l'implantation d'un réseau *CuNoC* et *QNoC* sur technologie *FPGA Xilinx Virtex IV* est également donnée. Ces résultats sont présentés respectivement dans le tableau 4.8 pour les routeurs *CU* et *Q-switch* et dans le tableau 4.9 pour les réseaux *CuNoC* et *QNoC* de taille 2×2 et 3×3 de différent format de données (de 8 à 32 bits). Ces résultats montrent clairement que d'un point de vue de ressources nécessaires, l'approche de réseau *CuNoC* nécessite beaucoup moins de ressources qu'une approche de réseau *QNoC*. Ainsi, un routeur *CU* nécessite environ 6 fois moins de ressources et possède une fréquence de fonctionnement maximale de 2 fois plus supérieure comparé à un routeur *Q-switch*. De même pour les réseaux de dimension plus élevée. Les ressources nécessaires pour un réseau *CuNoC* sont également d'environ 6 fois inférieure à des réseaux *QNoC* de même taille. Les fréquences de fonctionnement maximales des réseaux *CuNoC* sont de l'ordre de deux fois supérieur aux réseaux *QNoC* de même dimension.



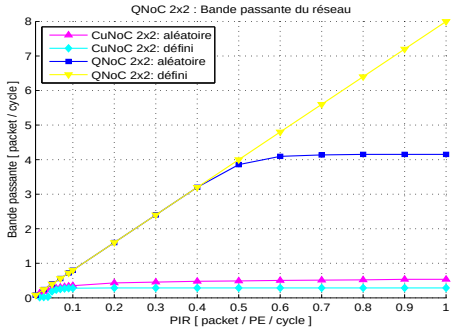
(a) CU vs *Q-switch* : latence moyenne



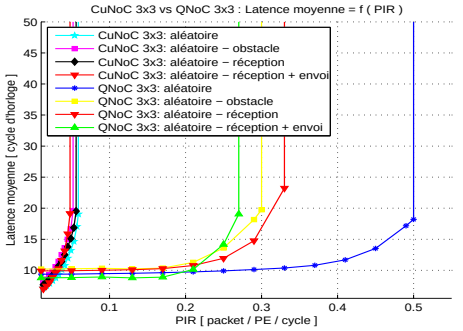
(b) CU vs *Q-switch* : bande passante



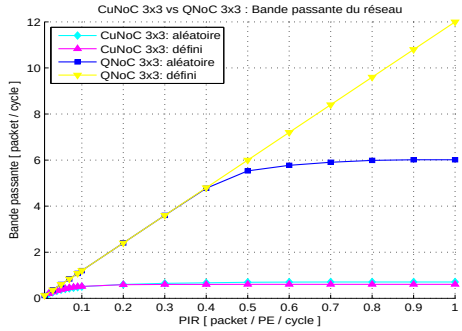
(c) *CuNoC* 2x2 vs *QNoC* 2x2 : latence moyenne



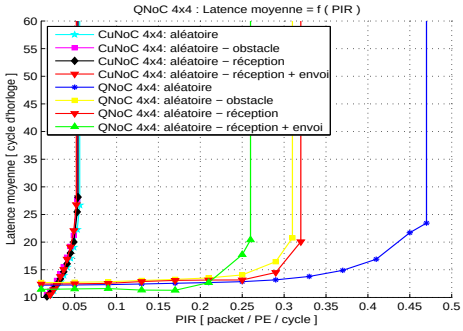
(d) *CuNoC* 2x2 vs *QNoC* 2x2 : bande passante



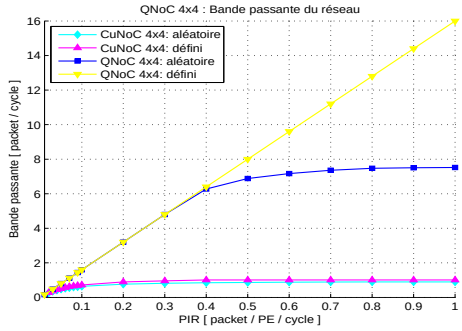
(e) *CuNoC* 3x3 vs *QNoC* 3x3 : latence moyenne



(f) *CuNoC* 3x3 vs *QNoC* 3x3 : bande passante



(g) *CuNoC* 4x4 vs *QNoC* 4x4 : latence moyenne



(h) *CuNoC* 4x4 vs *QNoC* 4x4 : bande passante

FIG. 4.53 – Analyse comparative entre un réseau *CuNoC* et un réseau *QNoC* en terme de performances.

4.3.5 Algorithme de routage des réseaux *CuNoC* et *QNoC*

4.3.5.1 Introduction

Dans l'introduction de ce chapitre, un point sur les principales catégories d'algorithmes de routage a été présenté dont les algorithmes tolérants aux fautes. Ces algorithmes permettent le routage de paquets de données dans des réseaux ayant des routeurs ou nœuds de routage défaillants. Ces nœuds défaillants rendent la structure et la topologie du réseau irrégulière, compliquant ainsi considérablement le routage des paquets. Comme mentionné dans l'introduction, la plupart des algorithmes de routage ne sont pas tolérants aux fautes. En effet, des modifications importantes sont nécessaires pour les rendre tolérants aux fautes.

Le routage dans les réseaux ayant des routeurs défaillants ou inaccessibles est similaire au routage dans les réseaux *CuNoC* et *QNoC*. En effet, la procédure de création de ces réseaux et le placement des modules de calcul au sein de ces réseaux sont les principales causes aboutissant à la formation de structures de réseau hétérogènes. Ces structures de réseau, composées de routeurs et de modules de calcul placés selon des règles prédéfinies, sont similaires à des réseaux homogènes ayant des nœuds de routage défaillants ou des *régions*[16] de routeurs. C'est pourquoi, cette section met principalement l'accent sur les aspects de routage tolérants aux fautes dans les réseaux homogènes.

4.3.5.2 Les algorithmes de routage tolérants aux fautes : Etat de l'art

Dans la littérature, plusieurs types d'algorithmes de routage tolérants aux fautes ont été proposés. La plupart d'entre eux peuvent, avec des modification mineures, être adaptés et utilisés dans les réseaux ayant des *régions*. Un exemple d'algorithme de routage tolérant aux fautes est l'algorithme basé sur l'algorithme adaptatif *negative-first* [GN93]. Cet algorithme de routage est destiné à des structures maillées (*mesh*) et peut tolérer jusqu'à $(n-1)$ nœuds de routage défaillants dans le réseau. Une de ses particularités est qu'il n'utilise pas de canaux virtuels (*virtuel channel*). Cependant, son plus grand inconvénient est qu'il ne peut pas être utilisé pour le routage de paquets de données autour de plusieurs nœuds de routage défaillants.

Un autre type d'algorithme tolérant aux fautes est l'algorithme partiellement adaptatif basé sur le modèle *odd-even* [Wu03]. L'utilisation de cet algorithme de routage est possible pour les réseaux possédant des *régions* ou plusieurs nœuds de routage défaillants formant des structures rectangulaires. Ces structures rectangulaires sont composées de routeurs défaillants et de nœuds de routage nommés *unsafe*. Ces derniers se trouvent à proximité des nœuds de routage défaillants et sont utilisés pour former des structures rectangulaires avec les routeurs défaillants. Ces nœuds *unsafe* ne sont pas accessibles par les autres routeurs du réseau considéré.

[16] eng. *region* - zones dans les réseaux ayant une taille supérieure à la taille réservée pour un module de calcul (eng. *tile*)

Des approches similaires à celle présentée ci-dessus mais utilisant des canaux virtuels peuvent être trouvées dans la littérature [BC95]. Certains de ces algorithmes de routage proposés utilise les *f-chains* et *f-rings*, sortes d'anneaux qui se forment autour de nœuds de routage défaillants d'un réseau, et qui servent de chemins pour tous les paquets passant par ces zones de nœuds défaillants. Une approche similaire, basée également sur l'utilisation des *f-chains* et *f-rings* est présentée dans le [CC98b]. Tous ces algorithmes de routage reposent sur l'utilisation de canaux virtuels afin de faciliter la résolution d'apparition des blocages dans les réseaux (*deadlocks*). D'autre part, les nœuds utilisant des canaux virtuels sont plus susceptibles aux fautes. Plus les ressources sont utilisées, plus grande est la probabilité d'apparition de fautes dans les circuits. Un autre inconvénient est que la défaillance d'une liaison physique entre deux routeurs entraîne une inutilisation de tous les canaux virtuels reliés à cette liaison.

Un exemple d'algorithme de routage tolérant aux fautes et entièrement adaptable (*fully adaptive*) est présenté dans [SZBR07]. Le routage entre la source et la destination dans cette approche est basé sur l'utilisation de *tables de routage*. Chaque nœud de routage possède une *table de routage* qui contient toutes les informations de routage vers tous les nœuds de routage du réseau. Ces tables de routage sont construites dans la phase d'initialisation du réseau. Lorsqu'un routeur reçoit un paquet d'entête (*header flit*), il le transmet vers son nœud voisin en échange d'un signal de confirmation si dans sa *table de routage* il existe une indication de cheminement vers la destination finale indiquée dans l'entête du paquet. Dans le cas contraire, le nœud de routage transmet le paquet vers les autres nœuds de routage voisins jusqu'à qu'il trouve le chemin pour le paquet considéré. Ainsi, le routeur décide du cheminement que le paquet va emprunter. De plus, pour distribuer uniformément les paquets dans le réseau, chaque nœud de routage possède des « tables de pénalité » (*penalty table*), dans lesquelles sont contenues les valeurs relatives aux trafic de chaque port de sortie d'un routeur. Ainsi, les chemins surchargés sont évités. L'absence de blocage dans les réseaux utilisant ce type d'algorithme n'est pas prouvée.

4.3.5.3 L'algorithme de routage *MPA* des réseaux *CuNoC* et *QNoC*

Nous proposons un algorithme de routage partiellement adaptable, tolérant aux fautes et permettant également le routage de paquets de données dans les réseaux à des *régions* ou des nœuds de routage défaillants [JTWB09]. Cet algorithme est sans blocage (*deadlock-free*) et n'utilise pas de canaux virtuels. L'originalité de cet algorithme de routage est qu'il permet le routage de paquets dans les réseaux possédant des *régions* pas nécessairement rectangulaires et vers le nœuds qui ne sont pas complètement bloqués par des nœuds défaillants. Notre approche est basée sur le modèle de type *turn* et sur l'algorithme *XY*. L'algorithme de routage proposé est nommé *MPA* (*Module Proximity Algorithm*) et fait référence à des entités (nœuds de routage défaillants, *régions*, modules de calcul) qui peuvent être placées statiquement ou dynamiquement

dans le réseau. La topologie adaptée à l'algorithme *MPA* est la topologie de réseau maillée *2D*.

Dans un réseau maillé *2D* $n \times n$, un nœud de routage K est représenté par un vecteur à deux dimensions (k_x, k_y), $0 \leq k_x, k_y \leq n-1$, où k_x et k_y sont respectivement les coordonnées de l'axe x et y. Chaque routeur possède 5 ports d'entrée et de sortie, à travers lesquels il peut transmettre et recevoir des paquets de données. Les quatre paires de ports vers les nœuds adjacents sont respectivement nommés *West*, *North*, *East* et *South* et le cinquième paire présente l'accès local du routeur à un module de calcul associé au routeur. Plusieurs définitions ont été établies pour la mise en œuvre de l'algorithme *MPA*.

Définition 1 *Dans un réseau maillé 2D, un nœud de routage est appelé* ***nœud pair*** *(even), ou nœud* ***impair*** *(odd), si la somme de ses coordonnées (x et y) représente respectivement un nombre pair ou impair.*

Le placement valide des nœuds de routage paires et impaires dans un réseau est présenté dans la figure 4.54. Dans cette structuration des routeurs, chaque nœud d'un type (pair ou impair) est entourés par des nœuds de routage de l'autre type et vice versa.

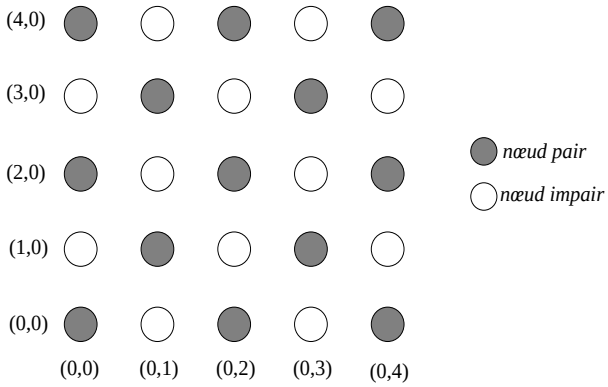


FIG. 4.54 – Placement valide des nœuds de routage pairs et impairs dans un réseau

Deux types de fonctionnement pour chaque nœud de routage sont distingués : un mode *activé* (*activated*) et un mode *désactivé* (*deactivated*). Ainsi, les notions de zone *activée* et de zone *désactivée* d'un réseau sont définies de la manière suivante :

Définition 2 *La* ***zone activée*** *d'un réseau est la zone minimale rectangulaire enveloppant tous les nœuds de routage défaillants ou régions dans le réseau.*

Si dans un réseau il n'y a pas de nœuds de routage défaillants ou de *régions*, alors ce réseau n'a pas de *zone activée*. De plus, un réseau peut avoir seulement une *zone activée*. Tous les nœuds appartenant à la *zone activée* d'un réseau sont **activés**. Une *zone activée* ne peut pas avoir simultanément un nœud impair (pair) au « coin » du réseau situé en haut à droite et un nœud pair (impair) au « coin » du réseau situé en bas à gauche du réseau.

Définition 3 *La **zone désactivée** d'un réseau est le reste du réseau n'appartenant pas à la zone activée.*

Si un réseau n'a pas une *zone activée* (pas de nœuds défaillants ni de *régions*), il est entièrement désactivé. Tous les nœuds appartenant à la *zone désactivée* sont **désactivés**.

La figure 4.55 illustre un réseau contenant une zone *activée* formée autour d'un nœud défaillant. Dans ce cas, uniquement les nœuds de routage enveloppant le nœud de routage défaillant changent de statut et deviennent activés. Les nœuds appartenant au reste du réseau ne changent pas leur mode et restent désactivés.

Un nœud *désactivé* achemine un paquet de données selon l'algorithme *XY*. Premièrement il achemine le paquet selon l'axe *X*, puis selon l'axe *Y*, jusqu'à ce que le paquet de donnée ne soit pas livré à la destination. Si le paquet arrive à la zone *activée* avant d'atteindre sa destination finale, les nouvelles règles de routage sont alors appliquées.

Les nœuds de routage *activés* n'obéissent pas aux mêmes règles de routage que les nœuds de routage désactivés. Ces règles sont décrites de la manière suivante :

Règle 1 *Un nœud pair et activé ne peut pas acheminer des paquets venant de la direction Nord (North) vers la direction Est (East) et vice versa.*

Règle 2 *Un nœud impair et activé ne peut pas acheminer des paquets venant de le direction Sud (South) vers la direction Ouest (West) et vice versa.*

Règle 3 *Tous les nœuds activés, par défaut, ne peuvent pas acheminer des paquets venant respectivement des directions Nord et Est vers les directions Sud et Ouest.*

Règle 4 *Tous les nœuds activés ne peuvent pas par défaut acheminer des paquets venant respectivement des directions Sud et Oest vers les directions Nort et Est.*

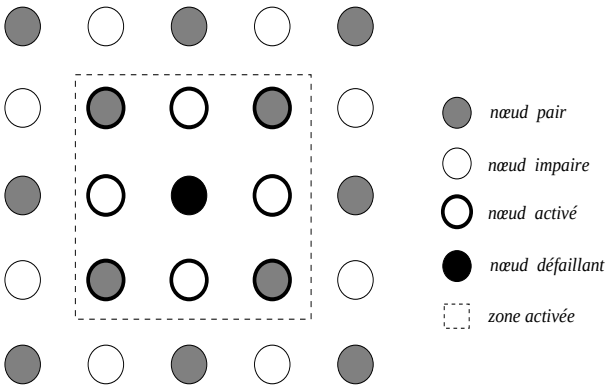


FIG. 4.55 – Exemple d'une *zone activée*.

Procédure utilisée pour former une zone activée
répète
scrutation du réseau entier
si *(il y a des nœuds défaillants ou des régions)*
enveloppe les nœuds défaillants ou les régions avec les nœuds
activés et met à jour les tables de routage
si *(il existe déjà une zone activée)*
fusionne la nouvelle zone activée avec l'ancienne
et met à jour les tables de routage
fin si
sinon
garde tous les nœuds désactivés
fin si
fin répète

FIG. 4.56 – Procédure utilisée pour former une zone activée.

Ces règles de directions non-autorisées pour l'acheminement de paquets dans un routeur s'appliquent systématiquement par les nœuds de routage dans une *zone activée* du réseau. Certaines règles peuvent ne pas s'appliquer en fonction des positions des nœuds défaillants ou des *régions* dans la *zone activée* du réseau. Ceci est particulièrement vrai pour les règles 3 et 4. En mettant un nœud de routage défaillant à côté d'un nœud *activé*, certaines directions initialement non-autorisées deviennent autorisées. Ainsi, certaines règles dans la zone *activée* sont simplifiées à travers les règles suivantes :

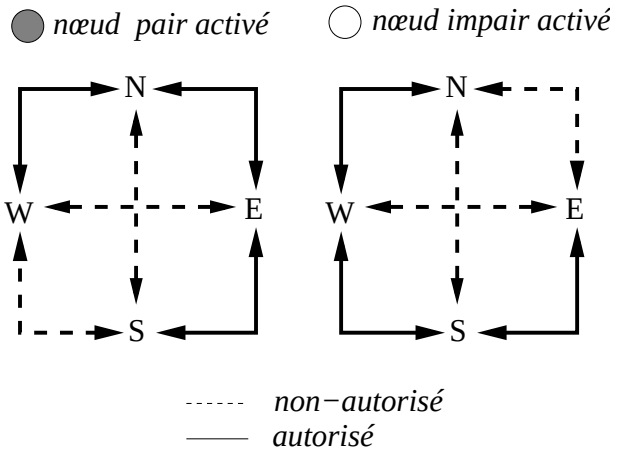


FIG. 4.57 – Les directions d'acheminement de paquets non-autorisés par défaut dans les nœuds de routage *activés*.

Règle 5 *Si un nœud de routage pair et activé à la position* (x, y) *est défaillant ou fait partie d'une région, les nœuds impairs et activés dans les positions* $(x+1, y)$ *et* $(x, y+1)$ *n'appliquent pas la règle 3.*

Règle 6 *Si un nœud de routage impair et activé à la position* (x, y) *est défaillant ou fait partie d'une région, les nœuds pairs et activés dans les positions* $(x-1, y)$ *et* $(x, y-1)$ *n'appliquent pas la règle 4.*

Règle 7 *Si un nœud de routage pair et activé, situé dans la position* (x, y)*, a dans les positions* $(x-1, y+1)$,$(x+1, y+1)$ *ou (*$(x, y+1)$ *et* $(x-2, y)$*) un nœud défaillant, il peut acheminer les paquets de l'Est vers l'Ouest.*
Réciproquement, si un nœud de routage impair et activé, situé dans la position (x, y)*, a dans les positions* $(x-1, y-1)$,$(x+1, y-1)$ *ou (*$(x, y-1)$ *et* $(x+2, y+1)$*) un nœud défaillant, il peut acheminer les paquets de l'Ouest vers l'Est.*

Règle 8 *Si un nœud de routage pair et activé, situé dans la position* (x, y)*, a dans les positions* $(x+1, y+1)$,$(x+1, y-1)$ *ou (*$(x+1, y)$ *et* $(x+1, y+2)$*) un nœud défaillant, il peut acheminer les paquets du Nord vers le Sud.*
Réciproquement, si un nœud de routage impair et activé, situé dans la position (x, y)*, a dans les positions* $(x-1, y+1)$,$(x-1, y-1)$ *ou (*$(x-1, y)$ *et* $(x, y-2)$*) un nœud de routage défaillant, il peut acheminer les paquet du Sud vers le Nord.*

Le réseau est scruté régulièrement, comme expliqué dans la procédure 4.56. Si un nouveau nœud de routage défaillant ou une nouvelle *région* sont détectés dans le réseau, les tables de routage des nœuds de routage *activés* sont mises à jour avec les nouvelles informations, contenant les positions des nouveaux nœuds défaillants ou *régions*. Les informations contenues dans ces tables de routage locales, aident les nœuds de routage *activés* dans les décisions de routage. Par exemple, une *zone activée* avec un nombre minimal de routeurs *activés*, qui forment autour des nœuds de routage défaillants une sorte d'anneau, n'achemine pas les paquets de données de la même manière qu'une *zone activée* avec plusieurs routeurs défaillants. Dans le dernier cas, les paquets utilisent non seulement l'anneau formé autour des routeurs défaillants pour atteindre la destination finale, mais également d'autres routeurs *activés* de la *zone activée*.

4.3.5.4 Routage des paquets dans un réseau utilisant l'algorithme *MPA*

Le routage dans un réseau utilisant l'algorithme de routage *MPA* est illustré par le pseudo-code présenté dans la figure 4.58. Lorsqu'un paquet arrive à la limite d'une *zone activée* avant d'atteindre sa destination finale, son routage est alors modifié pour évoluer de l'algorithme de routage *XY* classique vers l'algorithme *MPA*. Si il arrive de la direction *X*, il peut quitter la *zone*

Procédure : Routage MPA
*/*Message m_g est envoyé de la source S vers la destination D ;*
C est le nœud courant ;
SW, SE, NW et NE sont respectivement les coins Sud-Ouest, Sud-Est, Nord-Ouest et Nord-Est de la zone activée ;
X(coin) et Y(coin) sont les dimensions X et Y du coin ;
X(D) et Y(D) sont les dimensions X et Y de la destination/*
***si** (C est un nœud désactivé)*
  le routage XY
sinon
  le routage dans la zone activée
fin si
*/*Le routage dans la zone activée*/*
***si** (m_g de la direction X)*
  ***si** (D est dans la zone activée)*
    achemine vers la D en utilisant les règles
  sinon
    ***si** ($Y(D) > Y(NE)$ **et** $X(NW) \leq X(D) \leq X(NE)$)*
      route vers le nœud (X(D),Y(NW)) et envoie le m_g vers (X(D),Y(NW)+1) ;
    ***sinon si** ($Y(D) < Y(SE)$ **et** $X(NW) \leq X(D) \leq X(NE)$)*
      route vers le nœud (X(D),Y(SW)) et envoie le m_g vers le (X(D),Y(SW)+1) ;
    sinon
      route vers le premier nœud activé situé à la limite opposée de la zone activée
      dans le sens opposé de l'arrivée et envoie le m_g vers le nœud désactivé X(NE+1) ou X(NW-1)
    fin si
  fin si
***sinon** /* m_g de la direction Y */*
  ***si** (D est dans la zone activée)*
    route vers la D suivant les règles
  sinon
    route vers le (X(D),Y(NW)) ou (X(D),Y(SW)) et envoie le m_g vers (X(D),Y(NW)+1) ;
  fin si
fin si

FIG. 4.58 – Pseudo-code de l'algorithme de routage *MPA*

activée, soit dans la direction *X* mais du côté opposé de l'arrivée, soit dans la direction *Y* d'un nœud *activé*, ayant la même dimension *X* que sa destination. Si il arrive de la direction *Y*, il peut quitter la *zone activée* uniquement dans la direction *Y*, du côté opposé de la direction d'arrivée, du nœud de routage ayant la même dimension *X* que sa destination.

Le routage dans la *zone activée* dépend du nombre de nœuds de routage défaillants ou de *régions* dans le réseau. Dans le cas d'une *région* rectangulaire ou un groupe de nœud de routage défaillants formant un rectangle, les nœuds de routage *activés* forment un anneau autour de la

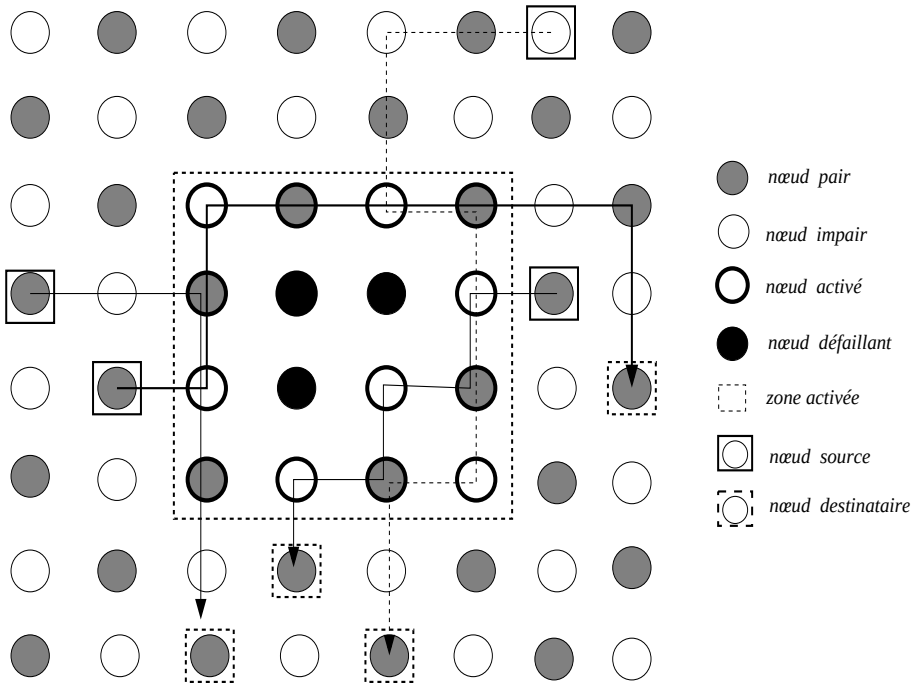


FIG. 4.59 – Le routage dans un réseau utilisant l'algorithme MPA dans le cas de 3 nœuds défaillants.

région de nœuds défaillants. Le routage dans la *zone activée* se réduit alors à un routage en périphérie de cet anneau formé.

La figure 4.59 illustre des situations de routage dans un réseau maillé *2D* de dimension 8×8 basé sur l'algorithme *MPA* dans le cas de 3 nœuds de routage défaillants. La *zone activée* et formée dans ce cas de figure correspondant à un rectangle de taille 4×4. La taille de le *zone activée* est donc fonction du nombre de nœuds défaillants dans le réseau. La figure 4.59 présente quelques exemples de cas de routage de paquets arrivant des directions *X* et *Y*.

4.3.5.5 Absence de blocage de paquets dans un réseau basé sur le routage *MPA* (*Deadlock freeness*)

Les situations de blocage de paquets de données dans les réseaux se produisent lorsque des paquets attendent le passage d'autres paquets (selon les priorités de passage des paquets définies par le routage dans un routeur) et que l'ensemble des paquets concernés par ces attentes forme une boucle dans le réseau. Ces situations sont donc la conséquence directe des directions des paquets autorisées pouvant être circulaires dans un réseau. Avec l'algorithme *MPA* proposé, l'absence de blocages est assurée par l'interdiction de certaines directions à des certaines positions pour les paquets dans un réseau. Ces interdictions sont spécifiées par les règles 1 à 8 de l'algorithme *MPA*. La figure 4.60 illustre un graphe de dépendance de canaux (*channel*

dependency graph) d'une zone activée de taille 4×4. On constate que dans ce graphe, l'algorithme *MPA* ne présente aucune direction circulaire (boucle fermée) suite aux règles adoptées. Par conséquent le théorème suivant est adopté.

Théorème 1 *L'algorithme de routage MPA est sans blocages (deadlock free).*

Preuve du théorème : Selon [DS87], un algorithme de routage est sans blocage si son graphe de dépendances de canaux n'a pas de boucles fermées (*circular waits*). Ainsi, l'algorithme de routage *MPA* est alors considéré sans blocage.

4.3.5.6 Résultats de simulation

Une évaluation de l'algorithme de routage *MPA* par simulation a été réalisée. Cette évaluation repose sur un simulateur *NoC* modélisé en langage *VHDL* et permettant à la fois la simulation de réseaux de topologies maillées régulières et irrégulières.

Le modèle de réseau utilisé dans les simulations réalisées est un réseau de dimension 10×10 utilisant la technique d'aiguillage de type *wormhole switching*. La taille des paquets de données est de 10 *flits*. Chaque routeur a des buffers d'entrée et de sortie acceptant au maximum 2 flits. Le délai minimal d'une liaison entre deux routeurs est de 3 cycles par *flit*, tandis que la bande passante maximale par liaison est d'un *flit* par cycle d'horloge (1 paquet / 10 cycles). La latence moyenne est la principale métrique relevée au cours des simulations. Grâce à ce simulateur, les performances de l'algorithme *MPA* ont été relevées à partir d'un trafic de 10,000 paquets de données transmis, après une phase d'initialisation de 2,000 paquets envoyés.

Les modèles de simulation développés peuvent générer des trafics *aléatoire* ou à *matrice transposée* (*matrix transpose pattern*). Le modèle de simulation à trafic *aléatoire* correspond à une simulation dans laquelle tous les modules de calcul connectés au réseau simulé transmettent

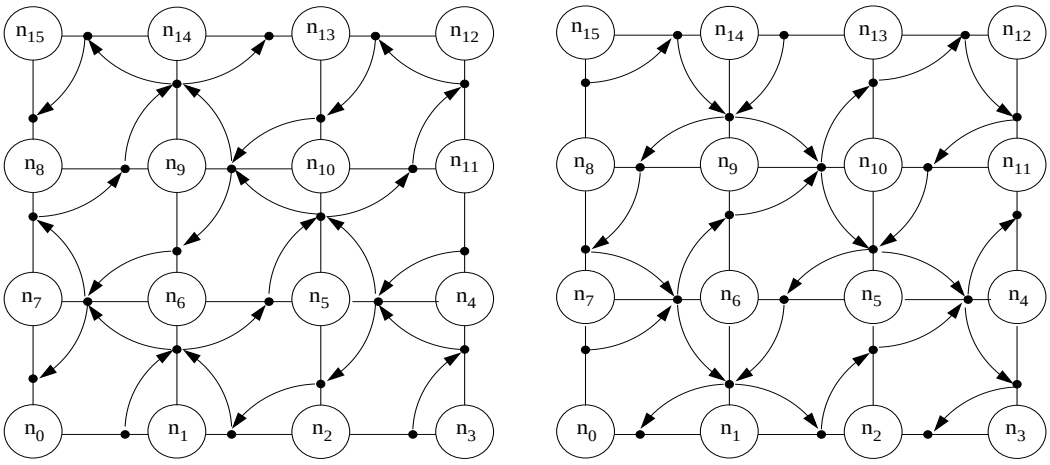


FIG. 4.60 – *Channel dependency graph* d'une *zone activée* de dimension 4×4

aléatoirement des paquets à travers le réseau simulé vers d'autres modules de calcul connectés également au réseau. Le modèle de simulation à trafic *matrice transposée* correspond à une situation de simulation où chaque module de calcul du réseau simulé de position (i, j) transmet des paquets au module de calcul situé à la position $(n - i, n - j)$, où n×n est la taille du réseau simulé. Le temps de latence entre l'envoi de deux paquets est uniformément distribué dans un intervalle de 5 à 10 cycles d'horloge.

Des exécutions de ces deux modèles de simulation ont été réalisées. Une première simulation de ces modèles est réalisée avec un réseau homogène. C'est-à-dire, dans le cas où le réseau considéré ne comprend pas de nœuds de routage défaillants ou de *régions*. Une deuxième simulation du second modèle sur un réseau hétérogène est ensuite mise en œuvre. Dans cette simulation 4 nœuds de routage défaillants placés dans le réseau sont considérés. Les résultats de simulation pour ces deux cas de figure de simulation sont comparés avec des simulations d'un réseau basé sur l'algorithme de routage *message routing* [CC98a]. Les figures 4.61 et 4.62 illustrent le premier cas de simulation d'un réseau homogène (sans nœuds de routage défaillants) respectivement pour les deux types de trafic (*aléatoire* et *matrice transposée*). Ces résultats montrent qu'en absence des nœuds de routage défaillants, l'algorithme *MPA* présente de meilleures performances en termes de latence moyenne comparé à l'algorithme de routage *message routing* [CC98a].

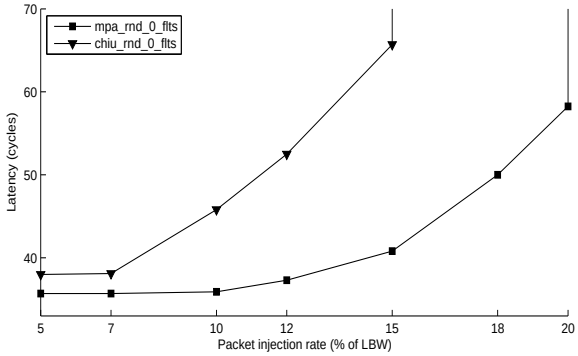


FIG. 4.61 – Résultats de simulation pour un trafic *aléatoire* sans nœuds de routage défaillants des réseaux basés respectivement sur les algorithmes *MPA* et *MR*.

Les figures 4.63 et 4.64 illustrent le deuxième cas de simulation pour un réseau hétérogène dans lequel 4 nœuds de routage défaillants sont placés au sein du réseau. Les deux types de trafic réseau ont été également appliqués pour ce cas de simulation de réseau. Ces figures montrent que les latences moyennes obtenues pour des réseaux basés sur algorithmes de routage *MPA* et *MR* sont supérieures à celles obtenues dans le premier cas de simulation. Ces résultats correspondant à un accroissement des temps de latence dans ces réseaux simulés s'expliquent par la présence de 4 nœuds de routage défaillants au sein de ces réseaux. Pour les deux types de

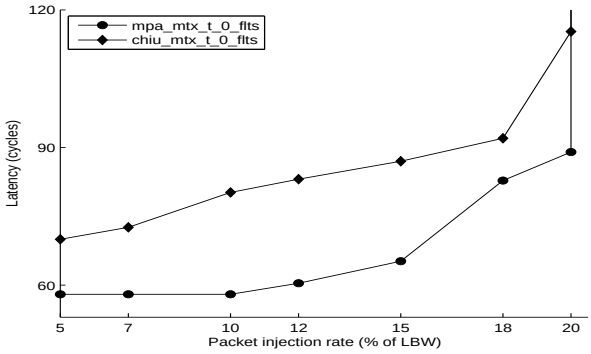


FIG. 4.62 – Résultats de simulation pour un trafic à *matrice transposée* sans nœuds de routage défaillants des réseaux basés respectivement sur les algorithmes *MPA* et *MR*.

trafic (*aléatoire* et *matrice transposée*) simulés, les réseaux basés sur les algorithmes de routage simulés montrent un taux d'injection de paquets inférieur par rapport au premier cas de simulation. Cependant, le réseau basé sur l'algorithme *MPA* permet un taux d'injection de paquets supérieur au réseau basé sur l'algorithme *MR*. En effet, pour un trafic *aléatoire*, l'algorithme *MPA* génère un taux d'injection de paquets de l'ordre de 15 %, tandis que l'algorithme *MR* fournit un taux d'environ de 12 %.

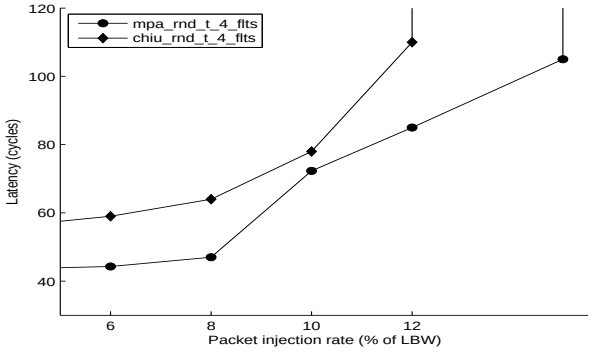


FIG. 4.63 – Résultats de simulation pour un trafic *aléatoire* dans des réseaux hétérogènes possédant 4 nœuds de routage défaillants basés respectivement sur les algorithmes *MPA* et *MR*

4.4 Conclusion

Après une brève présentation des notions fondamentales des moyens de communication sur une puce dans les systèmes de type *MPSoC*, nous avons détaillé les principaux concepts associés à des réseaux sur puce tels que les topologies d'un réseau, les techniques d'aiguillage de paquets de données, les algorithmes de routage, les architectures de routeurs, etc. Une analyse

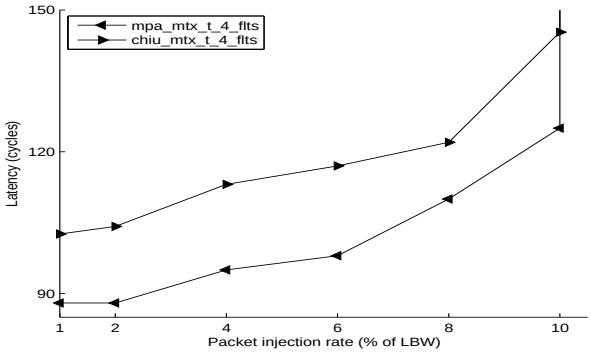


FIG. 4.64 – Résultats de simulation pour un trafic à *matrice transposée* dans des réseaux hétérogènes possédant 4 nœuds de routage défaillants basés respectivement sur les algorithmes *MPA* et *MR*

des réseaux proposés dans la littérature et basée sur des combinaisons de ces concepts a été menée. Il en résulte que la majorité des réseaux classiques proposés dans la littérature sont inadaptés pour la conception de système auto-organisé reconfigurable à base de technologie *FPGA*. Plus précisément, afin d'exploiter efficacement ces technologies reconfigurables, les approches *NoC* classiques doivent être adaptées dans le but d'intégrer dans ces réseaux les aspects de versatilité.

Dans ce chapitre, nous avons ensuite proposé deux architectures de *NoC* reconfigurables (les réseaux *CuNoC* et *QNoC*). L'approche de conception détaillée pour chacun de ces réseaux reconfigurables ainsi que leurs principaux avantages et inconvénients ont été présentés. La première approche de réseau sur puce proposée (réseau *CuNoC*) est caractérisé à la fois par un faible taux de ressources logiques nécessaires pour son implantation, par une politique d'arbitrage de son élément de base (routeur *CU*) reposant sur une règle de priorité à droite, par une connexion spécifique entre ses routeurs *CU* et les modules de calcul associés et enfin par un algorithme de routage adapté permettant l'acheminement des paquets de données (messages) dans un réseau dans les conditions de reconfiguration dynamique au cours de son fonctionnement. Cependant, un réseaux *CuNoC* présente des résultats insuffisants en terme de performances dans le cas de conception de réseau sur puce nécessitant des communications et bandes passantes élevées. La seconde approche de conception de *NoC* reconfigurable (réseau *QNoC*) améliore significativement les points faibles d'un réseau *CuNoC*. En effet, un réseau *QNoC* présente un gain élevé en terme de performances (débit de transmission et bande passante réseau) comparé au réseau sur puce *CuNoC*. Néanmoins, ce *NoC* reconfigurable présente un surcoût important en terme de ressources logiques d'implantation comparé à l'implantation d'un réseau *CuNoC*. Dans ce contexte de conception de réseau sur puce reconfigurable adapté à la mise en œuvre de système *MPSoC* auto-organisé, un compromis doit être recherché en terme de vitesse

de performance et de ressources d'implantation matérielle optimisée.

Outre la technologie d'implantation, la propriété d'auto-adaptation d'un réseau sur puce reconfigurable repose principalement sur sa capacité de poursuivre l'acheminement de paquets de données lors de phases de reconfiguration dynamiques de zone de réseau. Ces reconfigurations partielles correspondent à l'instanciation au cours de fonctionnement du réseau de nouveaux modules de calcul dans le réseau. Cette aptitude pour un réseau reconfigurable est caractérisée principalement par son algorithme de routage. Dans ce chapitre, nous avons proposé un nouvel algorithme de routage adapté aux approches *NoC* reconfigurables développés (réseaux *CuNoC* et *QNoC*). L'algorithme de routage proposé et nommé algorithme *MPA* (*Module Proximity Algorithm*) est un algorithme de routage de type tolérant aux fautes et adaptatif. Cet algorithme permet à la fois le routage de paquets de données dans des réseaux ayant des nœuds de routage défaillants ou des *régions* de réseau occupées par des modules de calcul de grande taille. Une présentation du fonctionnement de l'algorithme proposé et permettant l'acheminement de paquets d'un nœud de routage source vers un nœud de routage destinataire selon des règles de routage a été détaillée. L'algorithme proposé a été prouvé contre des situations de blocages de paquets dans les réseaux à base de cet algorithme (*deadlock free*). Cet algorithme de routage a également été validé au cours de simulations et comparé avec un algorithme de routage de même type. Ces résultats démontrent son adaptabilité et son efficacité pour la conception de réseaux sur puce reconfigurables. Il s'agit alors de valider l'association de l'approche générale de communication sur puce reconfigurable proposée avec les concepts architecturaux de mise en œuvre des propriétés d'auto-organisation pour la conception d'un système *MPSoC* reconfigurable auto-organisé. C'est l'objet du chapitre suivant de ces travaux de thèse, présentant des résultats expérimentaux sur exemple de conception d'un système auto-organisé de traitement d'images temps réel dans le but de valider l'ensemble des concepts présentés au cours de ces travaux pour la conception d'un système reconfigurable auto-organisé.

Chapitre 5

Validation expérimentale du concept d'auto-organisation architectural proposé

5.1 Introduction

Dans ce chapitre, nous présentons la conception d'un système auto-organisé appliqué spécifiquement à un traitement d'images dans le but de valider expérimentalement les concepts et solutions architecturales abordés et présentés dans les chapitres précédents.

5.2 Application traitée : Détection temps réel de contours d'images

5.2.1 Introduction

Nous avons choisi une application de traitement d'images de détection de contours. La détection de contours est une étape de pré-traitement à de nombreuses applications d'analyse d'images. En effet, les contours constituent des indices riches pour toute interprétation ultérieure d'images. Les contours dans une image proviennent des discontinuités de la fonction de luminosité (texture, ombre), des discontinuités de profondeur (bords de l'objet) et sont caractérisés par des discontinuités de la fonction d'intensité pixels d'images. Le principe de la détection de contours repose donc sur un calcul des dérivées de la fonction d'intensité dans l'image.

Plusieurs approches sont utilisées pour la détection de contours d'images en fonction de la méthode d'estimation des dérivées de la fonction d'intensité d'une image. Pour notre application, nous considérons la différence finie à base de l'opérateur de détection de contour de *Sobel* [FS68]. La figure 5.1 illustre le schéma synoptique du détecteur de contours de *Sobel* utilisé pour ce traitement.

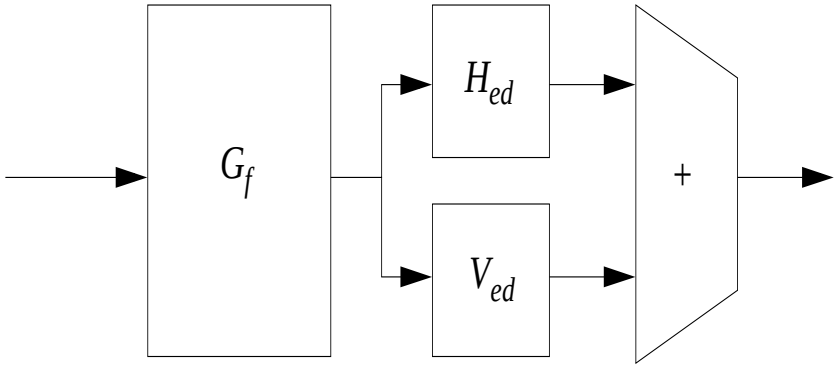


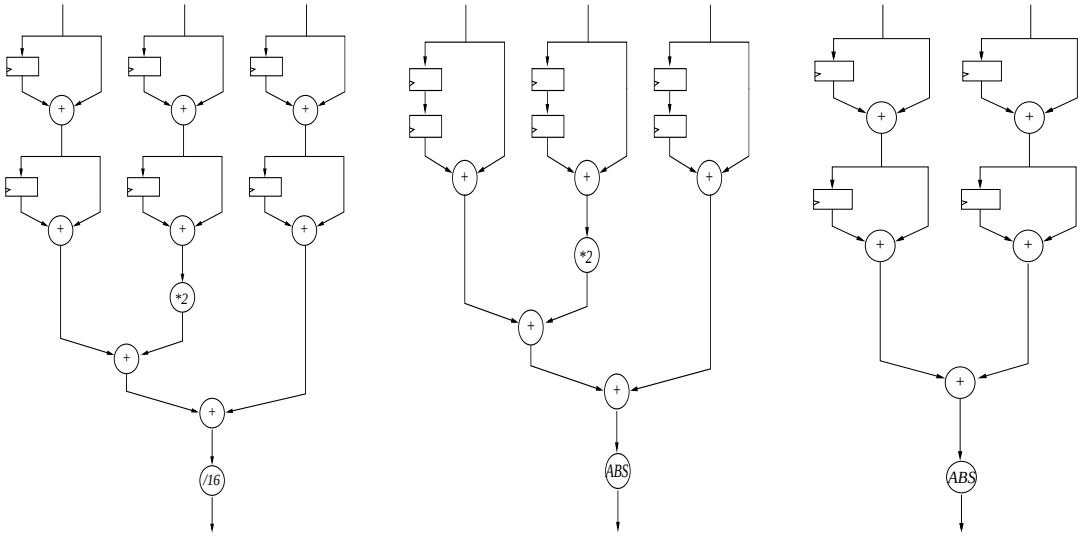
FIG. 5.1 – Schéma synoptique d'un détecteur de contours de *Sobel*.

Ce système est composé d'un filtre *Gaussien*, d'un bloc de détection de contours verticaux, d'un bloc de détection de contours horizontaux et d'un sommateur. Les trois premiers blocs peuvent être implémentés comme convolueurs avec les masques de détecteurs de contours suivantes :

$$G_f = \frac{1}{16} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \end{pmatrix}, H_{ed} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, V_{ed} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Ces masques correspondant à des filtres de détection de contours de taille 3×3 fournissent des résultats moins précis (contours détectés sont moins bien localisés et souvent épais) par rapport à d'autres types de filtres existants, mais les images ainsi obtenues sont généralement plus fiables et permettent des pré-traitements plus poussés.

Pour implanter cet algorithme de détection de contours, une structure binomiale de réalisation a été adoptée [LSC96]. Les graphes flots de données de chaque composant du détecteur sont présentés dans la figure 5.2. Pour la réalisation d'un filtre *Gaussien* à structure binomiale, le nombre d'opérateurs arithmétiques (additionneurs, multiplieurs) s'élève à 8. Les opérateurs de multiplication et de division par 2^n, $n \in N$ ne sont pas considérés dans ce comptage, car ils correspondent à des décalages logiques. Pour les modules de détection de contours verticaux et horizontaux de *Sobel*, le nombre d'opérateurs arithmétiques est pour les deux cas de 5 opérateurs.



(a) Filtre Gaussian

(b) Détection de contours verticaux - Ved

(c) Détection de contours horizontaux - Hed

FIG. 5.2 – Structure binomiale de détection de contours à base de l'opérateur de *Sobel*.

5.2.2 Proposition d'un système auto-organisé pour détection temps réel de contours d'images

5.2.2.1 Architecture structurelle

L'architecture de l'application de détection de contours selon l'approche d'auto-organisation est structurée et réalisée de la manière suivante. Le traitement considéré est décomposé en quatre parties réalisées chacune par un module. Quatre modules (M_1, M_2, M_3 et M_4) réalisent respectivement la fonction de filtrage *Gaussien*, la détection de contours verticaux, la détection de contours horizontaux et la somme des deux détections à partir des modules de détections horizontaux et verticaux. Le structure d'un module du système auto-organisé proposé est présenté dans la figure 5.3.

Chaque module est composé de 4 blocs. Le bloc «*Traitement principal*» qui effectue la fonction principale d'un module. Dans notre cas, cette fonction principale correspond soit à la fonction de filtrage *Gaussien* (module E_1), de détection de contours horizontaux et verticaux (modules E_2 et E_3) ou sommateur (module E_4). Ensuite, un deuxième bloc correspond à des ressources de calcul redondantes. Cette redondance fonctionnelle par des fonctions ou opérateurs de calcul dans un module est intégrée de manière que chaque module du système puisse être substitué ou remplacé par d'autres modules du système. Dans notre cas, un des modules réalisant le détecteur de contours peut être remplacé par la combinaison des trois autres mo-

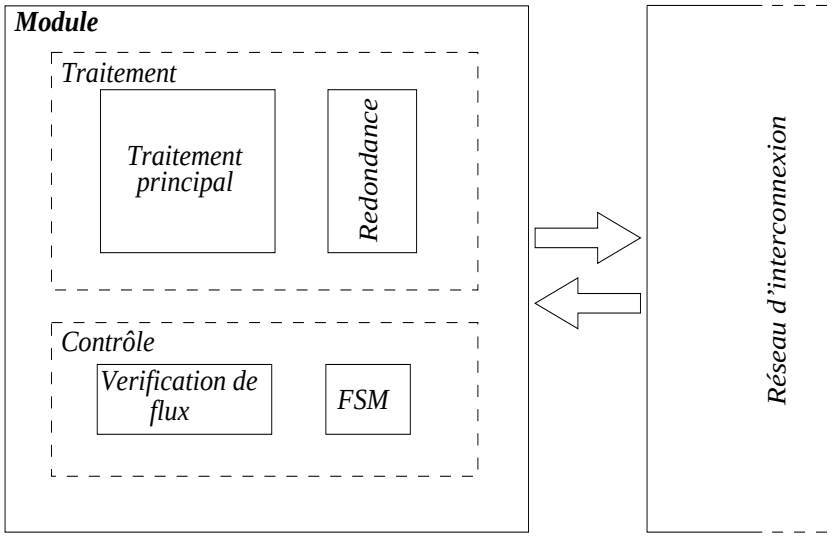


FIG. 5.3 – Schéma synoptique d'un module.

dules. Cette redondance fonctionnelle est définie par l'expression suivante :

$$\forall e_i,\ (1 \leq i \leq 4), \exists \sum_{j=1, j\neq i}^{4} e_j \Rightarrow e_i \subset \sum_{j=1, j\neq i}^{4} e_j. \tag{5.1}$$

Dans le cas applicatif considéré, la fonction de chaque module (E_i) est distribuée dans les trois autres modules (E_{jp})($j \neq i$). Aucun des modules, excepté dans notre cas module mettant en œuvre uniquement l'opération d'addition, ne peuvent pas être remplacés fonctionnellement par un seul autre module. Sa substitution est assurée par la combinaison des autres modules du système. L'objectif est de minimiser les ressources nécessaires à la conception du système (des modules) à travers une minimisation de l'intégration d'opérateurs locaux de redondance.

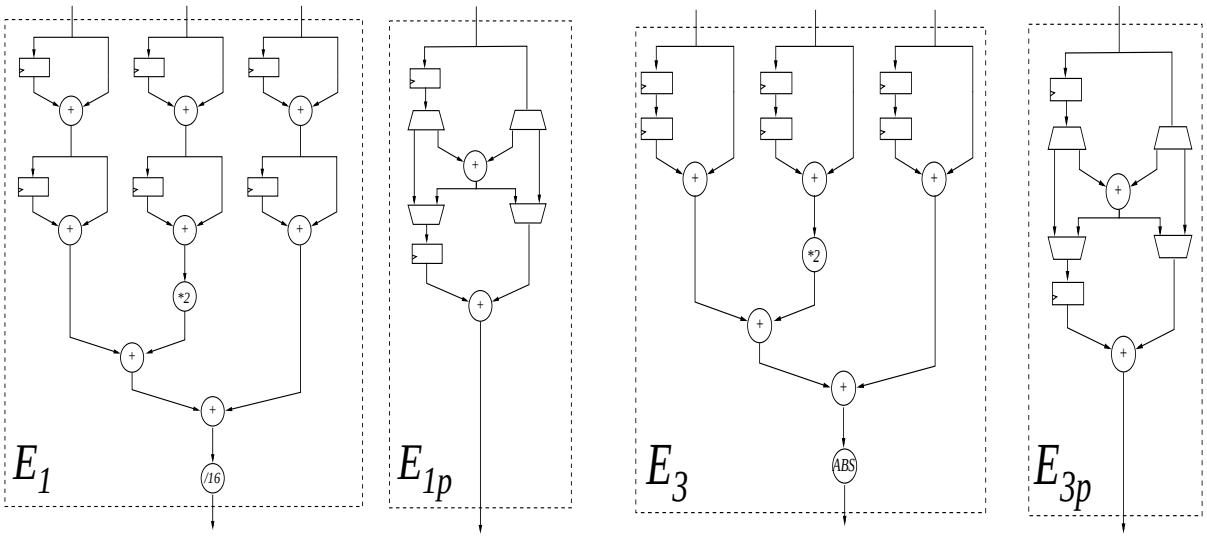
Pour la réalisation de notre système applicatif, la redondance fonctionnelle au sein des modules du système s'effectue par une analyse des fonctions principales de chacun des modules. Le but est de déterminer les parties redondantes suffisantes à intégrer dans les modules pour permettre plusieurs combinaisons des sous-fonctions principales locales des modules. Ainsi, une redondance d'implantation de la fonction principale globale du système selon la condition énoncée par l'expression 5.1 est réalisée. Les figures 5.4 et 5.5 illustrent les résultats de cette analyse en termes de blocs redondants intégrés à chaque module constituant le système auto-organisé applicatif de détection de contours.

Par exemple, la partie redondante du premier module E_{1p} contient 2 opérateurs arithmétiques d'addition, 4 opérateurs logiques multiplexeurs / démultiplexeurs et des registres. En fonction de signaux de contrôle sur les entrées des multiplexeurs / démultiplexeurs, ces opérateurs logiques et arithmétiques de redondance permettent de mettre en œuvre des sous-fonctions des fonctions principales des modules E_2 et E_3 (voir figures 5.4 et 5.5).

En plus d'un bloc de redondance, un module de système auto-organisé dispose de deux blocs de contrôle (voir schéma synoptique de la figure 5.3) : un bloc de « *vérification de flux* » et un bloc de contrôle *FSM*. Le bloc de « *vérification de flux* » a pour rôle principale la gestion de réception, de traitement et de retransmission du *flux informationnel* des modules du système auto-organisé vers les autres modules. Plus précisément, il s'agit de mise en œuvre de l'approche de concept d'auto-organisation de système matérielle défini dans le chapitre 3. Le bloc de contrôle *FSM* assure le contrôle local du module. A partir des informations renseignées par le *flux informationnel*, un bloc *FSM* gère la mise en œuvre des décisions sur le mode de fonctionnement local du module considéré. Ce bloc de contrôle est une machine d'états dont le fonctionnement et le graphe de transition sont détaillés dans la sous-section 5.2.2.4 suivante. L'ensemble des modules du système auto-organisé applicatif considéré communiquent à travers un réseau sur puce reconfigurable par transmission et réception de paquets de données dont la structuration est détaillée dans la sous-section suivante.

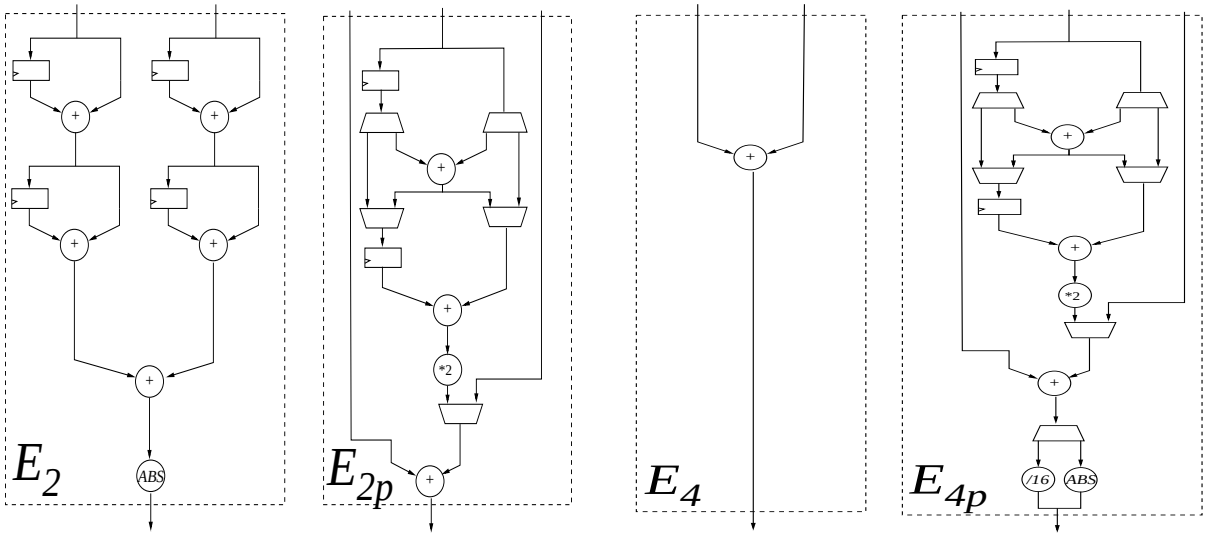
5.2.2.2 Réseau de communication inter-modules et format de paquets de données

Les modules du système auto-organisé proposé communiquent à travers un réseau sur puce reconfigurable de type *CuNoC* ou *QNoC* à travers des paquets de données de taille de 32 bits. La structure d'un paquet de données pour le système proposé à 4 modules, correspondant à une structuration de données adaptée aux réseaux *CuNoC* et *QNoC* (voir chapitre 4), est illustrée par la figure 5.6a. Le format de paquets correspond à un format modifié de paquets utilisés dans les approches *CuNoC* et *QNoC*.



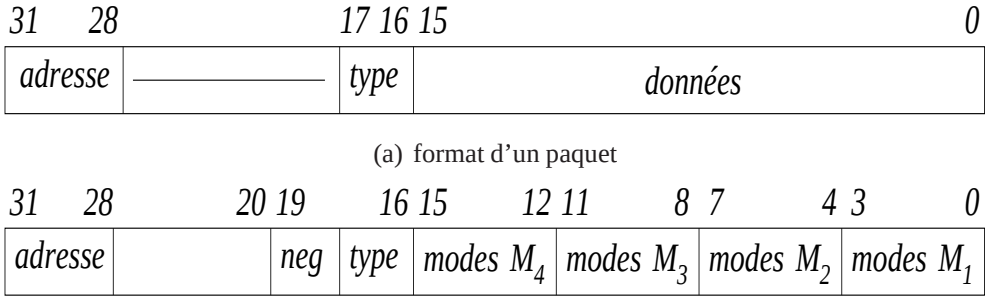
(a) Filtre Gaussian (b) Détection de contours verticaux - Ved

FIG. 5.4 – Blocs des traitement principaux et de redondance des modules M_1 et M_3 - I.



(a) Détection de contours horizontaux - Hed (b) Détection de contours horizontaux - sommateur

FIG. 5.5 – Blocs des traitements principaux et de redondance des modules M_2 et M_4 - II.



(a) format d'un paquet

(b) format du flux

FIG. 5.6 – Format de paquets de donnée du système auto-organisé applicatif proposé.

La particularité du format adopté, pour l'application considérée, est la distinction au sein d'un paquet du type de données transporté. On distingue alors 3 types de paquets :

– Les paquets de données pour le traitement principal,
– Les paquets de données pour la redondance fonctionnelle,
– Les paquets de données contenant le *flux informationnel*.

Chaque paquet de données, quel que soit son type de données transporté, contient une adresse de destination (voir figure 5.6). Cette adresse est contenu dans le champ situé entre les bits 28 à 31 d'un paquet. La distinction des trois types de paquets est codée à travers les bits 16 et 17. Plus précisément, des paquets contenant des données pour le traitement principal sont codés par "00", les paquets contenant les données pour la redondance fonctionnelle sont codés par "01" et les paquets contenant le *flux informationnel* sont codés par "11". Ainsi, en analysant

les bits 16 et 17 d'un paquet reçu, un module reconnaît le type de paquets de données considéré lui permettant soit la réception de ses données à traiter, soit de déterminer ses directives de traitement.

Pour les paquets de données de types traitement principal ou redondance fonctionnelle, les bits 0 à 15 contiennent les données à traiter soit par la fonction principale soit par la fonction de redondance d'un module selon son mode ou non de redondance fonctionnelle. Les autres bits 18 à 27 ne sont pas exploités pour ces deux types de paquets de données. Pour les paquets de données de type *flux informationnel*, les bits de 0 à 15 contiennent les modes de fonctionnement de chaque module du système. Dans notre cas applicatif, les quatre bits situés respectivement de 0 à 3, de 4 à 7, de 8 à 11 et de 12 à 15 sont associés et réservés aux modules M_1, M_2, M_3 et M_4 (voir figure 5.6b). Les bits 18 et 19 sont également utilisés par les paquets *flux informationnel* et servent à de « négociations » des modules lors des attributions des directives aux modules à travers le *flux informationnel* (voir sous-section 5.2.2.4).

5.2.2.3 Bloc de « vérification de flux »

Le bloc de « vérification de flux » est une machine d'états dont le diagramme de transition est présenté par la figure 5.7. Le *flux informationnel*, n'existant pas au démarrage du fonction-

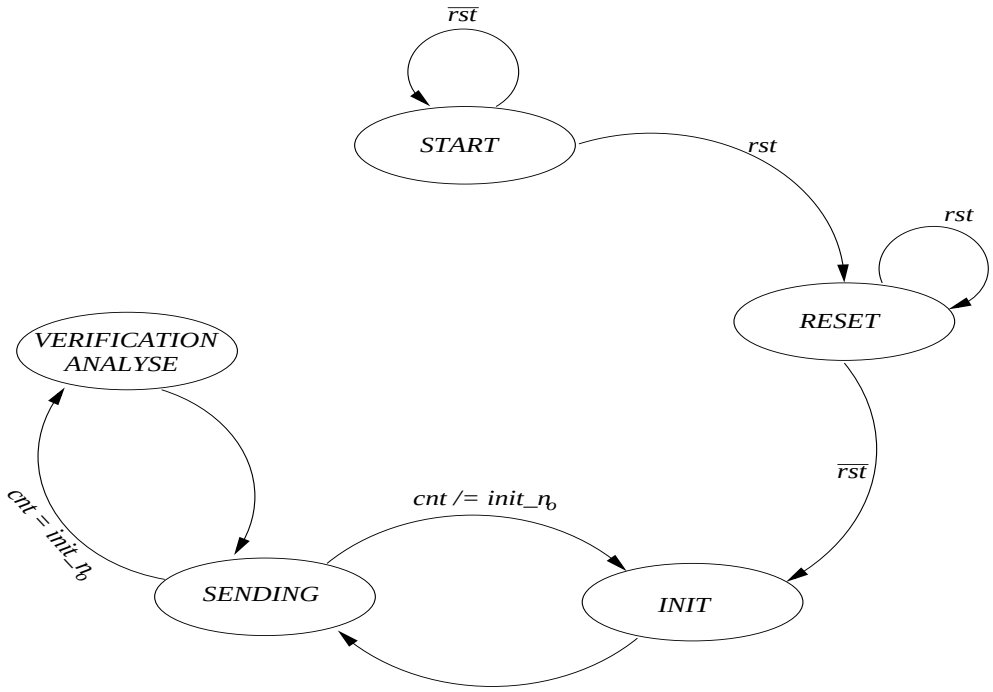


FIG. 5.7 – Diagramme de transition du bloc de « vérification de flux ».

nement du système (état *START*) est préalablement créé suite au démarrage du système. Cette étape est réalisée dans l'état initial *INIT* du bloc « vérification de flux ». Avant cette étape, un

module parmi l'ensemble des modules du système est désigné pour démarrer la création du *flux informationnel* au moment du démarrage du système (état *START*). Puis, un état *RESET* initialise à zéro l'ensemble des variables et signaux des modules du système. Dans notre cas applicatif, le premier module M_1 est désigné comme le module initiateur pour la création du *flux informationnel*. Au cours de cet état *INIT*, les autre modules du système restent inactifs jusqu'à la création effective du *flux informationnel*. Pendant la création du *flux*, le module initiateur intègre dans le *flux* les informations relatives à son état de fonctionnement (mise à jours de ses états initiaux). A l'issu de cette phase de conception du *flux informationnel* par le premier module, ce dernier transmet le *flux* initial crée vers les autres modules. Cette étape d'envoi s'effectue durant l'état *SENDING*. A la réception du *flux informationnel* initial par le second module, ce dernier intègre également les informations relatives à son état de fonctionnement initial avant de le transmettre à son tour vers les modules suivants. Cette procédure d'initialisation est effectuée pour l'ensemble des modules du système, jusqu'à ce que tous les modules du système aient indiqué leurs états initiaux au sein du *flux informationnel*. Ces informations d'états de chaque module seront ensuite accessibles à l'ensemble des modules du système à chaque nouvelle réception du *flux informationnel*. Cette étape d'initialisation caractérisée par les états *INIT* et *SENDING* se termine après plusieurs passages du *flux informationnel* à l'ensemble des modules du système. En effet, le *flux informationnel* devient globalement valide et à jour pour l'ensemble des modules du système qu'après un nombre de passage dans les modules (nombre de tours). Ce nombre de passage défini lors de la phase de conception correspond une latence opérationnelle du *flux informationnel* pour le système. Dans notre cas applicatif, le *flux informationnel* devient globalement opérationnel après quatre passages dans chaque module (4 tours). Le nombre de tours est une fonction du nombre de modules constituant le système et des problèmes possibles pouvant survenir au cours du démarrage du système (tel que non démarrage d'un module du système, etc). Dans le cas, où la création du *flux informationnel* dans la phase d'initialisation ne s'achève pas correctement, un signal d'erreur est alors généré (dans l'état *INIT*). La figure 5.8 présente un extrait de résultats de simulation illustrant la phase d'initialisation et de création du *flux informationnel* au sein du système auto-organisé proposé. On remarque bien qu'à l'issu d'un premier passage du *flux informationnel* dans les modules du système, les modes de fonctionnement de chaque module sont codés par une même valeur hexadécimale *F* correspondant au code du mode « état initial » de chaque module du système. Ceci représente le code de l'état initial de chaque module du système.

Après la création du *flux informationnel*, le système auto-organisé devient alors opérationnel et commence les traitements que doivent réaliser l'ensemble de ses modules. Initialement, après l'étape de remise à zéro (état *RESET*) et avant la phase de création du *flux informationnel* l'ensemble des modules du système sont opérationnels pour réaliser leurs traitements principaux mais ne disposent pas de données à traiter. En effet, ces traitements démarrent uniquement

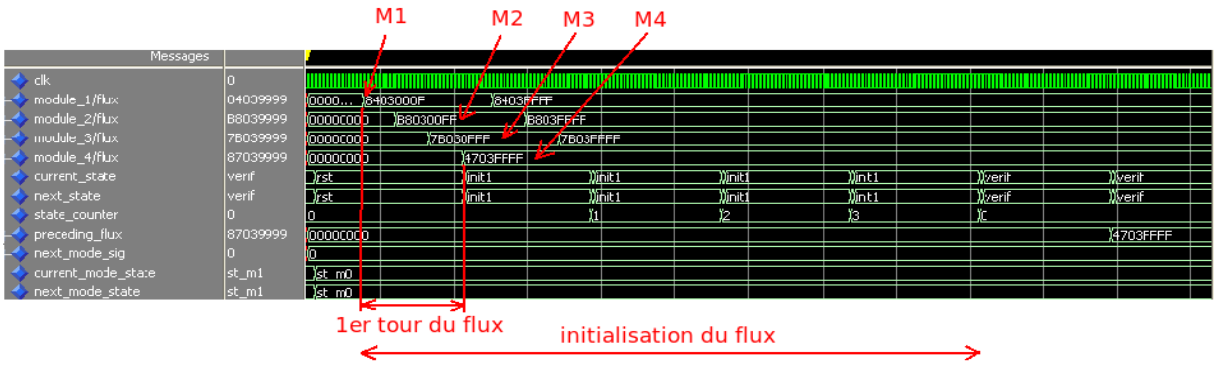


FIG. 5.8 – Extrait de simulation de la phase d'initialisation et de création du *flux informationnel* sous environnement *Modelsim*.

après les phases d'initialisation et de création du *flux* terminées. Un module interface entre les modules du système et une mémoire contenant les données à traiter par les modules est également initialisé à la fin de la phase de création du *flux* par le module initiateur (désigné lors de la procédure de création du flux). A la fin de la phase d'initialisation du flux, le module initiateur, envoie un paquet de confirmation au module interface lui indiquant que l'envoi des données pour traitement peut démarrer. La figure 5.9 illustre l'envoi d'un paquet de confirma-

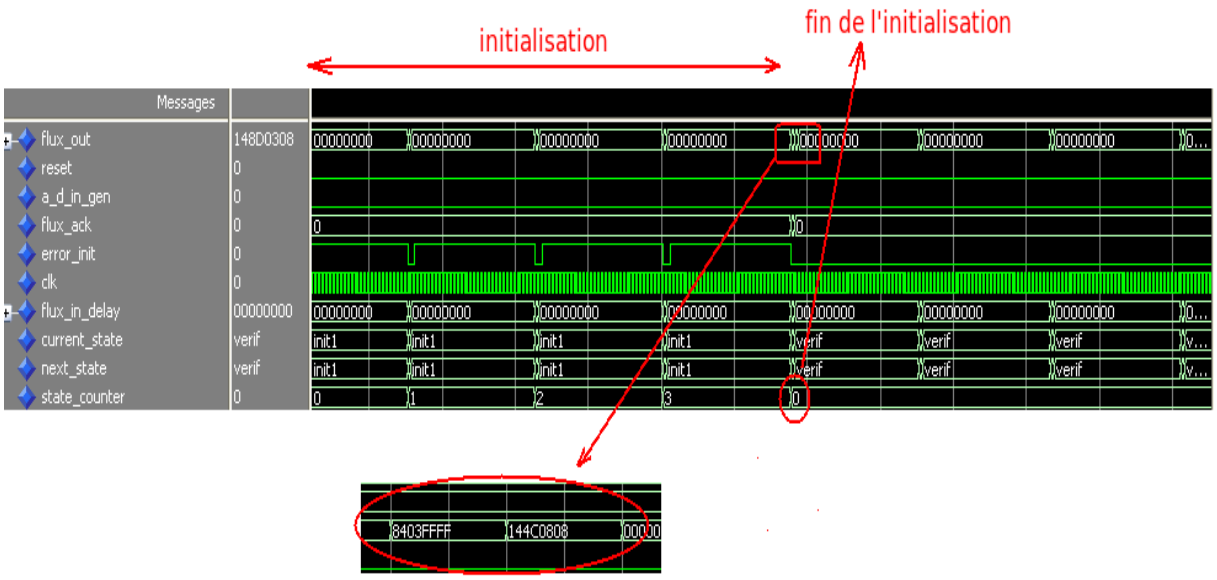


FIG. 5.9 – Extrait de simulation de la fin d'étape d'initialisation du *flux informationnel*.

tion du module initiateur (module M_1) au module interface. Le premier paquet du plan zoom de cette figure présente le *flux informationnel* contenant l'adresse du module M_2 et les modes de fonctionnement (valeurs initiales FFFF) de l'ensemble des modules du système. Le deuxième

paquet correspond au paquet de confirmation de la fin d'étape d'initialisation du *flux informationnel*. Ce paquet contient à la fois l'adresse du module interface et les informations contenant la vitesse d'envoi des données à traiter. Ces valeurs (deux octets du poids faible) peuvent varier selon plusieurs valeurs dont les détails seront donnés dans la section 5.2.2.5.

Après la phase d'initialisation, le bloc de « vérification de flux » alterne entre les états : *VERIFICATION & ANALYSE* et *SENDING* (voir figure 5.7). Le premier état correspond à l'état du module recevant le *flux informationnel* pour analyse et traitement dans le but de l'inviter à réaliser d'éventuelles modifications de traitement ou un nouveau traitement selon les informations contenues dans le *flux*. Ainsi, si un changement de mode de fonctionnement d'un module est signalé dans le *flux informationnel*, le module considéré est informé de cette nouvelle situation d'un module du système. Ensuite, à partir de la consultation de son *descripteur*, le module informé peut proposer une solution éventuelle dans la nécessité d'une réponse adaptée et correspondant à ce changement survenu. Dans le cas où le module n'a pas de réponses appropriées au changement survenu, il en informe les autres modules à travers une inscription dans le *flux informationnel* par un code prédéfini, ou bien en ne modifiant pas son état précédent. Dans notre application, la non résolution par un module, faisant suite à une diffusion par *flux informationnel* de modification de contexte, correspond au maintien du mode d'état de fonctionnement du module. Le module conserve donc son état précédent en cas de solutions appropriées inexistantes à des changements survenus et ne pouvant être proposées par le module. Dans le cas, où il n'existe pas d'informations de changements d'état des autres modules du système dans le *flux informationnel*, le module indique ses « intentions » de changements de modes de fonctionnements. En effet, une seule requête par module d'appel à modification de contexte peut s'effectuer simultanément. La phase de vérification du *flux* est prioritaire à une phase d'« alerte locale » d'un module indiquant sa variation d'état. Plus précisément, un module ne peut pas initier un changement et le faire savoir à travers le *flux informationnel*, si il existe déjà une demande de changement signalée également dans le *flux informationnel* par un autre module. Dans ce cas, le module considéré s'abstient jusqu'à ce que plus aucun nouveau changement soit signalé dans le *flux informationnel* par d'autres modules. Cette abstinence s'effectue également pour un module lorsqu'aucune proposition de résolution par ce module n'existe pour répondre à une demande de modification en cours par un module. Le *flux* traité est ensuite de nouveau renvoyé vers les autres modules du système pour information du nouveau contexte de résolution.

5.2.2.4 Bloc de contrôle de module *FSM*

Chaque module du système auto-organisé dispose d'un « contrôle local » nommé bloc *FSM* dont l'objectif principal est de piloter le module en considérant les informations des états de ses modules voisins du système (approche circulaire local de diffusion du *flux informationnel* - voir

chapitre 3). Ces informations sur les modules voisins sont fournies par le *flux informationnel*. Un module prend des décisions concernant son fonctionnement avec des accords mutuels avec ses modules voisins couverts par un même *flux informationnel*. Un bloc *FSM* d'un module correspond à une machine d'états dont une représentation partielle est présentée dans la figure 5.10.

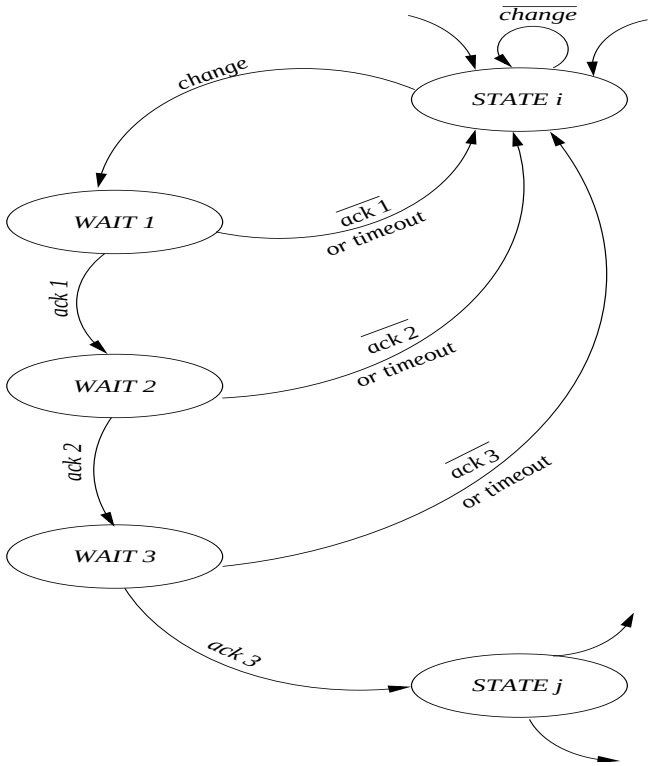


FIG. 5.10 – Extrait du diagramme de transition du bloc *FSM* d'un module.

Un module possède plusieurs modes de fonctionnement. Dans le cas de notre application, un module dispose au maximum de 8 modes de fonctionnement. Le nombre de bits utilisé pour représenter ces modes de fonctionnement dans le *flux informationnel* est de 4 (voir la figure 5.6). Les trois premiers bits du poids faible indiquent le mode de fonctionnement du module tandis que le quatrième bit du poids fort indique son état d'exécution. Si ce bit est dans un état '1', le module est en exécution, sinon les modules sont en phase de négociations. Le tableau 5.1 récapitule les modes de fonctionnement de chaque module du système. Les modes de fonctionnement réalisés par notre système pour l'application considérée sont les modes permettant le remplacement fonctionnel de modules au sein du système. Par exemple, le mode de fonctionnement "001" signifie que les modules M_2, M_3, et M_4 en plus d'exécuter leurs fonctions principales se combinent dans le but d'exécuter la fonction principale du module M_1 défaillant. Au démarrage, tous les modules du système sont par défaut dans leurs modes « fonction principale » illustrés par le code de fonctionnement de valeur "111" indiquant que tous les modules

TAB. 5.1 – Modes de fonctionnement des modules du système auto-organisé

mode de fonctionnement (3 bits du poids faible)	M_1	M_2	M_3	M_4
001	-	FP + M_1	FP + M_1	FP + M_1
010	FP + M_2	-	FP + M_2	FP + M_2
011	FP + M_3	FP + M_3	-	FP + M_3
100	FP + M_4	FP + M_4	FP + M_4	-
111	FP	FP	FP	FP

FP - fonction principale

exercent leurs modes de fonctionnements principaux.

Un module ne peut jamais changer de mode de fonctionnement sans signaler préalablement son intention aux autres modules à travers le *flux informationnel*. De plus, avant d'effectuer toute modification de traitement, un accord préalable entre tous les modules doit être établi. Pour obtenir un accord final des autres modules, un module initie des « négociations » avec ses modules avoisinants. La figure 5.10 illustre un exemple de changement d'états d'un module. La procédure de changements de mode de fonctionnement est la suivante : après la vérification par un module qu'aucune requête de changement existe pour un autre module signalée dans le *flux*, le module met à jour son champ associé dans le *flux* par des informations d'« intention » de changement de mode de fonctionnement. Il devient alors un module « demandeur ». Puis il transmet le *flux* modifié aux autres modules. Après passage du *flux informationnel* à travers l'ensemble des modules du système, le *module demandeur* reçoit de nouveau le *flux informationnel* et vérifie l'accord de sa requête par l'ensemble des modules du système. Ainsi, si aucune *objection* par les autres modules n'apparaît dans le *flux informationnel* concernant la requête de changement, la première phase de négociation est alors considérée comme un succès. La demande de requête s'applique et permet la modification du mode de fonctionnement du module « demandeur ». Dans le cas contraire, aucun changement n'est alors autorisé et les modules conservent leurs modes de fonctionnement en cours et continuent d'exécuter les mêmes fonctions avant la demande de requête.

Après une première phase de négociations réussie, le module passe à un état $WAIT_1$ dans le diagramme de transition du bloc de contrôle *FSM* (voir figure 5.10). Ensuite, le module « demandeur » propose à travers le *flux informationnel* aux autres modules de passer en mode *exécution*. Ce passage en mode *exécution* pour tous les modules du système entraîne la mise en œuvre des derniers préparatifs avant leurs passages véritablement dans le nouveau mode de fonctionnement. Si cette étape s'achève sans aucun problème et sans aucune contestation pouvant provenir de l'un des modules du système, le bloc de contrôle passe de l'état $WAIT_1$ à

l'état $WAIT_2$. Sinon, le module revient dans son état initial avant la requête (état $STATE_1$). Enfin, la dernière étape est le changement effectif du mode de fonctionnement par les modules. Ce changement de mode s'effectue concrètement à l'issu du troisième passage consécutif du *flux informationnel* dans les modules du système. Il est à noter que module « demandeur » de changement de mode assure la vérification des changements effectifs de mode de fonctionnement de l'ensemble des modules du système. En effet, après consultation du *flux informationnel* pour vérification des modifications effectives de mode d'exécution des autres modules du système, le module « demandeur » passe alors dans le nouveau mode de fonctionnement souhaité et correspondant à sa requête initiale. Il est à noter également que pendant les phases de « négociations », les modes de fonctionnements de tous les modules restent inchangés. C'est uniquement à l'issu d'un succès des phases de négociations que les modules peuvent passer d'un mode de fonctionnement à un autre pour éviter tout conflit de mode multiple interne au sein du système. En effet, ce conflit peut être à l'origine d'une autorisation de prise d'initiative propre des modules du système correspondant alors à un état de fonctionnement non organisé du système. La figure 5.11 présente un extrait des résultats de simulation illustrant le passage du mode de fonctionnement *ST_M*$_0$ (l'état initial) à l'état *ST_M*$_1$ (l'état correspondant à la substitution du module M_1 par les trois autres modules du système). Il apparaît clairement sur ces

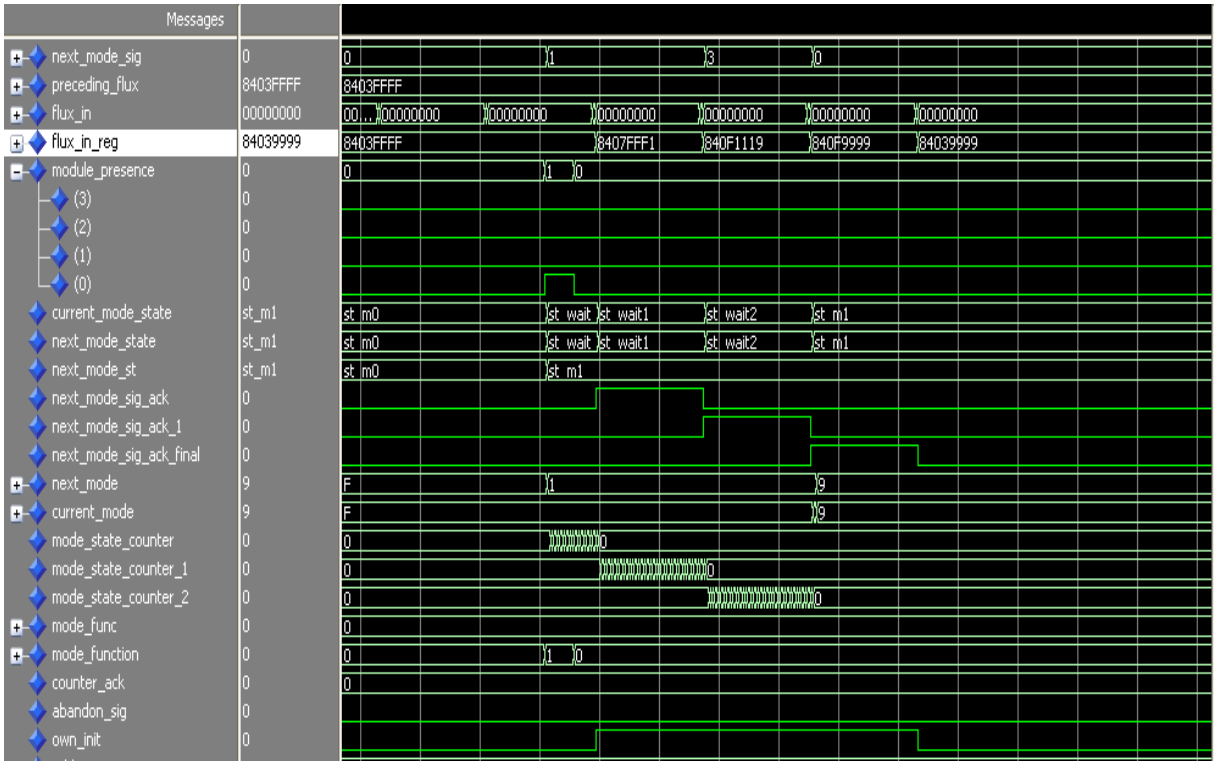


FIG. 5.11 – Extrait de résultats de simulation d'un passage d'un mode de fonctionnement à un autre du système auto-organisé proposé.

résultats que le passage à l'état *ST_M*$_1$, le module transite préalablement par les états $WAIT_1$, $WAIT_2$ et $WAIT_3$. Chaque état est validé par un signal d'acquittement correspondant à un état logique '1' des signaux *next_mode_sig_ack*, *next_mode_sig_ack_1* et *next_mode_sig_ack_final* dans les résultats de simulation présentés dans la figure 5.11. Dans ce diagramme de simulation logique, le signal *current_mode*, représentant le codage du mode de fonctionnement en cours par le module, prend la valeur du mode de fonctionnement *ST_M*$_1$ à la fin des phases de négociation (passage de la valeur hexadécimale "F" à la valeur "9" sur le diagramme de simulation de la figure 5.11).

5.2.2.5 Cas applicatif d'étude

Nous avons simulé le système auto-organisé proposé mettant en œuvre l'application de détection de contours considérée dans notre cas d'étude. Le système complet est composé d'une mémoire de données, de 4 modules (mettant en œuvre l'application divisée en quatre parties comme décrit précédemment), et connectés à travers un réseau QNoC [JTWB07b, JTWB07a]. Tous les paquets de données ou du *flux informationnel* entre les modules circulent à travers ce réseau QNoC où sont greffés ces modules en périphérie. La figure 5.12 illustre le système dans sa globalité.

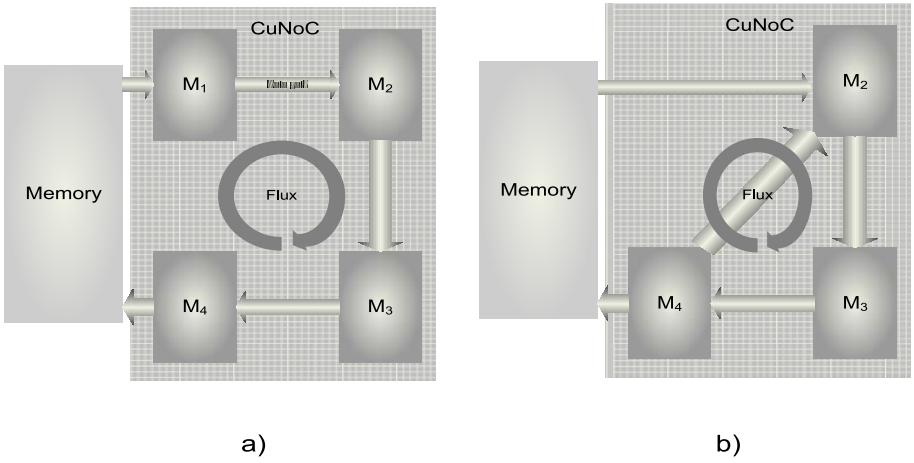


FIG. 5.12 – Cas de simulation considéré (a) avant (b) et après la défaillance du module M_1.

Une simulation de défaillance du module M_1 du système est réalisée en vue d'analyser et de valider le comportement auto-organisé global du système proposé. Dans ce cas d'étude, il est considéré que malgré la défaillance fonctionnelle d'un module (dans notre cas le module M_1), le bloc de « vérification de flux » du module défaillant reste néanmoins fonctionnel. Pour réaliser cette simulation, à un moment donné, le signal indiquant la défaillance du module M_1 est positionné à 1. Dans ce cas, les modules M_2, M_3 et M_4 à travers le *flux informationnel* deviennent informés de la défaillance survenue au niveau du module M_1. La figure 5.13 illustre

les changements se produisant dans le *flux* juste après le signalement de défaillance du module M_1. Nous constatons bien, qu'à chaque passage du *flux informationnel* à travers les modules,

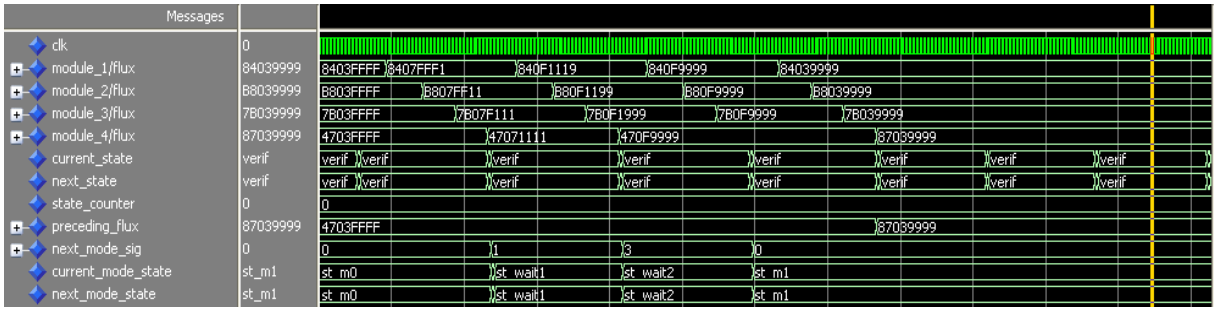


FIG. 5.13 – Changements dans le *flux informationnel* en cours de négociations entre les modules du système dans le cas d'une défaillance du module M_1.

les indications d'états des modules dans le *flux informationnel* changent de l'état FFFF (correspondant au mode de fonctionnement « fonction principale » pour chaque module) vers l'état "9999" correspondant au mode « fonction principale + substitution module M_1 » par les autres modules du système (mode 9 en hexadécimal, 1001 en binaire). Pendant la phase de négociation entre les modules, leurs modes de fonctionnement ne changent pas.

Avant le signalement d'indication de défaillance du module M_1 dans le *flux informationnel* (voir figure 5.13), les données sont traitées selon la séquence de traitement par les modules suivante : $M_1 - M_2 - M_3 - M_4$. Plus précisément, le cycle de traitement des données commence initialement par le module M_1, ensuite respectivement par les modules M_2, M_3 et M_4. Après le signalement de la défaillance du module M_1 et après l'organisation des autres modules pour le remplacer fonctionnellement, la séquence de traitement de données évolue et correspond au cycle de traitement suivant : $M_2 - M_3 - M_4 - M_2 - M_3 - M_4$. Ainsi, les paquets de données sont transférés successivement dans les modules M_2, M_3 et M_4 pour mettre en œuvre la substitution du module M_1. Puis à l'issu, les paquets de données sont ensuite de nouveau transmis successivement vers les modules M_2,M_3 et M_4 pour être traités par les fonctions principales de ces modules dans le but de finaliser le traitement globale du système. Dans ce cas de figure de défaillance, le temps de traitement est plus élevé que dans le cas initial, affectant donc les performances du système global. Impliquant par le même une cadence de données provenant de la mémoire moins élevée que dans le cas initial. A la fin des négociations entre les modules au cours de la défaillance, le bloc de « vérification de flux » du module initiateur (dans notre cas, le bloc du module M_1), envoie un message au module interface (situé entre les modules de traitement et la mémoire du système pour informer le module interface du cycle de changement de traitement). Ce message contient la nouvelle adresse du module devant recevoir les données à traiter (dans notre cas, non plus le module M_1, mais le module M_2) et la vitesse d'envoi de ces paquets de données. Cette procédure est illustré dans la figure 5.14. Si on compare les valeurs

contenues dans le paquet envoyé vers le module interface présentées respectivement dans les figures 5.14 et 5.9, on constate bien les données correspondant à la vitesse d'envoi des paquets de données sont différentes (les 2 octets du poids faible). La valeur de vitesse d'envoi dans le deuxième cas est approximativement deux fois moins élevée par rapport à la vitesse d'envoi initiale (0308 par rapport à 0808). On remarque également que le flux ne dessert plus le module M_1 et qu'il limite sa circulation sur les trois autres modules M_2, M_3 et M_4.

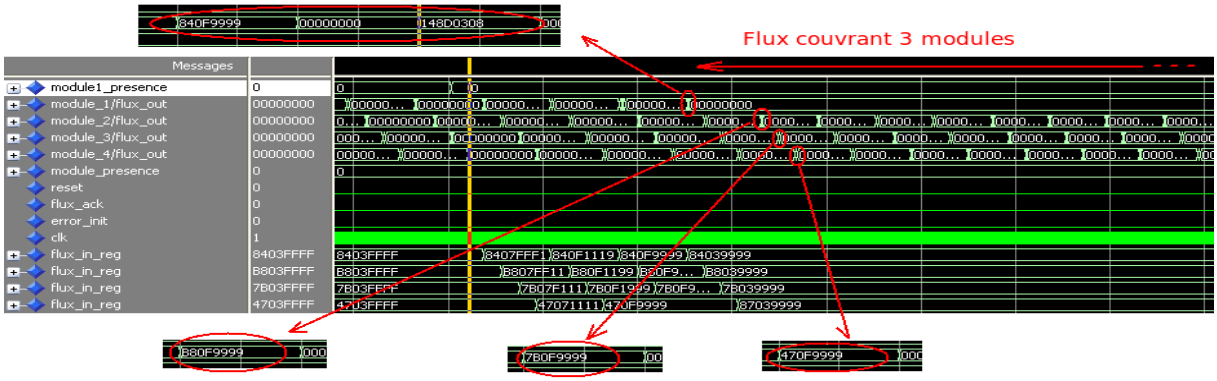


FIG. 5.14 – *Flux informationnel* desservant uniquement les modules M_2, M_3 et M_4 après défaillance du module M_1.

5.2.3 Implantation *FPGA* et résultats de synthèse

Le système auto-organisé d'application de détection de contours d'images présenté dans les sections précédentes a été implanté sur une plateforme *FPGA* de type Xilinx Virtex 5. La plateforme matérielle utilisée est la plateforme *ML505* dont une illustration est donnée dans la figure 5.15 [Xil07]. Cette plateforme est composée [Xil07] :

- D'un circuit *FPGA* Virtex 5 - XC5VLX50T-fg1136 caractérisé par des ressources logiques de 7200 en blocs logiques (CLB Slices), 36 Kbits de blocs RAM et de 480 entrées / sorties [Xil08],
- D'une mémoire DDR2 d'une capacité de 256 MB et d'une taille de bus de 64 bits,
- D'entrées / sorties vidéo (entrée VGA, sortie *DVI*), audio (codec AC97, entrée microphone, etc),
- Des connecteurs de protocole série (RS232), de souris PS/2 et de clavier,
- D'un afficheur LCD à 2 lignes de 16 caractères,
- D'un port de configuration JTAG permettant la programmation et configuration du circuit à l'aide de l'outil de développement Xilinx *EDK*[17],

[17] *EDK* - Embedded Developpement Kit, l'outil de Xilinx permettant le développement d'applications complètes

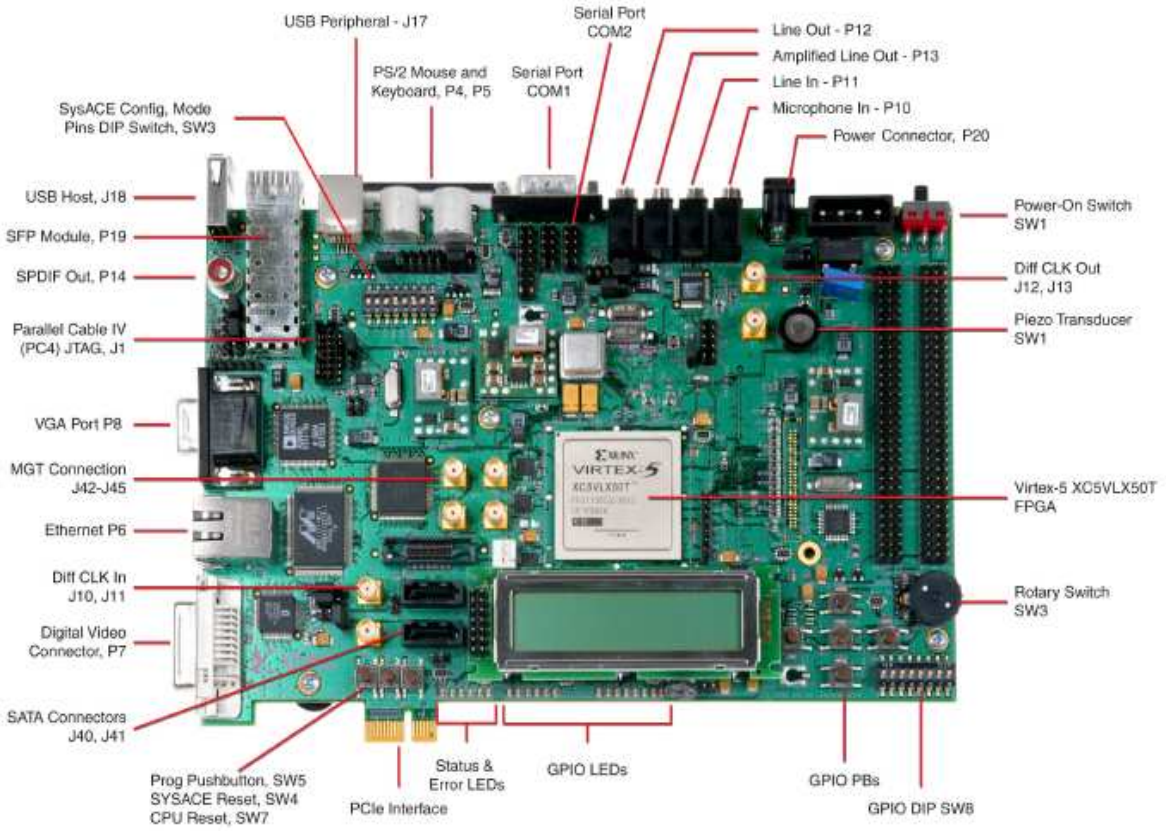


FIG. 5.15 – La plateforme d'évaluation *ML505* - vue d'ensemble.

– D'une interface Ethernet (RJ45).

Afin de valider expérimentalement les aspects abordés au cours de cette thèse à travers la conception d'un système auto-organisé de détection de contours d'images, d'autres modules permettant la gestion des périphériques de la plateforme sont également intégrés. La figure 5.16 présente l'architecture globale du système implanté. Cette architecture globale intègre également entre le système de détection de contours (composé de quatre modules de traitement, d'un réseau QNoC) et d'une mémoire externe *SDRAM* (externe au *FPGA*), Une architecture *SoC* à bus partagé de propriété Xilinx. Cette dernière est composée de blocs permettant le contrôle et la gestion des accès vers la mémoire externe *SDRAM* (contrôleur de mémoire *MPMC v4.03a* (*Multi-Port Memory Controller*), disponible parmi les *IP*[18] proposés par l'outil *EDK* 10.1). Ce contrôleur *SDRAM* est relié à un bus partagé local PLB v4.6 (*Processor Local Bus*), de taille de 128 bits et également disponible parmi les *IP* de la librairie *EDK Xilinx*. A ce même bus, un cœur de processeur *MicroBlaze Xilinx* et un module de débogueur matériel associé à ce dernier sont

à processeur embarqué et leur intégration dans un *FPGA*. Il permet également la programmation d'un ou plusieurs processeurs *MicroBlaze* et de leurs périphériques. Pour plus amples informations, se reporter à la documentation de Xilinx sur `http://www.xilinx.com`

[18] *Intelectual Property* - Résultat d'un travail de création de l'esprit qui fait l'objet d'un droit [GRD]

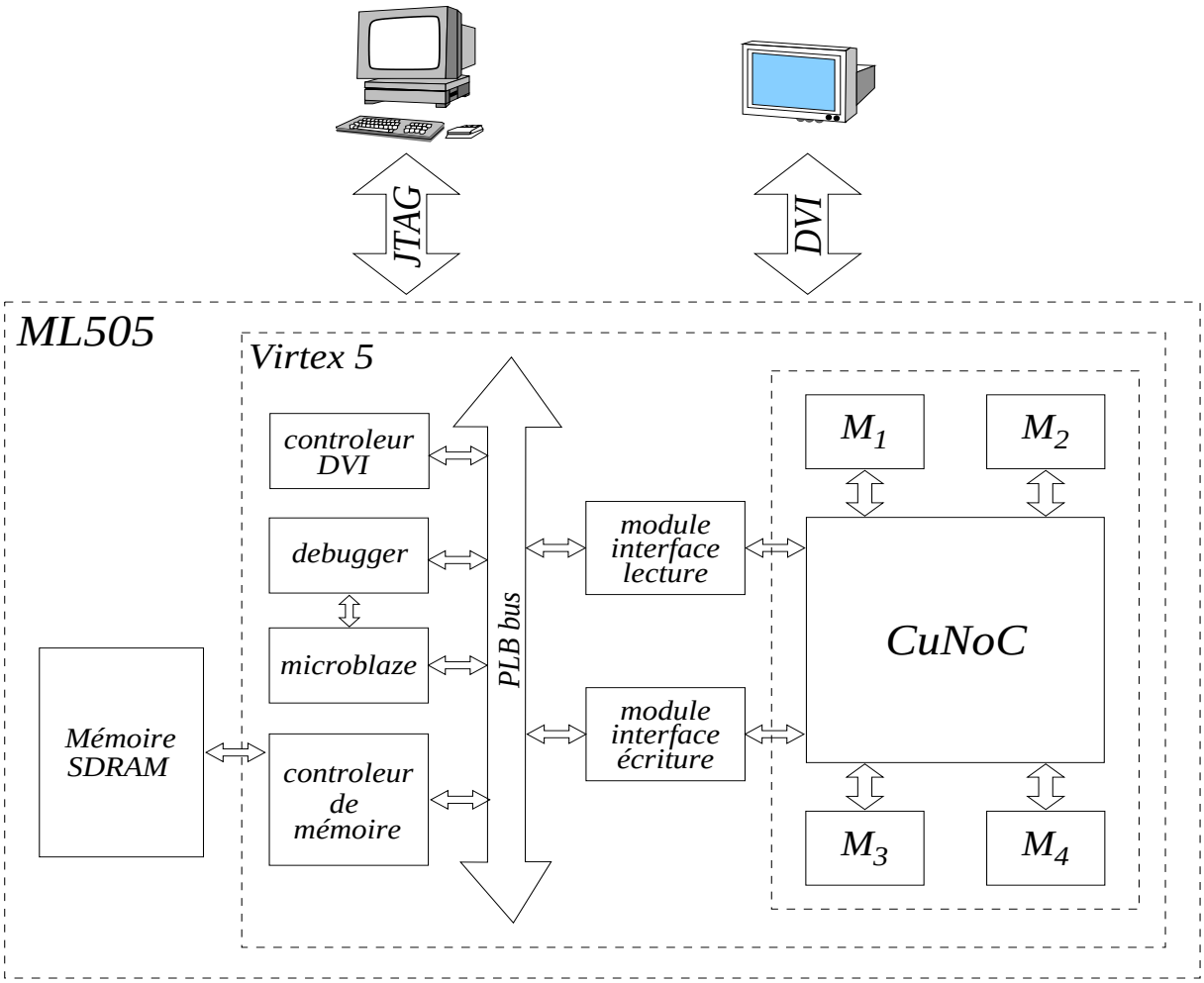


FIG. 5.16 – Architecture globale du système auto-organisé de détection de contours d'images implantée dans la plateforme d'évaluation *ML505*.

implantés (Microblaze 7.10d et d'ébogueur 1.00d). Les principaux rôles de ces deux modules supplémentaires sont à la fois d'assurer le remplissage de la mémoire externe *SDRAM* et permettre la visualisation de certains signaux internes au circuit *FPGA* à travers l'outil *ChipScope Analyzer* de *Xilinx*. Au démarrage de la plateforme, après sa mise sous tension, la mémoire externe *SDRAM* est chargée des données correspondant aux images à traiter. Ce chargement s'effectue à l'aide de l'outil *EDK*, qui permet d'accéder à la mémoire externe via le débogueur et *MicroBlaze* pour y stocker des données souhaitées. Les données traitées sont affichées sur un écran relié à la plateforme. L'affichage est géré par un contrôleur *DVI* permettant un affichage en résolution 600×480. Les données à afficher sont également stockées temporairement dans la mémoire externe *SDRAM*. En effet, cette mémoire externe *SDRAM* est répartie en deux zones. Une zone correspondant aux adresses de partie inférieure de la mémoire contenant les données à traiter, et une partie supérieure contenant les données pour visualisation. Deux mo-

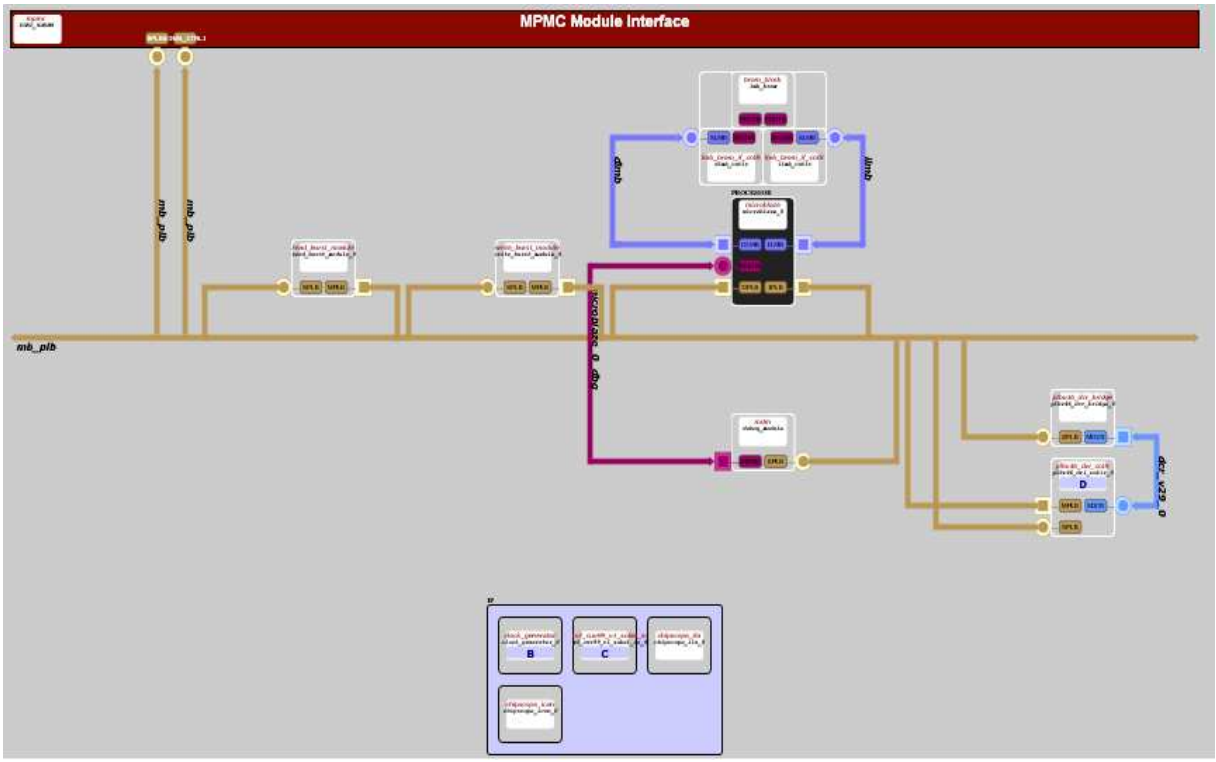


FIG. 5.17 – Vue globale du système de détection de contours dans l'outil *EDK* de *Xilinx*.

dules interfaces permettent les lecture et écriture des données dans la mémoire. Un « module interface lecture » initie en permanence (lorsque le bus est disponible) la lecture des données de la mémoire pour les écrire dans ses mémoires tampon. Un « module interface écriture » reçoit les données traitées de l'architecture auto-organisée et les stocke temporairement dans ses mémoires tampon pour ensuite les envoyer et stocker dans la mémoire externe *SDRAM* pour les afficher. La figure 5.17 illustre une vue d'ensemble sous forme de blocs diagramme des connexions entre les *IP* instanciés de l'architecture *SoC* à bus partagé et interfacé avec l'architecture auto-organisée de détection de contours d'images dans le circuit *FPGA* utilisé. Cette architecture *SoC* à bus partagé est entièrement générée avec l'outil *EDK* 10.1 de *Xilinx*. L'architecture auto-organisée (modules de traitement et réseau *QNoC*) n'est pas visible sur cette figure car cette architecture est câblée directement en *VHDL* avec les modules interfaces de l'architecture *SoC* à bus partagé.

Le tableau 5.2 présente les résultats de synthèse de l'implantation du système global de détection de contours sur la plateforme *ML-505*. Ces résultats montrent une utilisation de ressources supplémentaires par rapport à une implantation sans fonctionnalité de redondance. Ces ressources supplémentaires sont donc essentiellement liées à l'implantation des opérateurs logiques et arithmétiques de redondance rajoutée dans les modules de traitement et aux blocs de vérification de flux. On remarque également que les blocs de vérification de flux pour tous les

TAB. 5.2 – Résultats d'implantation sur la plateforme *ML-505*.

Module M	FP[a] (CLB[d])	FP/M (%)	R[b] (CLB)	R/M (%)	BVF[c] (CLB)	BVF/M (%)
M_1	18	9.2	6	3.1	23	11.8
M_2	26	12.6	6	2.9	23	11.2
M_3	16	13.3	6	3.1	23	11.8
M_4	2	1.8	7	6.1	23	20

[a] fonction principale
[b] Redondance dans le module M
[c] Bloc de vérification de flux du module M
[d] cellules de blocs logiques

modules sont de même taille, n'étant pas négligeable par rapport à leur taille initiale. La figure 5.18 illustre le *floorplan* d'implantation du système global dans le circuit *FPGA* utilisé. Ce plan d'implantation est généré automatiquement et obtenu par l'outil de synthèse, placement et routage *EDK*. On identifie la localisation de l'ensemble des modules implantés dans le circuit dans la figure 5.18. La grande partie du circuit est exploitée par la gestion de la mémoire externe *SDRAM* (contrôleur de mémoire, MicroBlaze et son débogueur, module *ChipScope* pour la visualisation des signaux internes à la puce *FPGA* et le contrôleur d'affichage).

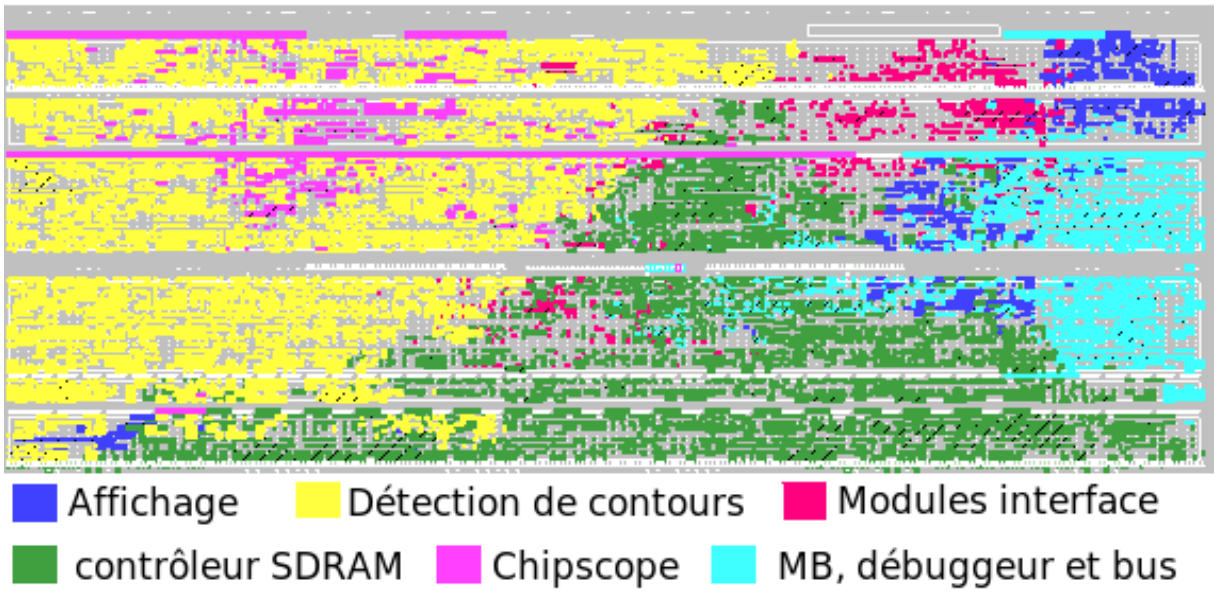


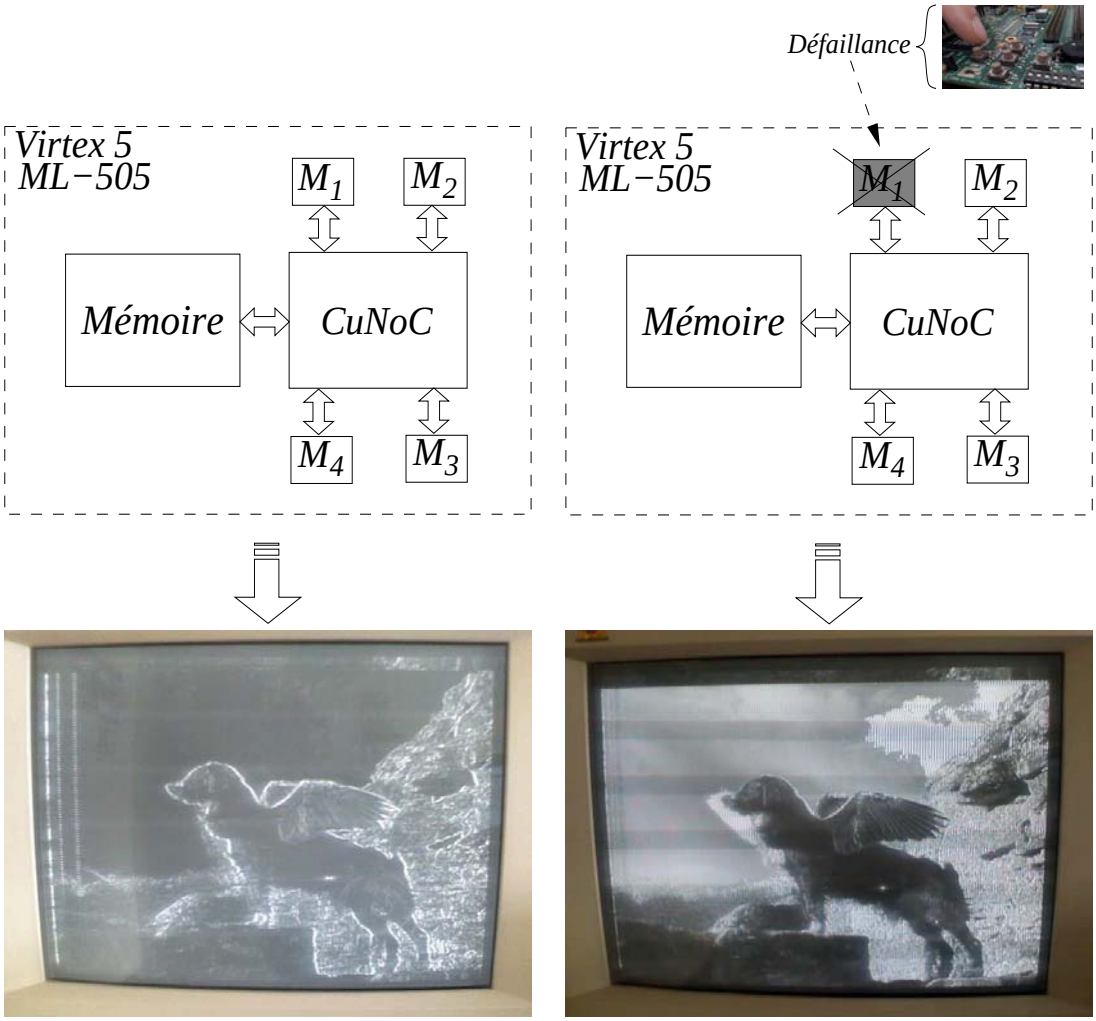
FIG. 5.18 – Visualisation du *plan d'implantation* (*floorplan*) du système de détection de contours sur le circuit *FPGA* de la plateforme *ML-505*.

5.2.4 Validation expérimentale

Le système complet présenté dans la figure 5.16 a été implanté sur la plateforme *ML-505* pour une validation expérimentale temps réel. Le système est cadencé avec une fréquence de 100 MHz. Les résultats de cette validation expérimentale sont présentés dans la figure 5.19.

Cette expérimentation s'effectue de la manière suivante : dans un premier temps, la mémoire externe *SDRAM* est chargée par une image. Le résultat du traitement de cette image est illustré dans la figure 5.19a. Ensuite, une défaillance du module M_1 de traitement du système auto-organisé est simulée. Cette simulation est réalisée à travers un bouton poussoir disponible de la plateforme où est implanté le système de détection de contours d'images (voir la figure 5.19b). Ce bouton poussoir signale physiquement un problème au module de traitement M_1, qui ensuite informe les autres modules du système du problème survenu à travers le *flux informationnel*. Une phase de négociation à travers l'échange d'informations par le *flux informationnel* entre les autres modules s'engage alors en vue de substituer le module M_1 considéré défaillant. Le résultat du cas de défaillance du module M_1 est présenté dans la figure 5.19b. En vue de mettre visuellement cette substitution (visualiser du passage d'un mode de fonctionnement à un autre des modules du système), le traitement d'images par les autres modules lors de cette substitution est volontairement et légèrement différent. Le retour au mode de fonctionnement initial pour l'ensemble des modules du système s'effectue également à l'aide d'un bouton poussoir disponible sur la plateforme matérielle utilisée.

Une vérification des valeurs du *flux informationnel* en temps réel, dans les cas des modes de fonctionnement des modules « fonction principale » et « fonction principale + substitution » , à travers une analyse des signaux internes au circuit a été réalisée. Cette vérification a été réalisée grâce à l'outil *ChipScope Analyzer* de *Xilinx*, qui permet la visualisation temps réel des valeurs des signaux préalablement choisis avant les phases de synthèse, placement et routage lors de l'implantation du système dans le circuit *FPGA*. Ces résultats sont présentés dans la figure 5.20. On constate que les valeurs du *flux informationnel* avant défaillance simulée sont identiques à ceux obtenues par simulation sous environnement *ModelSim* (voir la figure 5.20a). On constate également que la latence entre un envoi et une réception du *flux informationnel* entre deux modules de traitement est de l'ordre de 10 cycles d'horloge. L'acquisition des nouvelles valeurs du *flux informationnel* lors d'une défaillance simulée du module M_1 par appui sur le bouton poussoir de déclenchement du mode défaillant du système est présentée dans la figure 5.20b. Ces résultats montrent que les valeurs du *flux* changent et que le système de détection de contours d'images est en mode de fonctionnement « fonction principale + substitution M_1 » codé par le mode binaire "001".

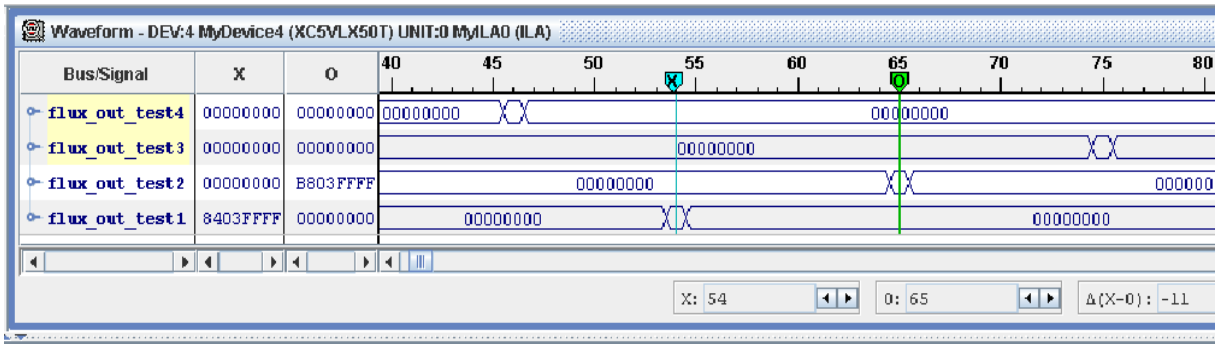


(a) avant (b) après

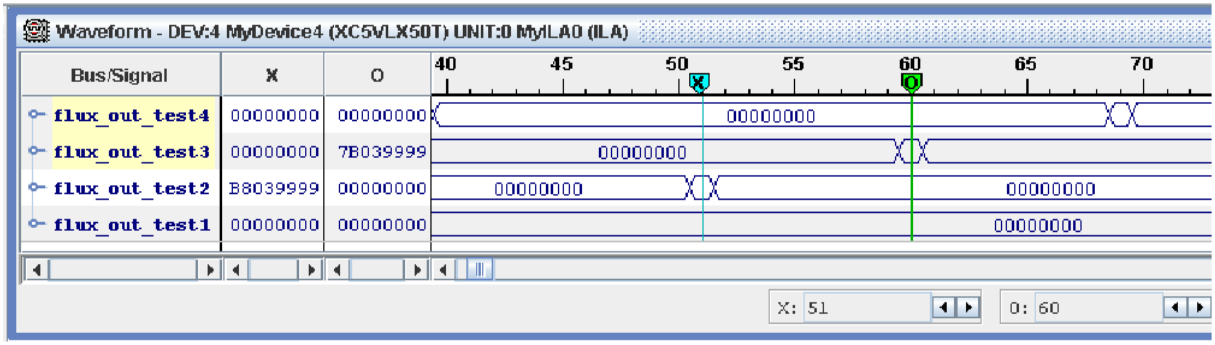
FIG. 5.19 – Résultats expérimentaux de l'implantation du système auto-organisé de détection de contours et de sa robustesse face à une défaillance sous-système.

5.2.5 Évaluation des performances

Une évaluation des performances en terme de temps de traitement d'images pour différentes tailles d'images est présentée dans le tableau 5.3. A partir de ces évaluations, un calcul du nombre d'images par seconde pouvant être traitées (*frames* par seconde) est déduit. Ces valeurs nous permettent de déduire la capacité du système auto-organisé proposé pour un traitement temps réel d'une séquence d'images vidéo. On constate que pour toutes les valeurs de résolution d'images (sauf pour le format *full HD*) et pour le cas d'implantation considéré (sans ou avec une défaillance simulée), le temps de traitement reste inférieure à la valeur maximale définie par le traitement de 25 images (*frames*) par seconde. Il faut également souligner que ces valeurs représentent des résultats d'évaluations issues de simulations et que seule une im-



(a) avant défaillance simulée du module M_1



(b) après défaillance simulée du module M_2

FIG. 5.20 – Acquisition des valeurs réelles du *flux informationnel* par de l'outil *ChipScope Analyzer* de *Xilinx*.

plantation peut donner des résultats réels de performances du système proposé. De plus, pour le traitement d'images de plus hautes résolutions où une quantité importante de données doit être traitée, une modification de l'architecture proposée doit être envisagée dans le but d'une adéquation algorithme-architecture vis-à-vis de la plateforme de prototypage ML505 utilisée. Cette modification architecturale doit principalement se centrer sur l'architecture *SoC* à bus partagé associé à l'architecture auto-organisée *NoC* proposée. En particulier sur sa partie interface entre la mémoire externe *SDRAM* et l'architecture de traitement *NoC*. En effet, ces interfaces de connexion sont de type bus de données, utilisées à la fois pour les transferts de données à traiter entre la mémoire et l'architecture *NoC*, l'architecture *NoC* et la mémoire et pour la visualisation des données traitées. Cette interface représente donc un véritable goulot d'étranglement des performances du système pour des résolutions d'images très élevées.

5.3 Conclusion

Nous avons présenté dans ce chapitre la conception et l'implantation d'une architecture auto-organisée appliquée au traitement d'images temps réel et basée sur les concepts d'auto-

TAB. 5.3 – Évaluation des performances du système global de détection de contours d'images vidéo.

Résolution de l'image		Temps de traitement		*Frames* par seconde	
		sans M_1 (ms)	avec M_1 (ms)	sans M_1	avec M_1
VGA	600×480	2.6	5.2	384.6	192.3
SVGA	800×600	4.8	9.6	208.33	104.17
XGA	1024×768	7.9	15.7	126.6	63.7
WXVGA	1280×768	9.83	19.7	101.7	50.9
full HD	1920×1080	20.7	41.5	48.2	24.1

organisation et de communication sur puce adaptée pour des systèmes sur puce reconfigurables. L'application considérée est un détecteur de contours d'images vidéo à base d'un détecteur de *Sobel* implanté à travers des filtres de structure binomiale dont les graphes flots de données ont été détaillés.

Nous avons explicité à travers cet exemple d'application, la stratégie de mise en œuvre de l'auto-organisation d'un système sur puce reconfigurable par application du concept de *flux informationnel*. L'ensemble des solutions d'implantation matérielle de la notion de l'auto-organisation ont été validées expérimentalement. En effet, nous avons montré l'auto-organisation matérielle de modules exécutant des fonctions principales de calcul et pouvant s'auto-organiser en vue de substituer un éventuel module défaillant au sein du système à travers des échanges locaux de données et sans aucun contrôle centralisé ou extérieur. Plus précisément, nous avons détaillé les démarches de subdivision architecturale d'une application dans un système constitué de quatre modules intégrant d'une redondance partielle structurée et permettant au système proposé de substituer à tout moment l'un des modules le constituant. Ainsi, chaque partie de l'application à réaliser, a été exécutée par un module du système auto-organisé par l'intégration d'opérateurs logiques, arithmétiques et d'éléments de mémorisation réalisant une fonction principale du module et correspondant à une partie du traitement à exécuter par le système tout en permettant éventuellement la réalisation de sous-fonctions d'autres modules du système dans le cas du déclenchement d'une auto-restructuration par le système. De même, nous avons détaillé la procédure de création d'un *flux informationnel* au démarrage du système et explicité les différents modes de fonctionnement de chaque module du système en vue d'un comportement global auto-organisé du système. Les blocs implantés et permettant la vérification de *flux* et de contrôle de modules ont été présentés en détails. La vérification des structures implantées pour la mise en œuvre de notre approche architecturale d'auto-organisation est validée. En effet, la vérification temps réel du contenu d'un *flux informationnel* et de son évolution au cours du temps permettant une auto-gestion des modules du système auto-organisé a été démontrée expérimentalement. De

plus, certains points de leurs fonctionnements ont été illustrés par des extraits de résultats de simulation. Le système auto-organisé proposé est considéré comme un *système reconfigurable auto-organisé* car il repose sur une structure de communication sur puce permettant une reconfiguration de sa topologie dans le but d'intégrer des modules dynamiques dans le système. En effet, les modules du système proposé communiquent à travers les paquets envoyés via un réseau *QNoC*. Plus précisément, nous avons détaillé les formats de paquets de données utilisé à la fois pour la communication des modules du système et pour véhiculer au sein du réseau un *flux informationnel*. Nous avons montré la validation expérimentale du système global par son implantation temps réel sur une plateforme de prototypage rapide *FPGA ML-505* de Xilinx. Sur l'exemple de simulation d'une défaillance d'un des modules, nous avons montré que le système continue de fonctionner malgré la présence de cette perturbation. Les résultats de synthèse du système proposé sur la plateforme de prototypage utilisée ont été également présentés ainsi que des évaluations de temps de traitement pour différents format d'images vidéo à traiter. Ces résultats démontrent la viabilité de la solution architecturale proposée à la fois en terme d'adéquation algorithme-architecture et d'auto-adaptabilité / auto-organisation.

En conclusion, l'ensemble des résultats expérimentaux présentés nous ont permis de valider concrètement la plupart d'aspects abordés au cours de cette thèse, notamment l'aspect de l'approche architecturale rendant un système capable de gérer de manière autonome et intelligente les éventuelles perturbations provenant de l'environnement l'entourant.

L'utilisation d'une structure de communication sur puce *QNoC* permet d'envisager la mise en pratique de l'auto-organisation de notre système dans le cas d'un placement dynamique d'un module au sein du réseau par l'utilisation de la reconfiguration dynamique de la technologie *FPGA Xilinx*. Ainsi, en perspective, une suite à ces travaux expérimentaux consiste à implanter simultanément une défaillance d'un module au sein d'un réseau en phase d'intégration d'un module dynamique.

Conclusion générale et perspectives

Avec l'évolution actuelle des systèmes complets intégrant des modules de nature différente sur une même puce (*SoC -System on a chip*), les technologies reconfigurables deviennent primordiales. En effet, les *SoC* ayant à l'origine une flexibilité matérielle limitée (structure de type *ASIC*) ont besoin à la fois de gérer leur fonctionnement de manière autonome et de s'adapter à des modifications de l'environnement dans lequel ils évoluent. De plus, les besoins en terme de puissance de calcul et de traitement ne cessent d'augmenter. Afin de faire face aux nouveaux besoins et aux nouvelles exigences, de nouveaux paradigmes et solutions architecturales basés sur des structures auto-adaptatives, auto-organisées sont à élaborer. C'est dans ce contexte que se situent ces travaux de recherche présentés dans ce mémoire de thèse.

Un système matériel auto-organisé peut être défini comme un système de traitement dans lequel le comportement global du système découle de la manière dont sont organisés les échanges et les interactions entre les modules bas niveau le constituant. Les règles d'interaction entre les modules constituant le système sont définies uniquement à partir d'informations localement véhiculées entre ces modules. Ces concepts généraux ne sont pas nouveaux (*calcul inspiré organique*, *systèmes de calcul autonome*, *systèmes multi-agents*, etc.) et ont déjà été mis en œuvre de manière logicielle.

A partir de l'analyse de caractéristiques d'auto-organisation dans le contexte des systèmes reconfigurables, nous avons établi une liste de besoins nécessaires pour la conception d'un système reconfigurable auto-organisé. Nous avons identifié et recensé les principaux aspects à développer pour aboutir à un système auto-organisé reconfigurable. Ainsi, l'aspect communication entre les modules constitutifs d'un système joue un rôle primordial dans la conception de tels systèmes. En effet, un moyen de communication robuste, scalable et adapté à la reconfigurabilité d'un système doit être utilisé. Le deuxième aspect à considérer dans la conception de systèmes auto-organisés reconfigurables est l'aspect approche architecturale mettant en œuvre les concepts liés aux propriétés d'une auto-organisation matérielle. Cet aspect comprend les mécanismes de distribution de contrôle d'un système, la coordination de ses entités et la manière dont les décisions au niveau système se prennent à travers les mécanismes locaux utilisés.

L'approche architecturale de calcul embarqué auto-organisée proposée dans ces travaux de thèse repose à la fois sur une structure de communication reconfigurable de type *NoC* (*Net-*

work on Chip) adaptée à la technologie FPGA et sur des structures originales de communications informationnelles contribuant à mettre en œuvre les principes d'auto-adaptation, d'auto-organisation. Concernant le concept informationnel système, la stratégie adoptée repose sur deux structures : une structure dynamique au sein du système que l'on nomme *flux informationnel* et une structure statique local (entité) nommée *descripteur*. On définit un *flux informationnel* comme une structure variable dont l'objectif principal est de recenser toutes les informations relatives aux fonctionnements des entités qui lui sont associées afin de les transmettre aux autres entités voisines. Le but d'informer l'ensemble des entités en vue de mettre en œuvre une émergence éventuelle par prise de décision individuelle des entités.

Au niveau de la structuration adaptative des communications, nous avons proposé deux architectures de *NoC* reconfigurables : *CuNoC* et *QNoC*. Nous avons détaillé pour chacune des approches leurs principaux avantages et inconvénients. La première approche de communication sur puce reconfigurable *CuNoC*, est caractérisé par un taux faible de ressources nécessaires pour son implantation, par une politique d'arbitrage au sein de ses routeurs reposant sur une règle de priorité à droite, par une connexion spécifique entre les routeurs du réseau et les modules de calcul et par un routage spécifique permettant l'acheminement des paquets et des messages dans les conditions de reconfigurabilité dynamique du réseau. La principale différence entre ces deux réseaux reconfigurable est qu'un réseau *CuNoC* montre des résultats médiocres en terme de débit dans des conditions de fonctionnement nécessitant des communications et des bandes passantes élevées. Cependant, son implantation matérielle sur technologie FPGA fournit un très bon compromis en terme de ressources matérielles nécessaires et sa complexité de mise en œuvre pour des applications embarquées à base d'une structure *NoC* reconfigurable. Dans le cas d'applications nécessitant de bonnes performances en terme de débit de transfert, nous avons introduit le réseau *QNoC* améliorant les faibles points du réseau *CuNoC* au détriment d'une consommation en terme de ressources matérielles plus élevées. En résumé, un réseau *QNoC* montre un gain considérable en terme de performances par rapport à un réseau *CuNoC*. Cependant, ces performances s'obtiennent au détriment des ressources utilisées comparé au réseau *CuNoC* dont les faibles ressources (nécessaires pour son implantation) sont considérées comme son principal atout. Néanmoins, il demeure adapté pour des applications embarquées à faible ressource comparé aux ressources nécessaires des *NoC* reconfigurables présentés dans la littérature.

Pour permettre la mise en œuvre de ces deux *NoC* adaptés à la conception de systèmes reconfigurables auto-organisés, nous avons développé et présenté un nouvel algorithme de routage adapté à ces deux structures *NoC*. L'algorithme de routage présenté est un algorithme de routage de type tolérant aux fautes et adaptatif, permettant à la fois le routage des paquets dans les réseaux ayant des nœuds défaillants et des *régions*. Nous avons présenté la manière dont l'algorithme achemine les paquets d'un nœud source vers un nœud destinataire et les règles de

base que nous avons introduites pour effectuer un routage. Nous avons également prouvé que l'algorithme présenté ne mène pas vers des situations de blocages dans les réseaux. Cet algorithme de routage est validé dans les simulations et comparé avec un algorithme de routage de même type.

L'ensemble des points conceptuels présentés dans ces travaux de thèse ont été validés non seulement au niveau de simulations mais également au niveau expérimental sur un exemple concret d'application de traitement temps réel d'images vidéo. En effet, nous avons présenté un exemple pratique mettant en œuvre la conception et l'implantation d'une architecture auto-organisée de détection de contours d'images reposant sur les concepts d'auto-organisation et de communications sur puce adaptés pour des systèmes sur puce reconfigurables. Nous avons réalisé une validation expérimentale du système global par son implantation réelle sur une plateforme de prototypage rapide *FPGA ML-505* de *Xilinx*. Sur un exemple de simulation d'une défaillance d'un des modules du système implanté, nous avons montré que ce dernier continue de fonctionner malgré la présence de cette perturbation. Nous avons explicité à travers cet exemple d'application, la stratégie de mise en œuvre de l'auto-organisation d'un système sur puce reconfigurable par application du concept de *flux informationnel*.

En conclusion, dans ces travaux de thèse nous avons proposer une nouvelle approche mettant en œuvre la notion d'auto-organisation et / ou d'émergence matérielle dans la conception architecturale des systèmes numériques de traitement. Nous avons montré qu'il faut envisager des solutions architecturales basées sur une structure auto-organisée (*Multiprocesseur sur puce - MPSOC* associé à un réseau sur puce reconfigurable - *RNoC*) offrant une adéquation entre une puissance de calcul suffisante et une grande flexibilité et adaptabilité aux évolutions des environnements de traitement. Nous avons montré également comment intégrer au sein de ces architectures des concepts architecturaux mettant en œuvre des propriétés tels que l'anticipabilité et un contrôle décentralisé. De plus, ces travaux démontrent l'intérêt de l'utilisation d'une structure de communication sur puce tels que les réseaux reconfigurables *CuNoC* et *QNoC*. Ces réseaux permettent la mise en pratique d'une auto-organisation matérielle par notre approche architecturale proposée dans le cas d'un placement dynamique d'un module de calcul au sein d'un réseau grâce à l'exploitation de la reconfiguration dynamique partielle des technologies FPGA disponibles (technologie *Virtex Xilinx*).

Les travaux menés durant cette thèse sont donc originaux en plusieurs points. D'abord, parce qu'il présente des travaux jusqu'alors peu explorés et correspondant à la conception d'une architecture sur puce transposant matériellement des aspects du principe d'auto-organisation des systèmes. Ensuite, parce que les travaux effectués prennent en considération les spécificités de la technologie reconfigurable de type *FPGA*.

En perspective, une suite de ces travaux théoriques et expérimentaux à mener consiste à développer l'implantation simultanée une défaillance d'un module au sein d'un réseau en phase

d'intégration d'un module dynamique par exploitation de la reconfiguration dynamique partielle des technologies *FPGA*. D'autres part, un autre aspect fondamental à prendre en compte lors de la conception d'un système reconfigurable auto-organisé et dont nous avons par ailleurs identifié dans la phase de conception de tel système, est l'aspect d'apprentissage permettant au système l'acquisition de connaissances de ses expériences et leur emploi pour ses réponses à la fois comportementales et auto-adaptative face à des environnements de traitement évolutifs.

Bibliographie

[ABD92] J.M. ARNOLD, D.A. BUELL et E.G. DAVIS : Splash 2. *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 316–322, 1992. 38

[ABT04] A. AHMADINIA, C. BOBDA et J. TEICH : A dynamic scheduling and placement algorithm for reconfigurable hardware. *Lecture notes in computer science*, pages 125–139, 2004. 39

[AEK06] B. AHMAD, AT ERDOGAN et S. KHAWAM : Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC. *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, pages 405–411, 2006. 91

[AG03] A. ANDRIAHANTENAINA et A. GREINER : Micro-network for SoC Implementation of a 32-port SPIN network. *Proceedings of the conference on Design, Automation and Test in Europe*, page 11128, 2003. 79, 85, 88, 90

[ALL05] D. AGRAWAL, K.W. LEE et J. LOBO : Policy-based Management of Networked Computing Systems. *IEEE Communications Magazine*, 43(10):69–75, 2005. 27

[Alt98] ALTERA : Databook. *San Jose, CA*, 1998. 36, 37

[Ama06] H. AMANO : A survey on dynamically reconfigurable processors. *IEICE Transactions on Communications*, 89(12):3179, 2006. 39

[And72] P.W. ANDERSON : More is different. *Science, New Series*, 177(4047):393–396, Aug 1972. Published by : American Association for the Advancement of Science. 15

[Ash47] William R. ASHBY : Principles of the self-organizing dynamic system. *Journal of General Psychology*, 37:125–128, 1947. 10, 14

[Ash62] W.R. ASHBY : Principles of the self-organizing system. *Principles of Self-Organization*, pages 255–278, 1962. 10, 11

[Atm98] ATMEL : Atmel 40k datasheet, 1998. 36, 37

[BA05] C. BOBDA et A. AHMADINIA : Dynamic interconnection of reconfigurable modules on reconfigurable devices. *Design & Test of Computers, IEEE*, 22, Issue 5:443 – 451, Sept.-Oct. 2005. 99, 107

[BAM+05] C. BOBDA, A. AHMADINIA, M. MAJER, J. TEICH, S. FEKETE et J. van der VEEN : Dynoc : A dynamic infrastructure for communication in dynamically reconfugurable devices. *In Field Programmable Logic and Applications, 2005. International Conference on*, pages 153 – 158, Aug. 2005. xiv, 94, 96, 106, 107

[BC95] R.V. BOPPANA et S. CHALASANI : Fault-tolerant wormhole routing algorithms for mesh networks. *Computers, IEEE Transactions on*, 44(7):848–864, 1995. 89, 149

[BDK+03] C. BOUTILIER, R. DAS, J.O. KEPHART, G. TESAURO et W.E. WALSH : Cooperative negotiation in autonomic systems using incremental utility elicitation. *Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 89–97, 2003. 26

[BDM01] L. BENINI et G. DE MICHELI : Powering networks on chips, 2001. 79

[BDM02] L. BENINI et G. DE MICHELI : Networks on chips : a new SoC paradigm. *Computer*, 35(1):70–78, 2002. 79

[Ber93] P. BERTIN : *Memoires actives programmables : conception, realisation et programmation*. Thèse de doctorat, Université de Paris 07, Paris, 1993. 38

[BGZ04] F. BERGENTI, M.P. GLEIZES et F. ZAMBONELLI : *Methodologies And Software Engineering For Agent Systems : The Agent-oriented Software Engineering Handbook*. Kluwer Academic Publishers, 2004. 16

[Bje05] T. BJERREGAARD : *The MANGO clockless network-on-chip : Concepts and implementation*. IMM, Informatik og Matematisk Modellering, Danmarks Tekniske Universitet, 2005. 87, 88

[BR96] S. BROWN et J. ROSE : FPGA and CPLD architectures : A tutorial. *IEEE Design & Test of Computers*, 13(2):42–57, 1996. 36

[BRM+99] J. BABB, M. RINARD, CA MORITZ, W. LEE, M. FRANK, R. BARUA et S. AMARASINGHE : Parallelizing applications into silicon. *Field-Programmable Custom Computing Machines, 1999. FCCM'99. Proceedings. Seventh Annual IEEE Symposium on*, pages 70–80, 1999. 2, 33

[Bru04] Philippe BRUNET : *Exploration multicritères d'architectures à reconfiguration dynamique*. Thèse de doctorat, Université Henri Poincaré - Nancy 1, Dec. 2004. 29, 30, 41

[CC98a] K.H. CHEN et G.M. CHIU : Fault-tolerant routing algorithm for meshes without using virtual channels. *Journal of Information Science and Engineering*, 14(4): 765–783, 1998. 157

[CC98b] Kuo-Hsuan CHEN et Ge-Ming CHIU : Fault-tolerant routing algorithm for meshes without using virtual channels. *J. Inf. Sci. Eng.*, 14(4):765–783, 1998. 89, 149

[CH02] K. COMPTON et S. HAUCK : Reconfigurable computing : a survey of systems and software. *ACM Computing Surveys (CSUR)*, 34(2):171–210, 2002. 2, 33, 36

[Cru94] J.P. CRUTCHFIELD : The calculi of emergence : computation, dynamics and induction. *Proceedings of the NATO advanced research workshop and EGS topical workshop on Chaotic advection, tracer dynamics and turbulent dispersion table of contents*, pages 11–54, 1994. 22

[CSV05] D. CHING, P. SCHAUMONT et I. VERBAUWHEDE : Integrated modelling and generation of a reconfigurable network-on-chip. *International Journal of Embedded Systems*, 1(3):218–227, 2005. 91

[CZF+07] Y-A. CHAPUIS, Lingfei ZHOU, Y. FUKUTA, Y. MITA et H. FUJITA : FPGA-Based Decentralized Control of Arrayed MEMS for Microrobotic Application. *Industrial Electronics, IEEE Transactions on*, 54:1926 – 1936, Aug. 2007. 58

[Dal04] W.J. DALLY : *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004. 89

[DBG+03] M. DALL'OSSO, G. BICCARI, L. GIOVANNINI, D. BERTOZZI et L. BENINI : Xpipes : a latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs. *Computer Design, 2003. Proceedings. 21st International Conference on*, pages 536–539, 2003. 85

[DC96] Y. DEMAZEAU et A.C.R. COSTA : Populations and organizations in open multi-agent systems. *1st National Symposium on Parallel and Distributed Artificial Intelligence*, 1996. 28

[Dem98] M. Beth L. DEMPSTER : A self-organizing systems perspective on planning for sustainability. Mémoire de D.E.A., University of Waterloo, 1998. 12

[DG06] F. DITTMANN et M. GOTZ : *Reconfiguration Time Aware Processing on FPGAs*. Internat. Begegnungs-und Forschungszentrum für Informatik, 2006. 39

[DPW99] D. DEMIGNY, M. PAINDAVOINE et S. WEBER : Architecture à reconfiguration dynamique pour le traitement temps réel des images. *Technique et Science de l'information Numéro Spécial Architectures Reconfigurables*, 18(10):1087–1112, 1999. 39

[DS87] WJ DALLY et CL SEITZ : Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on computers*, 100(36):547–553, 1987. 156

[DT01] W.J. DALLY et B. TOWLES : Route packets, not wires : on-chip interconnection networks. *Design Automation Conference, 2001. Proceedings*, pages 684–689, 2001. 79, 84

[DYN02] J. DUATO, S. YALAMANCHILI et L. NI : *Interconnection Networks*. Morgan Kaufmann, 2002. 84, 85, 88, 99, 130

[Edm95] Bruce EDMONDS : What is complexity ? - the philosophy of complexity *per se* with application to some examples in evolution. *The Evolution of Complexity*, 1995. 14

[EV62] G. ESTRIN et CR VISWANATHAN : Organization of a Fixed-Plus-Variable Structure Computer for Computation of Eigenvalues and Eigenvectors of Real Symmetric Matrices. *Journal of the ACM (JACM)*, 9(1):41–60, 1962. 2, 33

[Fer95] J. FERBER : Les systèmes multi-agents. *Vers une intelligence collective. InterEditions*, page 176, 1995. 28, 29

[FG97] S. FRANKLIN et A. GRAESSER : Is it an Agent, or just a Program ? : A Taxonomy for Autonomous Agents. *Lecture Notes in Computer Science*, 1193:21–36, 1997. 28

[FS68] G. FELDMAN et I. SOBEL : A 3×3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project*, 1968. 161

[Gas92] L. GASSER : An overview of DAI. *Distributed Artificial Intelligence : Theory and Praxis*, 9:9–29, 1992. 28

[GBTW97] H. GUERMOUD, Y. BERVILLER, E. TISSERAND et S. WEBER : Architecture à base de FPGA reconfigurable dynamiquement dédiée au traitement d'image sur flot de données. *GRETSI, Groupe d'Etudes du Traitement du Signal et des Images*, 1997. 39

[GDR05] K. GOOSSENS, J. DIELISSEN et A. RADULESCU : Aethereal network on chip : concepts, architectures, and implementations. Design & Test of Computers. *IEEE*, 22(5):414–421, 2005. 85, 87, 88, 91

[Ger02] Carlos GERSHENSON : Complex philosophy. *In In Proceedings of the 1st Biennial Seminar on Philosophical, Methodological & Epistemological Implications of Complexity Theory*, La Habana, Cuba, 2002. 15

[Ger05] Carlos GERSHENSON : A general methodology for designing self-organizing systems. *CoRR*, 2005. 12, 15, 16

[GG00] P. GUERRIER et A. GREINER : A generic architecture for on-chip packet-switched interconnections. *In Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, pages 250 – 256, March 2000. 79, 85, 90

[GH00] R. GHANEA-HERCOCK : Spontaneous group formation in multi-agent systems. *In Proceedings of Workshop on Self-Organisation in Multi-Agent Systems,* 2000. 18

[GHK+90] M. GOKHALE, B. HOLMES, A. KOPSER, D. KUNZE, S. LUCAS et R. MINNICH : Splash : A Reconfigurable Linear Logic Array. *International Conferance on Parallel Processing*, pages 526–532, 1990. 38

[GN93] GLASS et NI : Fault-tolerant wormhole routing in meshes. *In FTCS : Fault-Tolerant Computing : 23th Annual International Symposium*. IEEE Computer Society, 1993. 89, 148

[Gol99] J. GOLDSTEIN : Emergence as a Construct : History and Issues. *Emergence*, 1(1):49–72, 1999. 21, 22

[GRD] Le grand dictionnaire de l'Office Québécois de la langue française. `http://www.granddictionnaire.com/`. 9, 28, 33, 35, 81, 90, 91, 92, 94, 177

[HAC] Dictionnaire Hachette. `http://www.ehmelhm.hachette-multimedia.fr/`. 8, 19

[Hak84] H. HAKEN : *The Science of Structure : Synergetics*. Van Nostrand Reinhold Company, 1984. 21

[Hak00] H. HAKEN : *Information and Self-Organization : A Macroscopic Approach to Complex Systems*. Springer, 2000. 18

[HB06] M. HÜBNER et J. BECKER : Exploiting dynamic and partial reconfiguration for FPGAs : toolflow, architecture and system integration. *Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pages 1–4, 2006. 38

[HDM02] J. HU, S.Y. DENG et R. MARCULESCU : System-Level Point-to-Point Communication Synthesis Using Floorplanning Information. *ASP-DAC/VLSI*, 2002. 78

[Hey89] F. HEYLIGHEN : Self-organization, emergence and the architecture of complexity. *In Proceedings of the 1st European Conference on System Science,* pages 23–32, 1989. 18, 21, 24

[Hey03] Francis HEYLIGHEN : The science of self-organization and adaptivity. *In in : Knowledge Management, Organizational Intelligence and Learning, and Complexity, in : The Encyclopedia of Life Support Systems, EOLSS*, pages 253–280. Publishers Co. Ltd, 2003. 15, 16, 21, 22

[HG03] Francis HEYLIGHEN et Carlos GERSHENSON : The meaning of self-organization in computing. *IEEE Intelligent Systems, section Trends & Controversies - Self-organization and Information Systems*, pages pp. 72–75, 2003. 13

[HG07] A. HANSSON et K. GOOSSENS : Trade-offs in the configuration of a network on chip for multiple use-cases. *Proc. International Symposium on Networks on Chip (NOCS)*, 2007. 90

[HJ01] Francis HEYLIGHEN et Cliff JOSLYN : Cybernetics and second-order cybernetics. *Encyclopedia of Physical Science & Technology (3rd ed)*, 2001. 15

[HM04] J. HU et R. MARCULESCU : DyAD : smart routing for networks-on-chip, 2004. 89

[HMG05] K. HERRMANN, G. MUHL et K. GEIHS : Self management : the solution to complexity or just another problem ? *IEEE Distributed Systems Online*, 6(1), 2005. 27

[Hol98] JH HOLLAND : Emergence : From Chaos to Order. *AddisonWesley,Redwood City, CA*, 1998. 21, 22

[HSKB06] M. HUBNER, C. SCHUCK, M. KUHNLE et J. BECKER : New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits. *ISVLSI2006, Karlsruhe, Germany*, 2006. 39

[Ini94] O.M. INITIATIVE : PI-Bus Draft Standard Specification, 1994. 78

[JS00] D. JAGGER et D. SEAL : *ARM Architecture Reference Manual*. Pearson Education, 2000. 78

[JTBW09] S. JOVANOVIĆ, C. TANOUGAST, C. BOBDA et S. WEBER : CuNoC : A dynamic scalable communication structure for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems*, 33(1):24–36, 2009. 96, 97, 101, 103, 107, 109

[JTW08a] S. JOVANOVIC, C. TANOUGAST et S. WEBER : A new high-performance scalable dynamic interconnection for FPGA-based reconfigurable systems. *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. IEEE Conference on*, pages 61–66, 2008. 125, 126, 127, 128

[JTW08b] S. JOVANOVIC, C. TANOUGAST et S. WEBER : A New Self-managing Hardware Design Approach for FPGA-Based Reconfigurable Systems. *Lecture Notes in Computer Science*, 4943:160, 2008. 61, 67

[JTWB07a] S. JOVANOVIC, C. TANOUGAST, S. WEBER et C. BOBDA : CuNoC : A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs. *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 753–756, Aug. 2007. 96, 103, 174

[JTWB07b] S. JOVANOVIC, C. TANOUGAST, S. WEBER et C. BOBDA : A scalable dynamic infrastructure for dynamically reconfigurable systems. *ReCoSoC07, 2007.*, Jun. 2007. 96, 97, 103, 106, 107, 109, 174

[JTWB09] S. JOVANOVIC, C. TANOUGAST, S. WEBER et C. BOBDA : A new deadlock-free fault-tolerant routing algorithm for nocs. *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 326–331, 2009. 149

[KHHC07] A. KUMAR, A. HANSSON, J. HUISKEN et H. CORPORAAL : An FPGA design flow for reconfigurable network-based multi-processor systems on chip. *Proc. DATE*, 2007. 91

[KJCD03] O. KEPHART JEFFREY et M. CHESS DAVID : The vision of autonomic computing. *Computer*, 36(1):41–50, 2003. 25

[KJS+02] S. KUMAR, A. JANTSCH, J.-P. SOININEN, M. FORSELL, M. MILLBERG, J. OBERG, K. TIENSYRJA et A. HEMANI : A network on chip architecture and design methodology. *In VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, pages 105 – 112, April 2002. 87, 90

[KN06] H. KARINIEMI et J. NURMI : Arbitration and Routing Schemes for On-chip Packet Networks. *Interconnect-Centric Design for Advanced SoC and NoC*, 2006. 88

[KND02] F. KARIM, A. NGUYEN et S. DEY : An interconnect architecture for networking systems on chips. *Micro, IEEE*, 22(22379):36 – 45, Sept.-Oct. 2002. 79, 84, 90

[KNDR01] F. KARIM, A. NGUYEN, S. DEY et R. RAO : On-chip communication architecture for OC-768 network processors. *Proceedings of the 38th conference on Design automation*, pages 678–683, 2001. 84

[KPT+05] J. KIM, D. PARK, T. THEOCHARIDES, N. VIJAYKRISHNAN et C.R. DAS : A low latency router supporting adaptivity for on-chip interconnects, 2005. 89

[Lan90] C.G. LANGTON : Computation at the edge of chaos. *Physica D*, 42(1):12–37, 1990. 18

[Len64] G.G. LENDARIS : On the definition of self-organizing systems. *Proceedings of the IEEE*, 52(3):324–325, March 1964. 11

[LEO] Leonardo Spectrum Synthesis Tool Solution for ASICs and FPGAs, Mentor Graphics. `http://www.mentor.com/products/fpga/synthesis/leonardo`. 115

[Lew75] G.H. LEWES : Problems of Life and Mind, Vol. 2. *London : Trubner*, 1875. 21

[Liu08] Ting LIU : *Optimisation par synthèse architecturale des méthodes de partitionnement temporel pour les circuits reconfigurables*. Thèse de doctorat, Université Henri Poincaré - Nancy 1, 2008. 30, 37

[LPD03] O. LYSNE, T.M. PINKSTON et J. DUATO : A methodology for developing dynamic network reconfiguration processes. pages 77–86, 2003. 90

[LR06] P. LEITÃO et F. RESTIVO : ADACOR : a holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, 57(2):121–130, 2006. 58

[LSC96] W. LUK, N. SHIRAZI et PYK CHEUNG : Modeling and optimizing runtime reconfiguration systems. *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 167–176, 1996. 162

[Luc97] C. LUCAS : Transient Attractors and Emergent Attractor Memory. *CALResCo Group. URL : http ://www. calresco. org/transatr. htm*, 1997. 17

[MBAT05] M. MAJER, C. BOBDA, A. AHMADINIA et J. TEICH : Packet Routing in Dynamically Changing Networks on Chip. *Proc. Parallel and Distributed Processing Symp*, 2005. 94

[MBV+02] T. MARESCAUX, A. BARTIC, D. VERKEST, S. VERNALDE et R. LAUWEREINS : Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 795–805, 2002. 79, 84, 90

[Mes62] Mijailo MESAROVIC : *On self-organization systems*, chapitre in Self-Organizing Systems. Spartan Books, Washington 12, D.C, 1962. 11

[MG00] S. MCMILLAN et S.A. GUCCIONE : Partial Run-Time Reconfiguration Using JRTR. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 352–360, 2000. 115

[Mic00] ST MICROELECTRONICS : STBus Interconnect, 2000. 78

[MRF+03] S. MOSTEFAOUI, OF RANA, N. FOUKIA, S. HASSAS, G. DI MARZO SERUGENDO, C. VAN AART et A. KARAGEORGOS : Self-Organising Applications : A Survey. 2003. 18

[MS04] C. MÜLLER-SCHLOER : Organic computing : on the feasibility of controlled emergence. *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 2–5, 2004. 27, 28

[MSI] ModelSim - a comprehensive simulation and debug environment for complex ASIC and FPGA designs. `http://www.model.com/`. 118

[MVJS07] F. MARTINEZ VALLINA, N. JACHIMIEC et J. SANIIE : Nova interconnect for dynamically reconfigurable noc systems. *In Electro/Information Technology, 2007 IEEE International Conference on*, pages 546–550, May 2007. 92, 93

[MZ04] M. MAMEI et F. ZAMBONELLI : Self-Organization in Multi Agent Systems : A Middleware Approach. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 233–248, 2004. 24

[NB07] Mohammad Reza NAMI et Koen BERTELS : A survey of autonomic computing systems. *In ICAS '07 : Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 26, Washington, DC, USA, 2007. IEEE Computer Society. 25

[Neu66] John Von NEUMANN : *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966. 12

[New96] D.V. NEWMAN : Emergence and Strange Attractors. *Philosophy of Science*, 63(2): 245, 1996. 21

[Nic89] G. NICOLIS : Physics of far-from-equilibrium systems and self-organisation. *The New Physics*, pages 316–347, 1989. 21

[NM93] L.M. NI et P.K. MCKINLEY : A survey of wormhole routing techniques in direct networks. *Computer*, 26(4947):62 – 76, Feb. 1993. 127

[NOR03] D.A. NORMAN, A. ORTONY et D.M. RUSSELL : Affect and machine design : Lessons for the development of autonomous machines. *IBM Systems Journal*, 42(1):38–44, 2003. 26

[NP77] G. NICOLIS et Ilya PRIGOGINE : *Self-Organization in nonequilibrium systems*. John Wiley & Sons, New York, 1977. 15

[Ode02a] J. ODELL : Agents and Complex Systems. *Technology*, 1(2):35–45, 2002. 21, 22, 23

[Ode02b] J. ODELL : Objects and Agents Compared. *Technology*, 1(1):41–53, 2002. 21, 22, 23

[ORG] The organic computing page. `http://www.organic-computing.org/`. 27

[Par96] H.V.D. PARUNAK : Applications of distributed artificial intelligence in industry. *Foundations of distributed artificial intelligence*, 4:139–164, 1996. 58

[Par01] H.V.D. PARUNAK : Entropy and self-organization in multi-agent systems. *Proceedings of the fifth international conference on Autonomous agents*, pages 124–130, 2001. 21, 24

[PB04] H. PARUNAK et S. BRUECKNER : Engineering swarming systems, Methodologies and Software Engineering for Agent Systems,, 2004. 24

[PD08] S. PASRICHA et N. DUTT : *On-chip communication architectures : system on chip interconnect*. Morgan Kaufmann, 2008. 78

[PGIS03] P.P. PANDE, C. GRECU, A. IVANOV et R. SALEH : Design of a switch for network on chip applications. *In Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 5, pages V–217 – V–220 vol.5, 25-28 May 2003. 79, 85, 90

[PHF07] G. PEZZULO, J. HOFFMANN et R. FALCONE : Anticipation and anticipatory behavior. *Cognitive Processing*, 8(2):67–70, 2007. 16

[PKA06] T. PIONTECK, R. KOCH et C. ALBRECHT : Applying partial reconfiguration to networks-on-chips. *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pages 1–6, 2006. 93, 95

[PRE] Precision RTL Synthesis Tool for ASICs and FPGAs, Mentor Graphics. `http://www.mentor.com/products/fpga/synthesis/`. 132

[Pri68] I. PRIGOGINE : Dissipative processes, quantum states and field theory. *XIV Conseil de Physique Solvay, Fundamental Problems in Elementary Particle Physics, Interscience Publishers*, 1968. 12

[PST+05] G. PACIFICI, M. SPREITZER, AN TANTAWI, A. YOUSSEF, I.B.M.T.J.W.R. CENTER et NY YORKTOWN HEIGHTS : Performance management for cluster-based web services. *IEEE Journal on Selected Areas in Communications*, 23(12):2333–2343, 2005. 26

[Rab00] J. RABAEY : *«Busses and networking»*, volume 252. 2000. 78

[RAS+08] V. RANA, D. ATIENZA, M.D. SANTAMBROGIO, D. SCIUTO et G. DE MICHELI : A Reconfigurable Network-on-Chip Architecture for Optimal Multi-Processor SoC Communication. *16th IFIP/IEEE International Conference on Very Large Scale Integration*, October 13-15 2008. xiv, 93, 94, 95

[RL97] S. RUBINI et D. LAVENIER : Les architectures reconfigurables. *Calculateurs Parallèles*, 9(1):9–27, 1997. 36

[Ros85] R. ROSEN : *Anticipatory Systems : Philosophical, Mathematical, and Methodological Foundations*. Pergamon, 1985. 16

[RPB+] F. ROCHNER, H. PROTHMANN, J. BRANKE, C. MULLER-SCHLOER et H. SCHMECK : An Organic Architecture for Traffic Light Controllers. *Informatik 2006-Informatik fur Menschen*, 93. 27

[SBB+06] P. SEDCOLE, B. BLODGET, T. BECKER, J. ANDERSON et P. LYSAGHT : Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proceedings-Computers and Digital Techniques*, 153(3):157–164, 2006. 38

[Sch05] H. SCHMECK : Organic Computing–A New Vision for Distributed Embedded Systems. *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005). IEEE Computer Society*, pages 201–203, 2005. 27, 28

[Ser04] G.D.M. SERUGENDO : *Engineering Self-Organising Systems : Nature-Inspired Approaches to Software Engineering*. Springer, 2004. 17

[SF89] N. SADEH et M.S. FOX : CORTES : An exploration into micro-opportunistic job-shop scheduling. *Proceedings of Workshop on Manufacturing Production Scheduling*, 1989. 58

[SH09] Silicon Hives. Available from `http://www.siliconhives.com/`, 2009. 91

[Sha01] C.R. SHALIZI : Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata. *Unpublished doctoral dissertation, University of Wisconsin-Madison*, 2001. 17

[Sim06] Herbert A. SIMON : *Les sciences de l'artificiel*. DUNOD, 2006. 14

[Son02] I. SONICS : The Silicon Backplane, 2002. 78

[SS08] M.B. STENSGAARD et J. SPARSO : Renoc : A network-on-chip architecture with reconfigurable topology. *In Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 55–64, April 2008. 91, 95

[STG+01] Gilles SASSATELLI, Lionel TORRES, Jérôme GALY, Gaston CAMBON et Camille DIOU : The systolic ring : A dynamically reconfigurable architecture for embedded systems. *In FPL '01 : Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 409–419, London, UK, 2001. Springer-Verlag. 36

[SW01] D. DEMIGNY S. WEBER, Y. BERVILLER : *"Conceptions pour architectures à reconfiguration dynamique"*, chapitre Méthodes et architecture pour le TSI en temps réel, Ouvrage collectif, chapitre 13. HERMES, 2001. 2

[SZBR07] Timo SCHONWALD, Jochen ZIMMERMANN, Oliver BRINGMANN et Wolfgang ROSENSTIEL : Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures. *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 527–534, 29-31 Aug. 2007. 149

[Tan01] Camel TANOUGAST : *Méthodologie de partitionnement applicable aux systèmes sur puce à base de FPGA, pour l'implantation en reconfiguration dynamique d'algorithmes flot de données*. Thèse de doctorat, Université Henri Poincaré - Nancy 1, Oct. 2001. 2, 8, 30, 33, 38

[Tur52] A. M. TURING : The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society (B)*, 237:37–72, 1952. 12

[VBR+96] JE VUILLEMIN, P. BERTIN, D. RONCIN, M. SHAND, HH TOUATI et P. BOUCARD : Programmable active memories : Reconfigurable systems come of age. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(1):56–69, 1996. 38

[VdV97] G. Van de VIJVER : Emergence et explication. *Intellectica*, 25:7–23, 1997. 22

[WC96] RD WITTIG et P. CHOW : OneChip : An FPGA processor with reconfigurable logic. *IEEE Symposium on FPGAs for Custom Computing Machines, 1996. Proceedings*, pages 126–135, 1996. 36

[WH05] Tom De WOLF et Tom HOLVOET : Emergence versus self-organisation : Different concepts but promising when combined. *In Lecture Notes in Artificial Intelligence (LNAI)*, pages 1–15. Springer-Verlag, 2005. 13, 17, 18, 23, 24

[WHW+04] SR WHITE, JE HANSON, I. WHALLEY, DM CHESS, JO KEPHART, T.J.W.R. CENTER et W.P. IBM : An architectural approach to autonomic computing. *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 2–9, 2004. 27

[Wie48] Norbert WIENER : *Cybernetics - Control and Communication in the Animal and the Machine*. Hermann & Cie (Paris), 1948. 12

[WIK] Encyclopédie Wikipedia. `http://www.wikipedia.com/`. 9, 19

[Woo99] M. WOOLDRIDGE : *Intelligent agents*. MIT Press Cambridge, MA, USA, 1999. 28

[WSRS05] PT WOLKOTTE, GJM SMIT, GK RAUWERDA et LT SMIT : An energy-efficient reconfigurable circuit-switched network-on-chip. *19th IEEE International Parallel and Distributed Processing Symposium, 2005. Proceedings*, pages 155a–155a, 2005. 87

[Wu03] Jie WU : A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model. *Computers, IEEE Transactions on*, 52(9):1154–1169, Sept. 2003. 89, 148

[Xil94] I. XILINX : The Programmable Logic Data Book 1994, 1994. xiii, 37

[Xil00] I. XILINX : The Programmable Logic Data Book, 2000. 36, 37

[Xil07] I. XILINX : ML505/ML506 Evaluation Platform. UG347 (v2. 4) edn, 2007. 176

[Xil08] I. XILINX : Virtex-5 Family Overview, May 2008. 176

Liste de publications

Revue internationale

[R1] S. JOVANOVIĆ, C. TANOUGAST, C. BOBDA and S. WEBER, « CuNoC : A Dynamic Scalable Structure for Dynamically Reconfigurable FPGAs » , *Microprocessors and Microsystems*, Elsevier, Volume 33, Issue 1, February 2009, Pages 24-36.

Comunications Internationales

[CI 6] S. JOVANOVIĆ, C. TANOUGAST, S.WEBER and C. BOBDA, « A New Deadlock-free Fault-tolerant Routing Algorithm for NoC Interconnections », *The 19th International Conference on Field Programmable Logic and Applications,* IEEE Circuits and Systems Society, Prague, Aug 31 - Sep 2, 2009, A paraître.

[CI 5] S. JOVANOVIĆ, C.TANOUGAST and S.WEBER, « A New High-Performance Scalable Dynamic Interconnection for FPGA-based Reconfigurable Systems », *19th IEEE International Conference Application-specific Systems, Architectures and Processors (ASAP)*, July 2-4, Leuven, Belgium 2008.

[CI 4] S. JOVANOVIĆ, C.TANOUGAST, S. WEBER, « A New Self-Managing Hardware Design Approach for FPGA-based Reconfigurable Systems », *Reconfigurable Computing : Architecture, Tools, and Applications, 4th International Workshop, ARC 2008 Proceedings, Lecture Notes in Computer Science Vol. 4943*, march 2008.

[CI 3] S. JOVANOVIĆ, C. TANOUGAST, C. BOBDA and S. WEBER, « CuNoC : A Scalable Dynamic Infrastracture NoC for Simulataneous Communication in Dynamically Reconfigurable Devices », *17th International Conference on Field Programmable Logic and Applications (FPL 2007)*, IEEE Circuits and Systems Society, August 27-29, Amsterdam, NETHERLANDS, 2007, pp 753-756.

[CI 2] S. JOVANOVIĆ, C. TANOUGAST and S. WEBER, « A Hardware Preemptive Multitasking Mechanism Based on Scan-path Register Structure for FPGA-based Reconfigurable Systems », *IEEE NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, IEEE Computer Society, August 5-8, Edimburg, UK, 2007, pp 358-364.

[CI 1] S. JOVANOVIĆ, C. TANOUGAST, C. BOBDA and S. WEBER, « A Dynamic Communication Structure for Dynamically Reconfigurable FPGAs », *3rd International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC07)*, June 18-20, Montpellier, France, 2007, pp 98-105.

Communications nationales

[CN 3] S. JOVANOVIĆ, C. TANOUGAST and S. WEBER, « A Hardware Design Approach for Self-organizing SoC based on Reconfigurable Technology », *Le troisième Colloque Nationale du GDR System-On-Chip & System-In-Package (SOC-SIP) – CNRS*, 10-12 juin 2009, Paris Orsay, 2009.
[CN 2] S. JOVANOVIĆ, C. TANOUGAST and S. WEBER, « Self-organizing System-on-Chip based on Reconfigurable Technology », *Le deuxième Colloque Nationale du GDR System-On-Chip & System-In-Package (SOC-SIP) – CNRS*, 04-06 juin, TELECOM ParisTech, 4-6 juin, Paris, 2008.

[CN 1] S. JOVANOVIĆ, C. TANOUGAST et S. WEBER, « Self-organizing system achitectures based on FPGA reconfigurable technology », *Premier Colloque Nationale du GDR System-On-Chip & System-In-Package (SOC-SIP) – CNRS*, 13-15 juin, Jussieu, Paris, 2007.

Communication pédagogique

[PE 1] C. TANOUGAST, **S. JOVANOVIĆ**, F. MONTEIRO, C. DIOU et A. DANDACHE, « Initiation à la modélisation et co-simulation comportementale C-VHDL d'un Réseau de communication sur Puce (Network on Chip) », *10ème Journées Pédagogiques Coordination Nationale pour la Formation en Microélectronique (JPCNFM 2008)*, Saint Malo, 26-28 novembre 2008, ISBN 2-9522395-2-5, p.27-32.

Résumé

Afin de répondre à une complexité croissante des systèmes de calcul, due notamment aux progrès rapides et permanents des technologies de l'information, de nouveaux paradigmes et solutions architecturales basées sur des structures auto-adaptatives, auto-organisées sont à élaborer. Ces dernières doivent permettre d'une part la mise à disposition d'une puissance de calcul suffisante répondant à des contraintes de temps sévères (traitement temps réel). D'autre part, de disposer d'une grande flexibilité et adaptabilité dans le but de répondre aux évolutions des traitements ou des défaillances non prévues caractérisant un contexte d'environnement évolutif de fonctionnement du système. C'est dans ce cadre que s'insèrent les travaux de recherche présentés dans cette thèse qui consistent à développer une architecture auto-organisée de type *Reconfigurable MPSoC (Multi processor System on Chip)* à base de technologie FPGA.

Mots-clés: *Auto-adaptation et Auto-organisation matérielles, Calcul embarqué, Systèmes reconfigurables, Réseaux sur puce reconfigurables (RNoC), Algorithme de routage tolérant aux fautes, FPGA.*

Abstract

The growing complexity of computing systems, mostly due to the rapid progress in Information Technology (IT) in the last decade, imposes on system designers to orient their traditional design concepts towards the new ones based on self-organizing and self-adaptive architectural solutions. On the one hand, these new architectural solutions should provide a system with a sufficient computing power, and on the other hand, a great flexibility and adaptivity in order to cope with all non-deterministic changes and events that may occur in the environnement in which it evolves. Within this framework, a reconfigurable MPSoC self-organizing architecture on the FPGA reconfigurable technology is studied and developped during this PhD.

Keywords: *Self-organization and self-managing in hardware, Embedded computing, Reconfigurable systems, Reconfigurable Network-on-chips, Fault-tolerant routing algotithm, FPGA*