Conception architecturale pour la tolérance aux fautes d'un système auto-organisé multi-noeuds en réseau à base de NoC reconfigurables

Mikael Heil

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Conception architecturale pour la tolérance aux fautes d'un système auto-organisé multi-nœuds en réseau à base de NoC reconfigurables

THÈSE

présentée et soutenue publiquement le 4 décembre 2015

pour l'obtention du

Doctorat de l'Université de Lorraine

(Discipline: Systèmes électroniques)

par

Mikael HEIL

Composition du jury

Rapporteurs :	M. El-Bay BOURENNANE, Professeur, LE2I, Université de Bourgogne, Dijon M. Jean-Yves FOURNIOLS, Professeur, LAAS-CNRS, Toulouse
Examinateurs :	M. Bertrand GRANADO, Professeur, UPMC-LIP6, Paris M. Camel TANOUGAST, Professeur, LCOMS, Université de Lorraine, Metz M. Camille DIOU, Maître de Conférences, LCOMS, Université de Lorraine, Metz M. Loic SIELER, Maître de Conférences, LCOMS, Université de Lorraine, Metz

Laboratoire de Conception, Optimisation et Modélisation des Systèmes

Mis en page avec la classe thesul.

Remerciements

Je tiens avant tout propos à remercier la **Région Lorraine d'avoir co-financé ces travaux de thèse dans le cadre du projet Région Lorraine CAPTEX** avec le laboratoire LCOMS.

Je voudrais remercier sincèrement le Professeur Camel TANOUGAST de m'avoir si bien encadré durant ces trois années de thèse de part son engagement, sa disponibilité et ces précieux conseils. Il a su me soutenir, m'encourager et m'épauler afin d'aller jusqu'au bout de cette aventure qu'est le doctorat.

Je prie Monsieur El-Bay BOURENNANE, Professeur au LE2I de l'Université de Bourgogne, ainsi que Monsieur Jean-Yves FOURNIOLS, Professeur au LAAS-CNRS, d'accepter mes remerciements pour l'honneur qu'ils m'ont fait d'avoir été rapporteurs de ma thèse.

J'adresse mes remerciements à Monsieur Bertrand GRANADO, Professeur au LIP6-UPMC, pour avoir accepté d'examiner mes travaux de recherche et de participer à ma soutenance en tant que président du jury.

Je remercie également le Docteur Camille DIOU et le Docteur Loïc SIELER, dans un premier temps pour avoir accepté de participer à mon jury de thèse en tant qu'examinateurs mais également d'avoir pris sur leur temps pour m'apporter leurs conseils et leurs remarques pertinentes, principalement lors des nombreuses répétitions qui ont précédées la soutenance.

Je souhaite remercier le Docteur Cédric KILIAN de m'avoir accompagné dans la mise en œuvre de ces travaux qui poursuivent directement ses propres recherches de doctorat.

J'aimerais aussi remercier Christophe KIZIL et le Docteur Lucas CICERO, tant pour leur participation scientifique que pour les discussions partagées autour d'un café. Je souhaite à Christophe et autres doctorants du laboratoire bon courage pour la fin de leur doctorat.

Je tiens à remercier l'intégralité de l'équipe ASEC du laboratoire LCOMS ainsi que son directeur le Professeur Imed KACEM pour leurs contributions, de près ou de loin, à l'aboutissement de ces travaux.

Je terminerais par remercier du fond du cœur ma compagne Anaïs, mes parents Yves et Patricia, ainsi que mon frère Maxime et ma sœur Alicia, de m'avoir soutenue et supporté (dans tous les sens du terme) durant toutes ces années, que ce soit dans les bons ou les mauvais moments.

Table des matières

Liste des tableaux

Table des figures

Résumé

Afin de répondre à des besoins croissants de performance et de fiabilité des systèmes sur puce embarqués pour satisfaire aux applications de plus en plus complexes, de nouveaux paradigmes architecturaux et structures de communication auto-adaptatives et auto-organisées sont à élaborer. Ces nouveaux systèmes de calcul intègrent au sein d'une même puce électronique plusieurs centaines d'éléments de calcul (systèmes sur puce multiprocesseur - Multi-Processor System on Chip (MPSoC)) et doivent permettre la mise à disposition d'une puissance de calcul parallèle suffisante tout en bénéficiant d'une grande flexibilité et d'une grande adaptabilité. Le but est de répondre aux évolutions des traitements distribués caractérisant le contexte évolutif du fonctionnement des systèmes. Actuellement, les performances de tels systèmes reposent sur une autonomie et une intelligence permettant de déployer et de redéployer les modules de calcul en temps réel en fonction de la demande de traitement et de la puissance de calcul. Elle dépend également des supports de communication entre les blocs de calcul afin de fournir une bande passante et une adaptabilité élevée pour une efficacité du parallélisme potentiel de la puissance de calcul disponible des MPSoC. De plus, l'apparition de la technologie Field-Programmable Gate Array (FPGA) reconfigurable dynamiquement a ouvert de nouvelles approches permettant aux MPSoC d'adapter leurs constituants en cours de fonctionnement, et de répondre aux besoins croissants d'adaptabilité et de flexibilité. C'est dans ce contexte du besoin primordial de flexibilité, de puissance de calcul et de bande passante qu'est apparue une nouvelle approche de conception des systèmes communicants, auto-organisés et auto-adaptatifs basés sur des nœuds de calcul reconfigurables. Ces derniers sont constitués de réseaux embarqués sur puce (Network on Chip (NoC)) permettant l'interconnexion optimisée d'un grand nombre d'éléments de calcul au sein d'une même puce, tout en assurant l'exigence d'une tolérance aux fautes et d'un compromis entre les performances de communication et les ressources d'interconnexion.

Ces travaux de thèse ont pour objectif d'apporter des solutions architecturales innovantes pour la Sûreté de Fonctionnement (SdF) des systèmes MPSoC en réseau basés sur la technologie FPGA, et configurés selon une structure distribuée et auto-organisée. L'objectif est d'obtenir des systèmes sur puce performants et fiables intégrant des techniques de détection, de localisation et de correction d'erreurs au sein de leurs structures NoC reconfigurables ou adaptatifs. La principale difficulté réside dans l'identification et la distinction entre des erreurs réelles et des fonctionnements variables ou adaptatifs des éléments constituants ces nœuds en réseau. Ces travaux ont permis de réaliser un réseau de nœuds reconfigurables à base de FPGA intégrant des structures NoC dynamiques, capables de s'auto-organiser et de s'auto-tester dans le but d'obtenir une maintenabilité maximale du fonctionnement du système dans un contexte en réseau. Dans ces travaux, un système communicant multi-nœuds MPSoC reconfigurable capable d'échanger et d'interagir a été développé, permettant ainsi une gestion avancée de tâches, la création et

l'auto-gestion de mécanismes de tolérance aux fautes. Différentes techniques sont combinées et permettent d'identifier et localiser avec précision les éléments défaillants d'une telle structure dans le but de les corriger ou de les isoler pour prévenir toutes défaillances du système. Elles ont été validées au travers de nombreuses simulations matérielles afin d'estimer leur capacité de détection et de localisation des sources d'erreurs au sein d'un réseau. De même, des synthèses logiques du système intégrant les différentes solutions proposées sont analysées en termes de performances et de ressources logiques consommées dans le cas de la technologie FPGA.

Mots-clés: Auto-organisation, Auto-adaptation, Réseau sur Puces, FPGA.

Abstract

The need of growing performance and reliability of embedded System-on-Chips ($SoCs$) are increasing constantly to meet the requirements of applications becoming more and more complexes, new architectural processing paradigms and communication structures based in particular on self-adaptive and self-organizing structures have emerged. These new computing systems integrate within a single chip of hundreds of computing or processing elements (Multiprocessor Systems on Chip - MPSoC) allowing to feature a high level of parallel processing while providing high flexibility or adaptability. The goal is to change possible configurations of the distributed processing characterizing the evolving context of the networked systems. Nowadays, the performance of these systems relies on autonomous and intelligence allowing to deploy and redeploy the compute modules in real time to the request processing and computing power, the communication medium and data exchange between interconnected processing elements to provide bandwidth scalability and high efficiency for the potential parallelism of the available computing power of MPSoC. Moreover, the emergence of the partial reconfigurable FPGA technology allows to the MPSoC to adapt their elements during its operation in order to meet the system requirements. In this context, flexibility, computing power and high bandwidth requirements lead new approach to the design of self-organized and self-adaptive communication systems based Network-on-Chips (NoC). The aim is to allow the interconnection of a large number of elements in the same device while maintaining fault tolerance requirement and a compromise between parallel processing capacity of the MPSoC, communication performance, interconnection resources and tradeoff between performance and logical resources.

This thesis work aims to provide innovative architectural solutions for networked fault tolerant MPSoC based on FPGA technology and configured as a distributed and self-organized structure. The objective is to obtain performance and reliable systems on chips incorporating detection, localization and correction of errors in their reconfigurable or adaptive NoC structures where the main difficulty lies in the identification and distinction between real errors and adaptive properties in these network nodes. More precisely, this work consists to perform a networked node based on reconfigurable FPGA which integrates dynamic or adaptive NoC capable of self-organized and self-test in order to achieve maximum maintainability of system operation in a networked environment (Wireless Sensor Network (WSN)). In this work, we developed a reconfigurable multi-node system based on MPSoC which can exchange and interact, allowing an efficient task management and self-management of fault tolerance mechanisms. Different techniques are combined and used to identify and precisely locate faulty elements of such a structure in order tocorrect or isolate them in order to prevent failures of the system. Validations through the many hardware simulations to estimate their capacity of detecting and locating sources of error within a network have been presented. Likewise, synthesized logic systems incorporating the various proposed solutions are analyzed in terms of performance and logic resources in the case of FPGA technology.

Keywords: Self-organized system, Auto adaptive, Network on Chip, FPGA.

Introduction générale

Contexte général

Les systèmes complexes sur puce reconfigurables basés sur la technologie FPGA sont caractérisés par un haut niveau de modularité et de flexibilité, qui répond à l'évolution croissante des applications à mettre en œuvre, par des implémentions à durée déterminée et dynamiquement replacées (entièrement ou partiellement) au moyen de reconfigurations matérielles. Ainsi, vu l'importance et la complexité actuelle des systèmes embarqués, il est important de leur donner la possibilité de gérer les tâches et les calculs de manière intelligente et autonome. Par conséquent, face aux différents changements et évènements non-déterministes qui perturbent dynamiquement l'environnement des systèmes, une nouvelle approche de conception doit être adoptée. En particulier, une approche tournée vers les réseaux peut être une solution pour les demandes de calculs importants ou lors de la nécessité d'une auto-restructuration des calculs. Ces systèmes sont caractérisés par leur intelligence, leur autonomie et leur capacité à évoluer par eux-mêmes. On parle alors de systèmes auto-organisés capables d'auto-adaptation permettant de répondre aux exigences de flexibilité, de fiabilité, de performances et de contraintes.

Pour bénéficier du potentiel offert par ces systèmes reconfigurables, des réseaux sur puce (NoC) adaptatifs ou dynamiques sont utilisés comme structure de communication afin de permettre un degré élevé de parallélisme, de performances et d'évolutivité [AB15]. Un aspects important de ces NoC est de les rendre tolérants aux fautes grâce à la capacité d'échange de données et de fichiers de configurations permettant de transférer, délocaliser ou de substituer des tâches du système lors d'une défaillance localisée [CZM+10]. Cette délocalisation ou substitution est possible en combinant un mécanisme auto-organisé, réalisant des processus dynamiques et adaptatifs qui interagissent et travaillent de manière autonome et sans contrôle externe, avec la capacité de reconfiguration partielle ou totale des FPGA [HTCD13]. En effet, ces NoC qui correspondent à des routeurs et des interconnexions permettant la communication entre les unités (Processor Elements (PE), Intellectual Property (IP), contrôleurs de mémoire, etc) constituants les systèmes sur puce embarqués (System on Chip (SoC)), assurent les échanges de données selon un algorithme de routage partiellement ou totalement adaptatif [KTMD14]. Le résultat est un système intelligent, auto-organisé et sûr de fonctionnement, capable de gérer le transfert des tâches exécutées au sein d'un NoC reconfigurable.

Dès le début de la reconfiguration dynamique matérielle, l'équipe ASEC-LCOMS s'est impliquée dans l'exploitation du concept d'auto-organisation à base de reconfiguration dynamique pour les systèmes MPSoC adaptatifs comme approche de conception prometteuse [Jov09], notamment pour les systèmes en réseau à base de nœuds matériels à technologie FPGA [CTBD11]. En effet, à partir de l'analyse de caractéristiques d'auto-organisation dans le contexte des systèmes reconfigurables, les premiers travaux de thèse [Jov09] ont établi les besoins nécessaires pour la conception d'un système auto-organisé, et ont permis d'identifier et de recenser les principaux aspects à développer pour aboutir à un système auto-organisé reconfigurable. Les travaux de thèse suivants [CTBD11] se sont focalisés sur les aspects communications entre les modules constitutifs d'un système auto-organisé et leurs rôles primordiaux dans leur conception. En particulier, les aspects liés aux mécanismes de communications scalables à structure de flux dynamiques permettant une distribution de contrôle et une interopérabilité des tâches du système. En particulier, grâce à la coordination de ses entités et la manière dont les décisions au niveau système se prennent à travers des mécanismes locaux utilisés. La poursuite de ces travaux [Kil12], a permis d'élaborer une structure de communication sur puce robuste, scalable et adaptée à la reconfigurabilité d'un système auto-organisé. Plus précisément, en mettant en évidence la nécessité d'intégrer des mécanismes de tolérance aux fautes au sein même des structures de communication sur puce des nœuds constituant ces systèmes auto-organisés. Ainsi, plusieurs solutions SdF ont été développées pour des détections et localisations précises des éléments fautifs constituant un NoC adaptatif au sein d'une structure MPSoC dynamique. Notamment, en proposant à la fois d'une part, un concept de solution SdF hybride associant une technique en-ligne et hors-ligne afin d'obtenir une approche conceptuelle combinant les avantages de ces techniques complémentaires. D'autre part, en proposant une approche originale d'analyse par zone du réseau grâce à la factorisation des éléments de contrôle SdF permettant de minimiser les ressources tout en assurant un taux élevé de couverture d'erreurs. L'ensemble de ces travaux ont permis le développement de la plateforme RKT au sein du LCOMS et correspondant à un système auto-organisé en réseau de nœuds reconfigurables communicants à base de plateformes FPGA à structures NoC dynamiques RKT [Kil12]. Cependant, ces premiers travaux ont montré d'une part que nous sommes tributaires de la technologie disponible, et d'autre part qu'il n'existait peu ou pas de mécanismes d'auto-organisation de tolérance aux fautes intégrés dans les systèmes MPSoC reconfigurables en réseau.

La problématique de ces travaux de recherche s'inscrit dans le contexte de la sûreté de fonctionnement dans un système de nœuds auto-organisés en réseau. Plus précisément, l'objectif de ces travaux est la réalisation d'un réseau de systèmes/capteurs communicants où chaque nœud a la capacité de s'auto-reconfigurer partiellement en vue de répondre à des sollicitations de tâches délocalisées. Ce réseau est entièrement décentralisé (pas de station de base). Dans le concept considéré, ce sont les nœuds du réseau qui communiquent entre eux afin de s'échanger les fichiers de configuration (*bitstream*) ainsi que les données nécessaires associées et permettant l'exécution de tâches ou calculs. Dans le cas de ces systèmes communicants en réseau, la fiabilité ou la sûreté de fonctionnement sont des aspects primordiaux, compte tenu de l'évolution et de la complexité croissante des SoC qui sont de plus en plus sensibles aux phénomènes de génération de défauts de

nature permanente, transitoire ou intermittente (électro-migration, interférences électromagnétiques, diaphonie, rayonnement cosmique, impacts de particules alpha). En effet, la structure de communication centrale du MPSoC peut alors être affectée, et conduire à une défaillance de l'ensemble du système en réseau.

Motivation

Dans ces travaux de recherche, nous proposons d'élaborer un nouveau mécanisme auto-organisé pour la tolérance aux fautes des structures NoC adaptatives considérées comme les parties dynamiques d'un système en réseau multi-nœuds reconfigurables. Plus précisément, l'approche proposée repose sur l'auto-détection par tests et corrections d'erreurs au sein des structures NoC des nœuds communicants. Dans l'approche considérée, nous dissocions les tests et la fiabilité des parties statiques et dynamiques des nœuds reconfigurables. La fiabilité des parties statiques repose sur des tests inter-nœuds par des communications, conduisant à la désactivation d'un nœud défaillant par les autres nœuds du réseau. Pour les parties dynamiques, l'approche proposée repose sur une technique hors ligne (*offline*) de testabilité (Design for Testability (DfT)) par injection de vecteurs de tests dans les NoC des nœuds du réseau préalablement détectés comme défaillants. La principale originalité est que l'IP de test est dynamiquement mis en œuvre à distance dans un nœud non-défaillant afin de tester le NoC d'un nœud détecté fautif par une technique en ligne (*online*) de détection d'erreurs pendant le fonctionnement du même nœud. La procédure de mise en œuvre de l'IP de test dans un nœud à distance repose sur la capacité d'auto-organisation du système multi-nœuds à mettre en œuvre la détection des défauts d'un NoC désactivé, et cela afin de localiser et d'isoler les parties défectueuses pouvant conduire à une défaillance globale du système.

Contribution

Ces travaux de thèse contribuent à proposer des solutions architecturales innovantes pour la sûreté de fonctionnement (SdF) des systèmes auto-organisés en réseau à base de MPSoC. De manière plus précise, ces travaux consistent à réaliser un réseau de nœuds reconfigurable à base de FPGA intégrant des structures de communication sur puce (NoC) dynamiques ou adaptatives, capable de s'auto-tester dans le but d'obtenir une maintenabilité maximale de fonctionnement dans un contexte en réseau (WSN). La stratégie de recherche employée consiste à développer un système multi-nœuds reconfigurables capable d'échanger et d'interagir, et permettant ainsi une gestion avancée de tâches, la création et l'auto-gestion de mécanismes de tolérance aux fautes au sein même du réseau. Deux approches sont proposées dans ces travaux. La première est le développement d'un réseau permettant une organisation autonome des nœuds en fonction de différents facteurs tels qu'une défaillance. Une application dans un contexte sans fil de type WSN démontre la faisabilité de l'approche auto-organisée proposée où chaque nœud est composé d'un FPGA associé à un module de communication sans fil au protocole *Zigbee*. Nous utilisons la capacité de reconfiguration du FPGA, permettant la modification d'une partie

de l'architecture en cours de fonctionnement, afin d'organiser dynamiquement les tâches du système selon des mécanismes de répartition et de délocalisation au sein des différents nœuds constituant le système en réseau. La deuxième approche proposée consiste à intégrer une structure de routage de type NoC RKT dans chacun des nœuds du réseau. L'objectif recherché est d'atteindre la meilleure capacité de réorganisation des tâches. Une particularité du réseau sur puce sûr de fonctionnement proposé est l'intégration d'un test hybride alliant les avantages de différentes techniques de tests de NoC adaptatifs. Plus précisément, des modifications architecturales sur les structures NoC sont apportées afin de pouvoir efficacement détecter, localiser puis corriger d'éventuelles erreurs présentes dans les routeurs.

Plan du manuscrit

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

Dans le **premier chapitre**, nous rappelons la définition d'un système auto-organisé et recensons les principales propriétés associées (adaptabilité, autonomie, robustesse, flexibilité, anticipabilité, contrôle décentralisé). Nous détaillons ensuite la vision conceptuelle d'une architecture reconfigurable MPSoC auto-organisée (Reconfigurable Self-Organized System (RSS)) exploitant la reconfiguration dynamique de la technologie FPGA pour la mise en œuvre matérielle de ces propriétés. En particuliers, nous détaillons le concept d'auto-organisation par reconfiguration, incluant l'évolution des supports de communication utilisés dans ces systèmes. Les exigences des aspects réseaux, nécessitant l'élaboration de nouveaux paradigmes basés sur des structures de communication sur puce adaptatives et décentralisées selon le concept RSS d'interaction entre nœuds, sont également présentés. Enfin, nous concluons ce chapitre sur l'intérêt d'intégrer des mécanismes de tolérance aux fautes dans les structures sur puce des nœuds constituant un RSS. Plus particulièrement, nous présentons l'intérêt de la mise en œuvre de nouveaux mécanismes et techniques architecturales basés sur des structures de communication adaptées à la technologie FPGA et exploitant l'auto-organisation partielle afin d'assurer une fiabilité dans l'objectif d'une maintenabilité maximale du fonctionnement global d'un réseau WSN.

Le **deuxième chapitre** rappelle brièvement les notions de sûreté de fonctionnement (SdF) par la tolérance aux fautes de systèmes électroniques embarqués. Les principaux concepts et techniques de base et la taxonomie de la SdF sont présentés. Ensuite, ce chapitre détaille les concepts architecturaux SdF proposés et appliqués aux NoC dynamiques, en présentant le contexte de la tolérance aux fautes dans les réseaux intégrés. L'objectif est de localiser les éléments fautifs afin d'inhiber la propagation des erreurs. Pour ce faire, dans le cadre de la plateforme RKT, des blocs de détection et de localisation d'erreurs sont implantés au sein des réseaux pour effectuer des tests d'intégrité (au démarrage ou pendant le fonctionnement). Les principales méthodes de détection d'erreurs dans les NoC sont alors présentées dans ce chapitre. Puis, une approche architecturale d'optimisation de l'intégration des blocs de détection et de correction d'erreurs, stratégiquement placés dans le réseau, est détaillée. Enfin, nous concluant sur l'évidence de la nécessité de déve-

lopper des mécanismes d'auto-organisation de tolérance aux fautes dédiés aux structures basées MPSoC reconfigurables.

Le **troisième chapitre** présente l'approche architecturale adoptée permettant une solution matérielle de mise en œuvre de la tolérance aux fautes par auto-organisation d'un réseau de nœuds reconfigurables (Reconfigurable System on Chip (RSoC) ou reconfigurable MPSoC). En particulier, pour répondre à la principale difficulté qui réside dans l'identification et la distinction entre des erreurs réelles et des fonctionnements variables ou adaptatifs des éléments constituant les nœuds en réseau. Dans un premier temps, nous proposons une approche architecturale caractérisée par la mise en œuvre du principe d'auto-organisation à travers le développement matériel d'une structure pour communications informationnelles, circulaires localement et globalement au système auto-organisé proposé (notion de flux). Ce concept original permet des échanges d'informations entre les modules constituant le système sans un contrôle centralisé. Le but est de répondre à tout changement ou évolution du système dans le cas de défaillances. Puis, différentes techniques sont combinées et permettent d'identifier et localiser avec précision les éléments défaillants d'une telle structure, dans le but de les corriger ou de les isoler pour prévenir toutes défaillances du système. L'objectif visé est la conception de systèmes sur puce communicants sûrs de fonctionnement par l'intégration de mécanismes distribués et délocalisés de techniques de détection, de localisation et de correction d'erreurs des structures NoC reconfigurables ou adaptatives en réseau.

En vue de valider l'approche auto-organisée de tolérance aux fautes proposée et associée aux concepts, mécanismes et structures de communication sur puce adoptés, le **quatrième et dernier chapitre** présente les résultats de l'intégration des techniques de localisation d'erreurs proposées. Un réseau sans fil multi-nœuds reconfigurables capables d'échanger et d'interagir a été développé, permettant ainsi une gestion avancée de tâches, la création et l'autogestion des mécanismes de tolérance aux fautes adoptés. Les mécanismes détaillés dans le chapitre précédent sont ainsi mis en œuvre à travers un exemple concret de fonctionnement auto-organisé au cours d'une défaillance temporaire ou permanente d'entités constituant le réseau multi-nœuds basé sur le protocole sans fil *ZigBee*. Les approches utilisées ont été validées au travers de nombreuses simulations matérielles afin d'estimer leur capacité de détection et de localisation des sources d'erreurs au sein d'un réseau. De même, des synthèses logiques du système intégrant les différentes solutions proposées sont analysées en termes de performances et de ressources logiques consommées dans le cas des technologies FPGA *Xilinx*. L'objectif principal de ce chapitre est une validation expérimentale de l'ensemble des aspects conceptuels abordés dans ces travaux de recherche. Le but est de rendre un système capable de gérer de manière autonome et intelligente les éventuelles perturbations des structures NoC d'un système auto-organisé en réseau provenant de l'environnement dans lequel il évolue.

Enfin, une **conclusion générale** dresse le bilan et discute de ces travaux de recherche en présentant les apports des concepts et mécanismes architecturaux proposés et en dégageant des perspectives, notamment en termes de vérification de la fiabilité même des paradigmes proposés.

Chapitre 1

Systèmes sur puce reconfigurables auto-organisés et en réseau

Introduction

L'évolution importante du nombre de transistors au sein des puces électroniques a permis aux concepteurs de systèmes électroniques d'intégrer de nombreuses fonctions formant ainsi des systèmes sur puces (SoC). Les SoC peuvent être constitués d'un ou plusieurs processeurs, coprocesseurs, Digital Signal Processor (DSP), mémoires ou IP spécifiques. Plus précisément, les IP sont des blocs matériels réalisant des fonctions de traitement spécifique (acquisitions de données, cryptage, *monitoring*, etc.) lié à un large spectre d'applications [N.N12] qu'un concepteur implante dans un SoC à partir d'une éventuelle bibliothèque. Les SoC constitués de plusieurs processeurs et/ou IP sont désignés commes des multi-processeurs sur puce (MPSoC) pouvant s'associer en réseau comme l'illustre la figure 1.1. Cette quantité croissante de ressources logiques permet donc de concevoir des systèmes de plus en plus complexes et performants capables de réaliser des tâches combinant plusieurs éléments de traitement au sein d'une même puce. Les performances du système sont donc considérablement augmentées par la facilité de parallélisation des tâches au sein d'un SoC pouvant intégrer jusqu'à une centaine d'IP mais surtout grâce à une approche de type réseau.

Cette progression technologique a permis une évolution vers la conception de systèmes matériels auto-organisés qui peuvent être définis comme des systèmes de traitement dans lesquels le comportement global découle de la manière dont sont organisés les échanges et les interactions entre les modules bas-niveaux les constituants. Les règles d'interaction entre les modules constituant les systèmes 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. Parmi les aspects à considérer dans la conception de systèmes auto-organisés reconfigurables, on trouve l'aspect approche architecturale mettant en œuvre les concepts liés aux propriétés d'une auto-organisation matérielle [LR15]. 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. Un second aspect, le plus fondamental, est lié à la communication entre les modules constitutifs d'un système qui 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. Or, dans les systèmes hétérogènes, la difficulté majeure réside dans le support de communication qui doit fournir une certaine performance et efficacité pour les échanges de données des IP constituant le SoC. Il est apparu que les architectures de communications standards tels que les bus (partagés ou hiérarchiques) ou les connexions Point à Point (PaP) ne peuvent faire face aux contraintes des SoC, étant donnée leur faible adaptabilité et leur difficulté pour le placement et routage. C'est pourquoi, 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 adaptée à la technologie FPGA [AD14], et sur des structures originales de communications informationnelles contribuant à mettre en œuvre les principes d'auto-adaptation et d'auto-organisation.

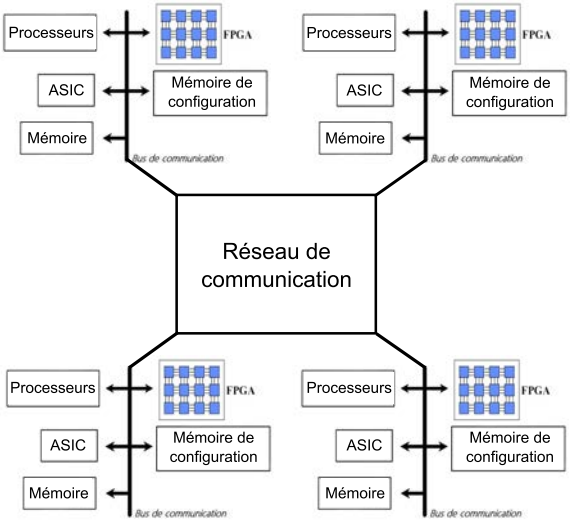


FIGURE 1.1 – Illustration d'un Système sur Puce Multi-Processeurs.

La suite de ce chapitre est structurée de la manière suivante. Nous rappelons une définition d'un système reconfigurable auto-organisé, puis les principales caractéristiques de la reconfiguration dynamique à base de technologie FPGA. Ensuite, 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 . Un recensement des principaux besoins pour la conception de systèmes reconfigurables auto-organisés est rappelé en montrant la nécessité d'élaborer de nouveaux concepts architecturaux basés sur des structures de communication sur puce de type NoC dynamique adaptées à la technologie FPGA. Enfin, nous concluons sur la nécessité primordiale de développer des mécanismes de sûreté de fonctionnement en cours de fonctionnement de l'ensemble des structures de communication, socle fondamental de la maintenabilité maximale du fonctionnement d'un système auto-organisé multi-nœuds en réseau.

1.1 Systèmes auto-organisés

1.1.1 Définition

On définit un système auto-organisé comme un système qui doit adapter son comportement aux changements éventuels par modification de sa structure et générer lui-même sa structure de fonctionnement. En effet, une structure ne présente qu'une disposition particulière des composants constituant le système 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 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 est trop important. 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 permettent au système une mémorisation de résolutions. Pour la mise en œuvre d'un tel système, il est donc primordiale de considérer l'importance de l'interactivité des entités du système à la fois au niveau local entre les éléments voisins et au niveau système. Dans le cadre de ces travaux, nous considérons ci-dessous une définition synthétique et plus globale de l'auto-organisation des systèmes [JTBW09] :

"L'auto-organisation d'un système est 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 des critères préétablis ".

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

1.1.2 Propriétés

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, et consiste à considérer un système composé d'éléments interactifs de sorte que le comportement global du système ne peut pas être déduit des comportements individuels de ses éléments [LMS91]. Un exemple de système présentant la propriété de complexité est une cellule biologique, qui manifeste la "vie" au niveau macroscopique tandis que ces éléments microscopiques constitutifs n'en manifestent aucune. 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 [A$^+$72].

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 associe cette propriété à 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 [Ger01].

La dynamique. Un système auto-organisé peut être considéré comme un processus dynamique évoluant dans le temps. Étant donné qu'un tel système évolue dans un environnement variable et dynamique, il doit lui-même 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 auto-organisé est caractérisé par une dynamique et une réaction à des changements plus importantes. Si il réussit à se maintenir dans un état qui lui permet un échange constant entre ses éléments constitutifs et son environnement, afin de produire une large variété d'actions menant à plusieurs configurations possibles [H$^+$01].

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.

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) [H$^+$01].

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 [Rob85]. 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 pré-adaptation 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 contrôle 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 hétérogènes [SKRZ04]. 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 ceux 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 des entités du système [DWH05]. C'est pourquoi, la propriété de contrôle décentralisé est souvent associée à la notion d'émergence.

L'augmentation de l'ordre d'un système. 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 manière à permettre l'accomplissement des fonctions globales possibles par le 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 et représente un état stable du système exprimant une de ses structures possibles et se manifestant à travers une fonction correspondant à une structure donnée du système [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 considérée comme un changement d'ordre du système. Si le changement d'ordre du système mène vers une structure spatiale, temporelle ou fonctionnelle 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 permettant de décrire l'augmentation de l'ordre d'un système est présentée dans [S$^+$01]. La complexité

statistique est une manière de mesurer la complexité d'un système, effectuant une fonction précise, à travers une mémoire historisée [GG00]. 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 [Hak06, 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 adapter son comportement par une réorganisation éventuelle de sa structure. Le système lui-même à travers ses éléments constitutifs initie d'une manière autonome et indépendante des actions à effectuer afin de répondre à des changements potentiels.

1.1.3 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 [DWH05]. La figure 1.2a 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. La figure 1.2b illustre un système manifestant uniquement des propriétés de l'émergence. Ces

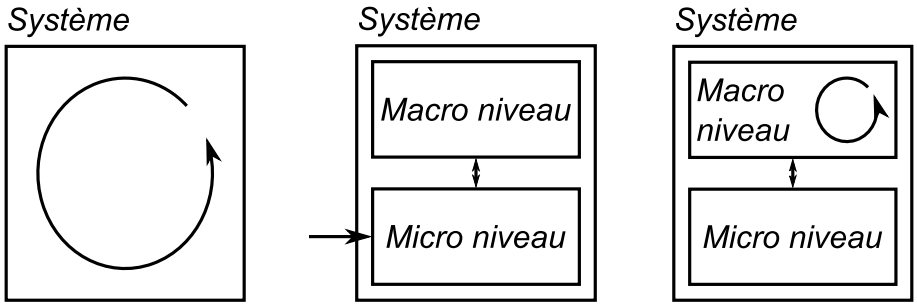


FIGURE 1.2 – Illustrations des systèmes auto-organisés et/ou émergents.

propriétés résultent des interactions entre les entités constituant le système. Ces entités peuvent évoluer sans entraîner un changement structurel du système initial. La figure 1.2c 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.

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 [VDPB01, 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.***

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 systèmes 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 systèmes circulent en sens inverse, du bas niveau vers le haut niveau de l'organisation. Plus précisément, 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 1.3.

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 présentent de bonnes caractéristiques en termes de productivité, dû essentiellement à leurs optimisations internes. En terme de

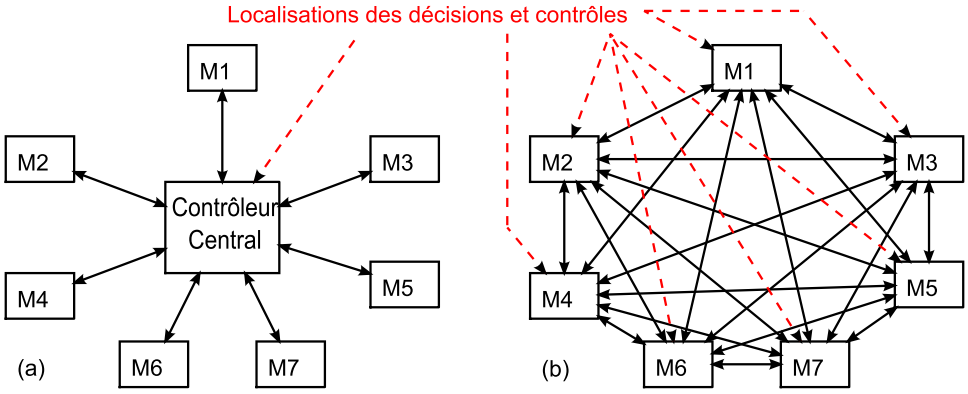


FIGURE 1.3 – Types de contrôle, (a) Centralisé, (b) Décentralisé.

	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

TABLE 1.1 – Comparaison des types de contrôle.

robustesse, si l'unité de contrôle d'un tel système a un dysfonctionnement au cours de son fonctionnement, le système considéré dans sa globalité, ne sera plus opérationnel. C'est la principale raison aujourd'hui que les approches traditionnelles de conception (initialement à base d'une centralisation de contrôle) s'orientent vers des approches à contrôle décentralisé. Dans une approche décentralisée, la gestion globale du système est distribuée dans tous ses modules constitutifs et est assurée par leurs interactions mutuelles. Le tableau 1.1 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. 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 (solutions logicielles). Il existe peu d'exemples de systèmes à contrôle décentralisé implantés dans des structures matérielles de type FPGA afin d'exploiter leur reconfigurabilité ou fléxibilité. 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 et permettant la réalisation des interactions mutuelles par des réseaux de communications fiables doivent être développées.

1.1.4 Technologie de mise en œuvre de l'auto-organisation

Une nouvelle approche de mise en œuvre du principe d'auto-organisation que les solutions micro-programmées est l'utilisation de technologies matérielles reprogrammables de type FPGA grâce à l'exploitation de la reconfiguration dynamique. Par la suite, nous adaptons 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, Tin08]. L'atout principal d'une telle solution technologique est l'apport d'une flexibilité permettant des traitements adaptés. Cependant, elle ne demeure pas moins déterministe dans la mesure ou la conception 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 permet 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 la contrainte 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 d'un system 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 grâce à l'interactivité de ses éléments constitutifs,
- Par sa complexité et sa modularité.

Parmi les solutions de mise en œuvre de tels systèmes, peuvent être citer les systèmes

de calcul inspirés organique (*organic computing*), les systèmes de calcul autonome (*autonomic computing*) et les systèmes multi-agents (*multi-agent systems*) [LR15]. 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 élevée. Cependant, ces systèmes présentent des performances insuffisantes dans certains domaines d'application temps réel. En effet, de façon plus concrète les systèmes existants reposent principalement sur l'association de processeurs modélisant des entités et permettant une forte flexibilité.

C'est pourquoi, nous considérons dans ces travaux l'auto-organisation à base de technologie FPGA. Plus précisément, c'est une mise en œuvre de la propriété d'auto-organisation et/ou émergence d'un système à travers la mise en place de mécanismes exploitant 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 de la suite de chapitre qui présente les concepts, les mécanismes et les besoins architecturaux et structurels pour la réalisation matérielle de systèmes auto-organisés.

1.2 Systèmes reconfigurables dynamiquement

La technologie FPGA fournit aux concepteurs un support de développement et de conception doté d'une très grande puissance de réutilisation. Nous nous focalisons dans ces travaux sur les FPGA Static Random Access Memory (SRAM) permettant de se reconfigurer indéfiniment. Certaines familles de FPGA disposent d'une technologie de reconfiguration partielle permettant de modifier en cours de fonctionnement une partie du système, augmentant ainsi en temps réel la flexibilité face aux besoins du système [Xil12]. En effet, ces reconfigurations partielles permettent d'implanter dynamiquement des accélérateurs ou des blocs matériels spécifiques afin de répondre à un besoin du système à un instant donné [CZM+10]. Cette technologie offre de nouvelles possibilités de partitionnement physique, temporel et fonctionnel d'une même application [RGFSC10]. Nous présentons dans cette section les principes de base de la reconfiguration dynamique exploitables par la réalisation matérielle du concept d'auto-organisation.

1.2.1 Structure d'un circuit FPGA

Les cicuits FPGA reconfigurables dynamiquement sont généralement basés sur des blocs logiques appelées Configurable Logic Block (CLB) [Che11, Xil12]. Chaque CLB est divisé en deux parties (*slices*) contenant chacune une Look-Up-Table (LUT) à 6 entrées

et huit registres si on considère la technologie *Xilinx Virtex VII*. Ces CLB sont connectés à une grille de routage au sein du FPGA par l'intermédiaire d'un commutateur matriciel (*switch matrix*) qui est configuré afin d'interconnecter plusieurs CLB (figure 1.4). Les interconnections de plusieurs CLB permettent de réaliser des fonctions logiques séquentielles ou combinatoires [Che11]. Lorsqu'un système doit être implanté dans un FPGA, un fichier de configuration (*bitstream*) est chargé. Ce fichier de configuration indique toutes les interconnexions à réaliser entre les différentes CLB du FPGA ainsi que le contenu des LUT et des registres des CLB utilisés.

Afin d'augmenter les performances des FPGA, un certain nombre de connexions internes et de blocs matériels (DSP, blocs *RAM*, etc.) sont intégrés et présents de manière permanente au sein du FPGA. Parmi ces blocs, on trouve le composant réalisant la reconfiguration dynamique et partielle du FPGA. Ce module est appelé Internal Configuration Access Port (ICAP) et permet d'accéder à la mémoire de configuration du FPGA contenant le fichier de configuration (*bitstream*). Ainsi, lorsque l'application configurée dans le FPGA doit être modifiée, le module ICAP remplace le fichier de configuration contenu dans cette mémoire.

1.2.2 Reconfiguration dynamique et partielle des FPGA

L'utilisation de la reconfiguration dynamique et partielle permet à un FPGA de modifier pendant son fonctionnement une partie de son architecture sans interrompre son fonctionnement et sans modifier l'état des autres parties ne subissant pas la reconfiguration. Pour ce faire, des zones spécifiques (Partial Reconfiguration Region (PRR)) doivent être définies physiquement au sein du FPGA lors de la phase de conception. Des fonctions électroniques représentées sous formes de modules (Partial Reconfiguration Module (PRM)) vont alors pouvoir être implantées dans les PRR préalablement définies. Plus précisément, les PRM sont des fichiers de configuration d'une PRR et sont généralement stockés dans une mémoire système pour être ensuite transférés à la mémoire de configuration via le module ICAP. La reconfiguration dynamique et partielle peut ainsi être comparée à une gestion de fichier [CZM+10]. Ainsi, un système peut être constitué de plusieurs PRR associées à de nombreux PRM stockés dans la mémoire système afin d'implanter dynamiquement les fonctions nécessaires aux besoins des applications. Ce dispositif de reconfiguration permet d'augmenter la flexibilité des MPSoC. Lorsque qu'un traitement est terminé, il peut être remplacé par d'autres. La principale contrainte de la reconfiguration dynamique partielle est la définition physique des PRR au sein d'un FPGA. En effet, les PRR et les PRM doivent avoir les mêmes interfaces pour être connectés à la partie statique du FPGA. La connexion entre la partie statique et dynamique est effectuée par l'intermédiaire de "bus macros" [SH10]. Ces derniers sont réalisés via des CLB décrits comme point d'accès des signaux devant être connectés à un PRM.

La réalisation d'un Système Reconfigurable Dynamiquement (SRD) est décrit par le flot de conception illustré en figure 1.5 [RGFSC10]. En partant d'une description en langage de description Hardware Description Language (HDL) à un niveau Register Transfer Level (RTL), la première étape à réaliser est le partitionnement du système. Le système

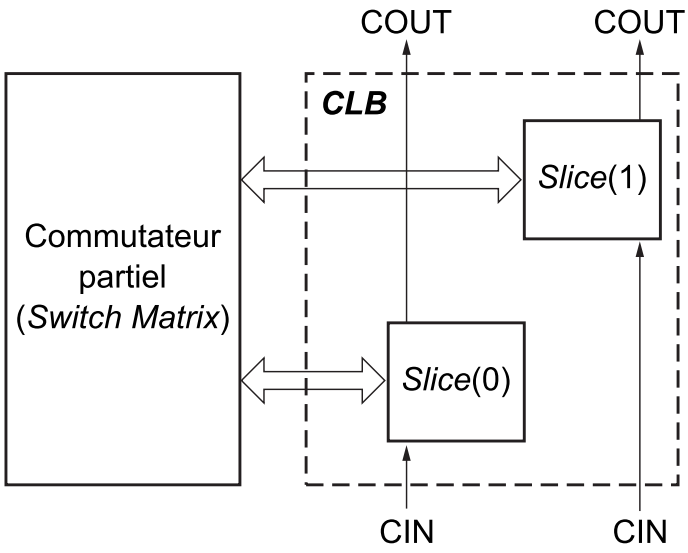


FIGURE 1.4 – CLB : élément de base des FPGA *Virtex* de *Xilinx* [Xil12].

est ainsi divisé manuellement en plusieurs modules. Certains modules sont fixes et sont implantés de manière permanente dans la partie statique du FPGA. Les autres modules correspondent à des PRM qui seront instanciés dynamiquement dans les PRR. L'étape suivante consiste à effectuer la synthèse des différents modules individuellement via une procédure automatique de l'outil de conception associé (*ISE*, *Xilinx*). Après ces synthèses, l'étape de placement des entrées et sorties est effectuée via un outil graphique (*Floorplanner*). Cet outil va ensuite générer le fichier des contraintes du système. L'avant dernière étape consiste à effectuer le placement et routage de la partie statique, c'est à dire des bus macros et des PRM au sein des PRR. Finalement, dans la dernière étape du flot de conception, une fusion de la partie statique avec les différentes PRR permet d'obtenir le fichier de configuration du système global, ainsi que les fichiers de configuration partielle correspondant à chaque PRM.

Actuellement, la conception d'un SRD est complexe par le manque d'outil et d'automatisation [RGFSC10]. La majorité des tâches du flot de conception d'un SRD doit être effectuée manuellement. Afin de faciliter l'utilisation de la reconfiguration dynamique partielle d'un SoC, des outils sont en cours de développement afin d'automatiser le flot de conception de ce genre de système. C'est notamment le cas de l'outil *DynoPlace* qui permet de générer automatiquement le placement et l'allocation des modules au sein des PRR [RGFSC10].

1.3 Système auto-organisé reconfigurable : modélisation simplifiée et formalisation

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

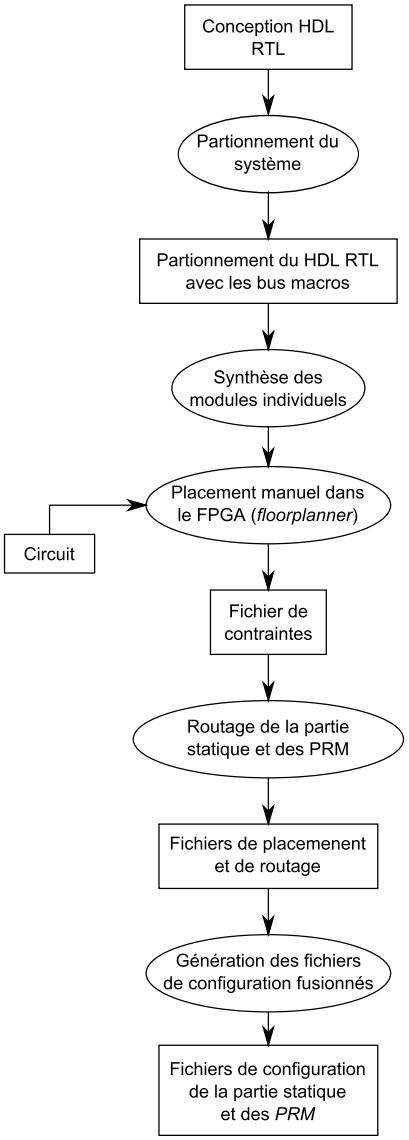


FIGURE 1.5 – Flot de conception d'un SRD [RGFSC10].

propriétés des technologies reconfigurables doivent être réalisées. Plus particulièrement, dans le cas de traitements non deterministes où il faut apporter des configurations adaptées à ses traitements évolutifs. En effet, elle permet 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, et d'apporter une flexibilité permettant des traitements adaptés. 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. La reconfigurabilité dynamique des technologies FPGA est actuellement la technologie la plus prospective autour des systèmes adaptatifs et auto organisés.

Un système auto-organisé reconfigurable (RSS), doit donc être capable 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 extérieur et dans le but d'un fonctionnement optimal par rapport à des critères prédéfinis,
- 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 face à des événements susceptibles de survenir.

L'utilisation de la technologie reconfigurable pour mettre en œuvre un système répondant aux critères ci-dessus doit donc lui permettre à :
- Adapter son fonctionnement à des perturbations survenues,
- Distribuer, en cas de défaillance d'une de ses entités, les fonctions ou tâches vers les autres entités,
- Remplacer ses parties ou entités défaillantes par le biais d'une phase auto-reconfiguration dynamique,
- Détecter le changement de type de données d'entrée à traiter et adapter son fonctionnement global à partir des fonctions possibles de ses entités,
- Organiser d'une manière autonome ses entités de sorte qu'elles "anticipent" les traitements à venir,
- Agir de sa propre initiative pour optimiser selon des critères prédéfinis l'utilisation de ses ressources matérielles.

En considérant la reconfigurabilité d'un système comme une architecture matérielle capable de changer de fonction au cours du temps, 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.

Par conséquent, pour qu'un système reconfigurable soit conforme aux propriétés et aux exigences imposées par le concept d'auto-organisation et/ou émergence, il doit posséder principalement :

- 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. Ce moyen de communication doit assurer les interactions entre les modules du système en favorisant l'apparition de la propriété 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.
- 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 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 spécifique pour les échanges entre modules.
- 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 modification au cours de fonctionnement une partie matérielle du système sans dégrader le fonctionnement du reste du système permet une auto-restructuration du système comme solution réelle d'auto-adaptation.

L'illustration de la modélisation adoptée dans ces travaux d'un RSS intégrant les aspects présentés ci-dessus est donnée par la figure 1.6, et représente un système qui :

- adapte son comportement à toutes perturbations,
- distribue, en cas de défaillance d'un nœud, les fonctions ou tâches à d'autres modules,
- remplace ses modules en utilisant l'auto-reconfiguration,
- détecte la modification de données entrantes et adaptent son comportement global à travers les fonctions possibles des modules pour les nouveaux calculs,
- organise de façon autonome ses modules pour les rendre aptes à anticiper les futures tâches de traitement.

Dans ces travaux, nous considérons un système auto-organisé comme un concept d'intelligence matérielle coopérative basé sur des structures d'échanges dynamiques informationnels appelés *Flux* permettant une mise en œuvre des principaux mécanismes d'auto-

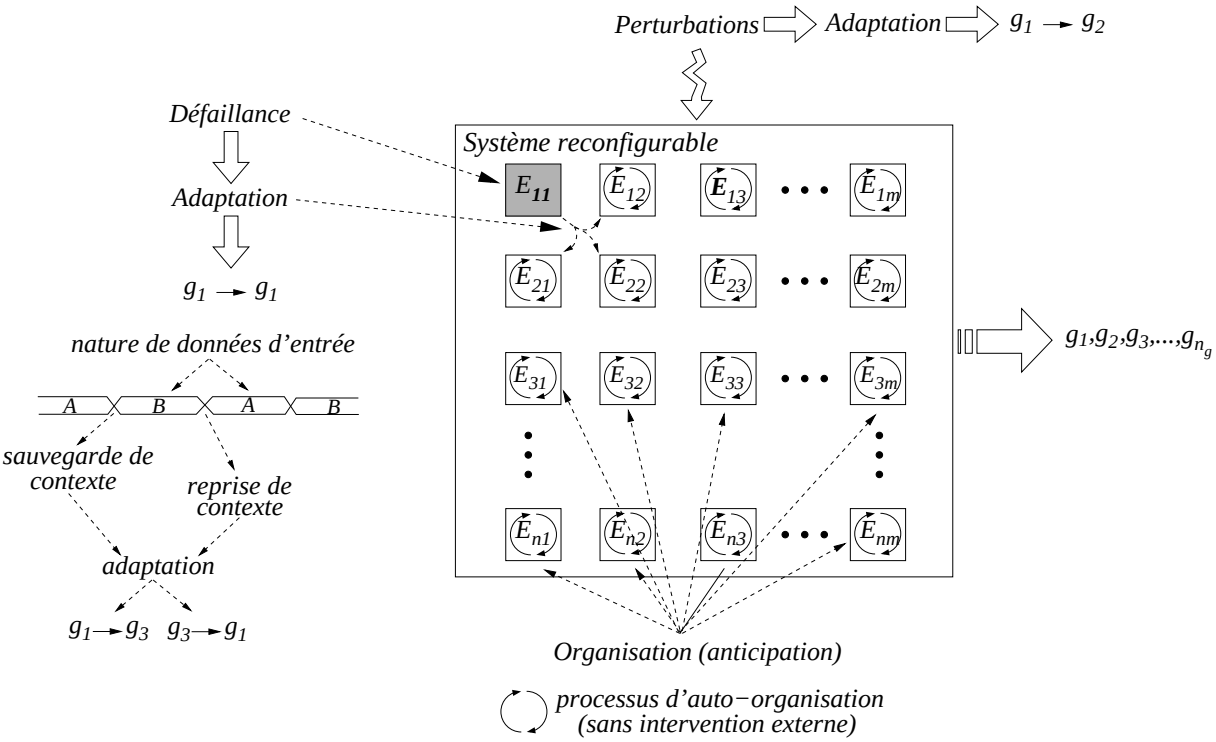


FIGURE 1.6 – Illustration de la formalisation d'un RSS.

organisation.

En termes de formalisation, on suppose un système en réseau S composé de N modules E_i $(1 \leq i \leq N)$, capable de communiquer et d'échanger des données (figure 1.7). Chaque module E_i réalise une fonction ou tâche $e_i(t)$ à un instant t. Chaque fonction du module représente une partie d'un ensemble de fonctions disponibles (ou de services) S_i qui peut être accompli par le module donné et contribue à réaliser une fonction globale $g(t)$ tel que $g(t) \in S$, où S représente l'ensemble des fonctions qui peuvent être mis en œuvre par les nœuds de réseau auto-organisé considérés ($g(t) \in S$, $S = \{\emptyset, g_1, g_2, ..., g_{n_g}\}$). Définie de cette façon, le système peut être décrit par l'ensemble des expressions suivantes [JTBW09] :

$$e_i(t) \in S_i,\ 1 \leq i \leq n,\ n \in N. \tag{1.1}$$

$$g(t) = \sum_{i,j=1}^{N} e_{ij}(t), 0 < i, j < N, n \in N. \tag{1.2}$$

où chaque tâche e_i peut être délocalisée et placée sur chaque nœud reconfigurable. Pour mettre en œuvre ce concept, les principaux besoins et mécanismes architecturaux ont été identifiés :

- Nouvelle approche architecturale,
- Une structure de communication adaptée à la technologie reconfigurable,
- Intégration de mécanismes d'auto-reconfiguration,

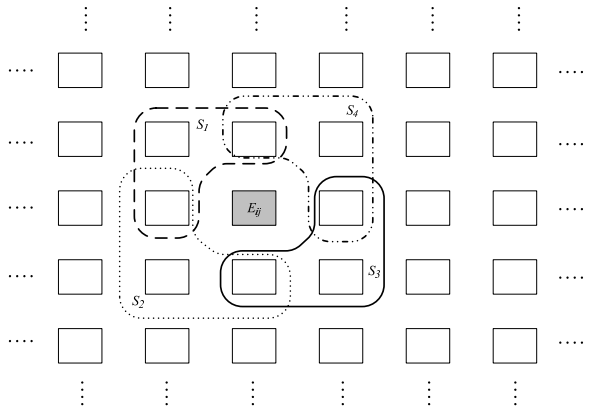


FIGURE 1.7 – Formulation du principe d'auto-organisation matérielle [Jov09].

- Mécanismes de sûreté de fonctionnement,
- Structure de type maillé (*mesh*),
- Redondance fonctionnelle et matérielle ou un ensemble de fonctions pouvant être exécutées par un module/système (*intelligence distribuée*),
- Un *contrôle décentralisé*,
- Une *autonomie* de gestion,
- Une *robustesse* aux défaillances basée sur une résolution *locale* et *progressivement* et permettant une *alerte* générale *progressive* en cas de problèmes. Il s'agit de considérer une *auto-surveillance* des modules par les modules du système permettant des *prises* de décisions *mutuelles* (selon l'illustration de la figure 1.8).

1.4 Conception matérielle de l'auto-organisation

Le système auto-organisé reconfigurable provient de l'association des concepts de l'auto-organisation et l'auto-reconfiguration [JTW08]. Pour réaliser ce concept, un système multi-FPGA réel, composé de nœuds intelligents en réseau est mis en œuvre. Dans ce réseau, les nœuds sont en mesure de choisir leurs propres tâches, en fonction des demandes des autres nœuds reconfigurables tout en assurant une certaine flexibilité et l'adaptabilité du système distribué. Deux nœuds sont également en mesure de communiquer directement pour échanger des données. Dans la pratique, chaque FPGA représente un nœud auto-organisé du réseau et permet de mettre en œuvre une architecture intégrée complexe qui se compose de cœurs de processeurs, blocs matériels (c. IP blocs), des unités de surface dynamiquement reconfigurables (Dynamic Reconfiguration Unit (DRU)) et d'un réseau sur puce (NoC). Chaque nœud est associé à une mémoire (externe et/ou intégré) contenant des fichiers de configuration (*bitstream*) locales, et est capable de supporter des applications dynamiques et de veiller à l'exécution de plusieurs tâches au sein de ses ressources.

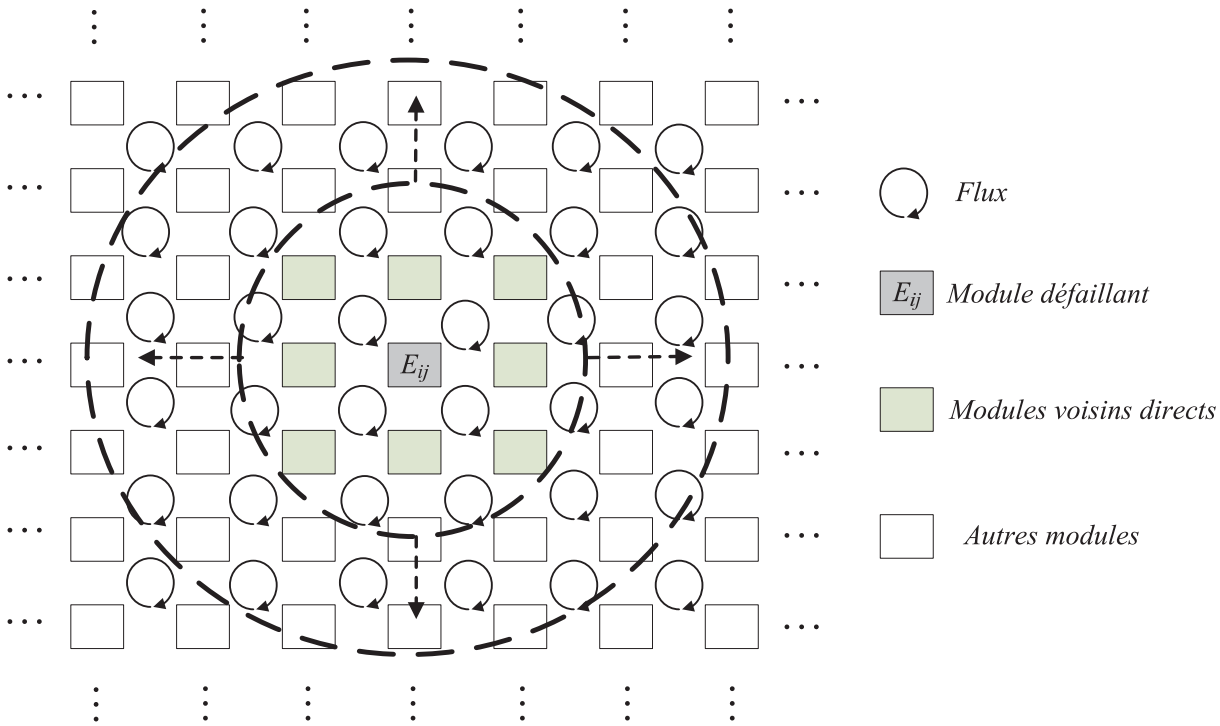


FIGURE 1.8 – Vision architecturale adoptée d'un système auto-organisé reconfigurable [Jov09].

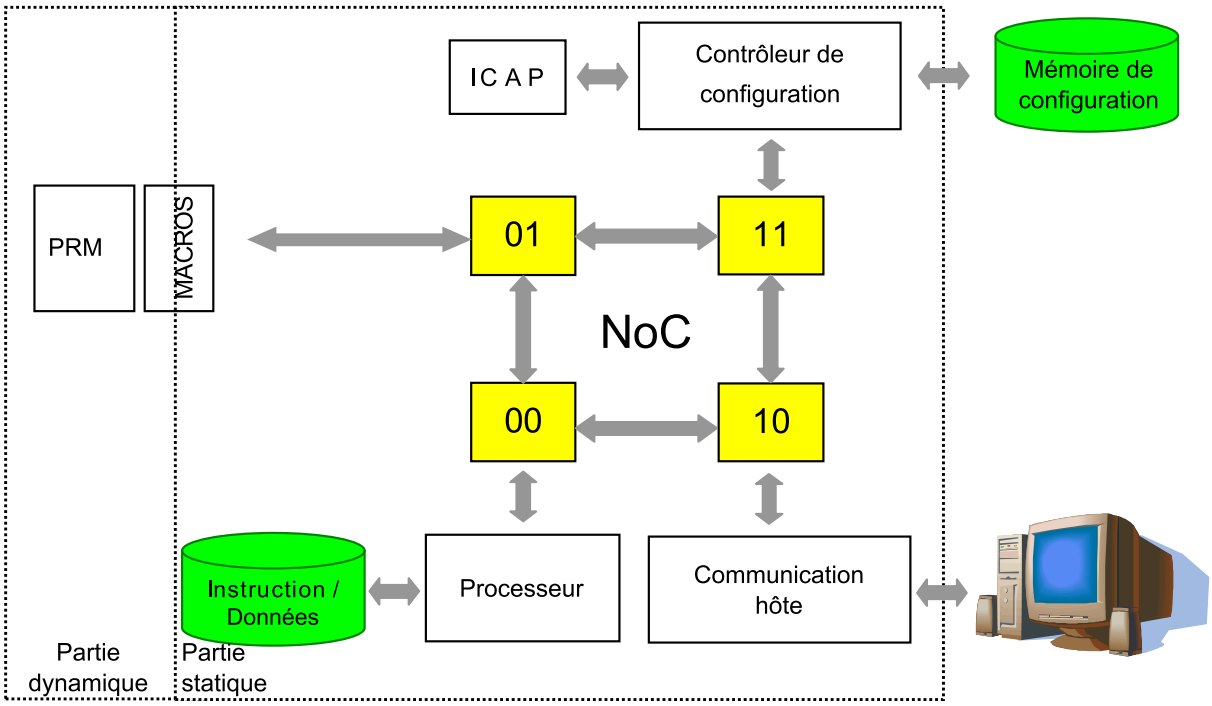


FIGURE 1.9 – Architecture reconfigurable dynamiquement *Artemis* [MGC+07].

1.4.1 Reconfiguration dynamique et partielle pour MPSoC basé NoC

La combinaison des architectures MPSoC utilisant un NoC pour leurs communications et la technologie de reconfiguration partielle, permet diverses solutions architecturales pour les systèmes. Une première solution est l'utilisation d'un réseau NoC fixe connecté à une ou plusieurs PRR [SH10]. Par exemple, l'architecture *Artemis* (voir figure 1.9) propose l'utilisation d'un NoC 2 × 2 maillé supportant les échanges de données entre une PRR, située dans la partie dynamique du FPGA, avec un processeur et un contrôleur hôte qui sont situés dans la partie statique [MGC+07]. Dans cette architecture, les constituants du réseau de communication ne peuvent pas se modifier. En effet, les PRR sont localisées aux périphéries du NoC et permettent aux différents IP de s'implanter dynamiquement.

Une autre solution est d'utiliser un NoC reconfigurable jouant un rôle central entre plusieurs parties statiques [RAS+10], comme illustré en figure 1.10. L'avantage de cette solution est de pouvoir adapter la topologie du NoC, ainsi que les tables de routage, en fonction de l'application et des besoins de communication entre les différents IP des parties statiques. Cependant, en dimensionnant des PRR selon la taille du plus grand IP, un surdimensionnement des ressources peut se produire. En effet, les routeurs du NoC sont généralement beaucoup plus petit en termes de surface logique que les IP intégrés dans le système.

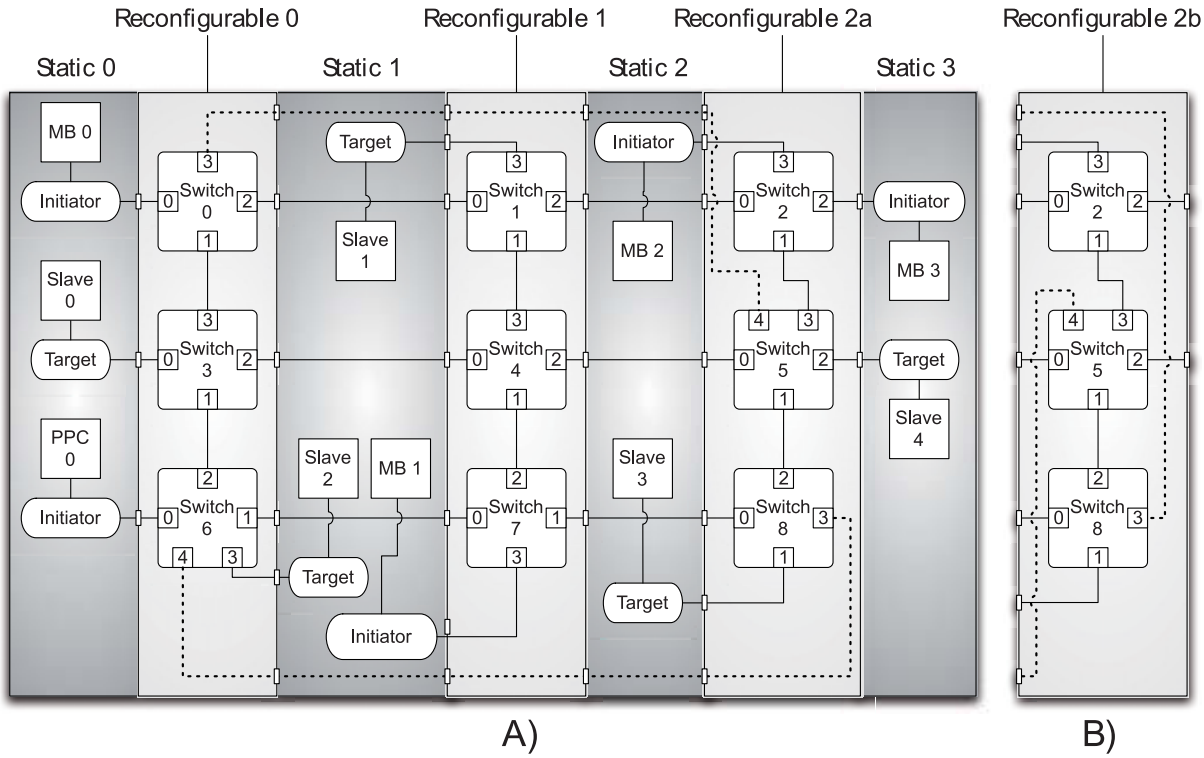


FIGURE 1.10 – MPSoC doté d'un NoC reconfigurable et d'IP implémentés en zones statiques : A) liens directs entre les routeurs 6 et 8 et entre les routeurs 0 et 5 ; B) PRM de la PRR2 réalisant une connexion directe entre les routeurs 0 et 8, et les routeurs 6 et 5 [RAS+10].

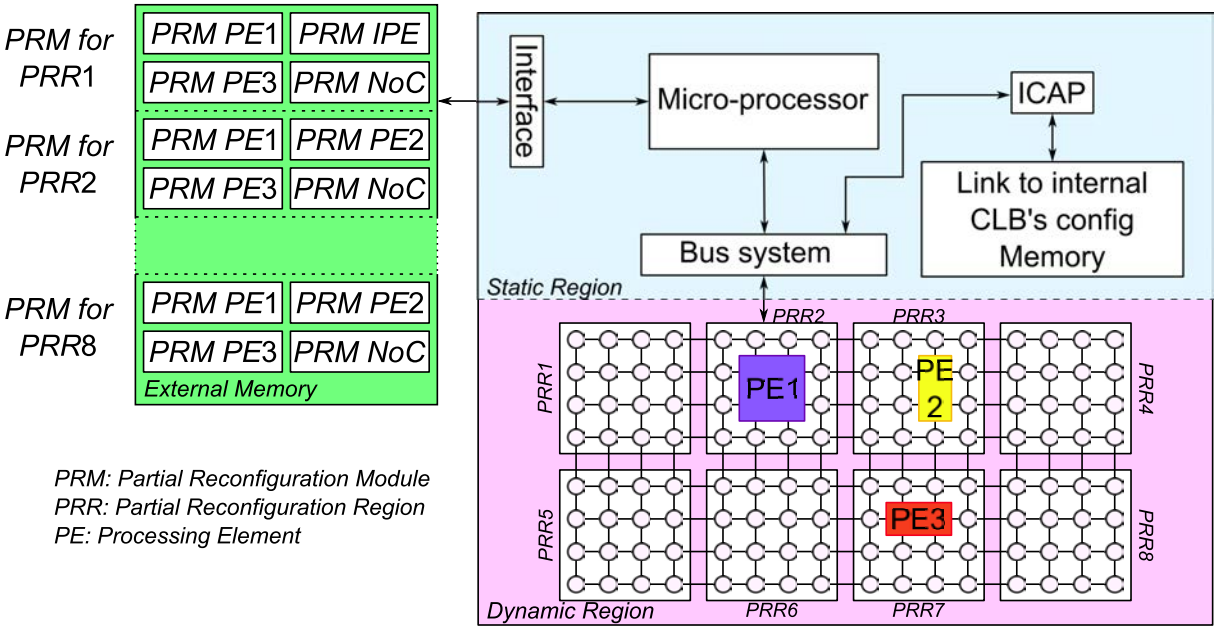


FIGURE 1.11 – Vue d'ensemble d'un nœud auto-organisé reconfigurable partiellement à base de technologie FPGA [Kil12].

1.4.2 MPSoC basé NoC reconfigurable dynamiquement

Pour augmenter considérablement l'adaptabilité des MPSoC [GOR15], il faut pleinement exploiter la technologie FPGA reconfigurable dynamiquement et partiellement pour modifier en temps réel les éléments de calculs constituant le MPSoC afin de répondre aux besoins de l'application. La principale contrainte réside donc sur la définition lors de la phase de conception des PRR dans lesquelles les PRM vont s'implanter dynamiquement [MT12]. Afin qu'un élément de calcul puisse s'implanter dans plusieurs PRR, il faut que son interface (nombre et position des entrées/sortie délimitant sa région au sein du FPGA) concorde avec les PRR susceptibles de l'intégrer. Pour ce faire, nous considérons l'architecture RSS illustrée par la figure 1.11 . Dans cette architecture, des PRR de dimensions identiques sont définies au sein du FPGA et dans lesquelles des routeurs et des IP peuvent être implantés [HK12]. Chaque PRR est configuré soit par un NoC soit avec un IP entouré de routeurs. La taille de la PRR doit au moins correspondre à la taille du plus grand IP. Ainsi, l'interface de toutes les PRR est identique et correspond aux définitions de la périphérie d'un NoC. Ce type de solution architecturale est efficace pour la mise en œuvre d'un NoC reconfigurable et donc est adapté à l'approche auto-organisée de tolérance aux fautes que nous proposons dans le chapitre suivant. En effet, les interactions nécessaires pour l'augmentation de l'ordre de l'émergence d'un système auto-organisé repose principalement sur le support de communication permettant les inter-connections entre les entités constituant le système auto-organisé.

1.5 Conception de systèmes auto-organisés en réseau

Les nœuds des réseaux sans fil (WSN) ayant une capacité de traitement et stockage limitée, la reconfiguration matérielle dynamique est parfaitement adaptée. Elle consiste en une reconfiguration partielle du FPGA en cours de fonctionnement. Par conséquent, plusieurs configurations d'architectures peuvent être stockées dans la mémoire et être utilisées qu'en fonction du besoin courant. On trouve dans la littérature un grand nombre de travaux sur l'utilisation de la reconfiguration dynamique à base de FPGA dans des réseaux de capteurs comme l'auto-réparation [YQG+12][SCM+12]. Dans la majorité des cas, la reconfiguration est gérée de l'extérieur par un microprocesseur/microcontrôleur [JS10][PBEKCRA09] bien qu'il soit possible d'intégrer matériellement l'ensemble du système [GGRG09][CTA08]. L'objectif de ces travaux est d'obtenir un système multi-nœuds, indépendants les uns des autres, ayant la capacité de communiquer afin de s'auto-reconfigurer, de s'auto-réparer ou de délocaliser une ou plusieurs tâches en cas de défaillance, dans le but de maintenir un bon fonctionnement de l'ensemble des tâches du système.

Contrairement à la majorité des travaux menés sur les WSN et l'exploitation de la reconfiguration reposant sur des solutions micro-programmées (processeurs, co-processeurs, DSP, etc.), ces travaux se fondent sur le concept des systèmes à base de nœuds matériellement reprogrammable FPGA avec la possibilité d'intégrer une gestion par reconfiguration partielle [N.N12]. Nous considérons ainsi une structure WSN au travers d'auto-reconfigurations partielles de l'architecture matérielle des nœuds constituant le réseau. Un tel système s'inscrit au delà du cas particulier de la SdF du réseau en lui même, mais dans une SdF globale d'un système de structure WSN, dont le rôle principal de collecte d'informations, va permettre l'auto-organisation de l'ensemble des tâches des nœuds du réseau permettant de maintenir, et garantir le fonctionnement global du système malgré des défaillances.

Beaucoup de travaux sur la SdF dans un WSN sont axés sur la sécurité des communications et l'organisation du réseau et donc sur le cryptage [POdlT+10], les protocoles de communication, la localisation des nœuds [VWS+12], etc.. Dans notre approche WSN l'adaptabilité associée à des auto-reconfigurations des nœuds, est l'élément qui permet la SdF du système. En effet, le mécanisme WSN associé permet l'échange de données, relatifs aux tâches du système, entre les différents nœuds afin de garantir des fonctionnalités continues grâce à la possibilité de les modifier matériellement et dynamiquement et/ou de délocaliser leurs traitements au sein d'autres nœuds du réseau. Concrètement, les fonctionnalités d'"auto-test" et de délocalisation des tâches des nœuds matériels sont possibles grâce à l'échange des fichiers de reconfiguration partielle associés à chaque tâche entre les différents éléments du WSN. C'est donc une combinaison WSN/auto-reconfiguration partielle qui rend le système tolérant aux fautes et donc plus robuste. L'originalité du système réside dans sa gestion intelligente de l'échange des tâches entre les différents nœuds du réseau afin de maintenir l'intégralité ou au moins un certain degré de fonctionnement. La figure 1.12 illustre l'architecture auto-organisée en réseau adoptée, et qui est composée de nœuds reconfigurables matérialisés par des circuits FPGA.

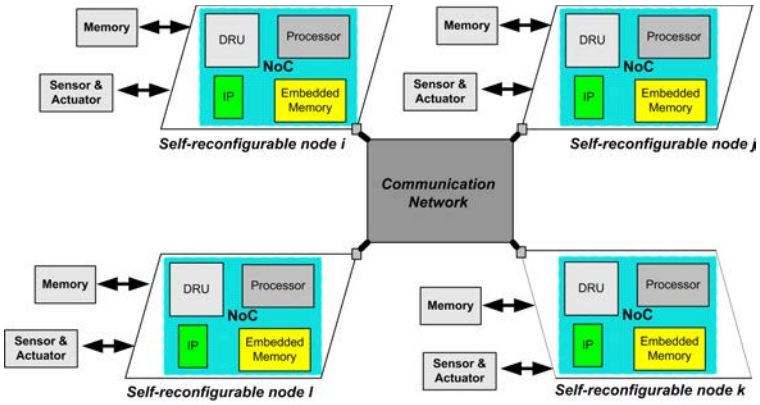


FIGURE 1.12 – Architecture réseau de nœuds auto-reconfigurables [CTBD11].

Le système multi-FPGA réseau considéré donc dans ces travaux est basé sur une répartition autonome des tâches en utilisant un flux d'échanges informationnels et permettant à chaque unité FPGA d'ouvrir des communications directes avec toute autre unité FPGA du réseau avec l'objectif de partager des données. En échangeant des données ou des fichiers de configurations partielles, notre système auto-organisé réalise la délocalisation des tâches de calcul avec un objectif de créer une entité qui peut se transformer en temps réel.

1.5.1 Concept de distribution des tâches par flux informationnel

Pour assurer une exécution efficace à la demande, le réseau auto-organisé doit dynamiquement gérer les tâches en utilisant les nœuds auto-reconfigurables. En raison de leur nature, ces nœuds sont limitées non pas par le nombre de tâches qu'ils peuvent traiter séquentiellement, mais par leurs régions reconfigurables pouvant contenir des logiques de traitement. Ces ressources ou régions logiques de traitement doivent préalablement être configurées afin de réaliser des fonctions de calcul. En outre, un nœud doit être en mesure d'effectuer plusieurs tâches et permettre à la gestion dynamique de plusieurs actions simultanément sans perturber le reste de la conception au cours des adaptations du système. En pratique, les demandes de ressources sont limitées à des informations sur la tâche elle-même. Dans ce contexte, la prise de conscience des nœuds voisins est émergente et provient des réponses potentielles. Pour toute réponse reçue, un nœud demandeur détecte automatiquement l'existence d'un autre nœud.

Pour permettre les configurations au sein du réseau, une approche intelligente coopérative adoptée est basée sur un flux informationnel qui se déplace d'un nœud à un autre et contenant des informations spécifiques : les numéros d'identification de la tâche et les noms des différents fichiers. Ces informations permettent aux nœuds de s'auto-organiser pour décider ou non de leur disponibilité et de leur capacité à contribuer ou non dans la mise en œuvre d'une adaptation du système. Il en résulte au travers de flux informationnel le dé-

clenchement de la procédure d'échanges permettant d'établir des liens de communication directe avec les nœuds demandeurs. Ces liens directs de communication sont établies pour permettre les transmissions de données ou des fichiers de données. Cette approche est possible que si le flux peut être mis à jour par tous les nœuds sollicitant l'assistance d'autres nœuds du réseau. La figure 1.13 illustre le mécanisme d'auto-reconfiguration basé sur le concept de flux. Elle montre la distribution des tâches et les flux de données point-à-point utilisés par le processus d'organisation des tâches dans le réseau. Le principal objectif de

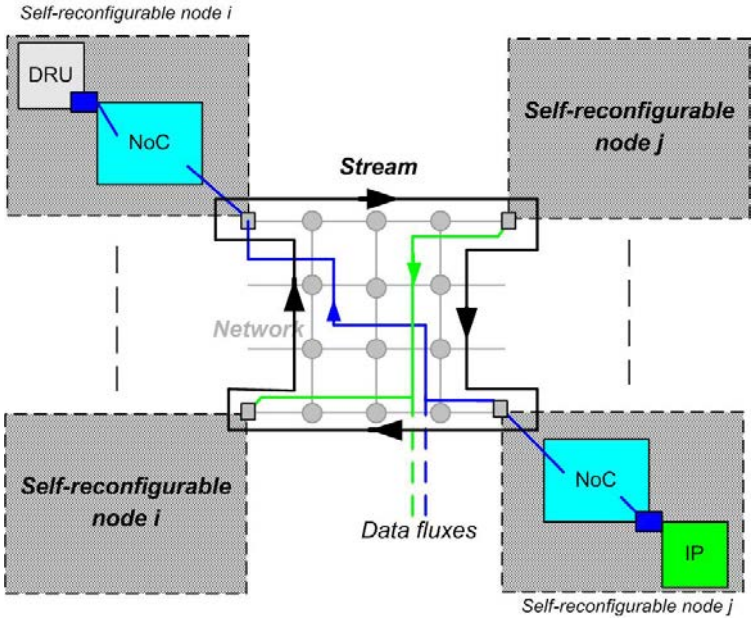


FIGURE 1.13 – Flux de transmission de requêtes de tâches, de données ou de *bitstreams* [CTBD11] au sein de l'architecture auto-organisée de nœuds reconfigurables.

ce flux est de recueillir des informations sur les tâches qui doivent être traitées par des nœuds et par extension l'entité réseau. Par exemple, si un $noeud_i$ reçoit une demande du $noeud_j$ et si le $noeud_i$ a une région reconfigurable libre, il peut répondre au $noeud_j$ en demandant les fichiers associés de la tâche concernée. En d'autres termes, une demande de ressources de calcul initiale est suivie d'une demande éventuelle de transmission de fichier.

Il est important de considérer que le flux n'est pas une structure de contrôle. Il ne fixe aucune décision sur les nœuds du réseau. Son rôle principal est de rassembler et d'informer des demandes. Le flux permet aux nœuds du réseau de mettre en place une répartition auto-organisée des tâches. Il est structuré en deux sous-flux. Le premier est le flux de demande et le second est un flux de données de point-à-point. Le flux de demande est établie pour relier l'ensemble des nœuds du réseau mais sans contrôle central et avec un protocole minimal. Le flux de données du fichier de donnée est établie pour échanger des fichiers de données de manière fiable. Ces deux types de flux sont conçus en utilisant des protocoles et technologies basés sur l'envoi de paquets. Toute norme de communication peut garantir une qualité de service supplémentaire pour les communications. La figure

1.13 illustre les différents types de communications. Le flux de demande qui atteint tous les nœuds du réseau est représenté en noir. Les flux temporaires créés pour transférer des données ou flux binaires entre les nœuds sont représentés en bleu et vert.

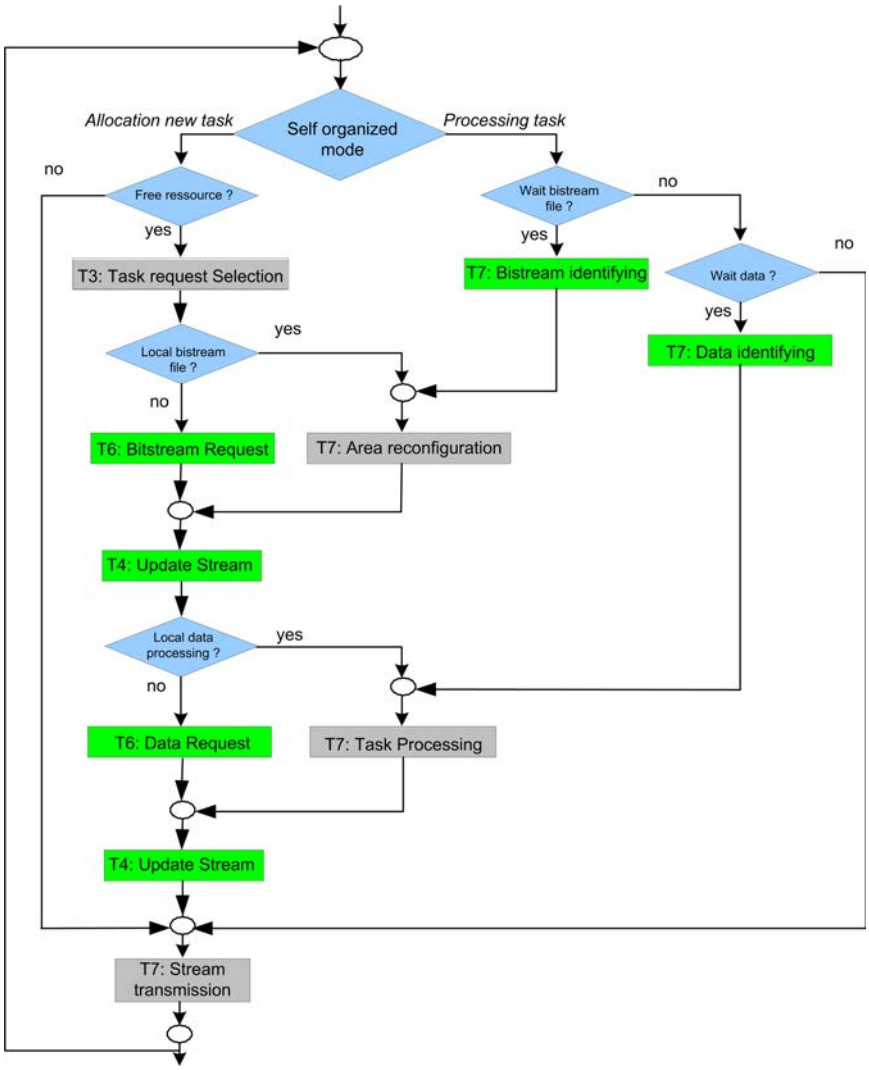


FIGURE 1.14 – Graphe de contrôle d'un nœud auto-organisé [CTBD11].

En résumé, quand un nœud accepte de traiter une tâche, il demande des données et le *bitstream* pour reconfigurer sa zone reconfigurable. Par exemple, un $noeud_i$ est choisit pour exécuter une tâche spécifiée indiquée dans le flux. Il renvoie une demande de fichier pour échanger des données et le *bitstream* avec le $noeud_j$. En outre, le concept de ressources et la demande de fichier est complètement dépendant de l'intelligence du nœud qui constitue une intelligence distribuée et un contrôle du matériel non-centralisée nécessaire à la conception d'un système auto-organisé. En effet, en donnant à chaque nœud la possibilité de gérer leurs propres tâches, il est possible de créer des propriétés d'auto-organisation sans aucune sorte de nœud. La figure 1.14 représente l'organigramme de commande du

mécanisme d'auto-organisation sur la base de nœuds dynamiquement reconfigurables et qui produit un mécanisme intelligent coopératif. Dans ce graphique, deux principales directions peut être suivie : l'attribution d'une nouvelle tâche ou le traitement d'une tâche.

Dans le cas de l'allocation d'une tâche (côté gauche), il est d'abord vérifié si les ressources disponibles existent et peuvent être partagées. Par conséquent, le fichier de données et le fichier de reconfiguration associé sont présents. Ces vérifications impliquent la découverte de la localisation de la tâche. Si les fichiers ne sont pas détectés au niveau local, cela signifie que les demandes doivent être transmises afin d'échanger ces fichiers. Cela correspond à la deuxième partie du graphe (côté droit), où le nœud est en attente de l'arrivée du fichier. Lorsque le fichier de reconfiguration est reçu, le nœud peut se reconfigurer. Après cela, lorsque le fichier de données est reçu, il peut être traité sur le module nouvellement configuré réalisant la tâche demandée.

Le flux de distribution des tâches est la partie la plus importante de l'entité reconfigurable auto-organisée. C'est un système d'auto-géré qui unifie tous les nœuds en formant un groupe. Toutes les informations nécessaires pour prendre une décision et communiquer sont prévues pour un nœud quelconque. Son principal objectif est de fournir des informations en temps réel et de créer une organisation émergente. C'est pourquoi le flux est conçu pour gérer un nombre de tâches évolutif en fonction de la charge de travail et des ressources disponibles au sein du réseau.

Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche mettant en œuvre la notion d'auto-organisation matérielle dans la conception architecturale des systèmes numériques de traitement. 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 une architecture 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, sa capacité à 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 l'architecture structurelle a été détaillée. Ensuite, à partir de l'analyse des caractéristiques d'auto-organisation dans le contexte des systèmes reconfigurables, nous avons rappelé 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. Notamment, les mécanismes de distribution de contrôle et de coordination des modules constitutifs, et la manière dont

les décisions au niveau système se prennent à travers les mécanismes locaux utilisés. Ensuite, nous avons mis en évidence l'aspect fondamental lié aux communications entre les éléments constitutifs d'un système auto-organisé. En effet, un moyen de communication robuste, scalable et adapté à la reconfigurabilité est primordial à développer. Nous avons ainsi montré l'intérêt de considérer qu'il faut envisager des solutions architecturales basées sur des structures MPSoC associé à des réseaux sur puce reconfigurables (Reconfigurable Network on Chip (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, et l'intérêt de l'utilisation de structures de communication sur puce reconfigurables. Ces réseaux permettent la mise en pratique d'une auto-organisation matérielle par une approche architecturale basée sur des placements dynamiques de modules de calcul au sein d'un réseau grâce à l'exploitation de la reconfiguration dynamique partielle des technologies FPGA disponibles (technologie *Virtex Xilinx*). Il est donc primordiale à veiller à la sûreté de fonctionnement de ces structures de communication sur puce. En effet, la fiabilité globale de tels systèmes nécessite d'assurer la tolérance aux fautes des parties NoC reconfigurables étant donnée leur importance dans le concept de l'auto-organisation considéré basé sur les interactions informationnelles entre les éléments constitutifs d'un système RSS. C'est l'objet du chapitre suivant.

Chapitre 2

Sûreté de fonctionnement des réseaux sur puce reconfigurables (RNoC)

Introduction

L'augmentation constante du nombre de cellules logiques et de transistors dans les puces électroniques, ainsi que la diminution de la finesse de gravure, rendent les structures MPSoC actuelles sensibles aux phénomènes générant des fautes de nature permanente, intermittente ou transitoire. Ces fautes peuvent générer des erreurs au sein des blocs fonctionnels constituant le système sur puce, à la fois au niveau du contenu des données transitant dans le NoC, ou sur des décisions de routage effectuées par les routeurs du réseau. La Sûreté de Fonctionnement (SdF) des NoC intégrés dans les MPSoC est un élément clé pour pouvoir étendre leurs utilisations dans le contexte d'une conception de systèmes auto-organisés.

Actuellement, dans les NoC proposés, pour prévenir les fautes sur les paquets de données pouvant provoquer leurs pertes au sein du réseau, des Code Correcteur d'Erreurs (CCE) sont généralement intégrés dans le réseau. De plus, des mécanismes de localisation d'erreurs (en cours ou hors fonctionnement du réseau) permettent d'analyser le réseau afin d'identifier les éléments fautifs. Cette localisation des éléments fautifs, permet de les isoler afin d'arrêter la génération des erreurs. Le contournement des éléments isolés est réalisé par des contournements des paquets de données transitant dans le réseau grâce à l'utilisation d'algorithmes de routage adaptatif. Cependant, les techniques proposées dans la littérature montrent des limites en termes de localisation des erreurs permanentes [Kil12]. En effet, pour certaines structures, des routeurs entiers sont déconnectés du réseau alors qu'un seul élément constitutif est fautif. Cela a pour effet de dégrader les performances du réseau par une surcharge des routeurs voisins au routeur fautif.

Actuellement, pas ou peu de travaux proposent la détection d'erreurs de routage des paquets de données pour des NoC dynamiques caractérisés par des algorithmes adaptatifs de routage. C'est dans ce contexte que des travaux ont été initié préalablement à ces travaux de thèse [Kil12]. Plus précisément, ces travaux initiaux ont permis de proposer

des techniques de détections en ligne (en cours de fonctionnement du réseau) permettant une localisation précise et garantissant des performances optimales pour le réseau, et cela même en présence d'erreurs permanentes. L'approche repose sur la mise en œuvre de procédés de désactivation temporaire ou permanente et de façon partielle ou globale des éléments fautifs du réseau afin de maintenir au niveau le plus élevé les performances du réseau. Néanmoins, si les travaux ont permis une meilleure converture de detection et localisation d'erreurs au niveau d'un seul réseau, des solutions restent à élaborer dans un contexte de SdF pour un réseau de structure NoC constituant un système auto-organisé multi-nœuds en réseau.

Dans ce chapitre, nous présentons la nécessité d'intégrer de la SdF dans les systèmes reconfigurables auto-organisés. Nous présentons, dans un premier temps la taxonomie et les concepts de base de la SdF. Dans un deuxième temps, nous détaillons les principales solutions de SdF appliquées aux NoC dynamiques correspondant à l'élément central d'un système auto-organisé en réseau. Enfin, nous proposons d'étendre la SdF dans le cas d'un réseau de nœuds auto-organisés structurés par des réseaux de NoC.

2.1 Généralités sur la SdF

Depuis plusieurs années, les concepteurs se sont focalisés sur la conception de systèmes de plus en plus performants, avec le soucis de garantir leur fonctionnement en les protégeant des erreurs. Ceci est particulièrement primordial pour les systèmes auto-organisés pouvant avoir un impact majeur sur les interactions des entités constitutifs.

Les origines des fautes dans les circuits électroniques sont très variées. La première est liée à la technologie employée pour graver les circuits intégrés. En effet, elle dispose d'une finesse de gravure réduisant considérablement l'échelle de conception (une finesse de gravure de 18nm en 2015) menaçant grandement la sensibilité aux radiations des futurs circuits [RYKO11]. Plus précisément, cette réduction de la finesse de gravure, et donc de la taille des transistors, rend les circuits sensibles aux rayonnements ionisants pouvant inverser l'état logique de leurs cellules [Muk08]. De plus, avec la finesse de gravure et la complexité croissante des systèmes, le nombre de transistors intégrés dans les circuits augmente ainsi que le nombre de fils d'interconnexion. La probabilité d'avoir alors des défauts de conception, et donc des transistors défectueux, augmente. La température est également un facteur pouvant augmenter le nombre de fautes transitoires ou permanentes dans les circuits intégrés, notamment en causant par exemple un vieillissement prématuré des puces. La dernière source pouvant provoquer des erreurs dans un système sont les défauts de conception liés à l'augmentation de la complexité des systèmes tel que l'on peut la trouver pour la réalisation de l'augmentation de l'ordre d'un système auto-organisé.

2.1.1 Définition de la sûreté de fonctionnement

Un système, quel qu'il soit, est défini comme sûr de fonctionnement si il a la capacité de limiter la fréquence et le niveau des défaillances à un niveau d'acceptabilité [ALRL04].

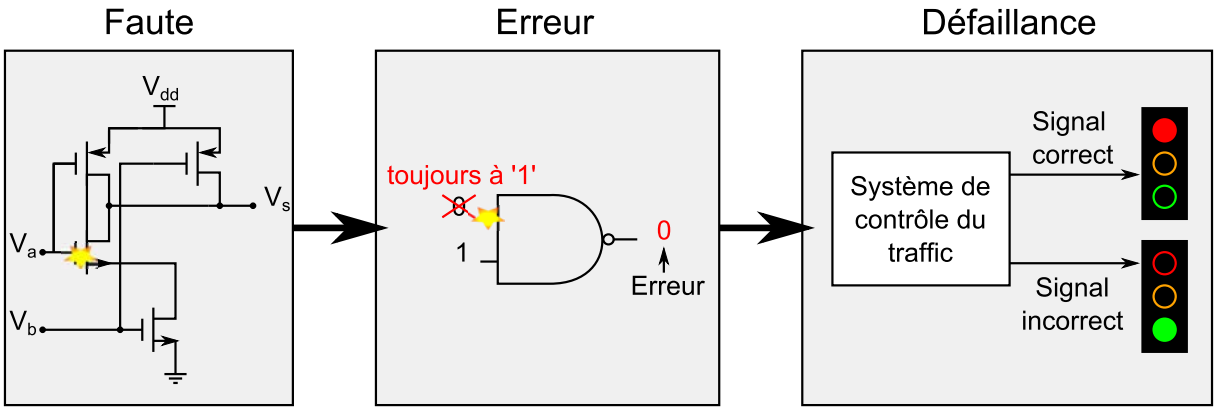


FIGURE 2.1 – Cycle faute, erreur et défaillance [ALRL04] et exemple sur un contrôle de trafic [Ami11].

On trouve 5 critères pour évaluer le niveau de SdF d'un système :

- La disponibilité : évalue la qualité du système à fonctionner lorsqu'un service est demandé,
- La fiabilité : évalue la continuité de fonctionnement du système,
- La sûreté : évalue l'impact d'une défaillance sur l'environnement et les utilisateurs,
- L'intégrité : évalue l'absence d'altération inappropriée du système,
- La maintenabilité : définit si un système est facilement modifiable et facile à remettre en œuvre après avoir subit une défaillance.

2.1.2 Relations entre fautes, erreurs et défaillances

Un système est défini défaillant si il n'arrive plus à réaliser les opérations qui lui sont demandées, ou à les effectuer en donnant un résultat erroné. La défaillance d'un système est le résultat de fautes et d'erreurs dans les composants du système. La figure 2.1 illustre le cycle faute, erreur, défaillance. La faute est définie comme un élément matériel du système ayant été modifié, dégradé ou endommagé par un élément externe (comme par exemple une particule alpha). Il se peut qu'une faute ne génère pas d'erreurs. Elle est alors définie comme *dormante* [ALRL04]. Au contraire, lorsqu'elle génère des erreurs au sein du système, on dit que la faute est *active*. Dans ce cas, les erreurs générées au sein du système vont se propager et générer une défaillance lorsque le système ne délivrera plus le résultat escompté. Il y a donc un lien de cause à effet entre la faute, l'erreur et la défaillance.

2.1.3 Origines et conséquence d'une faute

Plusieurs facteurs peuvent donc être à l'origine des fautes. Les circuits deviennent notamment de plus en plus sensibles aux rayonnements ionisants pouvant générer des

erreurs temporaires appelées erreurs logicielles, qui représentent 80% des erreurs dans un circuit [LLP07]. Trois principales sources d'erreurs logicielles sont répertoriées. Les particules alpha et les ions qui "touchent" le semi conducteur provoquent le résultat illustré en figure 2.2-a. La particule frappent le transistor Metal Oxide Semiconductor (MOS) et laisse derrière elle des paires d'électrons-trous qui génèrent des perturbations de charge à travers le substrat en causant une impulsion de courant proportionnelle à l'énergie de la particule. Une donnée dans une mémoire peut ainsi subir une inversion de bit (*bit-flip*). La figure 2.2-b illustre l'impact d'un neutron. Le neutron perturbe un transistor percutant un atome de silicium. Cet impact cause l'éjection d'ions ayant le même effet qu'une particule alpha, c'est à dire une impulsion de courant et une possible inversion de bit.

Concernant les FPGA SRAM, des supports matériels de mise en œuvre d'un RSS, des erreurs de type permanente peuvent être présentes dues à des défauts de fabrication (bits bloqués à un état haut ou bas) [SSC08]. Les erreurs transitoires sont dues à certains types de radiation comme pour les transistors MOS [SSC08]. En effet, si une particule alpha ou un neutron percute une cellule SRAM, cela provoque une inversion de bit qui affecte les données ou les fonctions comme illustré en figure 2.3 [Mic11]. De telles erreurs sont souvent corrigées à l'aide de CCE. Si une particule touche une cellule utilisée pour une fonction logique ou une matrice de routage, alors cette erreur peut changer le fonctionnement du circuit comme illustré en 2.3. Une telle erreur persistera tant qu'elle n'est pas détectée et corrigée, ou en reconfigurant de nouveau le FPGA avec la fonction considérée.

Le terme couramment utilisé pour désigner l'impact d'une particule est Single Event Effect (SEE) (Événement à Effet Unique). La conséquence d'une SEE peut entrainer plusieurs types d'erreurs. Le premier type d'erreurs est appelée Single Event Upset (SEU) (Evenement Unique d'Inversion) et correspond à l'inversion de l'état d'une mémoire ou d'un registre. Ce type d'erreurs correspond à des erreurs dites *logicielles* [FCCK06, BCT08]. Le circuit n'est pas endommagé de façon permanente, et la simple réécriture dans la mémoire ou dans le registre touché effacera l'inversion de bit. Une SEU peut se manifester en tant que Single Bit Upset (SBU) (Inversion d'Un Bit) ou Multiple Bit Upset (MBU) (Inversion de Plusieurs Bits). Dans le cas d'une SBU, seul un bit a sa valeur inversée, alors que dans le cas d'une MBU plusieurs bits sont affectés [BCT08]. On désigne par Single Event Transient (SET)(Événement Unique Transitoire) un signal transitoire provoqué par une SEE, et est souvent observé comme un parasite (*glitch*) dans les circuits [BCT08]. On désigne par Single Event Fonctional Interupt (SEFI) (Événement Unique d'Interruption de Fonctionnalité), un SEE générant une interférence dans le fonctionnement d'un circuit complexe, comme par exemple sa capacité de configuration ou sa fonctionnalité Joint Test Action Group (JTAG) [BCT08].

Les SEE peuvent également provoquer des erreurs permanentes [Ami11]. On désigne par Single Event Latchup (SEL) (Événement Unique de Verrouillage) le fait qu'un SEE provoque un court-circuit virtuel entre une tension d'état haut et une masse dans le circuit. Ces évènements provoquent donc un effet de destruction d'une cellule au sein d'un circuit. On désigne par Single Event Gate Rupture (SEGR) (Événement Unique de Rupture de Porte) un évènement provoqué par un ion isolé dans un Metal Oxide Semiconductor

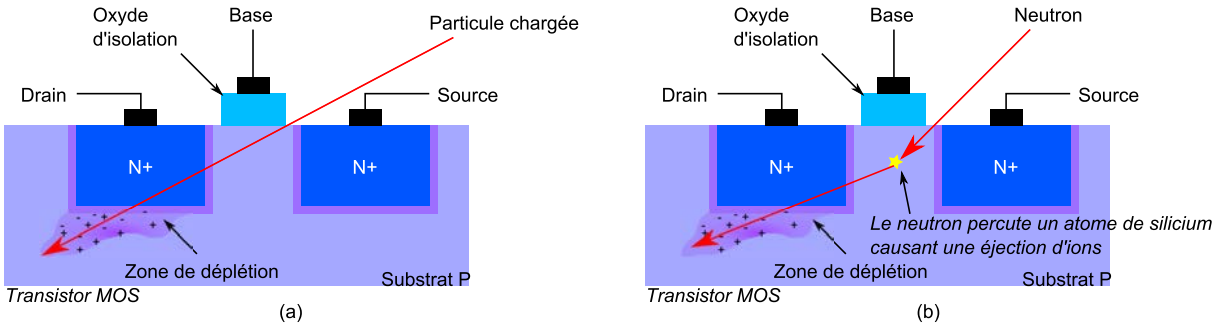


FIGURE 2.2 – Illustration des effets des radiations sur un transistor MOS : (a) impact d'une particule alpha, (b) impact d'un neutron [Mic11].

Field Effect Transistor (MOSFET) dont la puissance crée un chemin conducteur à travers l'oxyde d'une grille [Ami11]. Enfin, un fort courant pouvant causer la destruction d'un circuit dans un transistor de puissance est appelé Single Event Burnout (SEB) (Événement Unique de Surtension). Par la suite, ces erreurs sont définies par le terme générique d'erreurs permanentes.

2.1.4 Impact des erreurs sur un NoC

Des études ont été menées sur des architectures NoC afin d'analyser l'impact des erreurs sur cet élément central constituant les MPSoC auto-organisés. Ainsi, certains travaux ont analysé l'impact des SEU et des phénomènes d'interférences (*crosstalk*) sur des du routeurs [FCCK06]. Les phénomènes d'interférences apparaissent sur les bus de données physiquement proches sur une même puce dont les différentes valeurs des signaux peuvent se parasiter et générer des inversions de valeurs ou des délais lors des transitions de niveaux logiques.

Pour la procédure de simulation, les SEU ont été générées par des inversions de bits (*bit flips*) dans les logiques séquentielles du routeur. La position et la fréquence de ces SEU sont pseudo-aléatoires. Concernant les phénomènes d'interférences, ils ont été simulées comme parasites (*glitch*) et retardement des signaux. Le routeur test a été paramétré avec des *buffers* pouvant accueillir 4 *flit* et des bus de données de taille 8 bits.

Un total de 858 fautes ont été injectés dans une simulation où 78000 paquets de données ont été échangés. Le tableau 2.1 montre le résultat de la campagne d'injection des fautes au sein du routeur test. Ce tableau montre la répartition des erreurs par rapport aux différents éléments constituant le routeur ainsi que l'effet de la faute. Ces résultats montrent plusieurs effets générés par les fautes injectés. La perte de paquet survient lorsqu'un paquets de donnée est bloqué ou est écrasé par un autre paquet arrivant dans un *buffer*. L'erreur des données correspond à une modification du contenu d'un paquet de données. Les erreurs de routage correspondent à des paquets de données qui ne respectent pas un acheminement de routage théorique. Les erreurs de transmission correspondent

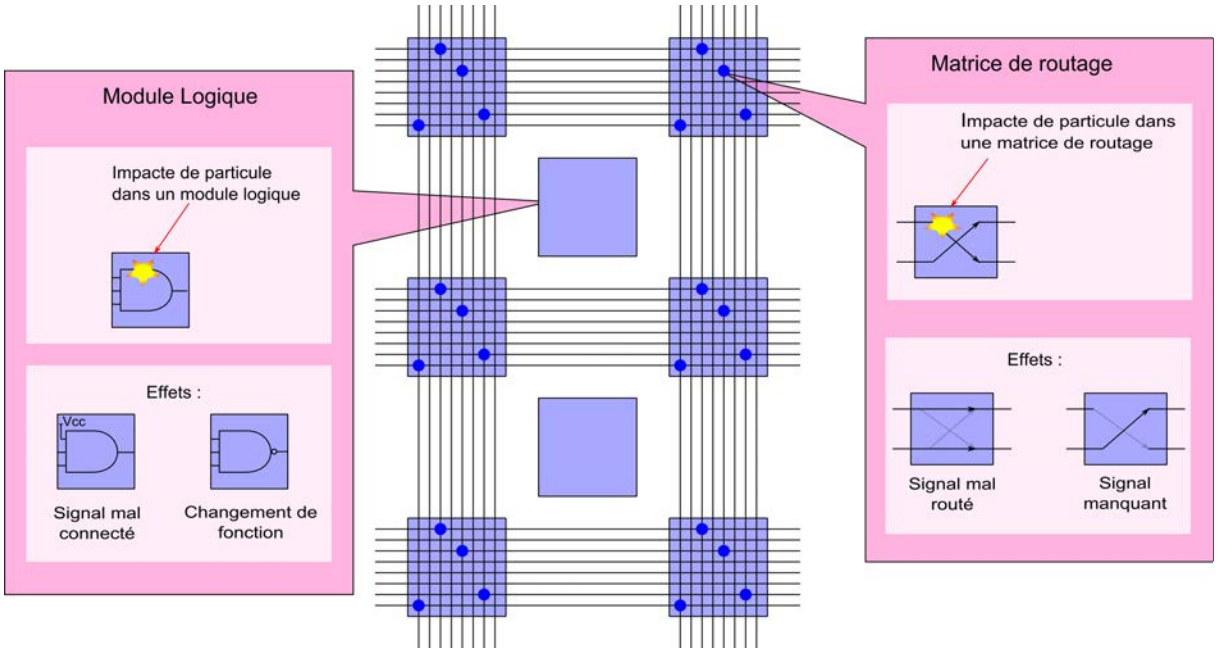


FIGURE 2.3 – Impact d'une particule dans un FPGA de technologie SRAM [Mic11].

à un non-respect des règles de transmission entre les routeurs. Enfin, la défaillance d'un routeur correspond à un arrêt total du fonctionnement du routeur et nécessite d'être redémarré pour fonctionner à nouveau. A partir de ces résultat, on peut noter que pour une durée de simulation de 700 ms, 6.58% des SEU ont généré des erreurs de routage, 9,47% des défaillances du routeur et 3,42% des erreurs sur le contenu des paquets de données.

Une étude similaire a été réalisée pour un routeur synchrone pouvant effectuer la transmission d'un *flit* en un cycle d'horloge [EYPZ09]. Plus précisément, ce routeur synchrone est composé de trois blocs : un *buffer* d'entrée, une logique de routage et un composant d'aiguillage. Le *buffer* et l'aiguillage sont pilotés par une horloge à front montant tandis que la logique de routage fonctionne sur fronts descendants. Les SEU sont injectés dans ces trois composants par l'application de différents modèles de fautes (SEU, erreurs permanentes (*stuck-at*), interférences (*crosstalk*)). Le tableau 2.2 montre les résultats des simulations d'injection de fautes. On constate que le taux de défaillance le plus élevé est obtenu pour la logique de routage. Cela est dû à son rôle de décision centrale au sein du routeur. Ainsi, la logique de routage est considérée comme l'élément le plus sensible du routeur. De plus, on constate que 48% des fautes injectées causent une défaillance du système et que 51% des erreurs ont été inhibées. Il reste donc 1% d'erreurs latentes ou dormantes.

2.1.5 Techniques de base de la tolérance aux fautes

La tolérance aux fautes est la capacité d'un système à corriger son fonctionnement lors de la présence de fautes [Ami11]. D'une manière idéale, un système tolérant aux fautes

TABLE 2.1 – Résultats de l'injection des fautes dans un routeur [FCCK06].

Type de faute	Position de la faute	Nb. de faute	Effet de la faute					
			Paquet manquant	Erreur de routage	Erreur de données	Erreur de transmission	Défaillance du routeur	Pas d'effet
SEU	Registre d'arbitrage de priorité	240	0,4%	0%	0%	0%	0,4%	99,2%
	Registre d'arbitrage de la *FSM*	120	2,5%	38,3%	0%	0%	40,8%	18,4%
	buffer d'entrée	280	0%	1,4%	9,3%	0%	5%	84,3%
	Registre de la *FSM* du port d'entrée	120	0,8%	0%	0%	14,2%	6,7%	78,3%
	Moyenne des effets sur 700ms de simulation		0,66%	6,58%	3,42%	2,24%	9,47%	77,1%
Interférences	Liens entre les routeurs	98	0%	6,1%	83,7%	0%	10,2%	0%

TABLE 2.2 – Résultats de l'injection de fautes dans un routeur synchrone [EYPZ09].

Position	Fautes annulées		Erreurs latentes		Défaillances	
	Nb.	%	Nb.	%	Nb.	%
buffer d'entrée	2015	69,76	5	1,14	845	29
Logique de routage	760	31,17	5	0,83	1665	68
Aiguillage	168	35,44	2	0,56	306	64
Total	2943	51	12	0,2	2816	48,8

FIGURE 2.4 – Redondance matérielle d'une fonction : (a) détection d'erreurs par duplication et comparateur, (b) masquage d'erreurs par triplication et voteur [Ami11].

fonctionne indépendamment de la présence d'erreurs [Ami11]. Dans des cas concrets de mise en œuvre de systèmes électroniques, il n'est pas possible de garantir à 100% le fonctionnement du système en présence d'erreurs. C'est pourquoi, les systèmes sont conçus afin de résister aux fautes les plus répandues pouvant survenir lors de leurs utilisations. Pour ce faire, diverses techniques de SdF peuvent être mise en œuvre. Le principe de base est la redondance, et est commun à toutes les techniques de SdF où l'ajout de ressources matérielles, temporelles ou informationnelles permet de faire face aux erreurs. L'objectif des systèmes tolérants aux fautes est donc de détecter et de localiser des erreurs et/ou corriger les erreurs afin de ne pas les propager dans le système pour éviter une défaillance.

a. Redondance matérielle

La redondance matérielle consiste à dupliquer la structure logique d'un circuit afin d'effectuer parallèlement plusieurs fois la même fonction [Ami11]. La figure 2.4-a illustre une méthode de détection d'erreurs basée sur la duplication de la structure d'une fonction et d'un comparateur. Si le résultat de l'opération des deux fonctions est différent, alors une erreur est détectée. Afin de pouvoir bénéficier d'une capacité de tolérance aux fautes, un masquage des erreurs survenant dans une fonction est possible. Pour ce faire, la triplication de la fonction est effectuée (voir figure 2.4-b). Ce type de technique permet de masquer une erreur survenant dans un des trois blocs. En effet, la présence d'un système de vote permet d'obtenir en sortie de la triplication le résultat obtenu majoritairement sur les trois blocs fonctions. Cette technique de redondance matérielle a un impact fort sur la surface logique nécessaire pour implanter le système sur une puce en augmentant les ressources par un facteur 2 pour la duplication, et par un facteur 3 à 3,5 pour la triplication [Ami11]. Cependant, elle reste une solution efficace pour protéger les éléments les plus sensibles d'une architecture embarquée.

b. Redondance temporelle

La redondance temporelle consiste à répéter consécutivement plusieurs fois un même traitement d'une fonction afin d'en comparer le résultat [Ami11]. Si une différence existe entre les deux résultats, alors une erreur est détectée. Comme illustré en figure 2.5, ce type de redondance nécessite un *buffer* mémorisant le résultat de la première opération afin de pouvoir la comparer et détecter une erreur. Ainsi, la redondance temporelle nécessite un coût logique faible, mais augmente la latence générale du système en doublant le nombre

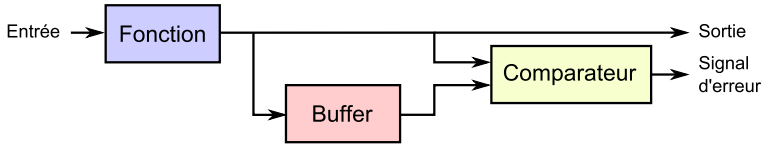


FIGURE 2.5 – Redondance temporelle d'une fonction [Ami11].

FIGURE 2.6 – Principe de la redondance d'information [Ami11].

d'opérations nécessaires pour la fonction à protéger par cette méthode. La puissance électrique consommée est également doublée. De plus, ce type de redondance ne peut pas détecter les erreurs permanentes. Pour corriger les erreurs, la fonction est répétée trois fois à trois intervalles différentes. Chaque résultat est stocké dans un *buffer* et un vote permet de définir par la majorité le résultat de l'opération. De manière similaire à la triplication matérielle, cette solution permet de masquer une erreur (à condition qu'il n'y a pas d'erreurs permanentes). Cette solution nécessite beaucoup moins de ressources logiques que la triplication matérielle mais diminue fortement les performances en termes de latence du système.

c. Redondance d'information

La redondance d'information consiste à ajouter des informations aux données transmises sur un canal au sein d'un réseau, ou stockées dans une mémoire [Ami11]. Ces données supplémentaires sont issues d'un codage mathématique réutilisé pour décoder les informations originales. La figure 2.6 illustre l'utilisation de la redondance d'information avec les deux principaux blocs nécessaires. L'encodeur définit les valeurs des informations supplémentaires à intégrer aux données, le décodeur vérifie l'exactitude des données. Parmi les techniques de redondance d'information, des codes permettent de détecter des erreurs comme les bits de parité ou le codage Cyclic Redundancy Check (CRC) (Contrôle de Redondance Cyclique). A l'inverse, les CCE permettent de corriger certaines erreurs par rapport à leur capacité de correction. Nous pouvons citer par exemple le codage de Hamming qui peut corriger une erreur d'un bit.

2.2 La SdF appliquée aux réseaux sur puce

2.2.1 Introduction

Pour rendre SdF les NoC, de nombreuses solutions sont proposées dans la littérature. Les différentes fautes que peut subir un NoC vont générer dans 48% des cas une défaillance générale du système, dans 51% des cas l'erreur sera inhibée avant de causer une défaillance, et 1% des erreurs vont être latentes [EYPZ09]. Les mécanismes de SdF se divisent en deux catégories : les solutions dites en ligne (*online*) et celles dites hors ligne (*offline*).

Les solutions hors ligne sont réalisées lors de la mise sous tension du système, ou lors d'événements spéciaux définis par le concepteur (par exemple exécutions périodique). Lors de l'exécution de ces tests hors ligne, aucune communication entre les éléments constituant le réseau peut être effectuée. En effet,dans cette phase de test du réseau, le système va s'auto-injecter des vecteurs de test afin d'analyser chacun de ses éléments pour détecter des parties fautives. La localisation des éléments fautifs peut donc être très précise, car chaque module constituant le NoC peut être testé indépendamment du réseau. L'inconvénient majeur des techniques de détections hors ligne est que seul les fautes permanentes peuvent être détectées. Toute faute survenant durant le fonctionnement du réseau ne sera pas localisée. De plus, il faut judicieusement choisir à quels moments effectuer les tests, car ils nécessitent un temps plus ou moins long pendant lequel aucune communication n'est possible et diminuent ainsi le temps de réponse du système.

La seconde catégorie de techniques de détection d'erreurs dans les NoC est appelée détection en ligne. Ce type de détection est réalisé en cours de fonctionnement du NoC, c'est à dire pendant que les éléments du NoC communiquent entre eux. Plus précisément, les paquets de données transitant sur le réseau sont utilisés pour analyser l'intégrité du système. Le principal inconvénient de ce type de technique est que les capacités de détection et de localisation dépendent directement du trafic. Cependant, les solutions en ligne permettent de détecter les erreurs survenant pendant le fonctionnement du système.

2.2.2 Détections en ligne

Contrairement aux techniques de détection hors ligne, les techniques de détection en ligne s'effectuent pendant que les éléments constituant le réseau communiquent. Plus précisément, les modules de détection utilisent le trafic du réseau pour analyser les éléments constituant le NoC. Le traffic est analysé selon plusieurs approches.

a. Approches basées Codes Correcteurs d'Erreurs (CCE)

Dans la grande majorité des NoC garantissant une certaine SdF, des CCE sont intégrés au réseau [GIS+06, MTV+05, KASN08]. L'objectif de l'utilisation des CCE est de protéger les données transitant sur le réseau grâce à leurs capacités de correction des erreurs. En effet, la tolérance aux erreurs des paquets de données est directement liée au

CCE utilisé. Par exemple, un bit de parité peut uniquement détecter des erreurs impaires sur les bits de données alors qu'un code de Hamming peut corriger un bit fautif. De plus, le CCE aura un impact direct sur l'augmentation de la consommation électrique, de la latence pour router les paquets de données, et de la surface logique du NoC.

Le deuxième objectif est de localiser les éléments fautifs générant des erreurs incorrigibles afin de les isoler et permettre aux paquets de données de les contourner à l'aide d'algorithmes adaptatifs de routage. La capacité de localisation dépend du nombre et de la position des CCE employés. On distingue trois principales techniques qui sont les techniques bout-à-bout, routeur-à-routeur et code-disjoint. La figure 2.7 illustre ces principales techniques employées dans les NoC concernant l'utilisation des CCE.

- Technique bout-à-bout : La technique appelée bout-à-bout (*End-to-End*) est illustrée en figure 2.7-a [MTV+05]. Dans cette technique, les CCE sont implantés dans les ports d'entrée des éléments de calcul du réseau. Ainsi, le contenu du paquet est vérifié uniquement à sa réception par le destinataire. Il est donc impossible de localiser dans quel élément du réseau s'est produit la faute. De plus, lorsqu'un paquet est détecté comme fautif avec une erreur incorrigible, une retransmission doit être effectuée pouvant surcharger le trafic. Pour permettre la possibilité d'une localisation des erreurs, un journal des suspicions est combiné à la technique bout-à-bout à base de CCE de type CRC [KASN08]. Ce journal contient l'adresse réseau des nœuds susceptibles de contenir une source d'erreurs. Dans cette technique, lorsqu'un paquet est réceptionné par un élément de calcul du réseau avec une détection d'erreur, les routeurs ayant été traversés par le paquet de données sont étiquetés "suspect" dans un journal d'évènements. Les routeurs traversés sont déduit de l'adresse de la source émettrice du paquet, indiquée dans un *flit* d'en-tête, et grâce à l'utilisation d'un algorithme déterministe via lequel un paquet de données emprunte toujours le même chemin à travers le réseau entre deux mêmes IP. Inversement, lorsqu'un routeur réceptionne un paquet sans erreur, les routeurs traversés sont retirés de la liste de suspicion. Après plusieurs itérations, l'élément fautif peut être localisé dans le réseau. L'inconvénient majeur de cette technique est la durée nécessaire pour la localisation des erreurs qui peut être plus ou moins longue en fonction de la taille du réseau et du trafic dans le NoC. De plus, cette technique ne peut pas être appliquée dans un NoC dynamique utilisant un algorithme adaptatif de routage. En effet, le concept de cette technique est de suspecter les routeurs traversés en les déduisant des chemins de routage déterministe. Par conséquent, lorsqu'un élément fautif est localisé, il est impossible de le contourner.

Pour appliquer la technique bout-à-bout dans un réseau utilisant un algorithme de routage adaptatif, une analyse de la probabilité des chemins possibles utilisés par un paquet de données réceptionné doit être effectuée [SGC11]. En considérant que l'algorithme de routage favorise les chemins minimaux de routage, et que chaque chemin minimal à la même probabilité d'être emprunté, une localisation dans le NoC de l'élément fautif est possible. Cependant, pour un algorithme adaptatif ou déterministe, lorsqu'un élément de calcul réceptionne un paquet ayant une erreur incorrigible, la lecture de l'adresse de la source dans le *flit* d'en-tête n'est pas garantie sans erreurs. En effet, il se peut que l'erreur

se situe dans l'adresse de l'émetteur indiquée dans le *flit* d'en-tête. Par conséquent, la déduction du chemin par lequel le paquet de données a transité et permettant de localiser le routeur fautif n'est pas fiable. Un tel scénario n'est pas évoqué dans les travaux précédemment cités concernant les techniques bout-à-bout.

- Routeur-à-Routeur : Une autre solution couramment utilisée dans les NoC est la méthode routeur-à-routeur (*Switch-to-Switch*) [MTV+05]. Dans cette méthode, un CCE est implanté dans chaque port d'entrée des routeurs constituant le NoC, comme illustré en figure 2.7-b. Ainsi, lorsqu'un paquet de données traverse le réseau, l'intégralité de son contenu est contrôlée à chaque routeur traversé. Cette technique a donc la capacité de localiser directement le routeur où s'est produit l'erreur et permet ainsi d'isoler rapidement les éléments fautifs du réseau. Cependant, lorsqu'une erreur est détectée, le routeur entier est déconnecté du NoC bien que seul un des éléments du routeur peut être fautif. La méthode routeur-à-routeur est couramment associée à un contrôle de flux de type *Ack*/*Nack* [LLP07]. En effet, dans les transmissions utilisant un contrôle de flux *Ack*/*Nack*, lorsqu'un *flit* d'un paquet de données est transmis à un nœud, une réponse est attendue avant de transmettre le prochain *flit*. Avec la méthode routeur-à-routeur, le *flit* réceptionné par un nœud est d'abord analysé par un CCE avant de générer une réponse *Ack*/*Nack* à l'émetteur.

Une technique combine l'utilisation des méthodes routeur-à-routeur et bout-à-bout [GGWT11]. Cette technique combinée permet d'optimiser les ressources matérielles du système. Les *flit* d'en-tête sont contrôlés également dans chaque routeur traversé, comme préconisé par la méthode routeur-à-routeur. Les *flit* constituant les données utiles du paquets sont quant à eux contrôlés uniquement lors de la réception du paquet part le destinataire.

- Code-disjoint : La dernière technique de positionnement des CCE dans un réseau est appelée code-disjoint [GIS+06, GAP+07]. Dans cette technique, un CCE est placé dans chaque port d'entrée et de sortie des routeurs du réseau comme illustré en figure 2.7-c. Un paquet de données traversant le réseau est donc contrôlé à la fois en entrant et en sortant d'un routeur. Ainsi, lorsqu'un CCE situé dans un port d'entrée détecte une erreur dans un paquet, l'erreur s'est donc produite sur le bus de données entre deux routeurs. Lorsqu'un CCE situé dans un port de sortie détecte une erreur, la source de cette erreur est située dans le routeur. Cette technique permet une localisation précise. En effet, elle est capable de localiser une source d'erreur sur un bus de données ou au sein d'un routeur. Cependant, lorsqu'une erreur est détectée dans un routeur, ce dernier est entièrement désactivé malgré un fonctionnement partiel possible du routeur. Cette technique conduit donc à une exploitation non optimale des ressources disponibles conduisant à une augmentation de la latence au sein du réseau et une diminution de la bande passante. De plus, le code correcteur employé est le bit de parité [GIS+06]. Il n'y a donc aucune possibilité de correction si l'erreur est détectée dans le port de sortie. Puisque cette technique ne prévoit pas de mécanisme de retransmission interne. Il en résulte que chaque paquet erroné sera systématiquement perdu.

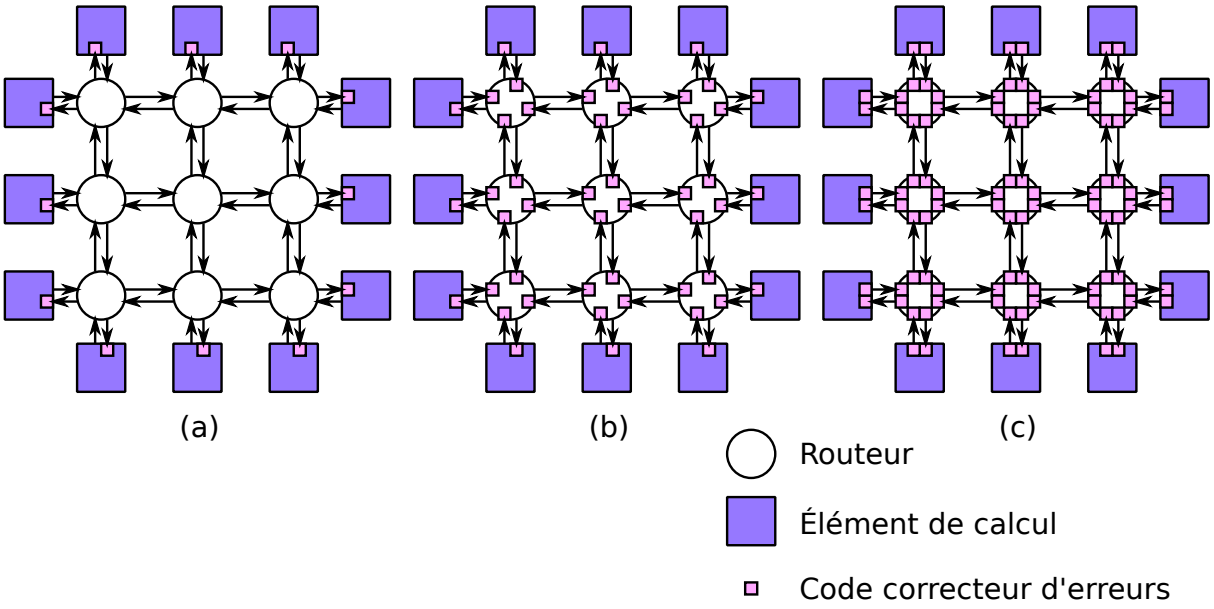


FIGURE 2.7 – Techniques d'utilisation des codes correcteurs d'erreurs dans un NoC maillé 2D : (a) bout-à-bout, (b) routeur-à-routeur, (c) code-disjoint.

b. Approche basée optimisation des ressources logiques

Comme l'utilisation de CCE au sein des routeurs implique nécessairement une augmentation des ressources logiques, des solutions proposent de limiter leurs impacts. Une solution est de combiner les CCE avec des algorithmes de routage basés sur la valeur du bit de parité [BK09]. Plus précisément, si le bit de parité vaut '1' le routage utilisé est l'algorithme XY. Dans le cas contraire, si le bit de parité vaut '0', l'algorithme de routage utilisé est le YX. Il s'agit d'une solution simple à mettre en œuvre. Cependant, elle n'est pas compatible avec l'utilisation d'algorithmes adaptatifs dans lesquels des zones de contournement locales peuvent utiliser des règles de routage spécifiques et indépendantes de l'algorithme XY ou YX. Cette solution d'optimisation n'est donc pas utilisable dans les NoC dynamiques.

c. Approche pour la distinction des erreurs permanentes et transitoires

Parmi les trois techniques présentées précédemment, aucune ne propose de solution pouvant distinguer si la source des erreurs est permanente ou transitoire. Ainsi, le moindre SEE occasionne la désactivation globale d'un routeur ou d'un port. Une solution permettant de distinguer si les sources d'erreurs sont permanentes ou transitoires est la sauvegarde des résultats des tests CCE [LLP07]. Cette solution utilise une technique de CCE basée routeur-à-routeur utilisant un contrôle de flux *Ack*/*Nack*. Les bus de données de 64 bits sont groupés en 4 mots de taille 16 bits. Chaque mot de 16 bits est encodé avec un CCE de Hamming. Le système est donc capable de détecter un MBU affectant jusqu'à 8 bits adjacents, et de 2 à 8 SEU si seulement deux d'entre elles affectent le même groupe de 16 bits. De plus, ces 4 parties sont entrelacées afin de répartir les effets d'un MBU.

	Sans SdF	*Ack/Nack*	*Spare*	*Split*
Latence (buffers vides) [ns]	2,61	7,71	8,48	9,65
Latence (buffers pleins) [ns]	4,93	14,98	17,82	20,31
Bande passante [Mmots/s]	492,6	204,5	180,5	152,7
Surface [(um)2]	6675	12386	25377	21602

TABLE 2.3 – Comparaison entre diverses techniques de sûreté de fonctionnement basées sur l'utilisation de CCE de routeur-à-routeur [LLP07].

Il y a principalement 2 techniques pour distinguer les sources d'erreurs permanentes ou transitoires. La première appelée *spare* (réserve) ajoute quatre bits de rechange par bus de données (un bit supplémentaire par mot de 16 bits). Lorsqu'un routeur réceptionne un *flit*, il analyse son contenu et stock le résultat du contrôle (syndrome de Hamming) dans un journal local associé au port d'entrée intégrant le CCE. Ce journal local peut stocker jusqu'à 3 syndromes. Si trois syndromes successifs non-nuls (donc *flit* avec une erreur) sont identiques, alors le routeur déduit une faute permanente située sur le bit correspondant au syndrome de Hamming. Si un mot de 16 bits d'un bus de données a un bit fautif de manière permanente, le routeur déduit l'existence d'un défaut physique d'un fil du bus et peut se reconfigurer afin de remplacer le fil défaillant par un fil de rechange. Pour ce faire, le routeur qui a détecté l'erreur permanente doit indiquer à son voisin ce changement de configuration. Cette opération s'effectue par l'échange d'un message de reconfiguration véhiculé à l'aide des signaux *Ack/Nack*.

La deuxième technique proposée dans ces travaux est appelée *split* (divisé) [LLP07]. Cette méthode divise chaque *flit* en deux parties. Chaque partie de *flit* est envoyée simultanément en double sur le bus de données. La transmission du *flit* est considérée comme réussie si une des deux parties d'une même transmission de la moitié d'un *flit* n'a pas de faute incorrigible. Le tableau 2.3 propose une comparaison de la latence, de la bande passante et de la surface du circuit pour ces différentes techniques [LLP07]. Les architectures *spare* et *split* sont notamment comparées à une architecture sans SdF et une architecture uniquement basé sur le contrôle de flux *Ack/Nack*. On constate très clairement que par rapport à une architecture sans SdF, l'architecture basée sur le contrôle de flux *Ack/Nack* nécessite le double de surface et diminue de plus de moitié la bande passante tout en triplant la latence des paquets de données. On constate également que l'architecture *spare* est celle nécessitant le plus de surface logique. En effet, chaque port d'un routeur dispose de 4 bits supplémentaires et 4 journaux pouvant stocker 3 syndromes de Hamming. L'architecture *split* quant à elle requiert moins de surface mais diminue de 68% la bande passante du circuit. Cependant, cette technique permet de distinguer les erreurs permanentes et transitoires, et permet une localisation précise des bits erronées ainsi que leurs remplacements pour maintenir les performances du réseau.

d. Approche de détection d'erreurs de routage

Bien que les logiques de routage sont des éléments les plus sensibles et les plus critiques des routeurs (voir section 2.1.4) [FCCK06, EYPZ09], peu de travaux dans la littérature proposent des solutions de détection en ligne permettant de localiser des blocs fautifs. Une des solutions principales consiste, à chaque réception d'un nouveau paquet de données dans un routeur, à contrôler si le routeur précédent a respecté le chemin de routage [KASN08]. Pour ce faire, la solution proposée a été implantée dans un NoC utilisant l'algorithme de routage déterministe XY. Ainsi, quand un routeur réceptionne un paquet de données, il effectue une comparaison des adresses (routeurs source et destination). Il est donc aisé de déterminer si la décision de routage effectué par le routeur précédent respecte ou non l'algorithme déterministe. En effet, un mouvement sur l'axe Y des adresses ne peut être effectuée que lorsqu'un routeur est dans la colonne de destination. Si cette règle n'est pas respectée, cela signifie que la logique de routage située dans le routeur précédent est fautive. Les principales insuffisances de ces travaux sont qu'ils ne présentent pas la manière dont est désactivé un bloc fautif, et sont uniquement applicables pour les algorithmes déterministes. Par conséquent, en cas de localisation d'une erreur, il n'est pas possible de désactiver le routeur fautif. En effet, avec l'algorithme XY, il est impossible de contourner des éléments fautifs au sein du réseau. De plus, cette technique ne peut être appliquée aux algorithmes de routage adaptatifs car les décisions de routage peuvent varier selon l'état (défaillant ou non) des routeurs constituant le réseau (contournements locaux au sein du NoC).

2.2.3 Détections hors ligne

Pour effectuer une détection hors ligne, aucune communication dans le NoC ne peut s'opérer. Des vecteurs de test sont injectés dans les routeurs du réseau et leurs réponses sont analysées afin de déterminer d'éventuels éléments fautifs [WWW06]. Ces méthodes permettent donc de tester plusieurs types de faute, comme les erreurs de routage et les erreurs dans le contenu des paquets de données. Deux méthodes sont utilisées.

La première méthode est appelée Design for Testability (DfT) et correspond à des NoC dont l'architecture dispose d'un ou plusieurs modules externes au réseau [WWW06, HBBN06, RUG07, RGU06]. Ces modules vont réaliser l'injection de vecteurs de test et l'analyse qui en découle pour localiser les éléments fautifs. Certains DfT nécessitent un seul module générant le vecteur de test connecté au réseau [HBBN06]. En effet, tous les routeurs du réseau sont identiques, un seul vecteur est transmis. Chaque routeur réceptionnant ce vecteur de test va l'appliquer localement et ensuite transmettre à ses voisins le vecteur de test ainsi que le résultat de son test local. Ainsi, les voisins vont comparer localement leurs résultats avec ceux des voisins. Si deux résultats ne concordent pas, un message d'erreur est renvoyé à un module principal connecté en bordure du réseau pour isoler les éléments fautifs. De manière opposée aux DfT propageant leurs vecteurs de test et afin de limiter les temps de propagation, une solution consiste à connecter à tous les routeurs du NoC, un module d'injection de vecteur et un module de comparaison des réponses.

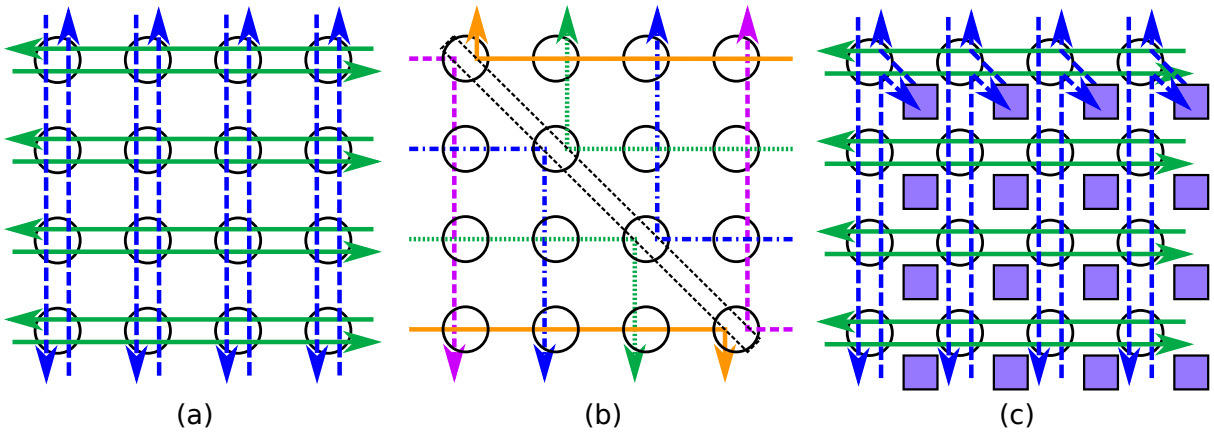


FIGURE 2.8 – Illustration d'une technique de localisation d'erreur hors ligne dans un NoC maillé 2 dimensions [RUG07] : (a) transmission des vecteurs de tests en ligne, (b) transmission des vecteurs en mode *XY*, (c) test des connexions aux éléments de calculs (ports locaux).

Une autre technique proposée dans la littérature est de disposer d'un NoC entièrement entouré par des modules d'injection de vecteurs de test [RUG07]. L'analyse se déroule en trois étapes. Dans la première phase de cette technique, les paquets de données sont envoyés à l'autre bout du réseau comme illustré en figure 2.8-a. Ainsi, si un module de test situé en bordure de réseau ne reçoit pas de paquets (où s'il reçoit des paquets ne correspondant pas aux paquets envoyés), un des routeurs de la ligne est considéré fautif. Dans la deuxième phase, les paquets de données sont envoyés selon l'algorithme de routage *XY* (voir figure 2.8-b). Tous les tournants des routeurs sont testés. Dans la dernière phase, illustrée par la figure 2.8-c, se sont les connexions aux éléments locaux de calcul qui sont testées (dans le cas de routeurs 5 ports). Le temps d'exécution de ces phases de test dépend directement de la taille du réseau. Le taux de couverture d'erreurs pour cette approche est très élevé.

La deuxième méthode de test hors ligne consiste à utiliser des auto-tests intégrés (Built-In Self-Test (BIST)). Ces derniers sont intégrés directement aux routeurs constituant le NoC. L'architecture de routeur présentée en [LSH+09] contient deux mécanismes assurant le test et l'isolation des parties fautives. Plus précisément, le mécanisme de test est composé de deux blocs intégrés dans chaque routeur. Le premier est un générateur de tests et le second est un bloc d'analyse des réponses. Il permet de tester les buffers et les multiplexeurs afin de localiser précisément ceux qui sont fautifs. Enfin, des circuits d'isolation des parties fautives permettent d'inhiber l'utilisation des éléments défaillants. Plus précisément, des blocs d'isolation des requêtes d'entrée et de sortie sont implantés dans chaque routeur comme illustré en figure 2.9 dans laquelle uniquement le port *Nord* est représenté. Ces modules d'isolation des erreurs masquent les signaux de requête des ports d'entrée ou de sortie fautifs. Ces signaux d'activation sont générés par une logique

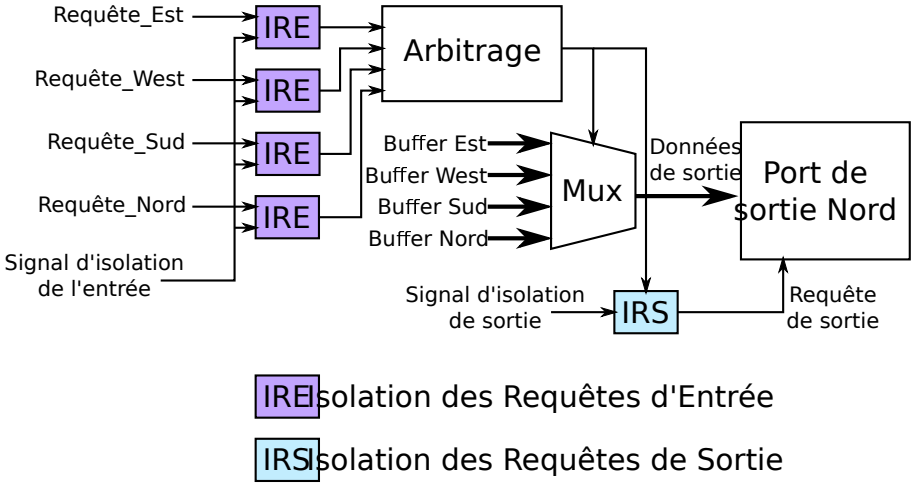


FIGURE 2.9 – Utilisation de module d'isolation des parties fautives par masquage des requêtes de transmission dans le port *Nord* d'un routeur [LSH+09].

de localisation des erreurs. Cette technique permet de détecter jusqu'à 97,8% des erreurs. Une autre technique BIST décompose le test en deux phases [PO07]. Dans une première phase, les routeurs ayant une adresse paire testent les bus de données tandis que les routeurs d'adresse impaire testent leurs logiques internes de contrôle. De plus, les routeurs impairs configurent leurs ports d'entrée en mode réflexion afin que les paquets réceptionnés soient renvoyés vers les routeurs pairs, dans le but de tester les deux bus connectant deux routeurs. Dans une deuxième phase les rôles sont inversés. Une comparaison entre ces différentes techniques DfT et BIST met en évidence des durées de test très différentes puisque allant de 117 à 405000 cycles d'horloge. Bien que la capacité de localisation des erreurs par ces techniques est élevée [LSH+09].

2.3 Détection hybride de fautes pour NoC dynamique

Cette approche se focalise sur la détection et la localisation d'erreurs de données permanentes. L'objectif de l'approche hybride proposée par [Kil12] est de combiner les avantages des techniques hors-ligne et en-ligne. Le tableau 2.4 montre la comparaison entre les techniques hors-ligne, en-ligne et hybride. Les techniques hors-ligne sont les plus efficaces pour localiser les sources d'erreurs permanentes. Cependant, il n'y a aucune solution pour les effectuer aux moments les plus opportuns, et sont généralement réalisées au démarrage du système ou de manière cyclique.

Les détections en-ligne sont quant à elles efficaces pour détecter des erreurs survenant pendant le fonctionnement. Cependant, pour garantir une certain précision dans la localisation des sources d'erreurs, il faut augmenter considérablement le nombre de CCE dans un NoC. Cette augmentation du nombre de CCE accroit la surface d'implantation du système et la consommation dynamique engendrée par la multitude des tests subis par

un paquet de données traversant le réseau. Ces détections sont limitées en localisation. Effet, par exemple, même en possédant 8 CCE comme pour la technique de code-disjoint [GIS+06], la capacité de localisation des sources d'erreurs reste plus faible qu'avec les techniques hors-ligne. Par exemple, lors d'une erreur interne à un routeur, la technique de code-disjoint déconnecte tout le routeur, alors que la technique proposée par Lin et al. permet de localiser le multiplexeur ou le *buffer* fautif [LSH+09]. L'approche hybride combine la précision de localisation des solutions hors-ligne avec la capacité de détection des erreurs survenant pendant le fonctionnement du réseau. L'inconvénient principal des détections hors-ligne est ainsi compensé par l'utilisation conjointe d'une solution en-ligne afin de déterminer les moments les plus opportuns pour effectuer une procédure de localisation hors-ligne d'une partie du réseau.

Les concepts de base de cette approche sont présentés en considérant un réseau maillé 2 dimensions. La détection en-ligne est effectuée par l'utilisation de CCE stratégiquement implantés dans le réseau. Plus précisément, le NoC est découpé en plusieurs Zone Globalement Sûre (ZGS) dont la taille est définie lors de la conception du système. Dans chacune de ces ZGS, seul les routeurs situés en périphérie contiennent des CCE. Plus précisément, uniquement les ports d'un routeur connectés à une périphérie d'une ZGS disposent d'un CCE. Ainsi, un paquet de données est contrôlé uniquement lorsqu'il entre ou quitte une ZGS. Ce contrôle permet de détecter si une erreur est survenue durant la traversée d'une ZGS. La figure 2.11-a illustre le fonctionnement d'un NoC maillé 2D 9x9 décomposé en 9 ZGS de taille 3×3. Ce réseau permet la communication entre trois IP. Si un paquet de données contient une erreur incorrigible lorsqu'il est réceptionné dans une ZGS, l'erreur s'est produite dans la zone voisine. Inversement, lorsqu'une erreur incorrigible est détectée par le port de sortie d'un routeur de périphérie, l'erreur s'est produite dans la ZGS qu'il tente de quitter. Dans les deux cas, la ZGS où s'est produite l'erreur est définie comme zone fautive et ne peut plus être utilisée temporairement. Cette zone sera dès lors déconnectée du réseau et contournée à l'aide d'algorithme de routage adaptatif comme illustré en figure 2.11-b. Les paquets de données restant dans la zone fautive sont évacués, mais aucun nouveau paquet ne peut être accueilli. Comme la zone fautive ne réceptionne de paquet de données, alors une localisation précise des erreurs peut être effectuée via une détection hors-ligne comme illustré en figure 2.11-c.

Le principal avantage de cette technique hybride est qu'au cours de l'exécution de la procédure de test hors-ligne d'une ZGS, les éléments de traitement constituant le MPSoC continuent à communiquer à travers les zones du réseau disponibles. Lorsque le test hors-ligne est achevé, la ZGS en test est réactivée et l'élément fautif est localisé selon la capacité de la technique hors-ligne utilisée (voir figure 2.11-c). Cette technique permet donc de définir le moment le plus opportun pour effectuer un test hors-ligne d'une partie du réseau. En testant uniquement une partie du NoC, le système continue toujours à fonctionner malgré des performances moindres. De plus, l'utilisation d'une structure de réseau en ZGS permet une optimisation de la latence des paquets de données, une diminution de la surface logique requise pour implanter le mécanisme SdF dans le NoC, et une diminution de la consommation dynamique et statique du système par rapport à des solutions en-ligne (routeur-à-routeur, code-disjoint, etc.).

Type de détection	**Coût en surface**	**Capacité de localisation**	**Inconvénients**	**Avantages**
Hors ligne	- Faible	- Très précis	- Ne peut détecter les fautes survenant pendant le fonctionnement - Tests effectués à des moments inopportuns	Très précis
En ligne	- Dépend du nombre de blocs de détection utilisés	Dépend du nombre de blocs de détection utilisés	Dépend du trafic	Peut détecter les erreurs pendant le fonctionnement
Hybride	- Plus faible que les techniques en-ligne	- Très précis	- Complexité de mise en œuvre	- Peut détecter avec précision les erreurs pendant le fonctionnement - Détections hors-ligne effectuées à des moments opportuns

TABLE 2.4 – Comparaison des techniques de détection d'erreurs hors-ligne, en-ligne et hybride.

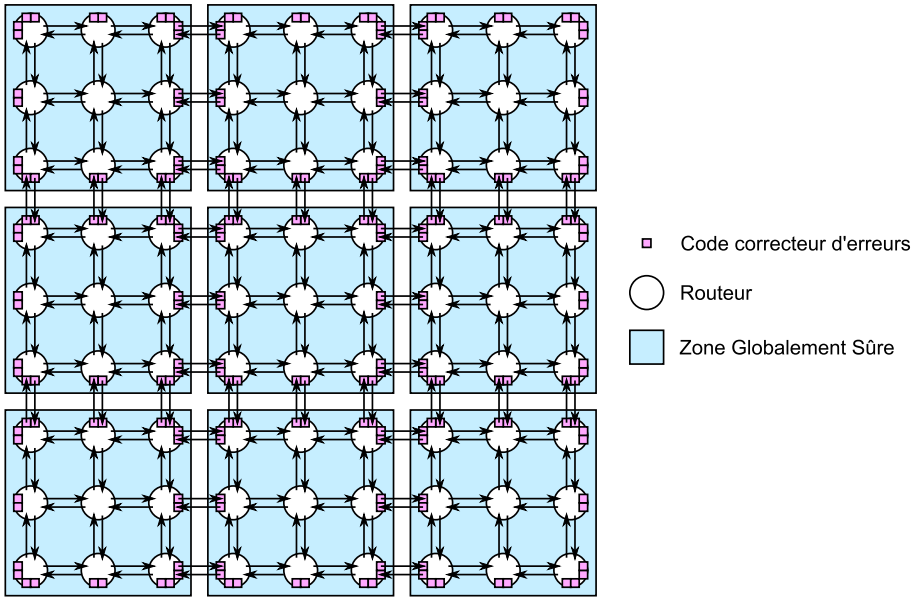


FIGURE 2.10 – Architecture de réseau embarqué sur puce sûr de fonctionnement selon l'approche en ZGS.

2.4 NoC RKT

Le NoC est une solution efficace pour effectuer la connexion de nombreux éléments de calcul au sein d'un MPSoC. En particulier, la mise en œuvre du concept d'auto-organisation défini dans le chapitre 1. La SdF est un élément clé garantissant le fonctionnement d'un système auto-organisé. Parmi les techniques proposées dans la littérature, celles effectuées hors-lignes ne garantissent pas une protection efficace du système. Plus précisément, lors du fonctionnement du MPSoC, toutes les nouvelles erreurs ne sont pas détectées, ou tardivement si les tests hors-ligne sont effectués cycliquement. Il n'y a donc pas de solution optimale permettant d'effectuer une procédure de tests hors-ligne au moment le plus opportun. Concernant les techniques en-ligne, la détection des erreurs de routage est peu traitée et sont principalement appliquées à des algorithmes de routage déterministes non compatibles avec des réseaux dynamiquement reconfigurables et des réseaux SdF dans lesquels des éléments doivent être isolés et/ou contournés.

La protection des paquets de données quant à elle est un sujet plus largement traité. L'utilisation de CCE est fréquent dans les NoC afin de protéger les messages transitant dans le réseau. Cependant, on constate une augmentation du nombre de CCE dans le réseau pour augmenter la capacité de localisation des sources d'erreurs. Cette précision garantie le maintient des caractéristiques intrinsèques du NoC tel que la latence des paquets de données, la bande passante du réseau, la consommation énergétique. En effet, si l'on emploie une technique de localisation des sources d'erreurs permanentes peu précise, des déconnexions d'éléments entiers vont être effectuées au sein du réseau. Par exemple, si un *buffer* est fautif dans un port d'entrée d'un routeur, une technique de détection d'er-

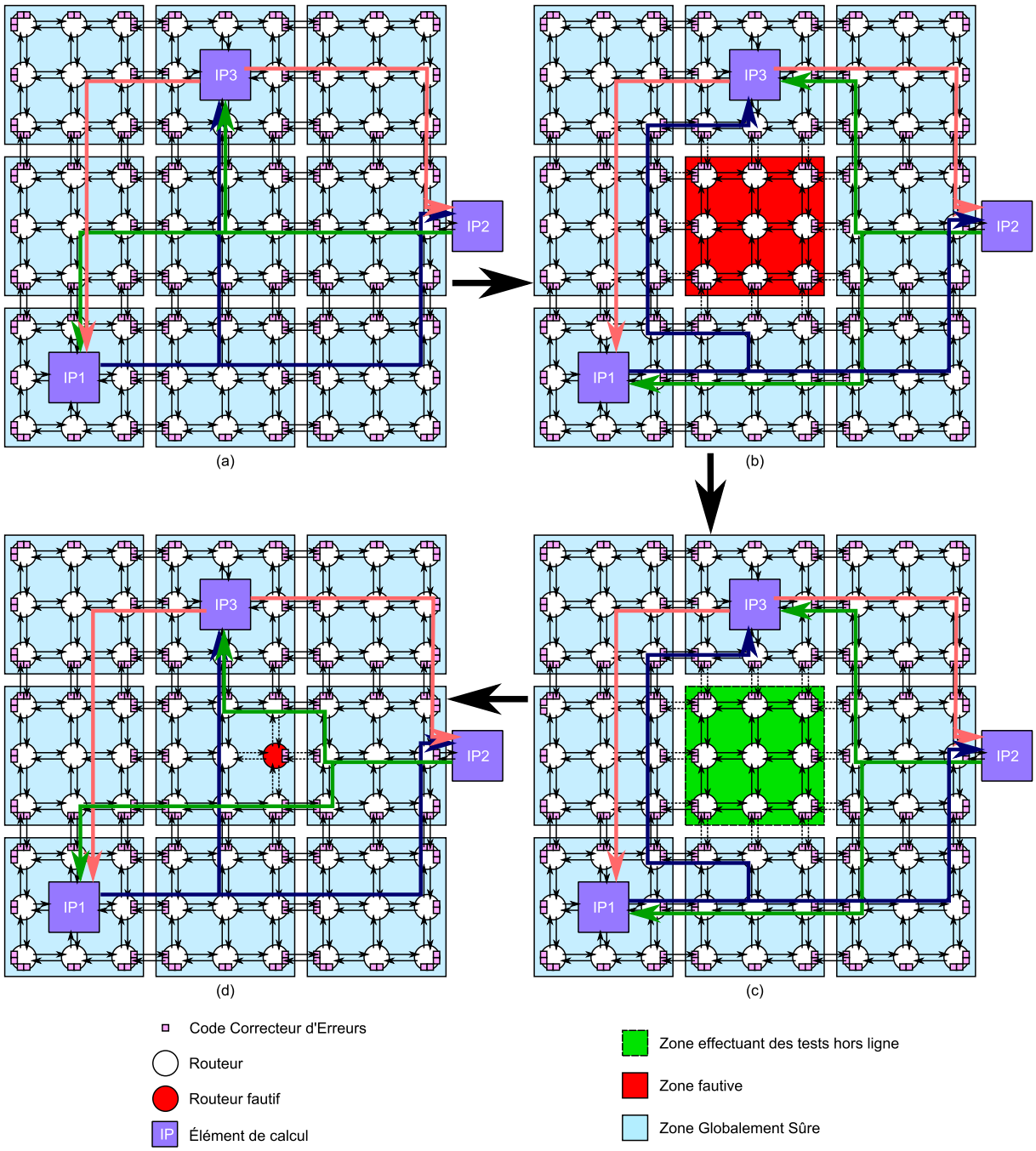


FIGURE 2.11 – Principes de fonctionnement d'un NoC basé ZGS : (a) fonctionnement normal, (b) déconnexion d'une zone fautive, (c) test hors-ligne d'une zone fautive, (d) réactivation d'une zone avec localisation précise de l'élément fautif.

reurs routeur-à-routeur va désactiver la totalité de ce routeur. Ainsi, au lieu de désactiver uniquement le port contenant l'erreur permanente, la totalité des ports seront condamnés. Cela conduit à une réduction du nombre de chemins disponibles dans le NoC pour router les paquets de données et créer des situations de congestion dues aux contournement des éléments défectueux. Pour augmenter la capacité de localisation, le nombre de CCE est augmenté, notamment via la technique de code-disjoint pouvant localiser si l'erreur est sur un bus de données ou dans un routeur. Cependant, cette technique basée sur le dédoublement du nombre de CCE comparée à une technique de détection routeur-à-routeur, n'est pas capable d'inhiber précisément un des éléments internes du routeur dans le cas d'une défaillance interne. En effet, si une erreur de données se situe dans un *buffer* d'entrée, le routeur sera déconnecté du réseau.

Dans ces travaux, nous considérons la structure NoC RKT qui est une architecture NoC de conception pour la détection en ligne des erreurs de routage pour des algorithmes adaptatifs, et des erreurs de paquets de données. Le NoC RKT est donc constitué de l'interconnexion de plusieurs routeurs RKT [Kil12]. L'architecture du routeur RKT est illustrée par la figure 2.12. C'est un routeur quatre directions (*Nord*, *Sud*, *Est* et *Ouest*) adapté pour des architectures de réseau maillé 2 dimensions. Chaque direction du routeur est composée de deux ports unidirectionnels : un port d'entrée et un de sortie. Un IP peut s'interconnecter directement à l'une des quatre directions.

Le NoC RKT repose sur une commutation par paquets de type Store and Forward (SaF). Bien que nécessitant des tailles de *buffers* plus importantes et une latence plus élevée pour router les paquets de données, cette solution est adaptée à une technique du rebouclage. Plus précisément, le routeur RKT repose sur l'intégration de module de rebouclage dans chaque direction du routeur afin de vider les *buffers* de sortie dans le cas d'un nœud voisin indisponible ou défaillant. Ces modules de rebouclage sont également utilisés avec des CCE afin de réaliser la détection et localisation très précise d'erreurs. Pour ce faire, le NoC RKT est basé sur un contrôle de flux *Ack*/*Nack* et l'intégration d'un CCE par port d'entrée. Un routeur RKT génère un *Nack* dès qu'un CCE détecte un *flit* erroné. Un *Ack* est généré uniquement à la réception totale d'un paquet de données. Cela permet d'envoyer les *flit* sans attendre un signal *Ack* entre chaque transmission de *flit*. De plus, cette commutation par paquets est idéale pour des réseaux adaptatifs, dans lesquels les routeurs constituant le réseau peuvent être amenés à être remplacés dynamiquement par un IP. En effet, avec une commutation SaF, un paquet de données est intégralement contenu dans un seul routeur. Ainsi, si un routeur doit être substitué, il suffit simplement de vider le contenu de ses *buffers*, contrairement à la technique *Wormhole* où un paquet de données est répartie dans plusieurs routeurs conduisant à un temps plus important pour vider l'ensemble de ces *buffers* réparties dans le réseau.

Le routeur RKT utilise une logique de routage par port d'entrée. De plus, ce routeur dispose de *buffers* en entrée et en sortie permettant d'obtenir une grande bande passante, de 4 logiques de routage et des blocs logiques de détections d'erreurs de routage. Ces blocs disposent chacun de trois journaux correspondant aux logiques de routage du routeur voisin connecté au port d'entrée et pouvant envoyer un paquet dans cette direction [Kil12].

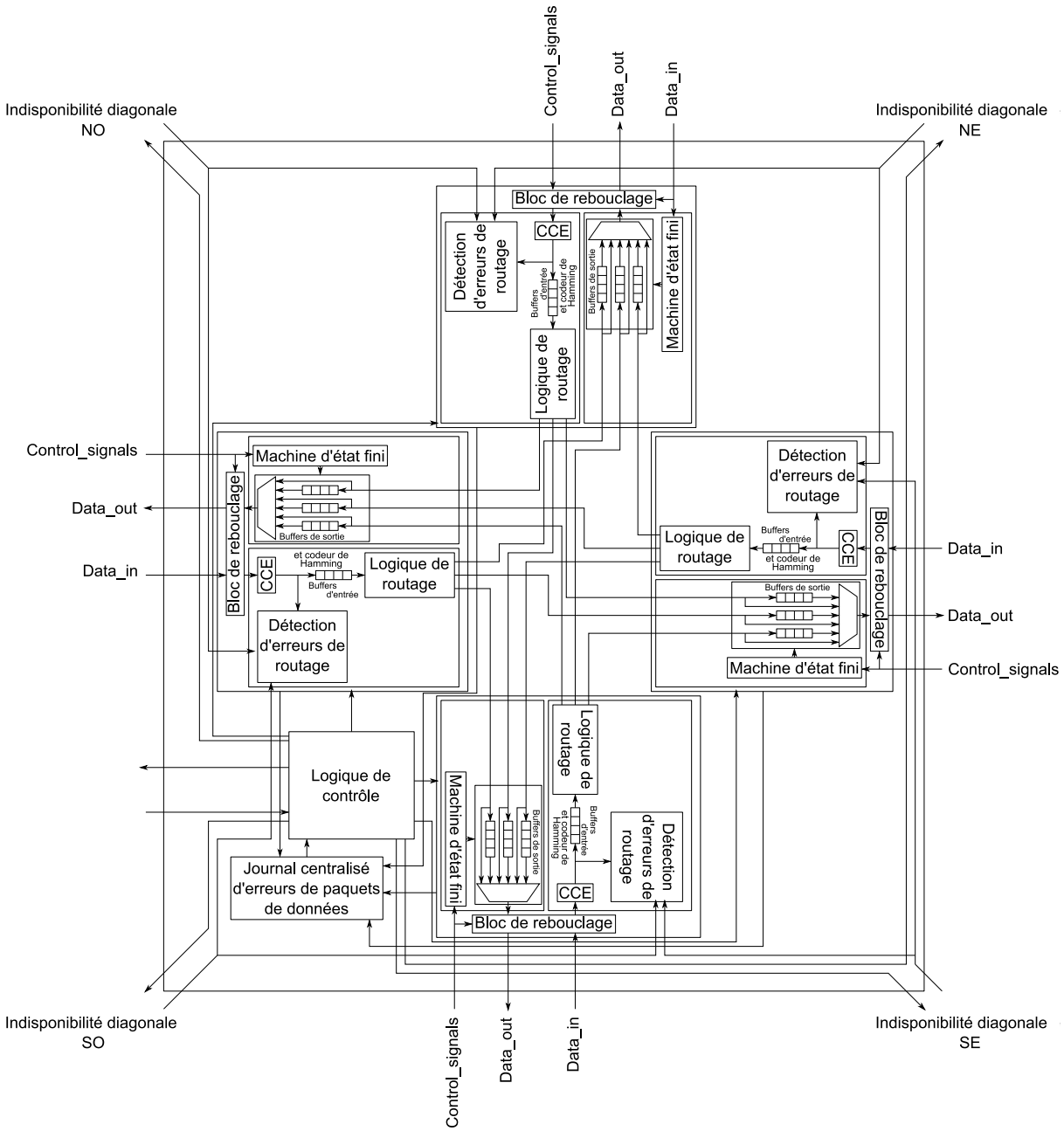


FIGURE 2.12 – Architecture du routeur tolérant aux fautes RKT [Kil12].

Pour satisfaire le concept de détection d'erreurs de routage pour les algorithmes adaptatifs, le routeur RKT signale son état (disponible ou indisponible) à ses 8 routeurs voisins directs. Ce signalement est réalisé aux quatre voisins directes et diagonaux via l'utilisation des connections spécifiques. Comme chaque port d'entrée peut être désactivé individuellement sans déconnecter entièrement le routeur. En effet, lorsqu'un routeur contrôle l'état de disponibilité d'un nœud situé sur une diagonale, il doit connaître précisément le nombre de ports désactivés (0, 1 ou 2 ports désactivés possibles). Le routeur RKT est un routeur de type *non-boucing*, donc un paquet de données ne peut pas être routé dans la même direction que celle de son arrivée [RGU06]. Ainsi, lorsqu'un routeur est entouré de trois voisins directs indisponibles, il devient également indisponible. Chaque port d'entrée intègre également une logique de détection d'erreurs de routage. Les tests de détection sont effectués en parallèle aux buffers d'entrée, ce qui permet de limiter l'augmentation de la latence des paquets de données.

Concernant le CCE employé, le code de Hamming fournit un bon compromis entre son coût en surface et sa capacité de détection et de correction d'erreurs. Plus précisément, chaque *flit* est codé avec un code Hamming et un bit de parité. Cette association permet de corriger une erreur d'un bit dans un *flit* et de détecter une double erreur. Cette capacité de correction d'une erreur d'un bit permet aux transmissions utilisant un contrôle de flux *Ack/Nack* de ne pas effectuer de retransmission pour chaque *flit* contenant une erreur d'un bit. En effet, si l'on considère un simple de bit de parité, chaque *flit* contenant une erreur engendrera une retransmission accroissant la latence des paquets de données dans le NoC.

En résumé, ce routeur RKT permet une localisation précise des éléments fautifs au sein des routeurs d'un NoC. En effet, si on considère la figure 2.13 en déconnectant uniquement le port fautif, le routeur(2,2) fonctionne partiellement en conservant trois ports d'entrée sur les quatre disponibles. Ainsi, seulement un paquet de données doit effectuer un contournement du port fautif limitant l'augmentation de la latence des paquets de données et conserve au maximum la bande passante du réseau.

2.5 Limitations et discutions

Une faute dans un module de détection d'erreurs peut avoir deux conséquences : la non-détection des erreurs, ou la génération de fausses détections. De plus, ces deux situations de génération d'erreurs peuvent être transitoires ou permanentes. L'approche adoptée se base sur l'utilisation de journaux et d'un seuillage pour définir si une erreur est permanente afin d'établir une déconnexion globale ou partielle de blocs de l'architecture. Cette stratégie conduit à ce que les erreurs transitoires soient ignorées.

Concernant les fautes générant une incapacité de localisation des erreurs survenant dans un CCE, notre réseau RKT considéré dans ces travaux est basé sur la subsidiarité de ses éléments. En effet, si un CCE d'erreurs ne détecte pas un paquet contenant une

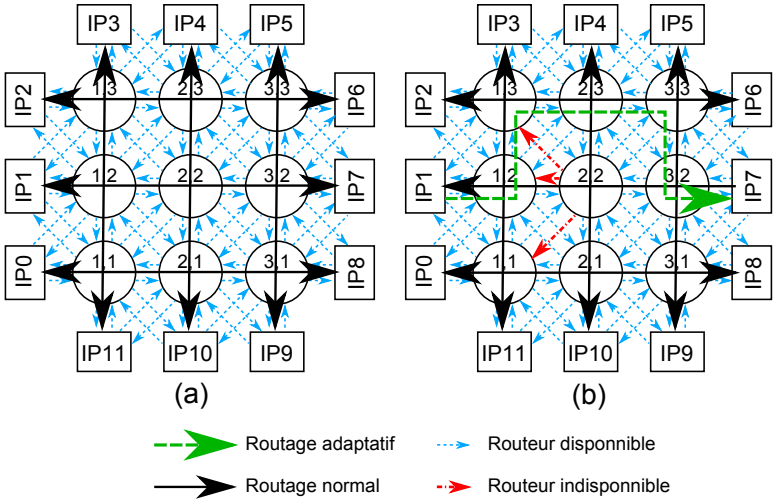


FIGURE 2.13 – Illustration de la simulation d'un NoC RKT de dimensions 3x3 :(a] échange normal entre les 12 IP, (b) erreur permanente dans un entrée d'un routeur.

erreur, elle sera détectée lors de la transmission du paquet de données erroné au nœud voisin. Ce dernier détectera l'erreur et générera des demandes de retransmission par l'envoi de signaux spécifiques (contrôle *Nack*). Dans ce cas, le paquet est donc rebouclé et le port contenant la faute générant les non-détections sera alors considéré comme fautif. Si une faute générant des non-détections d'erreurs est située dans un bloc logique de détection d'erreurs de routage, ou sur une Indication Diagonale de Disponibilité (IDD), le routeur ne pourra pas agir et les erreurs de routage seront non-détectées. Cependant, des solutions classiques de tolérance aux fautes peuvent être mise en œuvre pour pallier à ces problèmes. Comme par exemple la redondance matérielle en dupliquant des blocs/IDD) [Han05]. Concernant les fautes générant des fausses détections, nous avons intégré dans le NoC RKT une logique permettant de déconnecter très précisément les blocs fautifs. La déconnexion d'un bloc sain dans le réseau n'a pas un impact majeur sur les performances générales du NoC.

Un réseau NoC RKT permet de détecter des erreurs survenant en cours de fonctionnement du réseau en analysant les paquets de données transitant dans le NoC. Une analyse de la capacité du taux de détection et de localisation des erreurs a été réalisée à la fois pour les erreurs de données et de routage [Kil12]. Plusieurs comparaisons entre un NoC sans SdF et le NoC RKT ont également été réalisées afin de mesurer l'impact sur les performances d'un MPSoC basé NoC RKT. Ainsi, le NoC RKT permet de localiser les erreurs de routage au sein du réseau avec une efficacité proche de 96%. De plus, lors d'une détection d'une source d'erreurs permanentes, seul le port d'entrée contenant la logique de routage défaillante est déconnecté de manière précise du réseau. Ainsi, le routeur est partiellement déconnecté et continue à utiliser ses 3 ports d'entrées restant afin de limiter l'augmentation de la latence induite par le contournement adaptatif de l'élément défectueux.

L'architecture du routeur RKT permet également la protection des données transitant dans le réseau, et la localisation des sources d'erreurs des données. L'utilisation d'un module de rebouclage associé avec des CCE de Hamming permet de localiser précisément si la source d'erreurs est dans un port d'entrée, de sortie ou sur les bus de données. La précision obtenue présente des meilleurs performances comparée aux principales méthodes en ligne de la littérature. Les performances du routeur mettent en avant l'efficacité du NoC RKT. En effet, ce NoC repose sur un routeur intégrant deux techniques de SdF et possède une bande passante maximale de 41,72Gbit/s pour des bus de données de 64 bits avec une fréquence de fonctionnement de 441MHz.

Conclusion

La sûreté de fonctionnement est un élément clé des MPSoC. Étant donné l'évolution de la technologie et de la complexité croissante de ces systèmes, la protection des communications des NoC, élément central du MPSoC, est primordial. L'objectif dans les NoC est de localiser les éléments fautifs permanents afin de les isoler et ne pas propager ou générer de nouvelles erreurs. En effet, 48% des fautes subies par un NoC entraine une défaillance générale du système.

Au niveau de l'état de l'art, concernant les solutions proposées pour les NoC, deux familles de techniques existent. Il s'agit des solutions en ligne et hors ligne. Les solutions hors ligne offrent une bonne précision de localisation mais sont inefficaces pour détecter des erreurs survenant pendant le fonctionnement du réseau. Ces solutions testent le système lorsqu'aucune communication ne s'opère et offrent comme résultats des taux de détection et de de localisation très élevés. Ces solutions sont principalement effectuées au démarrage du système, les rendant inefficaces face aux erreurs survenant en cours de fonctionnement. Il est à noter qu'effectuer ces tests de manière périodique n'est pas une solution efficace car les erreurs peuvent aisément se propager au sein du réseau entre deux tests.

Les solutions en ligne s'opèrent au cours de fonctionnement du réseau et sont capables de détecter en permanence les erreurs. Actuellement, les solutions existantes ne permettent pas d'offrir un taux de détection et de localisation d'erreurs élevé comparé aux techniques hors ligne. En effet, de part leurs manques de précision, la localisation d'un élément fautif au sein d'un routeur entraine une déconnexion globale de ce dernier. Ainsi, même les éléments sains du routeur sont rendus inutilisables et des ressources matérielles du système sont perdues. Le fait de ne pas exploiter les routeurs partiellement défaillants en un mode "dégradé" conduit à une exploitation non optimale des ressources logiques, qui se traduit notamment par une augmentation de la latence des paquets de données induite par une diminution du nombre de chemin de routage au sein de l'architecture NoC.

Concernant l'utilisation des techniques de SdF, l'ajout dans le réseau de blocs assurant les tests des éléments constituants le NoC impacte directement sur le nombre de ressources logiques nécessaire à l'implantation du système. Actuellement, les solutions d'optimisation

proposées dans la littérature ne sont pas adaptables avec les NoC dynamiques, dont la nature des éléments du réseau peut changer en cours de fonctionnement par l'utilisation d'une reconfiguration dynamique et partielle ou bien par la déconnexion d'éléments fautifs.

En résumé, vu les résultats des campagnes d'injection de fautes dans l'architecture NoC RKT, on constate que les erreurs les plus fréquentes sont les erreurs de routage et les erreurs de paquets de données avec respectivement 6,58% et 3,42% d'occurrences pour une SEU injectée [FCCK06]. De plus, la campagne d'injection de fautes réalisée par Eghbal et al., montre que 68% des fautes présentes dans la logique de routage entraine une défaillance totale du routeur [EYPZ09]. Il en résulte donc que les détections des sources d'erreurs permettent de couvrir les erreurs les plus significatives.

Les NoC présentent une architecture flexible et adaptable pour réaliser efficacement le support de communication des MPSoC. De nombreuses architectures NoC sont proposées dans la littérature et présentent de bonnes performances pour des SoC nécessitant une bande passante élevée. Les solutions architecturales employées doivent être judicieusement choisies par les concepteurs afin de réaliser un réseau répondant aux besoins des applications à implanter. Le rôle du NoC au sein des SoC est critique. En effet, sa position centrale fait de lui l'élément clé du système. Une défaillance du réseau entraine une défaillance du système. C'est pourquoi, des mécanismes de Sûreté de Fonctionnement doivent être intégrés au sein des NoC afin de garantir l'intégrité du système.

La sûreté de fonctionnent des NoC reconfigurables nécessite de distinguer les fautes réels à la propriété d'adaptation de ces structures. Si les techniques en ligne permettent les détections en cours de fonctionnement, ces dernières ne permettent pas une couverture globale des fautes ou des défauts possibles. Réciproquement, les techniques hors ligne, permettent de mieux couvrirent l'ensemble des fautes ou erreurs mais ne répondent pas à une détection en cours de fonctionnement de ces fautes. Dans ce chapitre nous avons rappelé une approche combinée de détection d'erreurs à base de structures ZGS et pour laquelle nous avons montré la nécessité de développer des mécanismes de tolérance dans le cas d'un fonctionnement en réseau et à partir de l'exploitation du concept d'auto-organisation. C'est l'objet du chapitre suivant qui propose l'intégration de mécanismes nouveaux pour la SdF des NoC dynamiques dans un contexte d'auto-organisation en réseau.

Chapitre 3

Concept architectural auto-organisé pour la tolérance aux fautes de NoC en réseau

Introduction

Dans ce chapitre, nous présentons une approche conceptuelle basée contrôle décentralisée de mise en œuvre SdF des structures NoC d'un système auto-organisé en réseau. L'originalité de l'approche adoptée est d'exploiter l'auto-organisation comme mécanisme de mise en œuvre de la sûreté de fonctionnement du système en réseau. Plusieurs raisons ont orientés ce choix. Nous proposons un test hybride des réseaux sur puce RKT adaptatifs de type *2D-Mesh* intégrés dans un réseau MPSoC auto-organisé sans fil. Le test est considéré hybride car il combine des tests en ligne et hors ligne. Nous utilisons plusieurs codes correcteurs d'erreurs pour détecter les éventuels défauts dans les réseaux sur puce des nœuds auto-organisés constituant le système en réseau. Une fois qu'une erreur est détectée, la partie du NoC incriminée est déconnectée pour être testé par un test hors ligne. La particularité de ce test est qu'il est partiellement délocalisé et géré de manière décentralisée selon l'approche conceptuelle adoptée dans ces travaux. La génération de tests et les prises de décision sont assurées par un IP testeur spécifique qui est implémenté dans un nœud délocalisé. L'intérêt d'une stratégie d'une délocalisation de la génération de vecteurs de test est de réduire les probabilités d'erreurs dans le test lui-même. La génération de test est composé de vecteurs de test qui sont injectés dans le NoC à tester en utilisant un procédé de type *Scan Chain*. Bien que l'on conçoit une architecture complexe de routeurs sûr de fonctionnement, tel que le routeur RKT développé, testé et validé, rien ne garanti la fiabilité des blocs intégrés spécifiques à la mise en œuvre de la tolérance aux fautes des routeurs. Le mécanisme de test peut être adapté à tous types de routeurs et est adapté pour la délocalisation d'un générateur de tests à travers la structure réseau auto-organisé proposée. En effet, seul le vecteur est nécessaire pour le test, il peut donc être généré à partir de n'importe quel nœud distant. En outre, le test n'est pas fixe et peut être modifié, ce qui est un réel avantage dans le cas de structures MPSoC en réseau. D'une part, nous pouvons générer dynamiquement des différents vecteurs de test, adaptés au réseau sur puce testé. D'autre part, la structure ajoutée au NoC peut demeurer dans un fonctionnement statique.

Pour résumer, l'objectif est d'intégrer un mécanisme de test hors ligne, en coexistence avec la détection en ligne déjà présente. Grâce à cette combinaison, un routeur peut fonctionner selon deux modes. Un mode de *test*, où dans les données transmises, des spécificités sont injectées afin de tester séparément les différentes parties de l'architecture. Un mode *régulier* où les routeurs fonctionnent en ignorant toute logique de test supplémentaire nécessaire à la partie hors ligne. Ce chapitre, détaille le mécanisme délocalisé de tests basés sur le système auto-organisé proposé et permettant d'assurer la SdF des structures de communications sur puce mettant en œuvre les interactions mutuelles de l'ensemble des entités constituants le réseau de nœuds auto-organisés reconfigurables.

3.1 Nœuds Reconfigurables

La capacité de reconfiguration dynamique et partielle de certains FPGA permet une distribution intelligente des tâches au sein d'un SoC. En effet, en allouant plusieurs zones

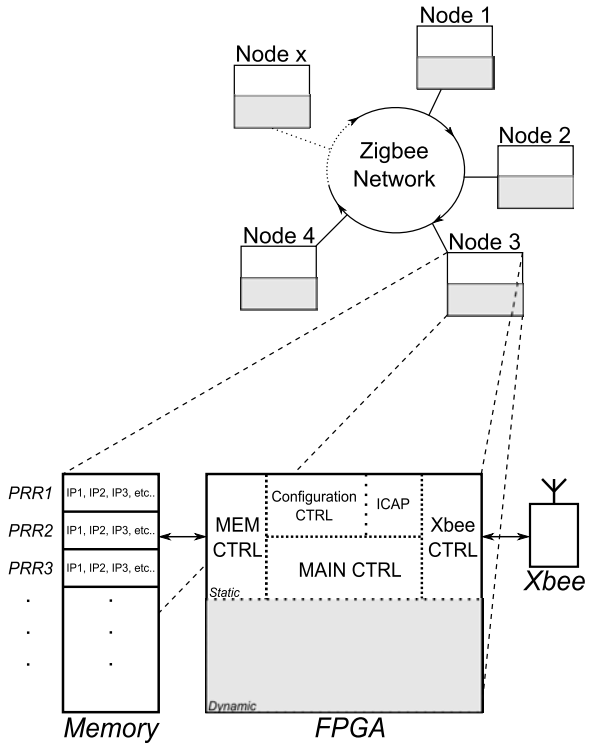


FIGURE 3.1 – Système auto-organisé multi-nœuds en réseau appliqué aux communications sans fil.

reconfigurables au sein d'un FPGA selon l'architecture de la figure 3.1, il est possible d'implémenter successivement un certain nombre d'IP en utilisant les fichiers de configuration (*bitstreams*) associés. De plus, dans le cas où le FPGA est localement défectueux, la logique initialement associée peut être réallouée dans une autre zone.

Pour se faire, l'architecture adoptée d'un nœud est composée de plusieurs modules statiques localisés dans une zone à configuration statique et d'une zone dynamiquement reconfigurable. La partie statique est principalement constituée de contrôleurs afin de gérer des ressources externes (mémoires, module de communication sans fil, etc.). La reconfiguration dynamique est également gérée à partir d'un module spécifique configuré dans dans cette région du FPGA en raison de son fonctionnement. En effet, un accès à l'Interface d'Accès au Port de Configuration (ICAP) [Xil08a] proposé par *Xilinx* est nécessaire. En outre, une gestion locale centralisée est mise en œuvre dans cette partie statique pour gérer le lien entre le réseau sans fil et la partie dynamique.

3.1.1 Composition d'un nœud

Un système auto-organisé et auto-reconfigurable implémentable sur plusieurs plateformes FPGA est proposé. Ces nœuds hétérogènes nécessitent cependant des ajustements en fonctions des technologies utilisées. D'une manière générale, nous retrouvons un certain nombre de périphériques présents sur chaque nœud du système communicant. L'élément

central pour chacun de ces nœuds est un FPGA, donc les caractéristiques nécessaires à l'implantation sont uniquement les ressources matérielles disponibles. En effet, l'architecture mise en place ne nécessite pas de minima en terme de rapidité d'exécution. De plus, la reconfiguration dynamique partielle n'est pas une exigence dans la mesure où seule une reconfiguration globale est exigée. La modification de la structure n'est pas possible dans ce cas mais la coopération avec les voisins reste active. Elle permet par exemple l'échange des fichiers de reconfiguration. La structuration en réseau des nœuds est illustrée par la figure 3.1, où la constitution d'un nœud est rappelée.

Afin de stocker les différents *bitstreams* des tâches à exécuter par le système, une mémoire externe est intégrée au nœud, pouvant être de technologie différente au même titre que le FPGA. Cette mémoire permet la gestion des fichiers de configuration mais également les mémorisations extérieures au FPGA des données intermédiaires. Le nombre d'emplacements mémoires à réserver au stockage des *bitstreams* dépend de la taille du système, du nombre de régions reconfigurables et du nombre de tâches. Plus de détails sur le stockage des fichiers sont donnés à la sous section 3.1.3.

Le système de communication utilisé est un réseau sans fils utilisant le protocole *Zigbee*. Le choix de la méthode de communication repose sur des modules *Xbee* proposés par *Maxstream* [Dig09], dont les détails sont donnés dans le chapitre 4.

3.1.2 Interface Configuration Acces Port

Lors de la conception d'un système reconfigurable partiellement, la frontière entre les parties statiques et dynamiques doit être définie. Pour la partie dynamique, la taille de la zone et la position de chaque région reconfigurable (Partial Reconfiguration (PR)) sont d'une importance capitale. Le développement d'un tel système a une influence directe sur la quantité de données nécessaires et par conséquent sur la taille des fichiers de reconfiguration (*bitstreams*). Les modules reconfigurables sont synthétisés spécialement pour une région reconfigurable spécifique. En conséquence, chaque module est synthétisé pour chaque région et pour chaque type de FPGA. Un ensemble de *bitstreams* partiels permet donc le déplacement d'un module dans toutes les régions reconfigurables, localement à un nœud ou bien de façon délocalisée dans un autre nœud du réseau communicant. Dans de tels systèmes, le nombre de *bitstreams* partiels peut augmenter rapidement. La taille d'un fichier de reconfiguration a un impact direct sur les performances du système. En outre, les ressources utilisées du FPGA, comme le nombre de cellules logiques (CLB, *Slices*) ou de mémoires internes embarquées dans le FPGA (Block Ram (BRAM)) sont directement liées à la mémoire de configuration. Ainsi, le temps nécessaire à la reconfiguration du FPGA dépend principalement de la taille du *bitstream*. Dans un système auto-organisé, l'ensemble de ces paramètres doivent être considérés lors d'implantation en Reconfiguration Dynamique (RD) aussi bien pour des blocs d'exécution matériels que logiciels.

La mise en œuvre de la reconfiguration partielle repose principalement sur la technologie reconfigurable SRAM du fabricant *XILINX*. La famille *FPGA Virtex* est sélectionnée

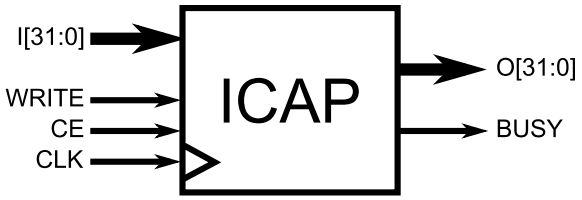


FIGURE 3.2 – Primitive de reconfiguration ICAP.

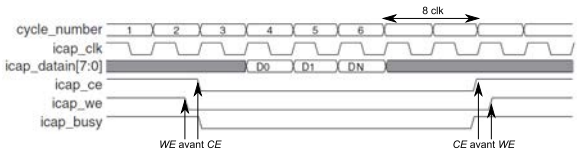


FIGURE 3.3 – Fonctionnement de la primitive de reconfiguration ICAP.

car elle propose des circuits parmi les plus performants tout en permettant la mise en œuvre de la reconfiguration dynamique partielle [Xil08b]. La technologie de reconfiguration partielle (PR) Xilinx est donc un élément clé de ce travail. Les concepts proposés et sa mise en œuvre en sont dérivés. Le bloc spécifique ICAP (Port d'Accès à la Configuration Interne) fournit par la *XILINX* pousse l'idée à la limite et permet au FPGA de se reconfigurer au cours d'exécutions [KPGdlT09].

A côté de cela, des outils de conception également fournis par la société *Xilinx*, nous permettent de jouer sur un grand nombre de paramètres et de possibilités. Ils sont constitués principalement des outils *ISE*, *ChipScope* et *PlanAhead* [Xil]. *ISE* est l'outil de base pour la conception des modules à partir d'un langage de description de matériel (*Hardware Description Language - HDL*). L'outil *ChipScope* est un analyseur de signal interne pour FPGA. *PlanAhead* est un outil "placement et routage" qui contribue également à la mise en œuvre des caractéristiques d'une reconfiguration partielle et à la génération des fichiers de reconfiguration associés. La primitive d'accès interne au port de configuration (ICAP) fournit à un utilisateur l'accès à la logique du FPGA, et ainsi d'accéder à la fois à des registres de configuration, à des données de configuration ou de reconfigurer partiellement le FPGA après que la configuration initiale soit terminée.

La figure 3.2 représente le schéma bloc de la primitive de reconfiguration ICAP. Son fonctionnement est assez basique. Il repose sur des bus de données d'entrées et de sorties de tailles 32 bits. Il est également possible de les utiliser uniquement sur 8 ou 16 bits. Nous retrouvons également une entrée *Chip Enable* (Chip Enable (CE)) qui va valider ou non la reconfiguration. Une entrée *WRITE* définit si le *bitstream* doit être écrit ou lu, car il est possible de récupérer les informations ou les données d'un *bitstream*. En sortie, nous avons un signal nommé *BUSY* qui informe de l'état de la primitive afin de connaître si elle est active ou non. Malgré que la primitive soit de type synchrone, le signal *WE* doit changer d'état avant le signal CE lors du début de la séquence de reconfiguration et inversement lors de l'arrêt d'une phase de reconfiguration comme l'illustre la figure 3.3.

3.1.3 Accessibilité aux fichiers de reconfiguration

L'un des éléments clés du système auto-organisé auto-reconfigurable adopté est l'accessibilité aux différents fichiers de reconfiguration permettant la délocalisation et l'organisation des tâches. Les périodes de réorganisation et l'efficacité de l'agencement des tâches dépendent en partie de la facilité et de la rapidité d'accès au bon *bitstream*. En effet, la nécessité physique de créer un fichier de reconfiguration par tâche et par emplacement, rend rapidement le nombre de ces derniers conséquent selon l'équation (3.1).

$$N_{bitstreams} = \sum_{n=1}^{N_{tches}} N_{emplacements}(n) \tag{3.1}$$

où $N_{bitstreams}$ est le nombre de *bitstream* total nécessaire au système. $N_{emplacements}(n)$ est le nombre d'emplacements disponibles ayant la capacité d'implémenter la tâche n. N_{tches} est le nombre de tâches du système.

Afin de limiter ce nombre de *bitstream* pouvant être important, le système auto-organisé dispose de plusieurs moyens de récupérer et stocker ces fichiers. Dans un premier temps, chaque nœud dispose d'une mémoire externe permettant une accessibilité et un stockage rapide de fichiers. L'utilisation de cette mémoire permet une reconfiguration rapide des *bitstreams* disponibles pour le nœud considéré mais avec un nombre limité de reconfigurations possibles lié à la capacité de stockage et à la taille des fichiers de reconfiguration. Les tâches ont toutes des rôles et donc des logiques différentes et conduisent à la génération des données de reconfiguration associées de tailles différentes. Il faut donc quantifier les tâches et les zones reconfigurables capables de les implémenter, mais également la quantité de données *bitstreams* à stocker. Cette limitation de stockage interne ne permet donc pas forcément l'accès à l'ensemble des fichiers de reconfiguration de l'ensemble des tâches exécutables dans l'ensemble des nœuds du système en réseau.

Dans le cas de la non-disponibilité locale d'un *bitstream*, le nœud concerné à la possibilité d'interagir par échanges de données avec les autres nœuds du réseau. Ainsi, si un *bitstream* n'est pas localement présent dans la mémoire, il peut être soit simplement récupéré et implémenté, soit récupéré, stocké (si de l'espace de stockage est disponible) et implémenté, ou alors récupéré pour être stocké en remplaçant dans la mémoire un autre *bistream* (si saturé) et implémenté. Grâce à cette flexibilité de circulation des fichiers de reconfiguration au sein du réseau entier, un nœud nécessitant sa reconfiguration partielle peut rapidement accéder aux données de configuration utiles. Cette rapidité peut être limitée par le système de communication utilisé. Par exemple, certains protocoles de transmission de données sans fil, tel que le protocole *Zigbee*, ont un débit de transfert relativement moyen. Bien que la réaction des nœuds afin d'échanger les données soit rapide, un transfert du fichier de reconfiguration en lui même est plus long et dépend de sa taille.

3.2 Gestion autonome des tâches

Dans un premier temps, il est important de rappeler que tous les nœuds constituants le réseau sont strictement identiques et intègrent dans un FPGA une structure architecturale permettant une auto-adaptation et une forte interaction des entités constitutifs du système selon l'architecture de la figure 1.1. Ainsi, en combinant l'auto-organisation du réseau avec la reconfiguration partielle du FPGA, nous réalisons un système auto-organisé en mesure de répartir les tâches parmi l'ensemble des nœuds du réseau (soit au sein même du nœud ou de façon délocalisée parmi l'ensemble des nœuds du réseau). En fonction du type d'échec, une tâche ne peut pas être déplacée dans une zone différente du même FPGA. Dans ce cas, le nœud défectueux effectue une demande aux autres nœuds pour distribuer les tâches de la zone non fonctionnelle. Pour ce faire, les fichiers de configuration des tâches incriminées et les données associées sont transférées à un autre nœud. Cependant, ce dernier doit respecter certaines règles avant de pouvoir accepter les fichiers :

- Ne pas être considéré comme un nœud fautif,
- Disposer de ressources logiques suffisantes afin d'accueillir l'implémentation de la nouvelle tâche,
- Avoir l'accessibilité au fichier de reconfiguration ou être capable de se le procurer à travers un autre nœud du réseau,
- Ne pas être occupé par des tâches prioritaires.

En résumé, le nœud doit évidement ne pas être lui même considéré comme fautif. Il doit disposer de ressources matérielles suffisantes (surface logique disponible) pour la configuration de l'IP. Il doit ensuite être en possession du fichier de configuration de l'IP à exécuter ou dans le cas échéant être capable de le recevoir ou de se le procurer auprès d'une mémoire de stockage associée à un autre nœud du réseau. La dernière règle établie est que ce nœud ne doit pas déjà être occupé par des tâches prioritaires comme par exemple assurer des tests d'un autre nœud. Une fois que toutes ces conditions sont réunies, le nœud dont l'IP est nouvellement affecté s'auto-reconfigure avec les fichiers de reconfiguration, et substitue le nœud défectueux dans le réseau. D'un point de vue global, le système maintient son fonctionnement, car aucun des traitements est stoppé sur une durée indéterminée.

3.2.1 Communications inter-nœuds

Le système auto-organisé développé repose également sur la façon dont les nœuds vont communiquer entre eux durant la période de fonctionnement, mais également lors des détections d'erreurs. Pour cela, nous avons mis en place un protocole de communication inter-nœuds adapté à la topologie du réseau de communication *Zigbee* utilisé. Lorsqu'une défaillance d'un nœud est identifiée et que le nœud concerné n'a pas la possibilité de réorganiser ses tâches afin de compenser la partie défaillante, l'IP implémenté dans sa PRR est délocalisé vers un autre nœud du réseau. La figure 3.4 illustre les principes de délocalisation de tâches et d'implémentation d'un IP test distant dans le cadre d'un test à distance d'un nœud considéré comme fautif. Pour se faire, une trame contenant un

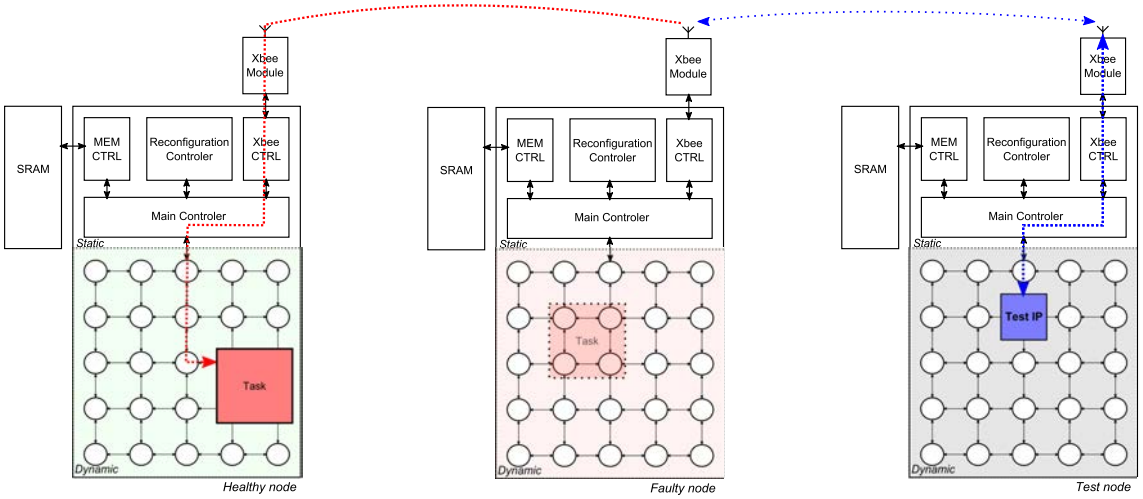


FIGURE 3.4 – Illustration du mécanisme de délocalisation de tâches à travers le réseau de communication *Zigbee* dans le cadre d'un test à distance.

identifiant permettant d'informer les autres nœuds du réseau d'une défaillance ainsi que la demande de délocalisation de tâches. Cette trame contient également les informations nécessaires aux autres nœuds afin d'analyser leur capacité à venir en aide. Parmi ces informations on trouve par exemple l'identifiant de l'IP permettant de connaitre la surface logique, pour technologie FPGA donnée, nécessaire à l'implémentation. En effet, comme il est précisé dans la section précédente, le nœud accueillant doit répondre à plusieurs règles afin d'assurer la maintenabilité du système. Lorsque la requête de délocalisation est réceptionnée par un nœud, celui-ci vérifie par conséquent sont aptitude à satisfaire cette requête. Dès lors où l'ensemble des conditions est respecté, une trame de retour est émise afin de valider la requête, mais également de préciser par identification le nœud contenant désormais la tâche. Si plusieurs nœuds ont la possibilité d'implémenter la tâche, c'est la première trame réceptionnée par le nœud défaillant qui est prise en compte et sélectionne le nouveau nœud accueillant la tâche délocalisée. Les transmissions étant effectuées en *broadcast*, la première trame reçue correspond soit au nœud le plus proche, soit au nœud ayant le moins de charge de travail, ce qui satisfait le protocole dans les deux cas. Lorsque le nœud de délocalisation (nœud "*helper*") est identifié, le transfert du fichier de configuration de l'IP (si disponible) et les données associées à la tâche sont transmis vers ce dernier. Si le transfert du fichier de configuration est indisponible via le nœud défaillant, une nouvelle communication s'effectue entre le nœud *helper* et le reste du réseau afin d'identifier un nœud du réseau disposant du *bitstream*. Une fois les données transférées, le nœud *helper* envoi un acquittement (*acknowledge*) afin d'informer la bonne réception de la trame et implémente ensuite l'IP à délocaliser. Dans l'objectif d'améliorer considérablement la capacité d'organisation des tâches, principalement au sein même d'un nœud, il est impératif de disposer d'un réseau de communication sur puce sûr de fonctionnement.

$BEGIN$	ID_d	ID_s	$TypeTrame$	$Data$	END

TABLE 3.1 – Structuration en champs d'une trame de communication directe entre deux nœuds du réseau.

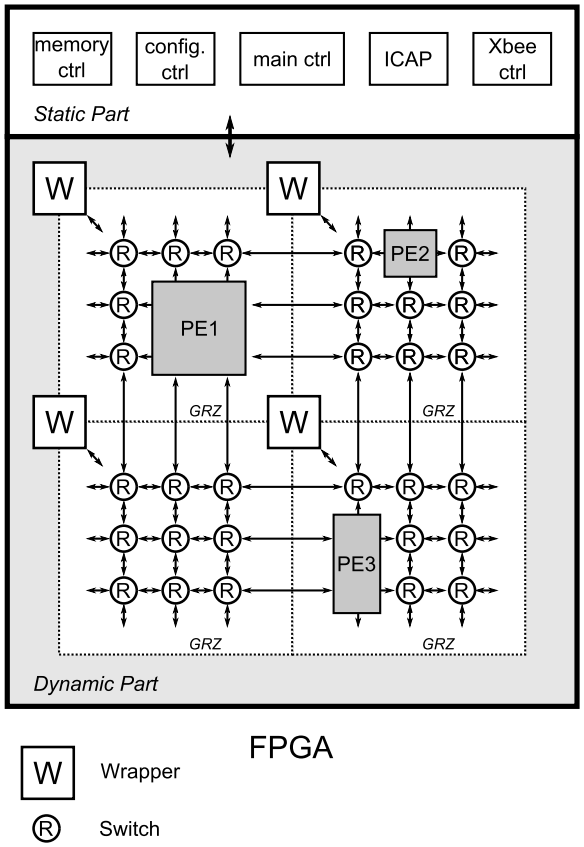


FIGURE 3.5 – Configuration d'un nœud auto-organisé.

3.2.2 Structure des trames

Le tableau 3.1 illustre la composition d'une trame de communication s'effectuant directement entre deux nœuds du système auto-organisé. Une trame est composée de plusieurs champs tels que les champs de début *BEGIN* et de fin *END*. De plus, elle contient les identifiants des nœuds destinataires ID_d et source ID_s, ainsi que des champs permettant d'identifier le type d'information $TypeTrame$ contenu dans le champ *Data*. Ce type de trame est utilisé par le système lors de l'identification d'une réorganisation de la structure reconfigurable des nœuds en cas de défaillances détectées ou répondant à l'évolution environnementale du système.

3.3 Fonctionnement global du test

Notre mécanisme de test fonctionne de la manière suivante. Le début de la procédure commence par une détection d'erreurs dans les NoC RKT, constituants les nœuds

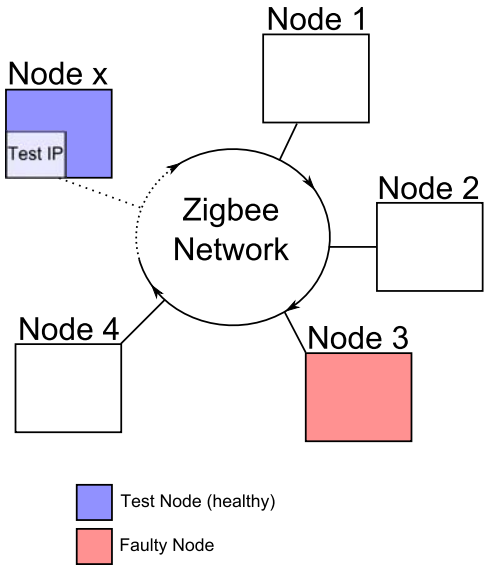


FIGURE 3.6 – Réseau auto-organisé et auto-reconfigurable au protocole *Zigbee*.

MPSoC réalisés par des FPGA, grâce à des tests en ligne basés sur des CCE. La figure 3.5 représente la structure interne du FPGA présente dans les différents nœuds du système auto-organisé. Une fois que le système détecte une ou plusieurs erreurs, les routeurs des *Zones Globalement Sûres* concernées passent dans un mode de test. Ensuite, une demande est envoyée aux autres nœud non-défectueux. Cette demande spécifique permet de connaître quels nœuds sont capables de mettre en œuvre et de faire exécuter un IP test. Lorsque le nœud test est identifié, le FPGA associé à ce nœud est dynamiquement reconfiguré avec le fichier de configuration de l'IP test. La figure 3.6 illustre le principe du test hors ligne proposé grâce à la délocalisation d'un IP test dans un nœud distant du réseau. Lorsque la reconfiguration est exécutée, ce nœud test est en mesure de générer des vecteurs de test. Ces vecteurs sont des séquences de bits qui induisent une réponse particulière à une architecture connue. Par conséquent, le nœud test commence par tester ses propres routeurs pour obtenir une réponse à un vecteur généré. La figure 3.7 résume l'utilisation des différents vecteurs utilisés par le mécanisme de test au sein d'un nœud. Cette réponse sert de *référence* en raison de l'état de santé du nœud. Les *vecteurs de test* et de *réponse* sont envoyés au nœud à tester. Le vecteur de test est alors injecté dans le NoC du nœud défaillant à travers un bloc spécifique dit *Wrapper*, qui établit le lien entre les parties statique et dynamique constituant le nœud à tester. Chaque routeur testé des ZGS défaillantes reçoit le vecteur de test. La référence associée est conservée par le *Wrapper*. Puis, une fois que le vecteur est injecté et que les réponses sont disponibles, ces résultats sont comparés localement avec la *référence* afin d'identifier à la fois les routeurs défectueux et en particulier ses éléments défectueux (*buffers*, logique de routage, etc.) générant des erreurs. Selon le nombre d'erreurs ou la localisation des erreurs, la réponse est renvoyée à l'IP test pour analyse. Ainsi, le *Wrapper* détermine les éléments défectueux du routeur grâce à des mauvaises positions de bits dans le vecteur réponse au vecteur de test. Par conséquent, une décision concernant le routeur défectueux est appliquée (reconfiguration ou désactivation partielle ou globale du routeur).

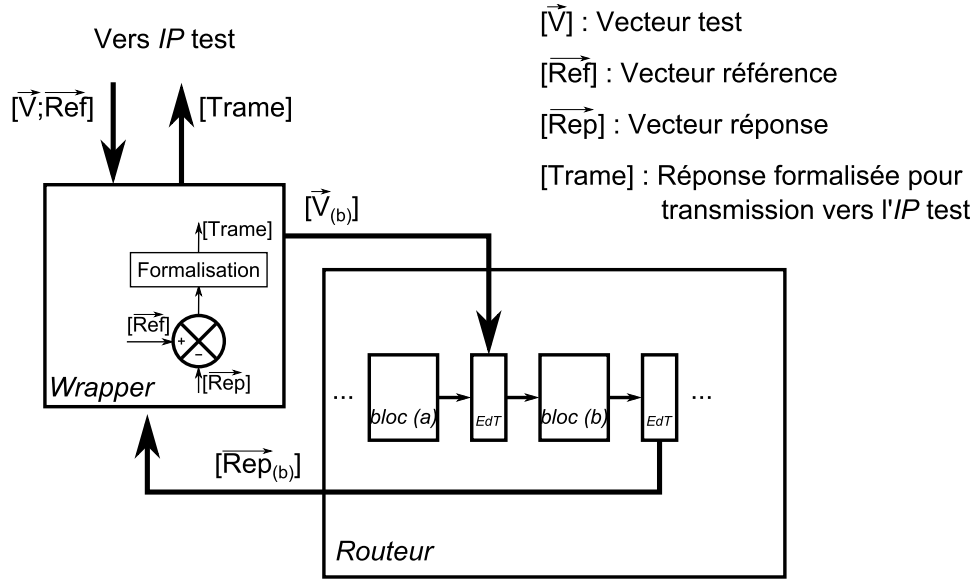


FIGURE 3.7 – Mécanisme de localisation d'erreur au sein d'un nœud.

3.4 Conception axée sur la méthode du Scan Chain

L'approche de test adoptée repose sur une méthode de chaine de test (*Scan Chain*) qui consiste à injecter des données (vecteurs de test), directement dans l'architecture à tester à travers des registres spécifiques en chaîne, dont l'accès s'effectue par les liaisons qui séries traversent l'ensemble des registres de la chaine. Ces données sont préalablement définies afin de connaitre les réactions du système lors de leur injection dans le but de vérifier la concordance des résultats obtenus avec ceux attendus. Ainsi, par comparaison des deux, il est possible de déceler des différences pouvant être liées à d'éventuelles erreurs ou défaillances.

Pour mettre en œuvre l'approche retenue, une adaptation de la méthode *Scan Chain* aux routeurs RKT est développée. Dans un premier temps, les modifications architecturales effectuées sont basées sur une méthode *Scan Chain* classique. C'est-à-dire, au travers d'une seule liaison série permettant le remplissage des registres par le vecteur. La figure 3.8 illustre cette application aux principaux blocs du routeur RKT. Ainsi, plusieurs registres sont présents entre les *buffers* d'entrées, la logique de routage et les *buffers* de sorties, avec la possibilité d'injecter des données en parallèle sur les entrées des *buffers* d'entrée. Toujours selon cette méthodologie, une chaine de test (*Scan IN*) relie l'ensemble des registres. Ainsi, en fonction du signal d'entrée *Scan Enable* les données peuvent circuler par décalage le long de la chaine.

Le principal inconvénient de cette configuration est le temps nécessaire pour remplir, ou vider, l'ensemble des registres dû à la communication en série. De plus, ce temps dépend principalement du nombre de registres à traverser, ainsi que de la taille des bus de données. Par conséquent, pour une architecture complexe utilisant un grand nombre d'entrées/sorties pour chaque bloc logique d'un routeur, cette méthode reste limitée en termes de performances temporelles.

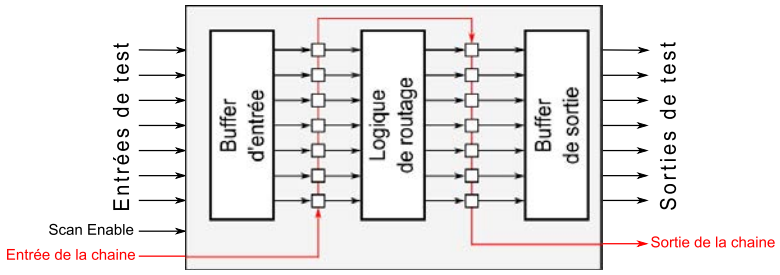


FIGURE 3.8 – Principe de base de la méthode "chaine de test" appliquée à un routeur RKT.

Afin d'adapter la structure *Scan Chain* classique, des modifications architecturales sont apportées au sein d'un routeur RKT. La figure 3.9 représente les modifications architecturales effectuées au routeur RKT par l'intégration des EdT en guise de registres permettant la récupération et l'injection des vecteurs, la capture et la transmission des réponses associées. De plus, afin de pouvoir injecter les données en entrées des *buffers*, des multiplexeurs ont également été intégrés. Une logique de gestion des vecteurs et des EdT a été développée et associée à chacun des routeurs.

Malgré la validation du fonctionnement des modifications apportées dans RKT et permettant la localisation de fautes dans le routeur, les ressources logiques nécessaires à l'implémentation d'une telle architecture sont trop importantes. En effet, le surcoût logique des EdT intégrés est d'autant plus important qu'il y a de modules de gestions de vecteur que de routeurs présents dans le réseau. Une exploration architecturale vers une solution optimale de conception est nécessaire et dont les détails sont données dans la section suivante.

3.5 Exploration architecturale

L'architecture développée reprend les EdT ainsi que le principe du module gestion des vecteurs au travers d'un module spécifique "*Wrapper*". L'objectif de ces changements est évidement de palier aux problèmes liés aux temps d'exécution trop long et la consommation excessive en ressources logiques. Comme mentionné précédemment, la structure modifiée du réseau NoC RKT est divisée en plusieurs *Zones Globalement Sûres* (ZGS). L'objectif de ce découpage est de créer une combinaison entre les tests en ligne et hors ligne. Nous utilisons des codes de correction d'erreurs dans chaque port d'entrée et de sortie des routeurs en bordure de la ZGS comme indiqué sur la figure 3.10. Cette solution de distribution des CCE permet de détecter les erreurs à l'intérieur de chaque ZGS du réseau sur puce. Bien que la localisation n'est pas précise, il y a plusieurs avantages à adopter cette répartition. Premièrement, c'est une solution économique en termes de ressources matérielles. En effet, contrairement à la mise en place de CCE pour tous les ports d'entrée/sortie de chaque routeur, le coût des ressources est plus ou moins importante en fonction de la taille des ZGS. Par exemple, pour une ZGS de taille 3x3 routeurs, 24 CCE

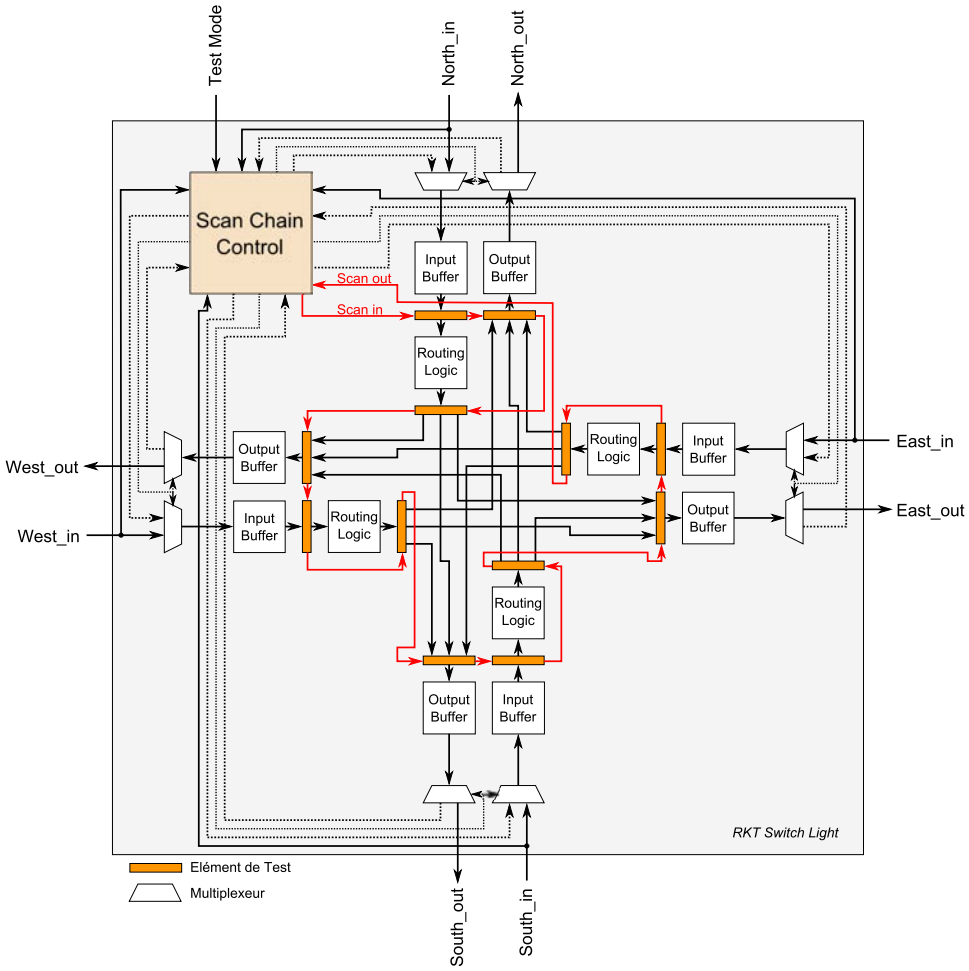


FIGURE 3.9 – Architecture initiale selon le principe du *Scan Chain* du routeur RKT.

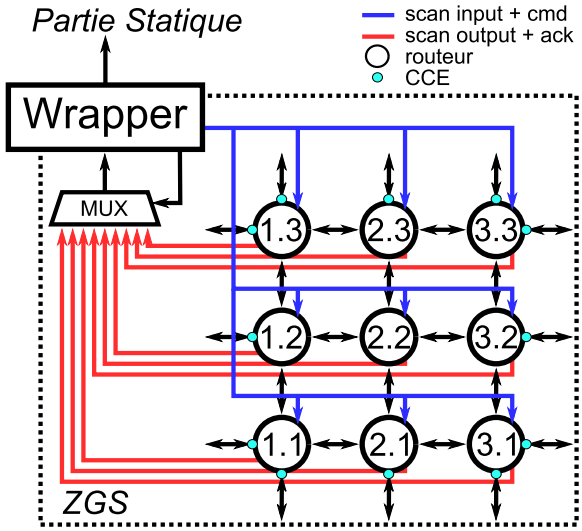


FIGURE 3.10 – Zone Globalement Sûre (ZGS).

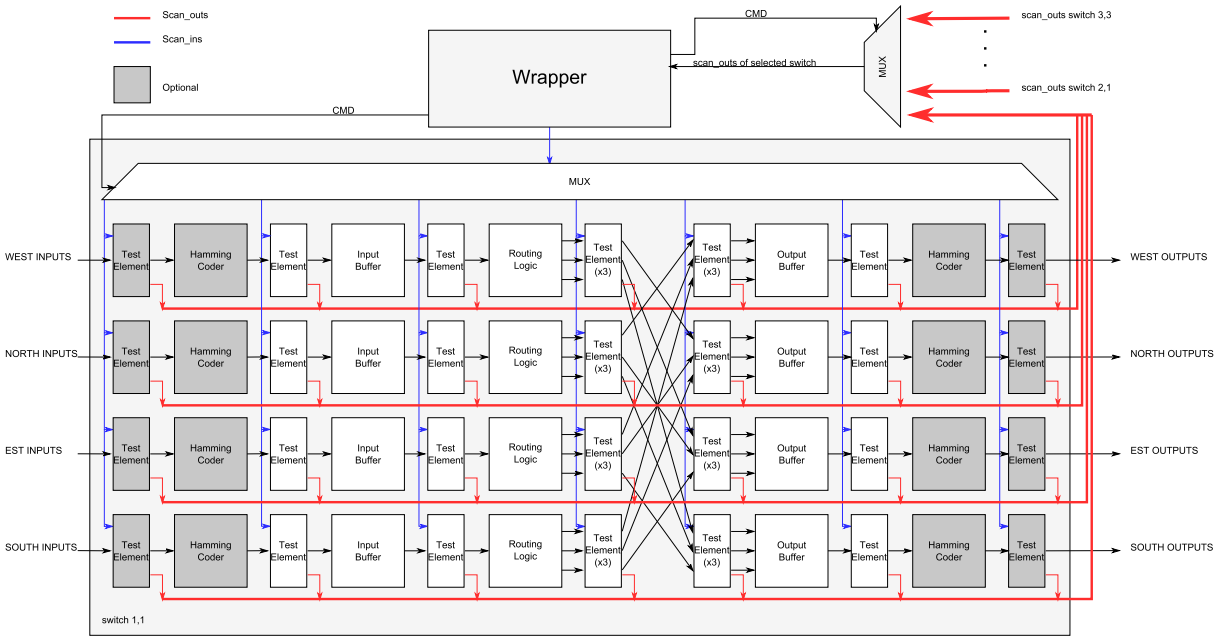


FIGURE 3.11 – Architecture modifiée du routeur RKT par l'intégration SdF du mécanisme délocalisé.

(6 par côté) sont nécessaires. La mise en œuvre d'un CCE par port et par routeur dans un NoC de taille 3x3, nécessite 72 CCE, soit 3 fois plus de ressources. Le deuxième avantage est la possibilité de désactiver une seul ZGS. Ce qui signifie une capacité d'effectuer un test hors ligne d'une partie du réseau tout en maintenant en fonctionnement le reste du NoC.

3.5.1 Bloc spécifique d'interconnexion : bloc Wrapper

Pour gérer tous éléments introduit par la mis en place du test, un module spécifique d'interconnexion dit "*Wrapper*" est mise en œuvre pour chaque ZGS selon l'architecture décrite en figure 3.11. Son rôle est multiple. Premièrement, le *Wrapper* commande l'ensemble des éléments de test constituant la ZGS qui lui est associé. La machine d'états de la structure est donnée à la figure 3.12. Quand une erreur est détectée par les CCE d'une ZGS, le *Wrapper* associé passe en mode test. A cet instant, la région est déconnectée du réseau et les EdT passent également en mode test. Il y a donc d'une part un mode *normal*, où le routeur fonctionne comme si il n'y avait pas d'EdT. D'autre part, un mode *test* où les EdT sont activés et attendent des instructions du *Wrapper*. Une fois qu'un vecteur de test est généré par l'IP test et envoyé au nœud défectueux, le *Wrapper* concerné est en charge de récupérer le vecteur ainsi que la référence associée. Lorsque cette étape est réalisée, la référence est conservée par le contrôleur, et le vecteur est distribué aux routeurs qui composent la ZGS. La distribution des données de test est détaillée dans la section 3.6. Le *Wrapper* doit ensuite contrôler les différentes étapes du processus de test à l'aide de plusieurs signaux, incluant les commandes des multiplexeurs. À la fin de ce processus, le *Wrapper* effectue une pré-analyse pour identifier les erreurs et les réponses

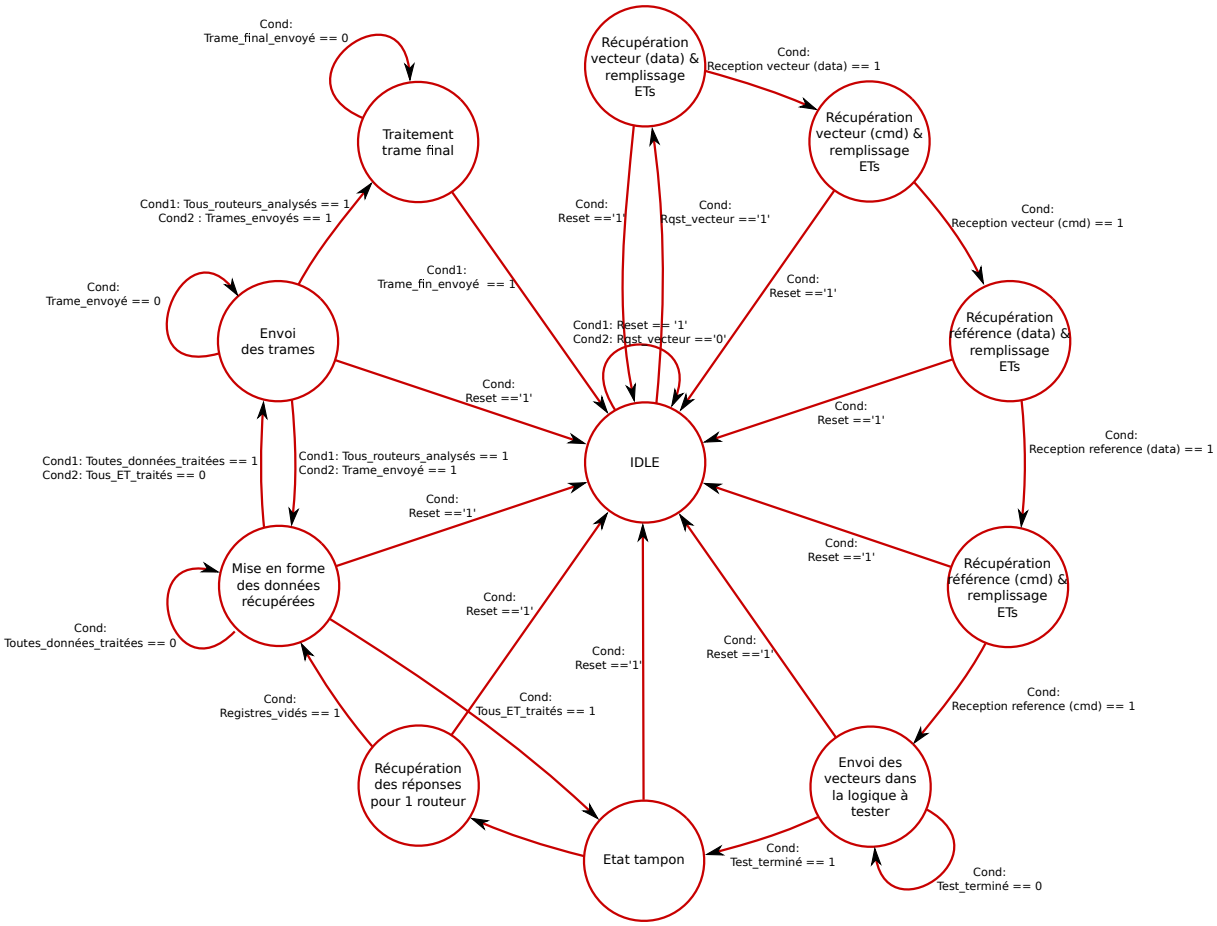
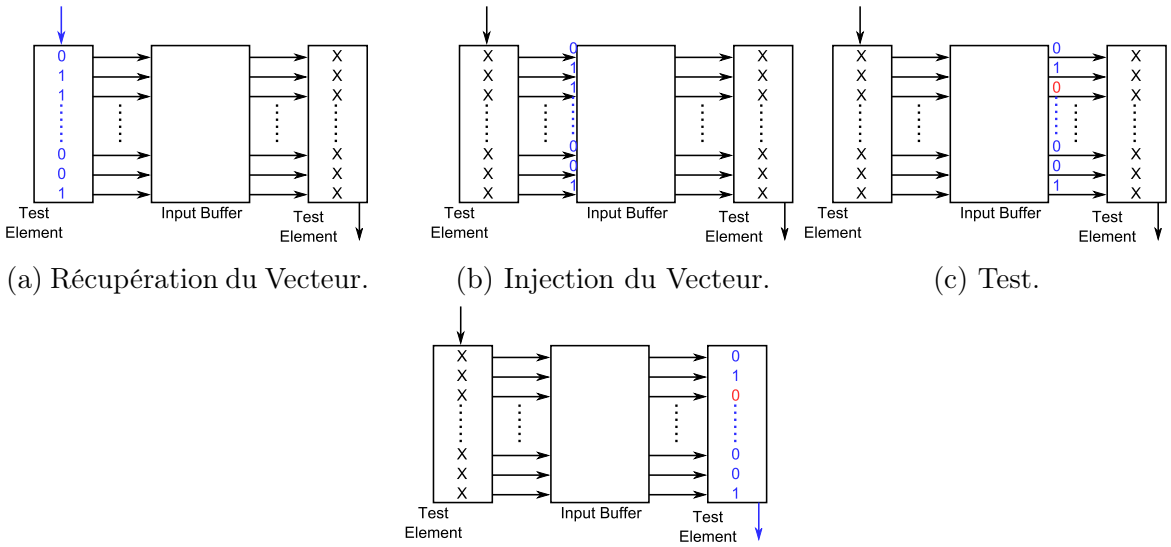


FIGURE 3.12 – Machine d'état de la structure *Wrapper*.

sont renvoyées à l'IP test. Cette analyse permet de classifier les réponses par le nombre d'erreurs par EdT dans le but d'optimiser la quantité de données à envoyer par l'intermédiaire des communications réseau. En effet, dans notre configuration, la transmission sans fil est la partie la plus lente de notre mécanisme. Il est donc nécessaire d'optimiser autant que possible la quantité de données à transmettre. En comparaison aux temps de transmission des données, le temps de test des routeurs est négligeable. Les détails de la classification et le formatage des réponses aux vecteurs sont donné dans les paragraphes suivants.

3.5.2 Principe de fonctionnement des blocs Éléments de Test

Pour être en mesure de localiser précisément une erreur dans l'architecture, des EdT ont été développés. Ces EdT sont des registres "améliorés" qui sont placés entre deux parties de l'architecture. Ils ont pour rôle de réceptionner et d'injecter les données spécifiques dans un des composants du routeur. Ces données, provenant d'un vecteur de test, sont appliquées sur les entrées du bloc à tester. Les étapes du fonctionnement des EdT sont détaillées à la figure 3.13. Les éléments de test fonctionnent en cinq étapes. La première est le remplissage des EdT (voir figure 3.13a), qui s'effectue sur une entrée série et qui est



(a) Récupération du Vecteur. (b) Injection du Vecteur. (c) Test.

(d) Capture de la réponse.

FIGURE 3.13 – Méthode de la chaine de test.

commandé par 2-bits de contrôle. Une fois que l'EdT a récupéré les données appropriées, la deuxième étape consiste à les envoyer dans l'architecture à tester (voir figure 3.13b). Ces données sont générées spécialement afin d'obtenir un résultat spécifique, comprenant des données, des signaux de commande, des acquittements, etc. La troisième étape est un état d'attente du traitement de l'architecture, comme dans le mode *normal*. Lorsque le traitement des données par l'architecture est terminé et que les données sont présentes sur ses sorties, elles sont saisies et maintenues dans l'EdT suivant (voir figure 3.13c). Les résultats présents dans les EdT composent le *vecteur réponse au vecteur de test*. Comme le même processus est effectué dans une même architecture, mais à l'intérieur d'un nœud sain et dans un nœud fautif, une analyse de la différence entre les deux réponses peut être réalisée afin d'y localiser d'éventuelles erreurs. Utilisant la positions des bits défectueux dans le *vecteur réponse*, l'IP test est alors en mesure de déduire, éventuellement après l'utilisation de plusieurs *vecteurs de test* différents, les éléments de l'architecture générant des erreurs. Au cours de la dernière étape, les *vecteurs réponses aux vecteurs test* sont renvoyés au bloc *Wrapper* par une sortie série.

Le pseudo-code d'un EdT mis en œuvre est décrit dans l'algorithme 1. En fonction du bus de commande *scan_ena* provenant du bloc *Wrapper* et pouvant prendre 4 valeurs différentes (2 bits), l'EdT effectue un traitement différent. Lorsque ce signal vaut le codage "01", les bits arrivant en série (*scan_in*) sont insérés dans le registre *scan_reg* et ceux déjà présents dans le registre sont envoyés sur le port de sortie *scan_out*. Pour injecter le vecteur dans l'architecture, le codage du signal de commande est "10". Ainsi, les données présentes dans le registre sont envoyées sur le bus de données *data_bus_out* connecté à l'architecture à tester. Lors de la capture de la réponse (*scan_ena* = "11"), les données présentes sur le bus de données entrant *data_bus_in* sont récupérées dans *scan_reg*. Enfin, lorsque le signal de commande vaut le codage "00", l'EdT reste inactif.

Algorithm 1 Pseudo code d'un Élément de Test.

Require: $Switch\_mode = test$
  $data\_bus\_out \leftarrow data\_bus\_test$
  if $scan\_ena = 01$ **then**
    $scan\_out \leftarrow scan\_reg(MSB)$
    $scan\_reg(MSB \rightarrow 1) \leftarrow scan\_reg((MSB - 1) \rightarrow 0)$
    $scan\_reg(0) \leftarrow scan\_in$
    $ack \leftarrow 0$
  else if $scan\_ena = 10$ **then**
    $data\_bus\_test \leftarrow scan\_reg$
    $ack \leftarrow 0$
  else if $scan\_ena = 11$ **then**
    $scan\_reg \leftarrow data\_bus\_in$
    $ack \leftarrow 1$
  else
    $ack \leftarrow 0$
  end if

Position des bits	1	2	3	4	5	6	7	8	9	10	11	12	13
Message	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5	D_6	D_7	D_8	P_5

TABLE 3.2 – Illustration du codage de Hamming pour 8 bits de données.

3.5.3 Bloc de Hamming

Ce bloc consiste à détecter les erreurs présentent dans le NoC et de les corriger. La solution proposée est simple à mettre en œuvre et est basée sur l'implantation d'un code correcteur de Hamming et d'une parité. Afin de simplifier l'explication du codeur mise en œuvre, les détails ci-dessous sont appliqués pour une taille réduite de message par rapport à la taille réelle des données utilisées dans le routeur. Ainsi, l'ajout de 4 bits de Hamming et un bit de parité, permet de détecter jusqu'à deux erreurs et de corriger une erreur. Ce bloc de détection et de correction est décrit en VHSIC Hardware Description Language (VHDL) et intégré à l'ensemble des routeurs constituants le réseau. De même, des blocs de codage de Hamming sont également implantés dans les blocs IP de transmission et de réception de paquets de données. Les spécifications de conception du principe du codage et décodage de Hamming + 1 bit de parité sont décrites ci-dessous. Pour un message initial codé sur 8 bits $[D_1D_2D_3D_4D_5D_6D_7D_8]$, 4 bits de Hamming (P_1, P_2, P_3, P_4) et un bit de parité (P_5) sont générés pour une transmission finale sur 13 bits dont la structure est donnée dans la table 3.2.

Le calcul de ces parités repose sur l'hypothèse d'une parité paire entre les bits du massage D_i et les bits de parité P_i telle que :

$$P_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7; \tag{3.2}$$

$$P_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_{11}; \tag{3.3}$$

$$P_3 = D_2 \oplus D_3 \oplus D_4 \oplus D_8; \tag{3.4}$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_8; \tag{3.5}$$

$$P_5 = P_1 \oplus P_2 \oplus D_1 \oplus P_3 \oplus D_2 \oplus D_3 \oplus D_4 \oplus P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8 \tag{3.6}$$

Le principe du décodeur de Hamming + 1 bit de parité est réalisé à la réception du message. Les bits de vérification Vi sont calculés de la même manière que pour le codage des bits de parité Pi. Un bit de parité globale P est également calculé.

$$V_1 = P_1 \oplus D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7, \tag{3.7}$$

$$V_2 = P_2 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_{11}, \tag{3.8}$$

$$V_3 = P_3 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_8, \tag{3.9}$$

$$V_4 = P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8 \tag{3.10}$$

$$P = P_5 \oplus P_1 \oplus P_2 \oplus D_1 \oplus P_3 \oplus D_2 \oplus D_3 \oplus D_4 \oplus P_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8 \tag{3.11}$$

A partir d'une analyse des bits de vérification et de la parité globale, des détections et corrections d'erreurs peuvent être réalisées en considérant le mot binaire $V = V_4V_3V_2V_1$. Quatre cas de figure se présentent alors :

- Si $V = 0000$ et $P = 0$: aucune erreur est détectée ;
- Si $V \neq 0$ et $P = 1$: une seule erreur pouvant être corrigée est détectée. Le codage V donne la position du bit erroné à inverser pour correction (Par exemple $V = 0110$ signifie inversion du digit à la 6^{eme} position) ;
- Si $V \neq 0$ et $P = 0$: deux erreurs sont détectées mais ne peuvent être corrigées ;
- Si $V = 0000$ et $P = 1$: une erreur est présente sur le bit de parité P.

Dans le cas d'une erreur détectée, le paquet est corrigé et la transmission est acquittée. Dans le cas d'une détection de deux d'erreurs, deux solutions sont alors envisagées :

- Le routeur est déclaré définitivement fautif. Il est isolé du reste du réseau en activant de façon permanente ses connexions d'indisponibilité aux routeurs voisins.

– Une mémorisation des syndromes de Hamming (valeurs V_i et P) et une demande de retransmission ($NACK$) est mise en œuvre. Si lors de cette seconde transmission du message, les mêmes syndromes sont obtenus, alors une erreur permanente est considérée et le routeur est déclaré fautif permanent.

3.6 Distribution des vecteurs de tests

La distribution des vecteurs de test est un élément clé du mécanisme en termes de temps d'exécution. En effet, il y a deux étapes dans le processus qui ont besoin de plus ou moins de temps en fonction de la façon dont ils sont mis en œuvre. La premier est la distribution des vecteurs de test pour les différents routeurs d'une ZGS. Habituellement, un vecteur est injecté en série à travers tous les composants d'une architecture testée. Dans le cas d'un grand nombre d'éléments à tester, le temps nécessaire pour remplir tous les EdT peut être très long. Pour parer à cet inconvénient, nous distribuons le vecteur avec une combinaison de transmission série et parallèle qui est détaillée dans le paragraphe suivant. La deuxième étape plus lente est le retour des *vecteurs réponses aux vecteurs de test*. Plus de détails sur le formatage des réponses sont donnés dans la section 3.7.2.

L'injection de données en série par *Scan Chain* est une méthode lente où le temps de chargement des données dépend de la taille du vecteur, et donc également du nombre d'éléments que la chaîne doit traverser. Par conséquent, plus le vecteur est long, moins le chargement est rapide. Pour réduire ce temps de chargement des données à travers la chaîne de test, nous injectons le vecteur partiellement en parallèle. Plus précisément, une entrée de chaîne de test est mis en œuvre pour chaque EdT. Une particularité du test d'un routeur correspond à la valeur constante des données qui traversent l'architecture. Seuls quelques signaux de commande sont différents d'un bloc à l'autre. Par conséquent, nous injectons les données du vecteurs de test en parallèle dans tous les éléments de test de tous les routeurs de la ZGS. Puis les signaux annexes sont successivement injectés dans chaque EdT placés entre mêmes blocs pour tous les routeurs de la ZGS. Par exemple, les signaux annexes du vecteur de test pour les EdT placés avant les *buffers* d'entrée (voir la figure 3.11) sont injectés simultanément dans tous les EdT situés à la même position dans tous les routeurs de la ZGS en cours de test.

Cette adaptation particulière de la méthode de la chaîne de test est possible grâce à un multiplexeur général mis en œuvre dans chaque routeur. Le bloc *Wrapper* sélectionne simultanément les EdT concernés pour tous les routeurs et les chargent un par un avec les données des signaux annexes. Cette solution est avantageuse en termes de temps d'exécution pour deux raisons. Premièrement, cette méthode de remplissage des EdT s'effectue de manière plus rapide que l'injection des vecteurs de test par une seule chaîne. Deuxièmement, comme nous utilisons des données communes à tous les EdT, nous réduisons le temps de transmission des données de l'IP test vers le bloc *Wrapper*. En effet, si nous voulons utiliser un vecteur de test différent pour chaque EdT, le temps de transmission serait multiplié par le nombre différent d'EdT par routeur, soit un total de 11 EdT si nous utilisons des vecteurs identiques pour chacun des quatre côtés (*Side*). De même, si

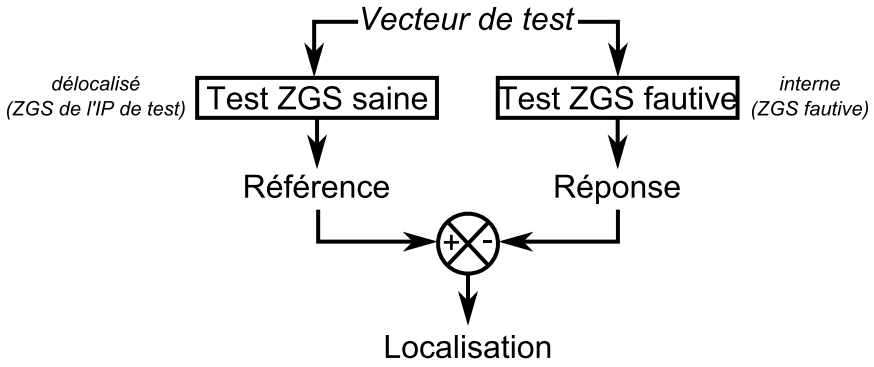


FIGURE 3.14 – Principe de fonctionnement de la localisation d'une erreur.

un vecteur unique par EdT est généré, le nombre de vecteurs différents s'élève à 44 EdT par routeur puisque que nous avons 11 EdT par *Side* et dont chacun est traité séparément. De plus, une référence différente pour chaque vecteur de test est nécessaire.

3.7 Traitement des réponses gérées par le mécanisme SdF proposé

3.7.1 Analyse des réponses et localisation des erreurs.

Les techniques et modifications architecturales présentées précédemment illustrent la méthodologie d'injection de vecteurs, préalablement défini, dans l'architecture d'une ZGS à tester . Elles permettent d'obtenir une réponse pour chacun des modules des routeurs. La localisation d'une ou plusieurs erreurs au sein de la structure du réseau sur puce est réalisée quant à elle par comparaison de bits entre les réponses générées et les références produites lors de la génération du vecteur de test. Le mécanisme est donc basé sur la détection de divergences entre les réponses à un vecteur de test pré-établies et ses références associées.

Les structures, ainsi que la méthodologie de test, étant strictement identiques, il en va de même pour les réponses aux vecteurs de test de l'ensemble des ZGS saines. Plus précisément, la même structure de différents routeurs produit une réponse identique à condition de ne pas être fautive. Par conséquent, grâce à cette homogénéité, nous pouvons localiser avec une précision avancée, le/les blocs ainsi que le/les signaux fautifs en fonction de l'emplacement du bit ou des bits erronés dans la réponse. La figure 3.14 illustre le principe de localisation des erreurs dans une réponse à un vecteur de test donné suivant le mécanisme de test délocalisé proposé.

3.7.2 Mise en forme des résultats

A la fin des tests locaux des routeurs des ZGS, les résultats sont contenus dans les EdT. Comme pour le chargement de ces derniers, les données sont récupérées séquentiellement

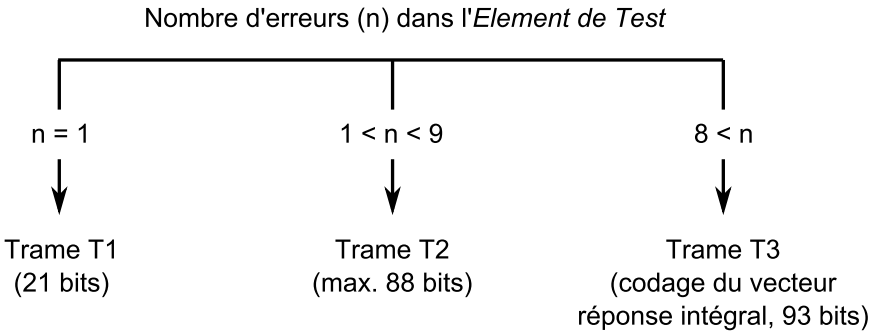


FIGURE 3.15 – Codage trame des vecteurs réponses d'un EdT en fonction du nombre d'erreurs détectées.

en parallèle. Les réponses sont ensuite envoyées au bloc *Wrapper* routeur par routeur. Pour chaque routeur, la sortie série de chaque EdT est utilisée pour former un bus de données en sortie vers le bloc *Wrapper*. En passant par un multiplexeur pour sélectionner le routeur, ce bus achemine les réponses au bloc *Wrapper*. Une fois que les données sont disponibles, la réponse de chaque EdT est analysée par comparaison avec la référence associée. Ensuite, la comparaison génère trois informations :

- les vecteurs réponses aux vecteurs de test,
- l'emplacement du ou des bits erronés dans les vecteurs réponses,
- le nombre de réponses non-fautives.

Quand une réponse d'un EdT est comparée et que les trois informations sont générées, le bloc *Wrapper* formate ces informations en fonction des résultats obtenus. En effet, pour réduire le temps de transmission des résultats vers l'IP test, les réponses aux vecteurs de test sont envoyées sous différentes formes de trames. La figure 3.15 résume le codage des vecteurs réponses d'un EdT en fonction du nombre d'erreurs détectées. Ainsi, en fonction du nombre d'erreurs par EdT, quatre formes différentes de trame de vecteurs réponses peuvent être envoyées par l'intermédiaire du réseau sans fil. Plusieurs champs de ces différentes trames sont identiques. Dans le cas où il y a une seule erreur dans le vecteur réponse d'un EdT, une forme de trame T1 est codé comme définie dans le tableau 3.3.

Switch ID	Side	EdT ID	Type	Localisation
6-b	2-b	4-b	2-b	7-b

TABLE 3.3 – Structure d'une trame T1 du vecteur réponse d'un EdT dans le cas de détection d'une seule faute.

Une partie de la trame est commune aux quatre formes. Les premiers champs commun à toutes les trames de vecteurs réponses sont l'identifiant du routeur (la position dans le réseau sur puce - *Switch ID*), le côté (*Side*) (Nord, Sud, Est ou Ouest du routeur), l'identifiant de l'EdT (l'emplacement de l'EdT dans le routeur - *ET ID*) et le type de faute qui indique quelle forme de trame est utilisée. On trouve ensuite un champ de taille 7-bits dédié à l'emplacement de l'erreur dans l'EdT. Ceci permet d'obtenir le temps de

Switch ID	Side	ET ID	Type	Nb Fautes	Loc. n_1	Loc. n_2	Loc. n_x
6-b	2-b	4-b	2-b	4-b	7-b	7-b	7-b

TABLE 3.4 – Structure d'une trame T2 de vecteur réponse pour 2 à 8 fautes détectées dans un EdT.

transmission le plus court possible car l'ensemble d'une trame de ce type nécessite que 21-bits à transmettre sur le réseau inter-nœuds.

La seconde structure de trame T2 concerne les vecteurs réponses dans le cas d'un nombre d'erreurs détectées dans l'EdT compris entre 2 et 8 erreurs. Cette structure est présentée dans le tableau 3.4. Comme la trame précédente, le vecteur réponse de l'EdT traité n'est pas envoyé à l'IP test dans son intégralité, seul les emplacements des erreurs sont transmis. Le nombre maximal d'erreurs acceptées pour des transmissions avec ce type de trame est de 8 car la taille d'une trame T2 est inférieure à celle d'une trame T3, correspondant à une trame contenant l'intégralité du vecteur réponse d'un EdT. La taille du codage en trame devient plus importante lorsqu'une 9ème erreur dans l'EdT est détectée et exprimée dans son vecteur réponse. D'une manière générale, le nombre de bits nécessaires pour le codage en trame du vecteur réponse à transmettre est donné par l'équation 3.12.

$$FrameSize_{me} = CC + NE + N_{fault} * TL \tag{3.12}$$

La taille de la trame dépend donc de la taille des champs communs (*Switch ID*, *Side*, etc.) CC (14-bits), de la taille nécessaire pour indiquer le nombre d'erreurs NE (4-bits), du nombre d'erreurs N_{fault} et de la taille de la trame de localisation TL (7-bits). Dans le cas d'une détection de 8 erreurs, la taille de la trame T2 est de 88-bits. Comme décrit dans le tableau 3.5, la taille de la trame à envoyer pour transmettre l'ensemble du vecteur réponse d'un EdT à l'IP test dépend des champs de trame communs et de la taille de cette réponse, correspondant à 72-bits de données et de 7-bits de signaux de commandes (acquittement, requêtes, etc.).

Switch ID	Side	ET ID	Type	Vecteur Réponse
6-b	2-b	4-b	2-b	79-b

TABLE 3.5 – Structure d'une trame T3 de vecteur réponse pour un nombre de fautes détectées supérieur à 9 dans un ET.

Par conséquent, la taille d'une trame T3, permettant la transmission de l'intégralité d'un vecteur réponse d'un EdT est de 93-bits, correspondant au cas où plus de 8 erreurs sont identifiées. Une dernière structure de trame T4 correspond à une trame de fin de test routeur afin d'y comptabiliser le nombre de réponses non-fautives. Les détails de cette trame sont donnés dans le tableau 3.6. Ainsi, à l'issue de l'analyse des vecteurs réponse des EdT d'un routeur, une trame réponse routeur est transmise à l'IP test spécifiant le nombre d'EdT non-défaillants selon une forme de trame T4. L'intérêt de cette trame est

double, elle permet dans un premier temps de signaler la fin de l'analyse d'un routeur, et également d'instaurer un mécanisme de contrôle permettant de s'assurer du test de l'ensemble des blocs logiques du routeur.

Switch ID	Side	ET ID	Type	Nb EdT Non-fautifs
6-b	2-b	4-b	2-b	7-b

TABLE 3.6 – Structure d'une trame T4 de vecteur réponse routeur.

3.7.3 IP de test et prise de décision

L'originalité du concept proposé repose sur la gestion délocalisée des tests grâce la capacité d'auto-organisation du système communicant. Malgré l'intégration architecturale des blocs dédiés à l'analyse du réseau sur puce, la génération des vecteurs de tests est effectuée à distance, par un nœud différent de celui à tester, puis transmis aux ZGS concernés. Ce concept permet l'implantation dynamique d'algorithmes de génération de vecteurs, adaptables en fonction de l'architecture à tester. En effet, nous proposons ce mécanisme de délocalisation d'IP test afin de localiser des erreurs survenues dans un NoC de type RKT. Cependant, ce concept reste valable pour d'autres types de structures NoC. La figure 3.16 illustre la fonctionnalité de l'IP test adapté au mécanisme de test d'une ZGS de routeurs RKT.

En plus de générer les vecteurs de tests nécessaires à l'identification de défaillances, l'IP test à également pour rôle d'analyser les résultats obtenus après injections des vecteurs dans le nœud concerné. Après l'utilisation de plusieurs vecteurs, et donc après l'obtention de plusieurs réponses, l'IP test est capable de déduire les types d'erreurs localisées (inversions de bits, bits bloqués, erreurs transitoires, permanentes, etc.). En fonction de l'identification des types d'erreurs, le module test délocalisé déduit la meilleure solution afin d'éradiquer ou de contourner la défaillance. Il dispose de plusieurs choix possibles en fonction de ces résultats :

- Isolement total ou partiel du ou des routeurs fautifs,
- Isolement de la ZGS,
- Reconfiguration totale ou partielle du ou des routeurs fautifs,
- Reconfiguration de la ZGS.

Il est également possible de réorganiser l'agencement des tâches implémentées dans la ZGS testée. Elles peuvent être déplacées vers d'autres ZGS au sein du même nœud ou de manière délocalisés vers un autre nœud en cas d'insuffisance de ressources matérielles.

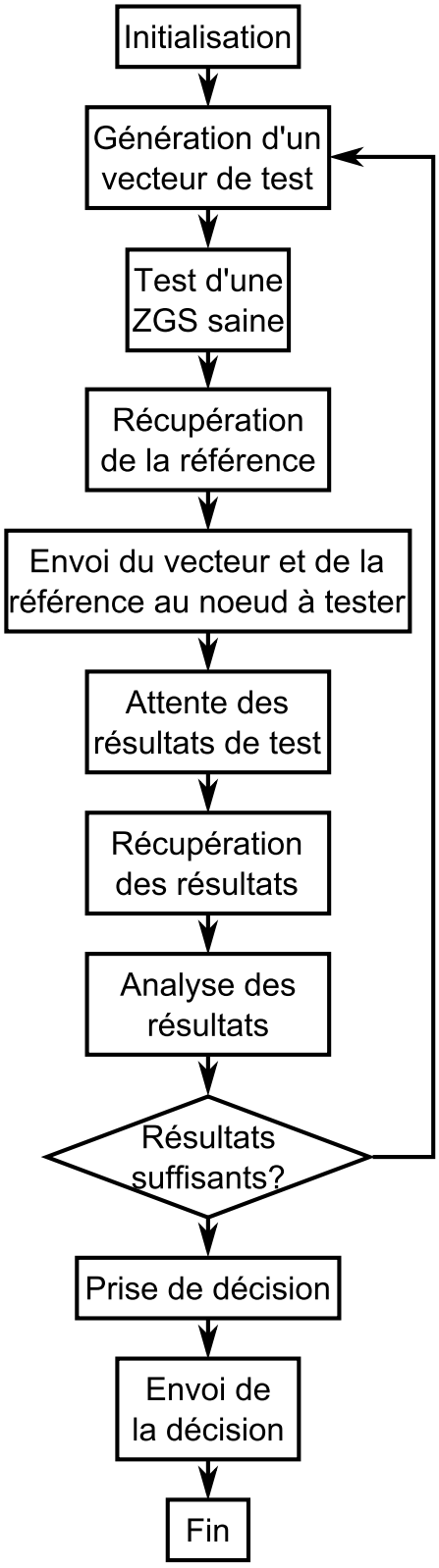


FIGURE 3.16 – Algorithme du principe de fonctionnement de l'IP test.

Conclusion

Dans ce chapitre, nous avons présenté et détaillé la conception de mécanismes logiques de tests adaptés à un contexte d'auto-organisation pour les structures d'intercommunication ou de communication sur puce (NoC) de systèmes reconfigurables en réseau à base de technologie FPGA. L'objectif est la mise en œuvre de tolérance aux fautes d'un réseau auto-organisé sûr de fonctionnement de systèmes/capteurs intelligents MPSoC à base de réseaux sur puce reconfigurables. Plus précisément, nous avons développé des mécanismes d'auto-organisation et de d'auto-tests des structures de NoC adaptatives d'un réseau de nœuds reconfigurables matérialisés par des circuits FPGA SRAM, dans le but d'une maintenabilité maximale du fonctionnement du système dans un contexte réseau WSN. Ainsi, nous avons proposé l'intégration de concepts architecturaux SdF délocalisables et appliqués à des NoC RKT. La principale originalité des techniques développées est qu'elles permettent la conception d'un système reconfigurable multi-nœuds capable d'échanger et d'interagir, et permettant ainsi une gestion avancée de tâches et une auto-gestion de mécanismes de tolérance aux fautes.

Dans un premier temps, l'approche architecturale d'optimisation de l'intégration des blocs de détection et de correction d'erreurs, stratégiquement placés dans le réseau, a été présentée. Cette architecture, de part sa structure originale découpée en zones, se révèle adaptée pour des réseaux reconfigurables dynamiquement. Dans un deuxième temps, nous avons présentés une solution innovante d'autogestion et d'auto-reconfiguration permettant de détecter les défauts des NoC identifiés au sein d'un réseau, au cours de fonctionnement du réseau et provenant de routeurs ou connections défaillants. Pour se faire, des mécanismes de tests délocalisés permettent d'identifier et de localiser avec précision les éléments défaillants de telles structures dans le but de les corriger ou de les isoler pour prévenir toutes défaillances du système. Ces tests reposent sur une approche hybride en-ligne et hors-ligne. L'originalité de l'approche générale est qu'elle est adaptée aux algorithmes de routage adaptatifs basés sur l'algorithme XY des NoC dynamiques constituant l'ensemble des nœuds du réseau. Pour évaluer l'ensemble des techniques SdF proposées, une validation, analyse et comparaison pour déterminer les principaux avantages et inconvénients doivent être menées. C'est l'objet du chapitre suivant qui propose une validation de l'approche globale proposée dans le cas d'un système auto-organisé multi-nœuds reconfigurables à base de NoC RKT.

Chapitre 4

Validation expérimentale des concepts

Introduction

Dans ce chapitre, une validation du mécanisme d'auto-organisation tolérant aux fautes pour MPSoC communicant en réseau est présentée. Pour réaliser cette validation, un réseau de nœuds reconfigurables a été développé et permet la délocalisation de tâches en cas de détection d'erreurs au cours de fonctionnement. Pour cela, il a été réalisé une implémentation de trois nœuds à base de plateformes FPGA *ML507*, *ML605* et *NEXYS3* afin d'obtenir un système hétérogène, où la simulation de détections d'erreurs permet l'analyse des caractéristiques du concept proposé. De plus, une validation du fonctionnement des réseaux sur puce sûre de fonctionnement NoC RKT intégrant un système hybride de détection/localisation d'erreurs décrit dans le chapitre précédent est détaillé, prouvant ainsi les concepts proposés par simulations et co-simulations. Une analyse sur la synthèse et les performances temporelles, ainsi que l'évaluation des capacités de localisation des erreurs sont également décrites.

4.1 Réseau de communication inter-nœuds

L'évolution importante des technologies sans fil permet le développement de nouvelles approches dans le domaine des communications à courtes distances afin de répondre à la demande croissante de systèmes embarqués plus performants, sécurisés, sûr de fonctionnement, et consommant un minimum d'énergie [YMG08]. Après les réseaux cellulaires ou autres réseaux locaux, nous disposons d'une nouvelle architecture de réseau : les Réseau de Capteurs Sans Fil (RCSF) (ou WSN pour *Wireless Sensor Network*) [YSM09].

Un réseau de capteurs sans fil est un réseau *Adhoc* composé d'un grand nombre de nœuds (*node*) miniaturisés et autonomes (nœuds capteurs) permettant la collecte d'informations diverses. Cette architecture est présente dans un nombre important de domaines (militaire, environnemental, sécurité, médical, etc.) [ASSC02]. Une particularité du réseau est la non-obligation de connaissance de la position des nœuds dans l'espace. Par conséquent, la recherche s'est souvent axée sur la détection et le positionnement des nœuds [VWS+12] ainsi que sur l'ordonnancement et le routage des données, que ce soit pour

de petits ou de grands réseaux [Kon11]. Cependant, ces réseaux sont adaptés pour assurer l'intercommunication d'un réseau auto-organisé de MPSoC. En effet, l'un des intérêts des nœuds de ce type de réseau sans fil tel que le protocole *Zigbee* est leur autonomie tant du point de vue de leur alimentation que de leur gestion. Il est donc évident que la consommation électrique doit être minimum et la capacité d'adaptation importante.

4.1.1 Protocole Zigbee

La technologie de communication sans-fil *Zigbee* est principalement caractérisée par une haute résistance aux bruits, une consommation d'énergie réduite, un débit de transfert de données limité et un faible coût. Ce protocole offre des spécifications attractives, principalement pour les réseaux nécessitant des communications bas débit tel que les réseaux de capteurs sans fils (WSN). *ZigBee* est une norme de transmission de données sans fil gérée par la communauté "*Zigbee alliance*" conçu pour transporter des données de manière fiable et sécurisée à travers un environnement bruité des radios fréquences (ondes hertziennes). Moins connue que les protocoles *Bluetooth* et *Wi-Fi*, sa très faible consommation électrique et ses coûts de production très bas en font une solution idéale pour la domotique, les réseaux de capteurs pour un large spectre d'applications telles que la surveillance de la santé des structures ou le monitoring multi-capteurs [S.K11], etc.. Elle tire tous les avantages de la norme *IEEE* 802.15.4 et opère dans les bandes de fréquences mondiales non réservées (donc libres de licences).

La norme 802.15.4 est un protocole de communication défini par l'organisme de normalisation *IEEE* [IEE09]. Elle est destinée aux réseaux sans fil de la famille des Low Rate Wireless Personal Area Network (LR WPAN) du fait de leur faible consommation, de leur faible portée et du faible débit des dispositifs utilisant ce protocole. Elle est utilisée par de nombreuses implantations de transmission de données basées sur des protocoles propriétaires ou sur IP, comme la norme *ZigBee*. Les caractéristiques des *LR WPAN* sont :

- La formation d'un réseau de type étoile ou maillé,
- L'allocation d'une adresse de 16 bits ou de 64 bits,
- La faible consommation d'énergie,
- L'utilisation de 16 canaux dans la bande de fréquence de 2.4 à 2.4835 GHz.

Les débits autorisés sont de 250 Kbits/s, mais c'est véritablement sa très faible consommation électrique qui en fait son atout. La norme *Zigbee* fonctionne sur la bande des fréquences situées à 2,4 GHz et sur 16 canaux. Sa portée est supérieur à 100 avec possibilité de rajouter des modules pour accroitre son champ d'action. Cette technologie a pour but la communication de courte distance telle que le propose déjà la technologie *Bluetooth*, tout en étant moins chère, plus simple et plus robuste à un environnement bruité. Le tableau 4.1 compare les principales caractéristiques du protocole *Zigbee* avec d'autres protocoles utilisant la bande de fréquences située à 2,4 GHz comme le *Wifi* et le *Bluetooth*. A travers ce comparatif, nous remarquons que le protocole *Zigbee* est plus performant dans la majorité des critères. Seule la vitesse de transfert est plus faible que ses concurrents. Il faut également préciser que certains modules utilisant la norme *ZigBee*

CaractéristiquesProtocoles	Zigbee	Blutooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g/n/n-draft
Besoins mémoire	4-32 Kb	250 Kb+	1 Mb+
Autonomie avec pile	Années	Jours	Heures
Nombre de noeuds	65 000+	7	32
Vitesse de transfert	250 Kb/s	1 Mb/s	11-54-108-320 Mb/s
Portée	100 à 1600 m	10 à 100 m	300 m

TABLE 4.1 – Comparaison des protocoles *Zigbee*, *Wi-fi* et *Bluetooth*.

ont une portée allant jusqu'à 1600 mètres.

4.1.2 Topologie utilisée

Un réseau *Adhoc* est un ensemble de nœuds équipés d'une interface de communication sans fil dont la particularité est la création de réseaux temporaires ne nécessitant aucune architecture réseau fixe, et où chaque nœud est indépendant. C'est-à-dire, qu'il y a aucune centralisation des informations. Contrairement aux réseaux de capteurs, les réseaux *Adhoc* n'accordent pas autant d'importance à la consommation d'énergie. En effet, ces derniers ont pour priorité les optimisations visant à améliorer la qualité de service comme le débit, la bande passante ou la mobilité des nœuds. Cette mise au second plan est due à la facilité de remplacement des batteries dans ce type de réseaux. De même, les communications se font à un débit moindre car l'objectif est de préserver la durée de vie des nœuds. De plus, les transmissions de données s'effectuent qu'à un certain intervalle de temps. Un tel réseau est en évolution constante car sa structure dépend de la position des nœuds, de la puissance de transmission ou encore des interférences entre les canaux de communication. Ce changement constant de structure va entraîner une déconnexion fréquente des nœuds. Les réseaux *Adhoc* sont principalement caractérisés par :

- Des contraintes énergétiques : Les nœuds sont alimentés par leur propre source d'énergie. Cette caractéristique doit être prise en compte dans toute action faite par le système.
- Une bande passante limitée : Le réseau est basé sur la communication sans fil où l'utilisation d'un canal de communication est partagée.
- Une topologie dynamique : La topologie des réseaux *Adhoc* change constamment dû au déplacement permanent des nœuds.
- Une sécurité physique limitée : Les réseaux *Adhoc* sont plus sensibles aux attaques qui menacent les données transmises au vu du moyen de communication.

De manière générale, ce type de réseau est composé d'un coordinateur, permettant l'interface entre le réseau et l'utilisateur, de routeurs, afin d'acheminer les données vers le coordinateur, et de nombreux *End Devices*, qui ont pour rôle de récolter les informations. Cependant, il est possible de modifier cette structure standard afin de l'adapter aux

FIGURE 4.1 – Module de communication *Xbee*.

besoins. De ce fait, nous utilisons uniquement des nœuds de type *End Device* pour créer un réseau *Ad-hoc*. Par conséquent, le réseau n'est plus dépendant d'un seul nœud. En effet, la mise en hors ligne du coordinateur dans une structure habituelle a pour conséquence la chute du réseau entier. Dans notre cas, chaque nœud est indépendant de ses voisins et la défaillance d'un nœud n'empêche aucunement le fonctionnement normal du reste du réseau. Dans cette configuration, un nœud peut être remplacé par n'importe quel autre sans changer la topologie du réseau. En combinant ce modèle de réseau sans fil avec la capacité de reconfiguration proposée par certain type FPGA, nous obtenons un système communicant capable de gérer les tâches entre les différents nœuds du réseau en créant un système inter-nœuds intelligent de répartition des tâches grâce au mécanisme d'auto-organisation.

4.1.3 Module Xbee et interface

Les modules utilisés sont des modules *MaxStream XBee* [Dig09] (photo en figure 4.1), employant le protocole *Zigbee* et dont les caractéristiques techniques sont détaillées à la figure 4.2. Le choix de l'utilisation de ces modules *Zigbee* est leur grande facilité de mise en œuvre. En effet, une des possibles utilisation est la communication directe par liaison série au protocole Universal Asynchronous Receiver Transmitter (UART). Il est ainsi aisé d'envoyer ou de récupérer des informations sur le réseau sans fil. Plus précisément, l'envoi d'une donnée s'effectue en transmettant directement l'information sur une broche *TX* du module selon le protocole défini par le constructeur. La figure 4.3 illustre la trame standard utilisée pour la communication entre une plateforme FPGA et un module de communication *Xbee* au protocole *Zigbee*.

Au niveau des communications, lors de l'envoi d'un nombre important de données, les bits sont regroupés en octets. Chaque octet est encadré d'un bit de *Start* à l'état bas et d'un bit de *Stop* à l'état haut. Il est également nécessaire de préciser qu'il n'y a pas de bit de parité pour ce mode de transmission. De plus, pour ce protocole le premier bit de donnée D_0 est le bit de poids faible et le dernier bit D_7 le bit de poids fort. Si par exemple la quantité de données à envoyer est de 32 bits, 4 trames d'un octet au protocole UART sont envoyées successivement avec à chaque fois les deux bits de synchronisation *Start* et *Stop*. Les trames doivent être obligatoirement transmises les unes derrières les autres, sans espacement entre les 4 trames. Ainsi, pour 32 bits de données utiles à transmettre, 40 bits au total sont générés.

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	up to 100 ft. (30 m)	Up to 300' (100 m)
Outdoor RF line-of-sight Range	up to 300 ft. (100 m)	Up to 1 mile (1500 m)
Transmit Power Output (software selectable)	1mW (0 dBm)	60 mW (18 dBm) conducted, 100 mW (20 dBm) EIRP*
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 - 115200 bps (non-standard baud rates also supported)	1200 - 115200 bps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	If PL=0 (10dBm): 137mA(@3.3V), 139mA(@3.0V) PL=1 (12dBm): 155mA (@3.3V), 153mA(@3.0V) PL=2 (14dBm): 170mA (@3.3V), 171mA(@3.0V) PL=3 (16dBm): 188mA (@3.3V), 195mA(@3.0V) PL=4 (18dBm): 215mA (@3.3V), 227mA(@3.0V)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 µA	< 10 µA
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
Operating Temperature	-40 to 85° C (industrial)	-40 to 85° C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector	Integrated Whip, Chip or U.FL Connector

FIGURE 4.2 – Caractéristiques techniques des modules *Xbee*.

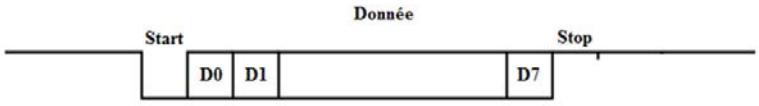


FIGURE 4.3 – Trame de transmission UART pour la communication avec les modules *Xbee* utilisés.

Pour les différentes manipulations, plusieurs paramétrages des modules sont nécessaires. La figure 4.4 présente l'environnement logiciel de paramétrage des modules. Ainsi, les paramètres *Channel*, *PanID*, adresses source et destination ainsi que le débit sont définis et adaptés à la création de notre système réseau. Ces paramètres permettent de créer une transmission entre les modules, sachant que le *Channel*, le *PanID* et le débit doivent être identiques et que les *Adresses de Destination* et de *Source* doivent correspondre pour établir des communications. La vérification des configurations réseau est assurée par un analyseur de réseau (*Network Analyser MICROCHIP*) où les trames *Zigbee* circulant dans le réseau peuvent être récupérées. Il est alors possible d'en analyser le contenu et d'y retrouver les différents paramètres définis ainsi que les données transmises. La figure 4.5 donne un exemple de trame *Zigbee* relevée à l'aide de l'analyseur de réseau [Mic08]. A titre d'exemple, dans cette trame, la donnée transmise se situe dans le champ *Invalid Data* et à pour valeur hexadécimale 0xB2.

En termes de sécurité des échanges, les modules *XBee* sont également capables d'effectuer un cryptage de type Advanced Encryption Standard (AES) qui permet de masquer les données transmises. Si on considère l'exemple de la trame de la figure 4.6, plusieurs

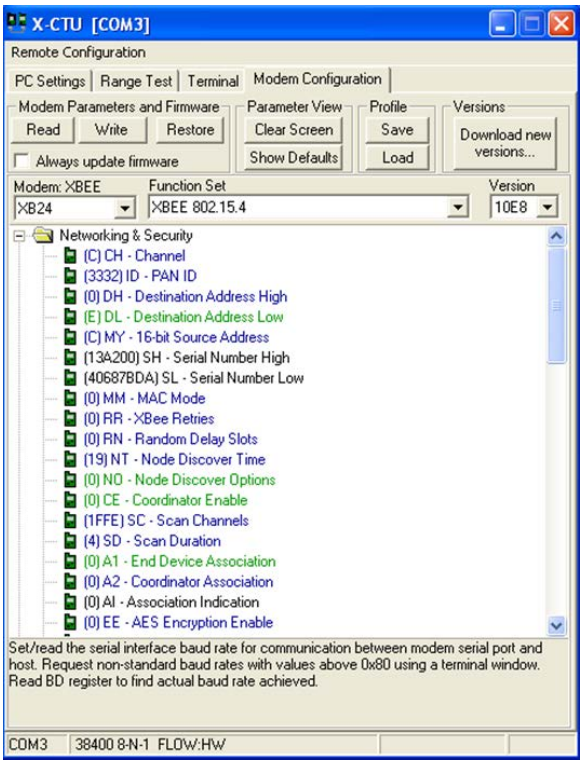


FIGURE 4.4 – Interface de configuration des modules *Xbee*.

Frame	Time(us)	Len	MAC Frame Control					Seq	Dest	Dest	Source	Invalid Data	FCS		
	+4256		Type	Sec	Pend	ACK	IPAN	Num	PAN	Addr	Addr	0x70 0x00	RSSI	Corr	CRC
00277	+1119328	14	DATA	N	N	Y	Y	0x6C	0x3332	0x000C	0x000E	0xB2	-06	0x6C	OK

FIGURE 4.5 – Exemple d'une trame au protocole *Zigbee*.

Frame	Time(us)	Len	MAC Frame Control					Seq	Dest	Dest	Source Address	Encrypted Data					FCS		
	+42112		Type	Sec	Pend	ACK	IPAN	Num	PAN	Addr		0x00	0x00	0x02	0x04	0x00	RSSI	Corr	CRC
00071	+2169984	27	DATA	Y	N	Y	Y	0xF8	0x3332	0x000E	0x0013A20040687BDA	0x80	0x23	0xEF	0x9F	0x54	-13	0x6B	OK

FIGURE 4.6 – Exemple d'une trame au protocole *Zigbee* utilisant un cryptage AES.

paramètres tels le *PanID* et l'adresse de destination sont visibles. Par contre, l'adresse source ainsi que les données transmises sont cryptées et indéchiffrables par l'analyseur de réseau. Seul le module *Xbee* correspondant à l'adresse de destination pourra décrypter la donnée.

Au niveau de la consommation électrique du réseau, le tableau 4.2 détaille les résultats des différentes consommations d'émissions et réceptions, avec ou sans cryptage pour différents débits (9600, 38400 et 115200 Bps). L'alimentation des modules est fixée à 3.82V. De plus, l'envoi de données est effectué par intervalles de temps de 500 fronts d'horloge entre chaque trame. On peut constater que lors de l'émission, l'option cryptage augmente légèrement la consommation. Cette dernière ne varie pas au niveau de la réception lorsque l'option cryptage est actif.

L'ensemble des paramétrages ont permis de mettre en œuvre un réseau de nœuds

Débit (Bps)	**9600**	**38400**	**115200**
Emission (mW)	168,1	190,4	189,1
Emission crypté (mW)	179,5	196,7	196,7
Réception (mW)	208,1	212	215,8
Réception crypté (mW)	208,1	212	215,8

TABLE 4.2 – Analyse de la consommation électrique des modules *Xbee*.

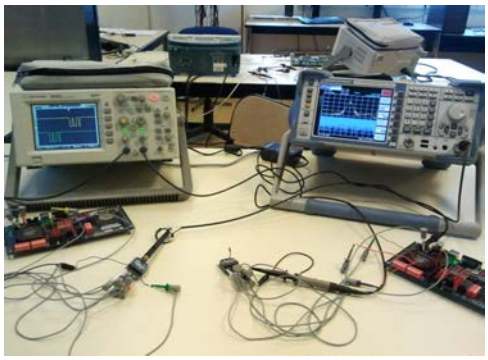


FIGURE 4.7 – Manipulation de la mise en place de la communication au protocole *Zigbee*.

FIGURE 4.8 – Manipulation de la mise en place d'une communication dans le cadre de l'envoi de données cryptées.

reconfigurables au protocole *Zigbee* comme illustré par les figures 4.7 et 4.8. Le fonctionnement et la validation du réseau ont pu être vérifiés expérimentalement selon plusieurs options possibles.

4.1.4 Protocole d'échange des données

La communication entre deux nœuds du réseau se fait de manière simple. Lors d'une détection d'erreurs de la partie communication sur puce, le nœud partiellement défaillant transmet sur le réseau un octet précis informant les autres nœuds d'une demande de délocalisation de tâches pour cause de défaillance. Lorsqu'un nœud est disposé à assurer cette délocalisation en veillant au préalable de sa capacité de réaliser cette délocalisation (ressources disponibles et accès aux données), il établit une communication pour disponibilité par l'intermédiaire de l'envoi de champs informationnels précis. Il s'en suit des échanges d'informations et de données entre les nœuds solliciteur et de substitution. C'est

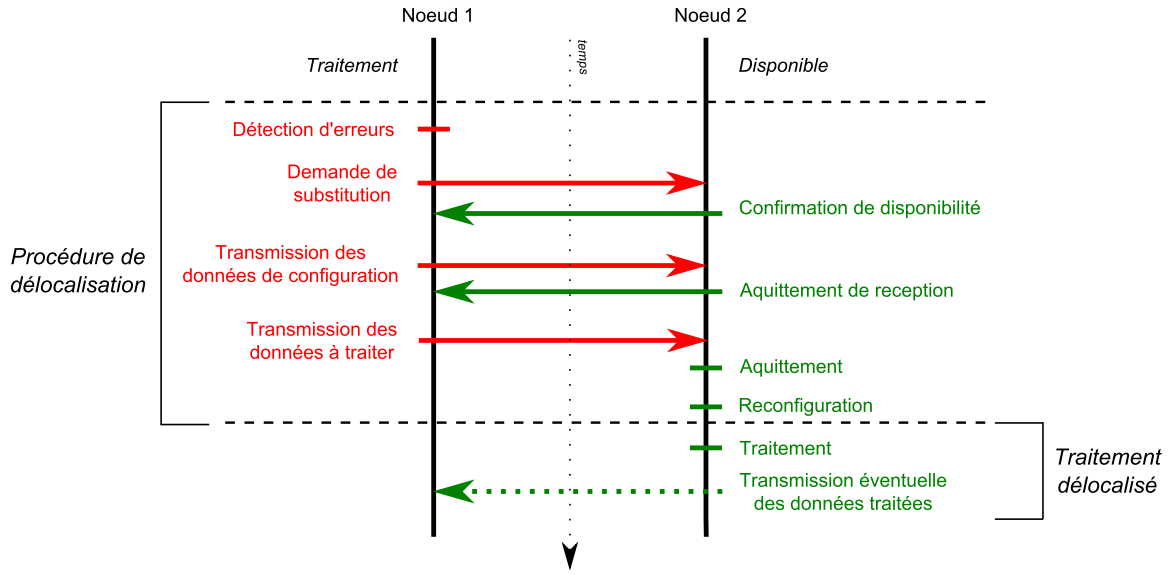


FIGURE 4.9 – Protocole de communication inter-nœuds pour la délocalisation d'une tâche.

au cours de ces échanges que les données de configuration de la tâche à délocaliser et des données de traitement associées sont transmises. La figure 4.18 représente le protocole de communication pour la délocalisation de traitement entre deux nœuds reconfigurables du réseau auto-organisé.

4.2 Validation fonctionnelle

Dans cette section, les méthodologies de validation des différentes parties du système auto-organisé tolérant aux fautes pour MPSoC sûre de fonctionnement sont développées. La première consiste à valider et à évaluer la capacité d'auto-organisation de notre système. Pour se faire, une architecture se basant sur le concept d'auto-organisation proposé est implémentée, en y appliquant une tâche de traitement de filtrage d'images en s'assurant préalablement de la fonctionnalité de la reconfiguration dynamique partielle. En simulant une détection d'erreurs, le comportement du système est observé dans l'objectif de valider son fonctionnement. Dans un second temps, différentes simulations sont proposées afin de valider le mécanisme de tolérance aux fautes basé sur le système auto-organisé décrit dans le chapitre précédent.

4.2.1 Gestion de la reconfiguration

Deux choix sont disponibles afin de gérer la primitive ICAP. La première est d'implémenter un coprocesseur et la seconde est de réaliser une description matériellement d'un contrôleur. Dans notre cas d'étude, cette dernière option est développée car moins complexe à implémenter, plus rapide et moins coûteuse en termes de ressources logiques occupées dans le FPGA. Dans un premier temps, la reconfiguration est manuelle, c'est-

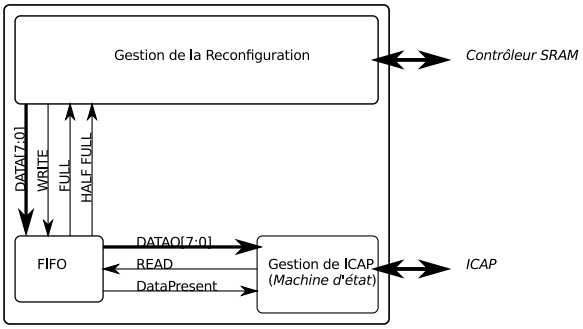


FIGURE 4.10 – Schéma de la gestion de la reconfiguration partielle mis en place sur plateforme *ML507*.

à-dire qu'une reconfiguration partielle s'exécute lors de l'activation d'un bouton poussoir (*BP*). Le contrôleur de reconfiguration fonctionne de la manière suivante. Lors d'une reconfiguration partielle (activation par impulsion d'un *BP*), le module de gestion de la reconfiguration va récupérer octet par octet le *bitstream* présent dans la mémoire pour le transmettre au bloc de gestion d'ICAP par l'intermédiaire d'une structure de type First In First Out (FIFO). Lorsque la première donnée se trouve dans cette dernière, la sortie *DataPresent* est activée pour informer au sous-ensemble suivant qu'un nouveau *bitstream* partiel va devoir être implémenté. L'architecture de gestion de l'ICAP permet uniquement de mettre en forme les signaux nécessaires à la primitive. La figure 4.10 représente le schéma bloc de la fonction assurant la reconfiguration partielle dans le cas de la plateforme *ML507*.
Lorsque le signal *WRITE* est actif, la donnée présente sur *DataI* est mémorisée dans la FIFO. Lorsque le signal *READ* est actif, la donnée présente sur *DataO* est lue dans la FIFO. Les signaux *HALF FULL* et *FULL* permettent respectivement d'indiquer l'état de la FIFO moitié pleine ou pleine (tous deux actifs à l'état haut).

4.2.2 Reconfiguration dynamique partielle

Dans un premier temps, la validation du fonctionnement de la reconfiguration dynamique partielle du système est développée afin de vérifier les structures de communication entre l'architecture développée et l'interface de reconfiguration (ICAP) proposée par *Xilinx*. Pour réaliser cette validation, une implémentation successive de différents IP dans la même région reconfigurable est mise en place. La figure 4.11 illustre la manipulation de validation dans le cas d'un exemple traité où deux tâches correspondant respectivement à un compteur et un décompteur sont successivement implémentées.

L'architecture présente sur le FPGA reste très similaire à celle présentée précédemment. Elle est composée :

- d'une partie statique, permettant la gestion de la reconfiguration, la communication externe ainsi que l'accès à la mémoire où sont stockés les fichiers de reconfigurations,
- d'une partie dynamique où les différentes tâches sont implémentées successivement.

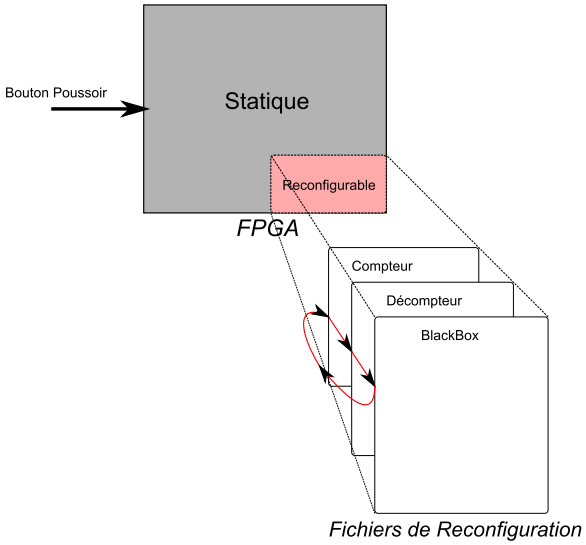


FIGURE 4.11 – Illustration de la reconfiguration dynamique partielle d'un compteur/décompteur.

Des signaux de contrôle sont également intégrés ainsi qu'un bus de données entre les deux ensembles afin de transférer les données d'une tâche reconfigurable à l'autre. Ces signaux mettent en "*pause*" la tâche à l'instant précédent la reconfiguration, afin d'en extraire les données utiles au module suivant, puis relancent le nouveau bloc IP implémenté après avoir fourni ces données utiles.

Afin d'obtenir à la fois une manipulation permettant de valider la reconfiguration par un résultat visuel, nous réalisons trois architectures différentes effectuant des actions sur des Light-Emitting Diode (LED) disponibles sur plateforme de développement *ML507*. Nous retrouvons donc une architecture qui incrémente de 1 une donnée sur 8 bits avec affichage sur 8 LED et à intervalle de temps régulier, de manière à être perceptible pour l'œil. La deuxième architecture décrémente cette même valeur de 1, et la dernière IP correspond à une ("*BlackBox*"), où aucune architecture est présente. La figure 4.12 illustre le développement de la RD partielle sous l'environnement *PlanAhead* proposé par *Xilinx*. Cet outil permet de définir la région reconfigurable pour un nombre de tâche défini. Sur cette figure 4.12, nous retrouvons les 3 IP présentés, les fichiers de reconfiguration générés pour ces modules permettant uniquement la reconfiguration dans la région reconfigurable sélectionnée.

La prise de décision de la reconfiguration partielle du FPGA s'effectue à l'aide d'un bouton poussoir. Lors de la première pression, le système implémente un compteur. Lors d'une seconde pression, le compteur s'arrête, sauvegarde les données, implémente un décompteur et restaure les données. Lors d'une troisième pression la *BlackBox* est implantée, puis le cycle recommence dans cet ordre à chaque action sur le bouton poussoir comme l'illustre la figure 4.13. Les trois fichiers de reconfiguration sont transmis préalablement

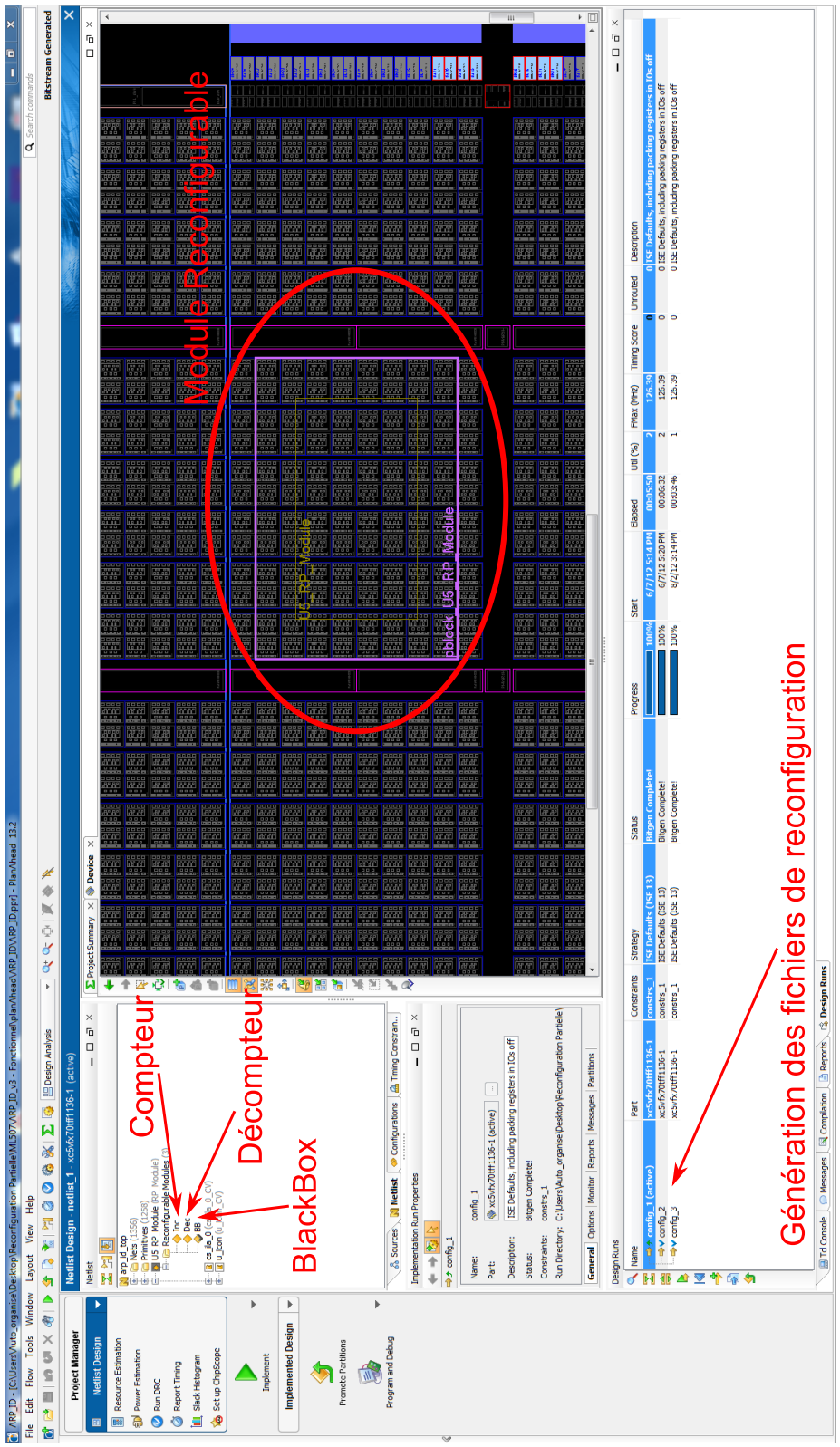


FIGURE 4.12 – Développement de la RD partielle sous environnement *PlanAhead*.

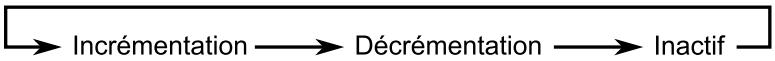
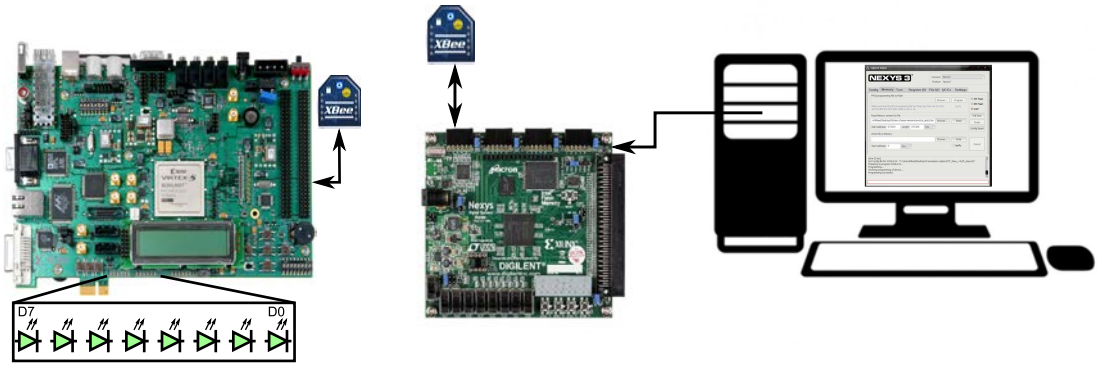


FIGURE 4.13 – Cycle d'implémentation des tâches.



(a) Plateforme ML507. (b) Plateforme NEXYS3.

FIGURE 4.14 – Validation expérimentale de la reconfiguration dynamique partielle.

au nœud reconfigurable grâce à la communication sans fil *Zigbee* via les modules *Xbee* et sont stockés dans sa mémoire.

Lors de cette manipulation, le fonctionnement de la communication inter-nœud ainsi que la gestion des *bitstreams* dans la mémoire RAM sont également validés. En effet, afin de fournir les 3 fichiers de reconfiguration à la plateforme *ML507*, ces derniers sont transférés en passant par un second nœud matérialisé par une carte *NEXYS3* comme illustré à la figure 4.14. La particularité de cette carte est son accès simplifié à une mémoire externe associée par l'intermédiaire de l'application logicielle *ADEPT* (voir figure 4.15). Lorsque les 3 *bitstreams* sont présents dans la mémoire, ils sont envoyés à travers le réseau sans fil *Zibgee* et stockés dans la mémoire de la plateforme *ML507*.

4.2.3 Délocalisation de tâches

La validation du mécanisme de délocalisation est explicitée par une application de traitement d'images. L'objectif du système consiste alors à réaliser en continue un traitement correspondant à un filtrage *moyenneur* des composants pixels couleurs codés 8 bits d'une image de dimension 194 x 141 pixels, afin d'obtenir une image lissée en niveau de gris. Le système réalise ce traitement par l'un de ses nœuds et doit maintenir ce traitement malgré une possible défaillance du NoC du nœud réalisant le traitement. Dans notre contexte, nous cherchons volontairement à simuler une panne lors du traitement afin que ce nœud défaillant puisse auto-gérer le problème selon le concept d'auto-organisation entre nœuds en réseau. On se focalise principalement dans le cas considéré, à une auto-organisation par l'intermédiaire d'une délocalisation du traitement et des données. Le chemin de données du traitement *moyenneur* d'images à réaliser est détaillé dans la figure 4.16.

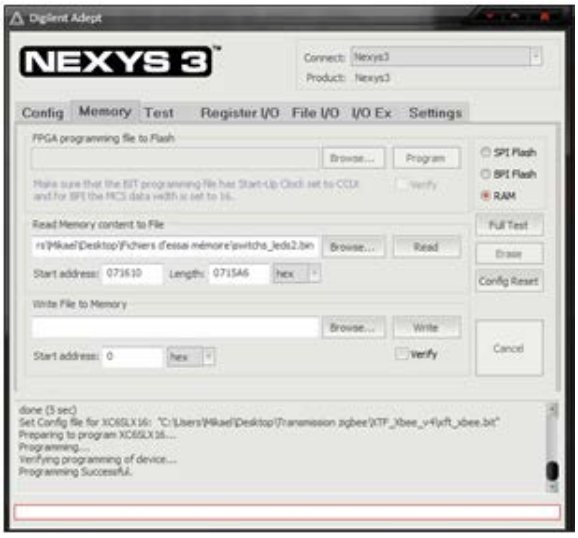


FIGURE 4.15 – Logiciel *DIGILENT ADEPT* de gestion mémoires SRAM de la plateforme *NEXYS3*.

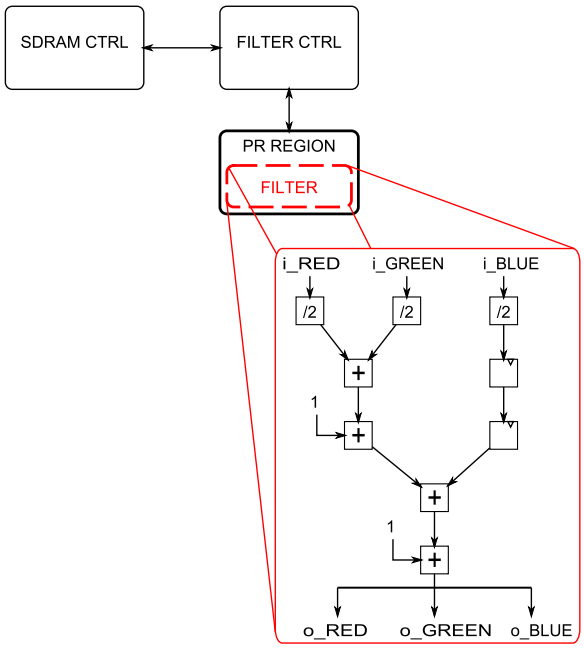


FIGURE 4.16 – Chemin de données (*Data-path*) du filtre *moyenneur*.

La figure 4.17 est une photographie de la manipulation réalisée. On y retrouve deux plateformes de développement (*NEXYS3* et *ML507*) associées à deux modules de transmission sans fil *Maxstream Xbee* ainsi qu'un analyseur de réseau qui va permettre de visualiser les informations échangées entre les deux nœuds considérés du système réseau. La plateforme *NEXYS3* ne permettant pas une auto-reconfiguration (compte tenu de la technologie du circuit *Spartan6* disponible), le nœud va devoir exporter sa tâche vers un autre nœud reconfigurable (la plateforme *ML507* permet un auto-reconfiguration étant

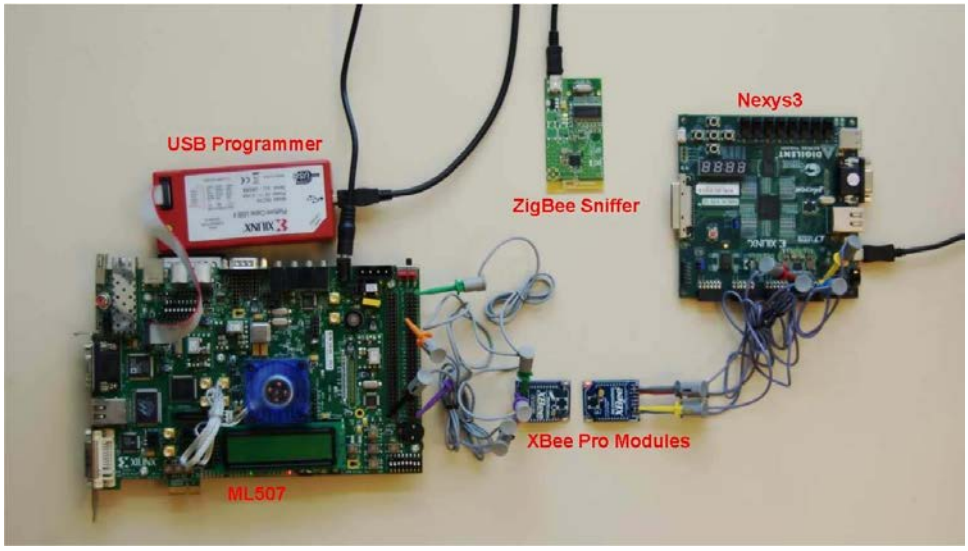


FIGURE 4.17 – Plateforme de démonstration.

donnée la famille *Virtex V* disponible) en commençant par lui transmettre le fichier de reconfiguration du filtre.

Le protocole d'échanges entre ces deux nœuds matérialisés par les deux plateformes considérées est détaillé par la figure 4.17. La communication entre deux plateformes repose sur celle définie entre nœuds du réseau. Dans notre cas d'étude, dès détection d'erreurs le nœud associé à ces erreurs (matérialisé par une plateforme FPGA) transmet sur le réseau un octet précis informant les autres nœuds d'une demande de délocalisation. Lorsqu'un nœud est disposé à établir la communication après avoir établi sa capacité à réaliser la tâche demandée, il envoie un octet spécifique dans le réseau pour faire savoir au nœud fautif sa disponibilité à recevoir la délocalisation. Ensuite, les deux plateformes vont alors échanger les informations utiles que sont le fichier de reconfiguration de la tâche à délocaliser, puis, si nécessaire les données à traiter ou associées. La figure 4.18 représente le protocole de communication de demande de réaffectation de tâche au sein du réseau mis en place dans le cas du système basé sur deux plateformes pour réaliser en alternance le traitement de filtrage *moyenneur* d'images.

Dans cet exemple précis, la plateforme *NEXYS3* réalise en continue le filtrage où nous simulons une panne. L'objectif est de transmettre le fichier de reconfiguration (*bitstream*) ainsi que l'image à filtrer au nœud capable de s'auto-reconfigurer. Après avoir établi la communication avec ce nœud (*ML507*), la plateforme *NEXYS* transmet dans un premier temps son *bitstream* puis, après avoir reçu confirmation de sa réception, envoie l'image à traiter. Une fois cette étape d'échange de donnée terminée, le nœud sollicité se reconfigure, exécute le traitement de filtrage d'images et transmet les résultats au nœud demandeur. La dernière étape consiste donc à envoyer les résultats pixels filtrés au nœud "défaillant" pour exécuter un tel traitement. La figure 4.19 représente le schéma bloc de l'architecture du nœud auto-organisé reconfigurable implantée dans la plateforme *ML507*. La figure 4.20

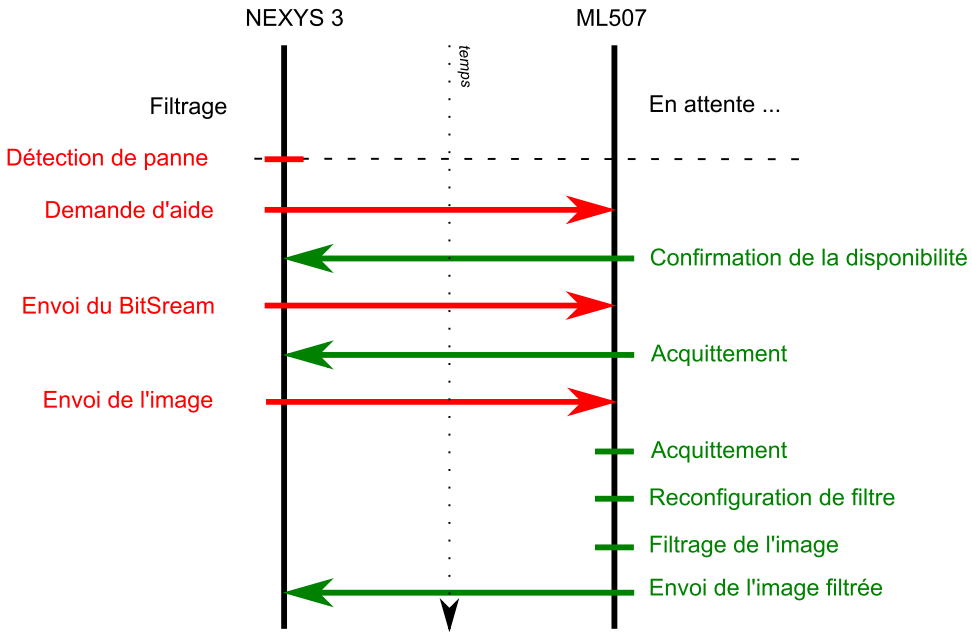


FIGURE 4.18 – Application du protocole de communication utilisé lors de la délocalisation du traitement de filtrage d'images.

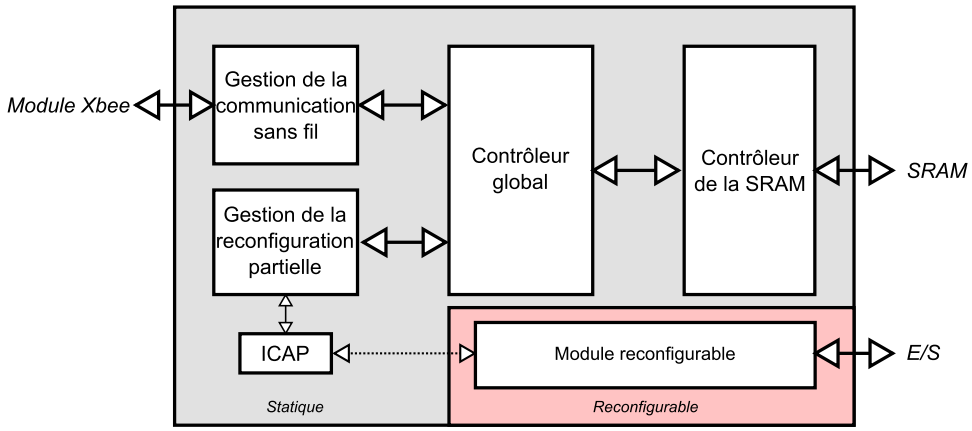


FIGURE 4.19 – Schéma bloc de l'architecture auto-reconfigurable de la plateforme *ML507*.

représente le schéma de la manipulation ainsi que les détails de l'architecture de traitement développée et implantée dans le nœud auto-organisé reconfigurable via la plateforme *ML507*. Les figure 4.21a et 4.21 représentent respectivement les graphes de gestion et de contrôle des nœuds implantés sur les plateformes *NEXYS3* et *ML507*.

D'un point de vu expérimental, dans l'état par défaut, le système réalise à intervalle régulier le filtrage d'une image stockée dans la mémoire *NEXYS3* depuis une adresse précise. La pression d'un bouton poussoir spécifique simule une défaillance ou panne du module exécutant le traitement. Le nœud déclenche alors une demande de délocalisation du traitement dans un autre nœud du réseau en envoyant une requête via une trame précise sur le réseau sans fil, puis se positionne dans un mode d'attente. Lorsqu'une ré-

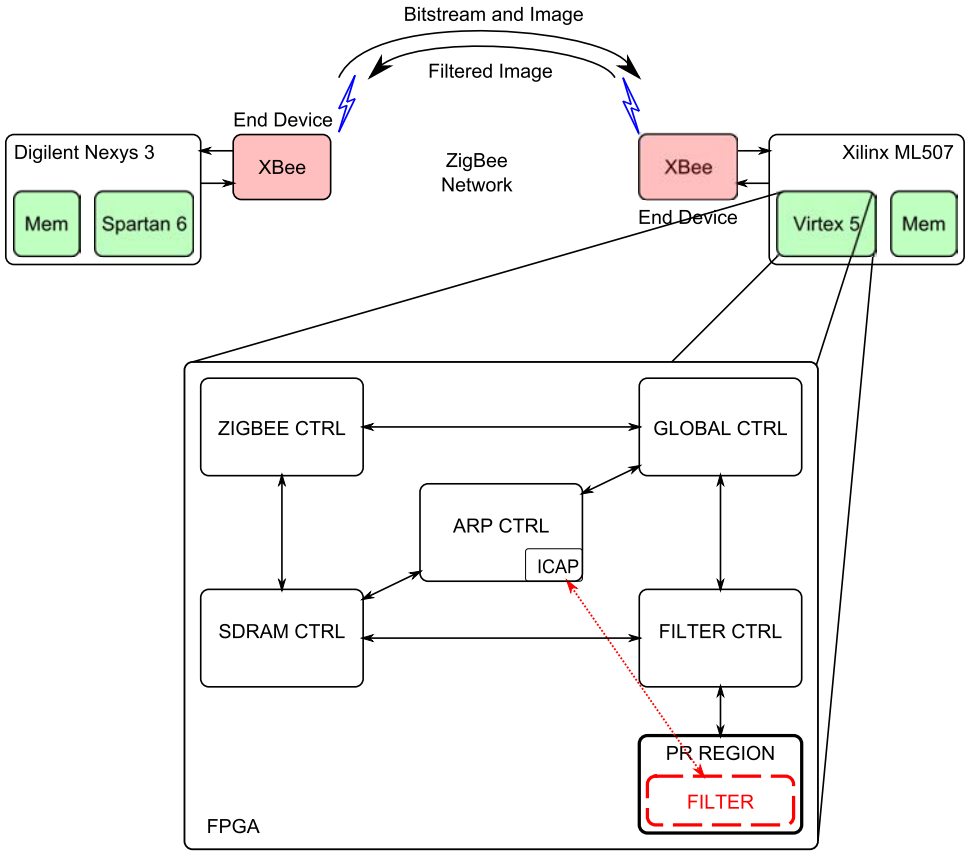
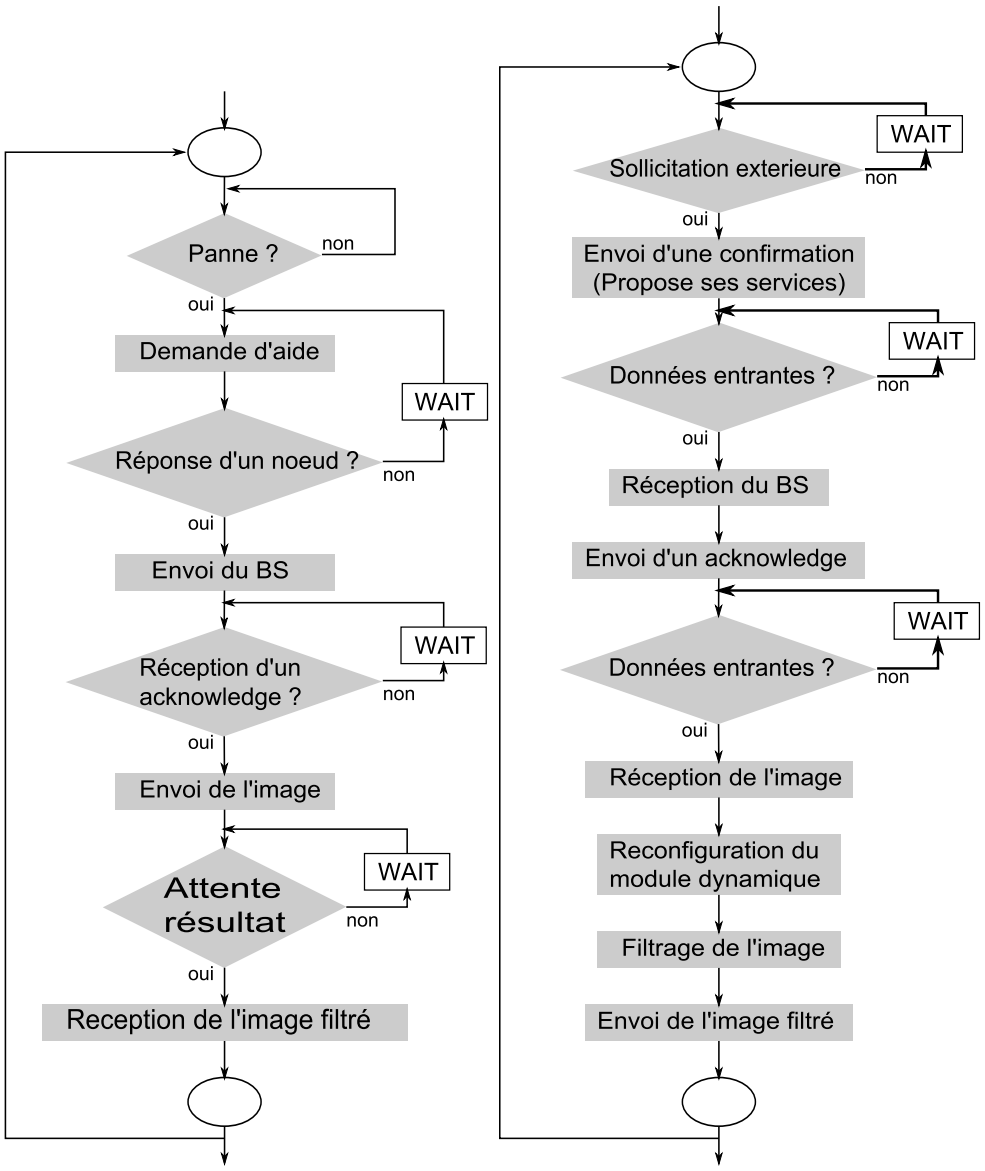


FIGURE 4.20 – Architecture de l'application implantée dans la plateforme *ML507*.

ponse est réceptionnée, le nœud demandeur transmet le fichier de reconfiguration du filtre *moyenneur* au nœud disponible pour la substitution. Une fois l'envoi terminé et après avoir reçu un acquittement pour confirmation, l'image à traiter est ensuite transmise. Après l'envoi du dernier octet de l'image à traiter, le nœud demandeur "défaillant" en traitement attend de recevoir les résultats pixels provenant du nœud de substitution qui a réalisé le traitement de manière décentralisée. A la réception, l'image filtrée est stockée dans la mémoire locale du nœud demandeur de la même façon et au même endroit que si la tâche avait été exécutée localement au nœud.

Du coté de la plateforme mettant en œuvre le nœud de substitution, l'état par défaut reste l'attente d'une sollicitation d'un nœud en difficulté. Lors de la réception d'une requête via une trame spécifique émise par un nœud défaillant, une confirmation de la capacité système à réaliser le traitement est émise sur le réseau. Cet acquittement est établi après la vérification de possibilité de stockage du *bitstream* de traitement (en l'occurrence ici l'architecture du *moyenneur*) et de la capacité de l'implanter de manière autonome. Dans un second temps, le nœud délocaliseur va attendre les différentes données (fichier de reconfiguration et images à traiter) en transmettant un accusé de réception (*acknowledge*) après la réception du *bitstream* (afin de bien distinguer les données de configuration et de traitement). Une fois l'ensemble des données reçues, le contrôleur principal commande



(a) Graphe de contrôle du nœud implanté dans la plateforme *NEXYS3*.

(b) Graphe de contrôle du nœud implanté dans la plateforme *ML507*.

FIGURE 4.21 – Validation expérimentale de délocalisation et de reconfiguration dynamique partielle de tâches.

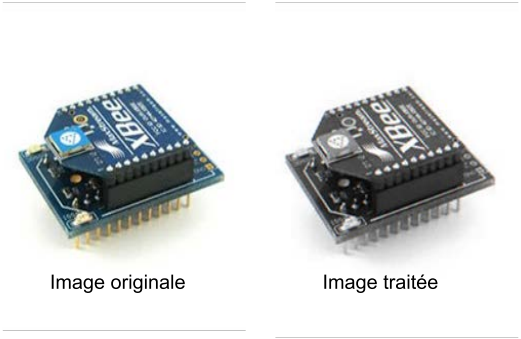


FIGURE 4.22 – Résultats du processus délocalisé de traitement d'images.

au bloc de gestion ICAP la réalisation d'une reconfiguration partielle de la zone reconfigurable avec le *bitstream* contenant le traitement à réaliser, c'est à dire la description du filtre *moyenneur*. Après avoir effectué le traitement, les résultats sont transmis et envoyés au nœud demandeur défaillant et le système reprend son fonctionnement en cours.

La figure 4.22 présente le résultat du traitement d'une image par le filtre *moyenneur* délocalisé dans un autre nœud. La partie de gauche représente l'image à traiter et celle de droite présente le résultat du filtrage image *moyenneur* délocalisé.

4.2.4 Mécanisme de test délocalisé

L'objectif de cette section est de valider plus spécifiquement le fonctionnement du mécanisme proposé de test d'un nœud afin de vérifier sa capacité à localiser une ou plusieurs erreurs au sein de son NoC et de transmettre les résultats obtenus. Pour cela, le comportement de la partie auto-organisation du système est simulé dans cette étape de validation.

Afin d'obtenir des données exploitables en termes de vérification du mécanisme, les différents vecteurs et réponses associés sont définis directement dans un *testbench*, ce qui permet une plus grande modularité des simulations. Cette souplesse au niveau de la génération des vecteurs permet d'établir un cycle de test où les résultats attendus sont connus, et par conséquent de vérifier l'exactitude des données produites par le système. Une fois le vecteur défini, les trames de données composées de paires de vecteurs de test et références, sont envoyées directement dans le NoC du nœud à travers les blocs *Wrapper* des ZGS à tester. Nous simulons donc la réception des données provenant du réseau de communication et passant originellement par la gestion de l'auto-organisation du nœud.

Dans le but de vérifier que la capacité de localisation d'une erreur, des tests par inversion volontaire de bits sont appliqués dans la référence transmise. En injectant une erreur à un endroit précis de la référence, la réponse et l'analyse par le système sont théoriquement connues. La figure 4.23 représente le vecteur de test utilisé, ainsi que la référence au vecteur et la réponse associée. La figure 4.24 est le résultat de simulation du mécanisme du point de vu du bloc *Wrapper*. Il est possible d'observer les différentes étapes décrites

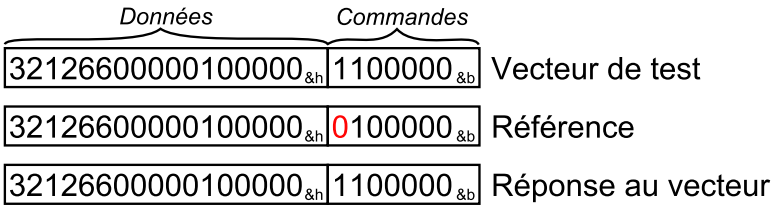
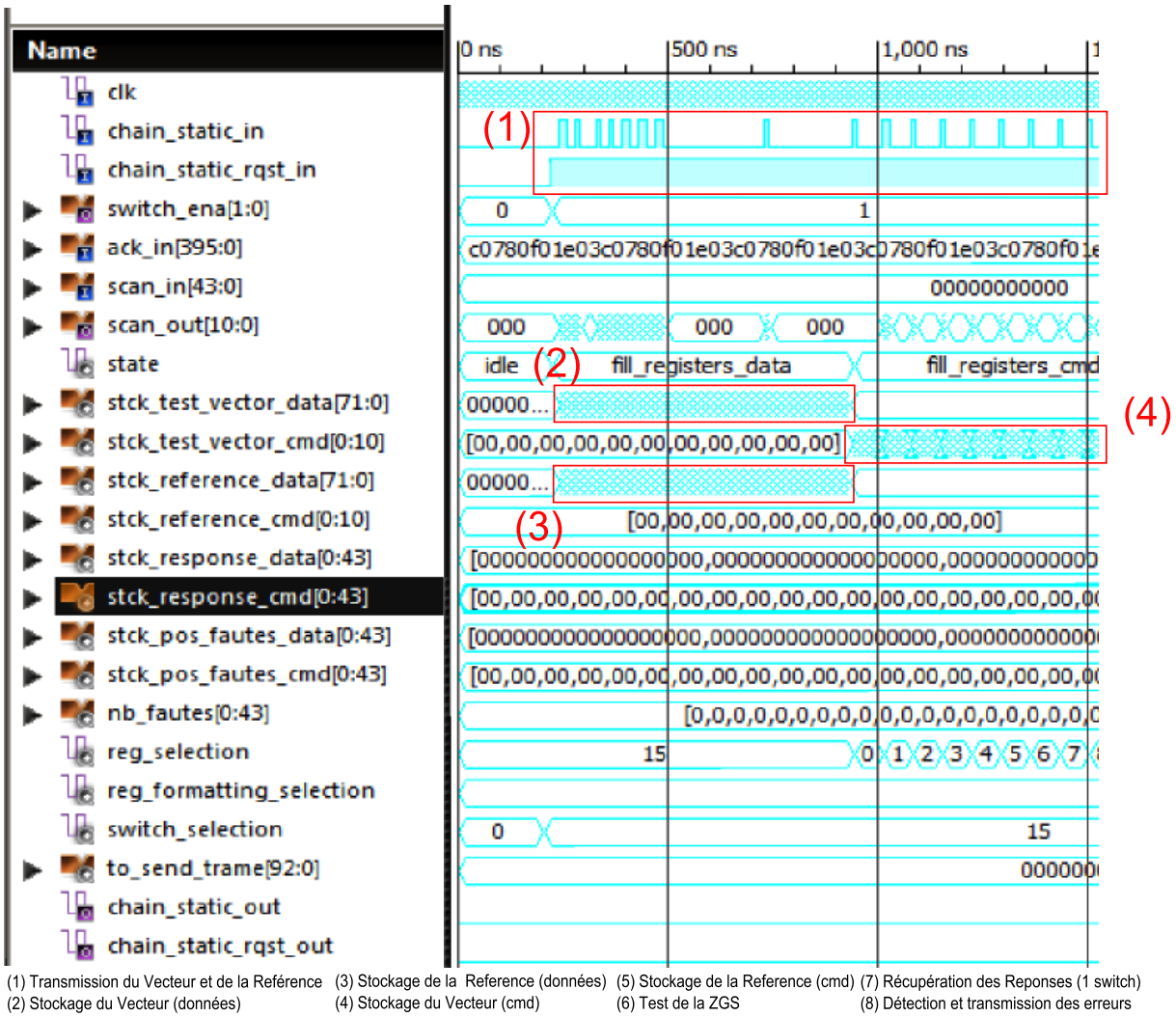


FIGURE 4.23 – Exemple de localisation d'une erreur injectée dans le vecteur référence.

précédemment comme suit :
- La première étape (1) montre la réception du vecteur de test en série via le signal *chain_static_in* ainsi que la réponse associée générée par le *testbench* et transmis au bloc *Wrapper* de la ZGS concernée (voir figure 4.24a).
- Dans les étapes (2) et (3), nous observons le stockage des parties *données* du vecteur dans un registre $stck\_test\_vector\_data[71:0]$ et de la référence dans un registre $stck\_reference\_data[71:0]$.
- Les étapes (4) et (5), décrivent les *signaux de commandes* qui sont sauvegardés respectivement dans les registres $stck\_test\_vector\_cmd[0:10]$ et $stck\_reference\_cmd[0:10]$ (voir figure 4.24b).
- L'étape noté (6) représente l'envoi du vecteur dans les différents blocs logiques de l'architecture de la ZGS à tester grâce aux signaux $switch\_enable[1:0]$ permettant le contrôle des EdT. Durant cette période, le bloc *Wrapper* se contente d'attendre la fin du traitement des données par les éléments architecturaux des routeurs.
- Lorsque l'ensemble des blocs des routeurs ont récupéré leurs réponses, ces dernières sont transmises au bloc *Wrapper* comme illustré en (7). Les réponses sont récupérées par le bloc *Wrapper*, routeur par routeur de la ZGS, pour analyses.
- Durant cette analyse, plusieurs erreurs sont localisées à l'étape (8) (voir figure 4.24c), où le signal $nb\_fautes[43:0]$ indique le nombre de fautes localisées par EdT du routeur testé. Les signaux $stck\_pos\_fautes\_data[0:43]$ et $stck\_pos\_fautes\_cmd[0:43]$ indiquent les positions des fautes dans les EdT suivant si elles se situent dans la partie *données* ou *commandes*.

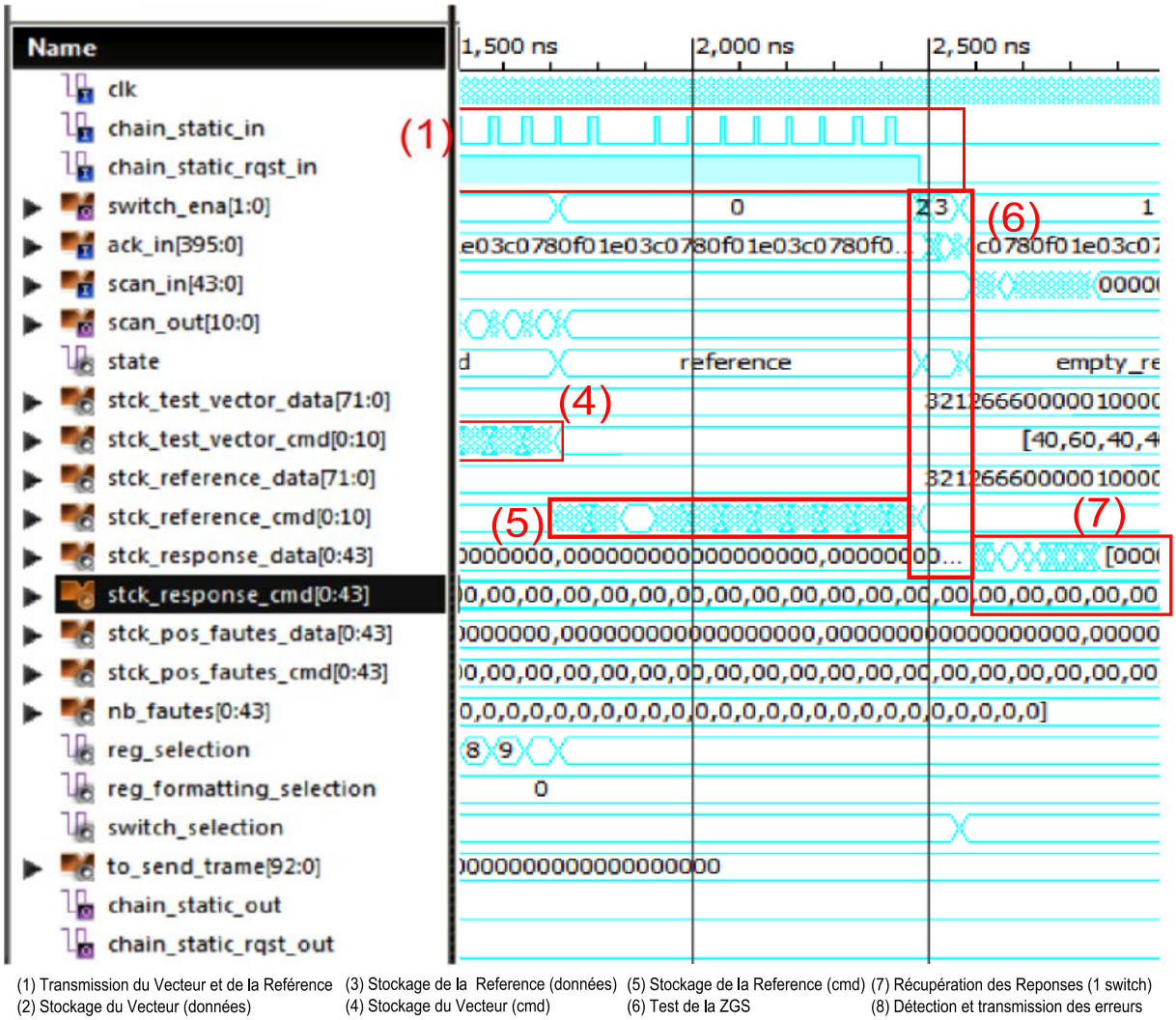
Au final, les positions des erreurs sont mises en forme puis envoyées à l'IP test afin d'être regroupées avec les résultats des autres routeurs de la ZGS testé. Pour cela, les trames formalisées sont envoyées vers le module de transmission via le signal *chain_static_out*.

La figure 4.25 présente les résultats de simulation du mécanisme de test au niveau d'un EdT. Sur cette figure, deux éléments distinctifs sont observés. Le premier est un EdT (en rouge), et le second est le *buffer* d'entrée qui le succède (en bleu). Sur cette simulation, nous observons les différentes étapes de l'injection d'un vecteur à travers le *buffer* d'entrée d'un routeur, ainsi que l'envoi de la réponse résultante au bloc *Wrapper*. L'étape de remplissage des EdT n'est pas représentée sur cette simulation. Il est cependant possible d'identifier le vecteur dans le registre $scan\_chain\_registers\_data[78:0]$. Les indicateurs (1) et (2) représentent respectivement les entrées et les sorties du *buffer*. La commande du bloc *Wrapper* à l'EdT via le signal $scan\_enable[1:0]$ conduit à l'injection du vecteur



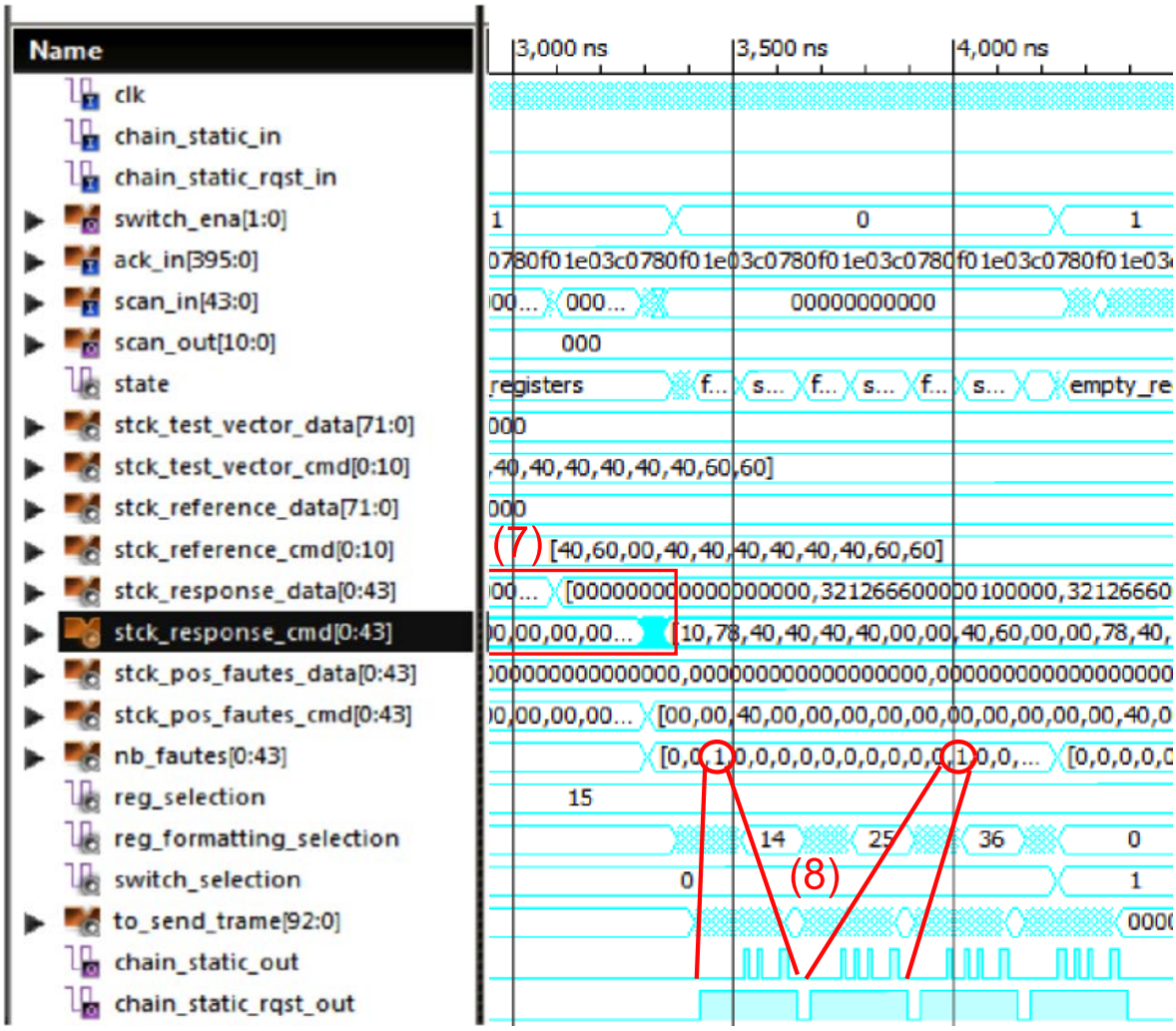
(a) Réception des vecteurs test et référence.

FIGURE 4.24 – Résultats d'une simulation du test d'une ZGS au niveau du bloc *Wrapper*.



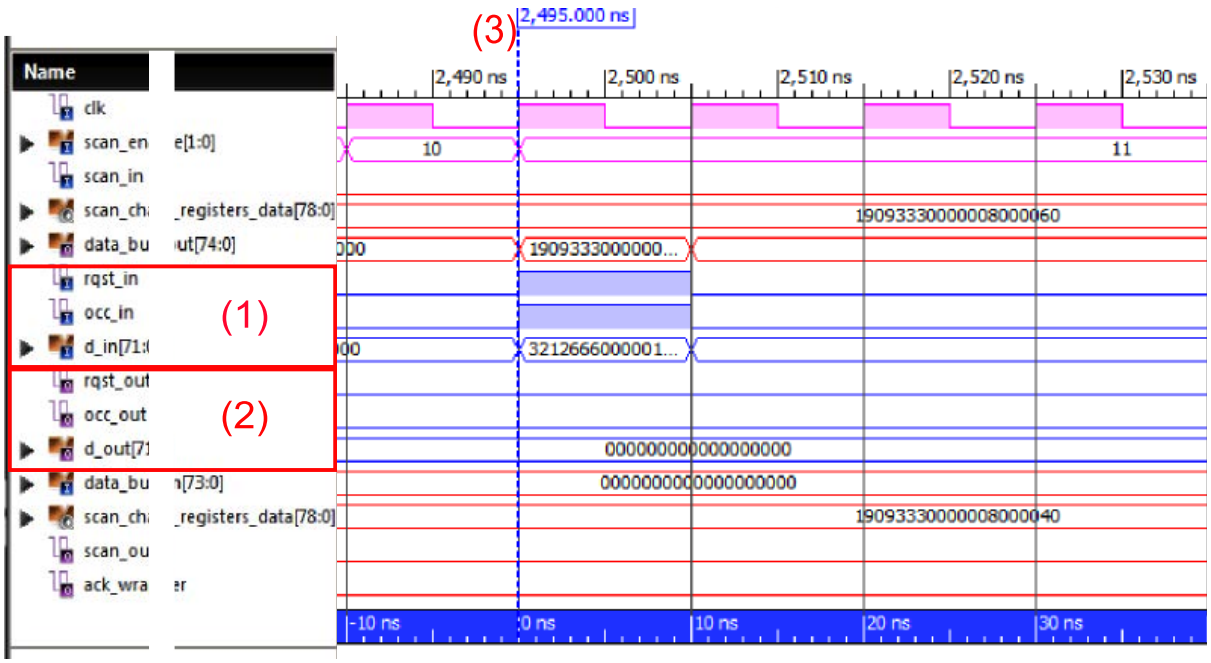
(b) Lancement du test d'une ZGS et récupération des premiers vecteurs réponses.

FIGURE 4.24 – Résultats d'une simulation du test d'une ZGS au niveau du bloc *Wrapper*.

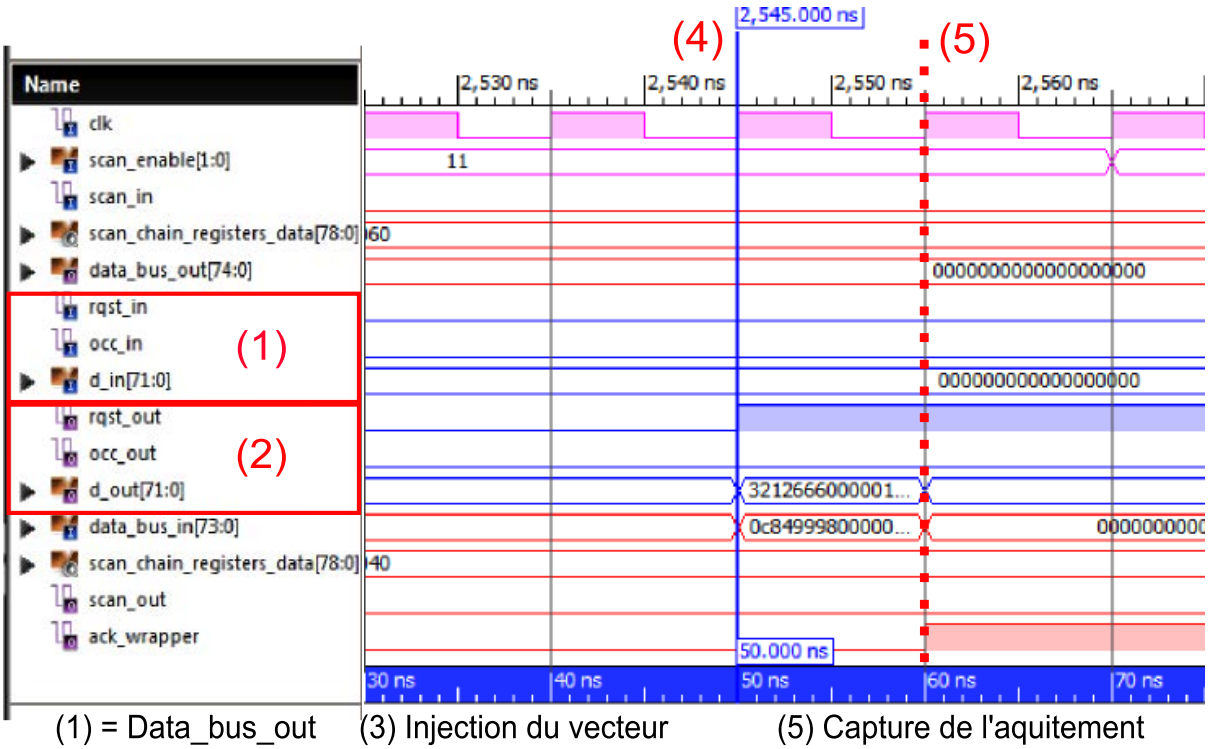


(c) Lancement du test d'une ZGS et récupération des premiers vecteurs réponses.

FIGURE 4.24 – Résultats d'une simulation du test d'une ZGS au niveau du bloc *Wrapper*.

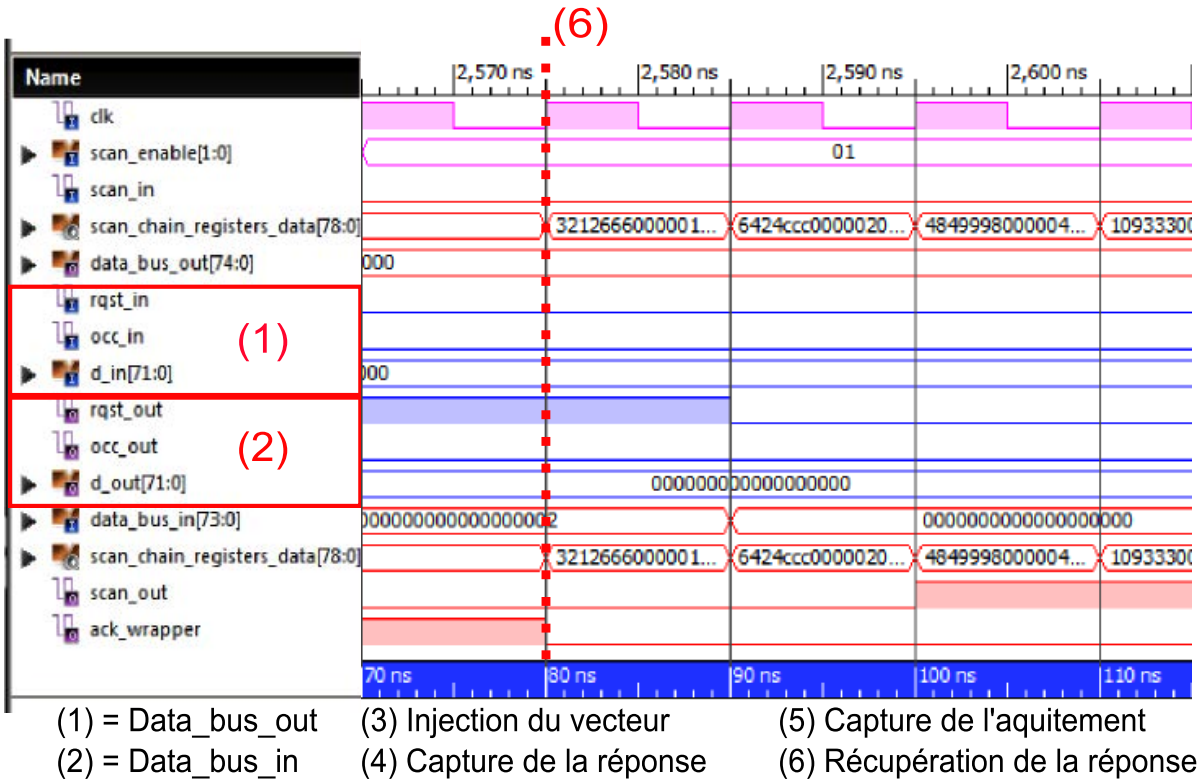


(a) Injection du vecteur de test dans d'un EdT.



(b) Capture de la réponse au vecteur de test injecté.

FIGURE 4.25 – Résultats d'une simulation du test d'une ZGS au niveau d'un EdT.



(c) Récupération du vecteur réponse via la sortie série.

FIGURE 4.25 – Résultats d'une simulation du test d'une ZGS au niveau d'un EdT.

Bloc	Registres	LUTs	Fréquence Max.
Contrôleur Principal	65	62	447 MHz
Gestion Mémoire Externe	45	110	516 MHz
Gestion Reconfiguration	87	233	276 MHz
Interface Xbee	165	238	310 MHz
Partie Statique	**362**	**643**	**276 MHz**
Filtre	268	309	196 MHz
Ensemble de l'architecture	654	963	197 MHz

TABLE 4.3 – Résultats de synthèses d'un nœud auto-organisé sans mécanisme de test pour la technologie FPGA *Virtex V*.

de test dans le *buffer* (3) (voir figure 4.25a). Les données présentent dans le registre sont ensuite envoyées via le bus de sortie de l'EdT $data\_bus\_out[74:0]$ vers l'entrée du *buffer*. Les 72 bits de poids forts de ce bus de sortie, correspondant aux données traversant usuellement le routeur, sont injectés sur le bus d'entrée du *buffer* ($d\_in[71:0]$). Les 3 bits de poids faibles quant à eux, correspondent aux signaux de commande de requête ($rqst\_in$), d'occupation (occ_in) ainsi qu'un dernier bit non-utilisé pour le *buffer*. Lorsque les données du vecteur de test traversent l'architecture du *buffer* (4), la réponse générée ainsi que les signaux de commande associés sont capturés (voir figure 4.25b). Afin d'informer le bloc *Wrapper* que la réponse de cet EdT est disponible, un acquittement (*Acknowledge*) est envoyé (5). Puis, lorsque le signal $scan\_enable[1:0]$ indique la possibilité de réception (voir figure 4.25c), cette réponse lui est transmise en série (6).

Ces simulations montrent clairement la capacité de localisation des erreurs à travers les différents blocs architecturaux qui composent un routeur. En effet, nous retrouvons dans la trame de sortie les localisations exactes des erreurs injectées.

4.3 Résultats de synthèse

Dans l'objectif d'évaluer les ressources logiques des différentes architectures proposées dans ces travaux, différentes synthèses ont été effectuées. Dans un premier temps, le tableau 4.3 présente les résultats de synthèse pour l'architecture de l'auto-organisation (technologie *Virtex V*). A travers ces résultats, il est possible d'observer que pour la partie statique de l'architecture proposée, le bloc consommant le plus de ressources logiques est l'interface de communication avec le module sans fil *Xbee* qui nécessite 165 Registres et 238 LUT. Concernant la fréquence maximum de fonctionnement, elle est limitée par le bloc de reconfiguration permettant la gestion de la primitive ICAP avec un fréquence maximale de 276 MHZ. L'architecture proposée est donc peu exigeante sur un FPGA de type *Virtex V*, cela est principalement dû au fait de l'utilisation de mémoires externes

	Virtex VII		**Virtex VI**		**Virtex V**	
Module	**Registres**	**LUT**	**Registres**	**LUT**	**Registres**	**LUT**
RKT_SW	12131	16760	11937	16455	11926	15980
RKT_W	11666	15953	11486	15662	11473	15115
RKT_NW	12131	16760	11937	16455	11926	15980
RKT_S	11666	15953	11486	15662	11473	15115
RKT	11200	15144	11020	14855	11011	14242
RKT_SE	12131	16760	11937	16455	11926	15980
RKT_E	11666	15953	11486	15662	11473	15115
RKT_NE	12131	16760	11937	16455	11926	15980
RKT_N	11666	15953	11486	15662	11473	15115
Wrapper	9029	14759	9023	14759	8237	15877
ZGS	**115417**	**160755**	**113735**	**158082**	**112844**	**154499**

TABLE 4.4 – Résultats de synthèses logiques des routeurs RKT intégrant le mécanisme de test pour les technologies *FPGA Virtex V, Virtex VI* et *Virtex VII*.

pour stocker les différentes données générées et ainsi limiter les registres. De plus, les données communicants sur le réseau sans fil sont envoyées/réceptionnées en série par paquets d'octets à une vitesse relativement faible par rapport à la fréquence de fonctionnement du FPGA. Plus précisément, grâce à cette différence entre la fréquence de fonctionnement et de transmission, les données sont traitées directement par flot de paquets limités à 8 bits et non sur l'ensemble des données, limitant ainsi les ressources matérielles nécessaires.

Des synthèses logiques ont également été réalisées pour l'implémentation d'un routeur RKT adapté aux tests hors ligne délocalisés sur cibles de technologies *FPGA Virtex V, VI* et *VII*, et sont présentées dans le tableau 4.4. Ces résultats de synthèses ont été obtenus pour des routeurs de taille de bus de 72 bits en considérant les bits de Hamming. Si l'on ne considère pas le codage de Hamming intégré à chaque paquet, la taille du bus de données est de 64 bits. Parmi ces résultats se trouve en premier lieu la synthèse d'un routeur RKT original, avant intégration de la logique de test. On notera que les fréquences de fonctionnement maximales pour ce routeur sont respectivement pour les technologies *Virtex V, VI* et *VII* de 306, 419 et 441 MHz. En comparaison, les versions modifiées fonctionnent pour ces mêmes technologies respectivement aux fréquences maximales de 149, 185 et 194 MHz. On peut également noter les fréquences de fonctionnement du bloc *Wrapper* qui correspondent à 61, 81 et 83 MHz, respectivement pour les technologies considérées.

Les coûts en surface logique de chaque bloc constituant le routeur RKT, ainsi que le module de gestion des vecteurs, sont détaillés dans le tableau 4.5 pour une technologie *Virtex VII*. Ce tableau permet d'analyser le coût en ressources logiques de chaque bloc de l'architecture. Par exemple, la surface logique nécessaire à l'implémentation des EdT, blocs supplémentaires dans un routeur RKT de base, constitue 55 à 56,4% de la surface totale d'un routeur. En d'autres termes, la surface nécessaire à la mise en place d'un routeur RKT intégrant le mécanisme de test proposé conduit à une augmentation en

Structure	**Nb de blocs**	**Slices**		**% du routeur**	
		Registres	**LUT**	**Registres**	**LUT**
Wrapper	1 pour 9	9029	14759	/	/
Elément de Test	44	154	258	55 à 56,4	66,7 à 68,9
Codeur de Hamming	0 à 4	77	139	0 à 2,5	0 à 3,3
Buffer d'entrée	4	369	378	12 à 12,3	8,9 à 9,2
Buffer de Sortie	4	865	879	28,1 à 28,8	20,6 à 21,3
Logique de Routage	4	75	23	2,4 à 2,5	0,5 à 0,6

TABLE 4.5 – Détails des synthèses logiques pour les différents modules constitutifs de l'architecture.

ressources logiques équivalent à environs 2 fois la version originale. Concernant le bloc *Wrapper*, dont l'implémentation permet la gestion du mécanisme pour une ZGS, il occupe une surface logique de 9037 registres et de 15056 LUT. Par comparaison avec les autres blocs de l'architecture, la surface logique de ce module est à peu près équivalent celle d'un *buffer* de sortie si on considère l'amortissement d'un bloc *Wrapper* mise en place pour une ZGS de taille 3x3 routeurs.

A partir des données de synthèses obtenues pour une ZGS de taille 3x3, les surfaces logiques nécessaires à différentes tailles de ZGS et pour plusieurs tailles de réseaux sur puce peuvent être estimées. L'objectif de cette comparaison est de pouvoir évaluer les différences de besoins entre des tailles de ZGS différentes pour un réseau de même taille. Le tableau 4.6 présente les estimations minimales obtenues pour des tailles de NoC 6x6, 12x12, 24x24 et 48x48 routeurs utilisant des ZGS de tailles 2x2, 3x3, 4x4, 6x6, 8x8, 12x12, 16x16 et 24x24. La figure 4.26 donne l'évolution graphique des données estimées. Il est ainsi possible d'observer une diminution de la surface logique nécessaire en élargissant les zones utilisées. En effet, une augmentation du nombre de routeurs par zone signifie une réduction du nombre de zones. Par conséquent le nombre de routeurs incorporant des CCE se trouve réduit par rapport au nombre de routeurs sans SdF en ligne. De plus, l'architecture du mécanisme de test hors ligne proposé comporte un bloc *Wrapper* par ZGS. En d'autres termes, la réduction du nombre de ZGS inclut la réduction du nombre de bloc *Wrapper* et par conséquent les ressources logiques consommées. Ainsi, l'utilisation de 4 ZGS de taille 24x24 routeurs pour un NoC de taille 48x48 en remplacement de 576 ZGS de 2x2 routeurs permet d'obtenir un gain en ressources logiques de 21,5%, passant de 33 150 528 à 26 019 844 registres.

		NoC 6x6	NoC 12x12	NoC 24x24	NoC 48x48
Taille ZGS	**Surface ZGS**	**Surface NoC**	**Surface NoC**	**Surface NoC**	**Surface NoC**
2x2	48524	517977	2071908	8287632	33150528
3x3	106388	461668	1846672	7386688	29546752
4x4	186652		1761129	7044516	28178064
6x6	414380		1693636	6774544	27098176
8x8	731708			6666633	26666532
12x12	1635164			6576772	26307088
16x16	2897020				26154441
24x24	6495932				26019844

TABLE 4.6 – Ressources logiques nécessaires pour différentes configurations de tailles de NoC et de ZGS pour la technologie *Virtex VII*.

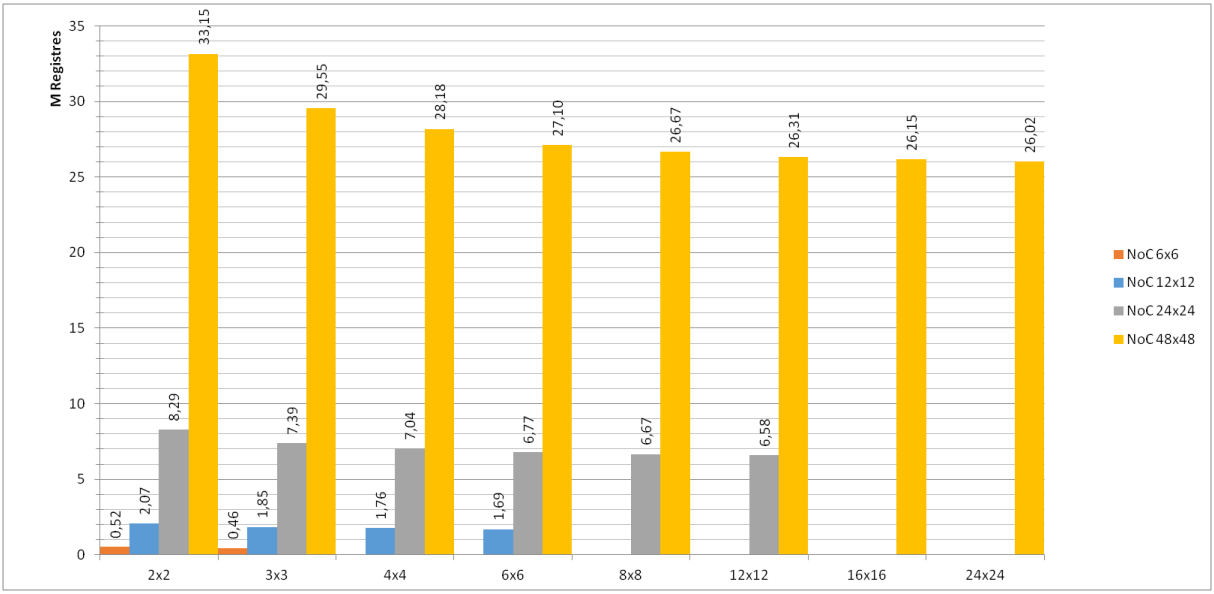


FIGURE 4.26 – Estimation des surfaces logiques nécessaires à différentes configurations de taille de NoC et de ZGS pour la technologie *Virtex VII*.

4.3.1 Impact du mécanisme sur le NoC RKT

Dans cette section, une comparaison entre un NoC RKT avec et sans mécanisme de test hors ligne est présenté au travers d'une série de résultats de synthèse et des évaluations de performances en termes de ressources logiques et de consommations électriques. Le tableau 4.7 résume les résultats obtenus pour différentes configurations.

Une analyse de ces résultats montre un surcoût d'un facteur 3 pour les besoins en registres et d'un facteur de 3,5 concernant les LUT, si on considère un NoC de taille 6x6 utilisant 4 ZGS de tailles 3x3. On note également une baisse importante de la fréquence maximale de fonctionnement relevée lors des synthèses, passant ainsi de 281 MHz à 83

		Registres	**LUT**	**f (MHz)**
NoC 6x6 avec ZGS de 3x3	*Original*	152340	178676	281,682
	Avec test hors ligne	461452	638438	83,289
	surcoût	303%	357%	-338%
Routeur RKT (2 CCEs)	*Original*	5977	7362	276,163
	Avec test hors ligne	12131	16760	193,825
	surcoût	49%	43%	-30%
Routeur RKT (1 CCEs)	*Original*	5825	6993	276,193
	Avec test hors ligne	11666	16266	193,825
	surcoût	50%	43%	-30%

TABLE 4.7 – Comparaison des ressources logiques d'un NoC RKT 6x6 original utilisant des ZGS de tailles 3x3 avec la version incorporant les mécanismes proposés sur technologie *Virtex VII*.

MHz. Cette chute de fréquence de fonctionnement est causée par le module *Wrapper*. En effet, comme précisé la fréquence de fonctionnement est imposée par le bloc le plus lent et correspondant au bloc *Wrapper* limité à 83 MHz, tandis que les différents types de routeurs ont une fréquence maximale de fonctionnement avoisinant en moyenne 200 MHz, soit environ 30% inférieure au routeur RKT. D'une manière générale, le bloc *Wrapper* contribue à diminuer les performances de l'architecture tant en termes de ressources logiques qu'au niveau de la fréquence de fonctionnement globale du système. Si l'on considère uniquement un routeur, la dégradation est moins significative comme le montre les résultats de synthèse du tableau 4.7 pour un unique routeur. Par exemple, dans le cas d'un routeur à 2 CCE (routeurs Nord-Ouest, Nord-Est, Sud-Est et Sud-Ouest), le RKT nécessite 5977 registres contre initialement 12131 registres en incluant la logique de test proposée. Nous obtenons donc un surcoût de 49% qui reste néanmoins nettement inférieur au surcoût global engendré par l'intégration du bloc *Wrapper*.

Une comparaison des consommations électriques a également été effectuée. Le tableau 4.8 donne les résultats de ces consommations électriques obtenues avec l'outil de synthèse *ISE 13.1* de *Xilinx*, pour différentes technologies de FPGA. Pour les technologies *Virtex VI* et *Virtex VII*, la consommation pour les différents routeurs reste similaire lorsque l'on intègre la logique de test au routeur RKT. On note cependant, une consommation moindre pour la technologie *Virtex V* concernant les routeurs intégrant le mécanisme de test hors ligne. Cette diminution de la puissance consommée est liée aux fréquences de fonctionnement plus faibles pour les routeurs proposés.

	Virtex V	**Virtex VI**	**Virtex VII**
RKT (1 CCE)	3,590 W	4,447 W	1,037 W
RKT avec mécanisme (1 CCE)	3,217 W	4,447 W	1,037 W
RKT (2 CCEs)	3,583 W	4,447 W	1,037 W
RKT avec mécanisme (2 CCEs)	3,223 W	4,447 W	1,037 W
Wrapper	3,170 W	4,447 W	1,037 W

TABLE 4.8 – Comparaison des consommations électriques d'un routeur RKT fiabilisé par incorporation des mécanismes SdF proposés et implantés dans les technologies *Virtex V*, *Virtex VI* et *Virtex VII*.

4.4 Évaluation des performances

Cette section met en avant les performances du système proposé en termes de temps d'exécution et de capacité de localisation d'erreurs. Pour cela, plusieurs méthodes sont utilisées. L'auto-organisation des nœuds est implémentée physiquement dans plusieurs plateformes de développement, intégrant un module d'acquisition des données. Le mécanisme de test délocalisé quant à lui est évalué par co-simulation, dans le but de déterminer ses performances pour un très grand nombre de vecteurs et d'erreurs.

4.4.1 Méthode d'injection des fautes

Afin d'évaluer les performances du système sûre de fonctionnement proposé, une méthodologie de test par co-simulation C-VHDL est utilisée. Pour se faire, le logiciel *ModelSim* [Gra02] est utilisé. L'objectif de cette méthode est l'injection d'un très grand nombre de vecteurs successifs. En effet, le langage de programmation C permet plus aisément la modulation d'un *testbench* en fonction du besoin. De plus, il est possible de lancer parallèlement des processus permettant de gérer plusieurs parties du système. Dans le cas de l'évaluation de la capacité de localisation des erreurs, ce mécanisme de parallélisation de processus permet d'injecter indépendamment des erreurs dans l'architecture de chaque routeur. La figure 4.27 illustre le principe de test de l'architecture d'une Zone Globalement Sûre par co-simulation. Sur ce schéma se trouve une architecture RTL à tester, en l'occurrence une ZGS dans ce cas précis, où le nombre de processus d'injection de fautes générés correspond au nombre de routeurs qui composent cette zone.

En outre, un processus principal permet de simuler l'utilisation d'un bloc IP test. Ce dernier reprend donc partiellement le fonctionnement de l'IP test précédemment décrite. Il permet la génération de vecteurs de tests, mais également de réceptionner, de décoder les trames et de les analyser. Ainsi, ce processus principal génère successivement des vecteurs puis les envoi dans l'architecture suivant le même protocole que l'IP test.

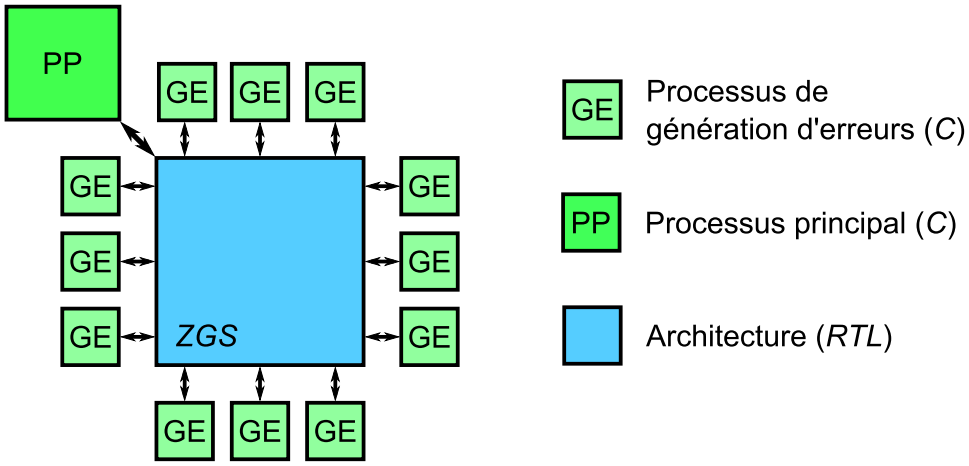


FIGURE 4.27 – Principe de la Co-Simulation C-VHDL appliqué à une ZGS.

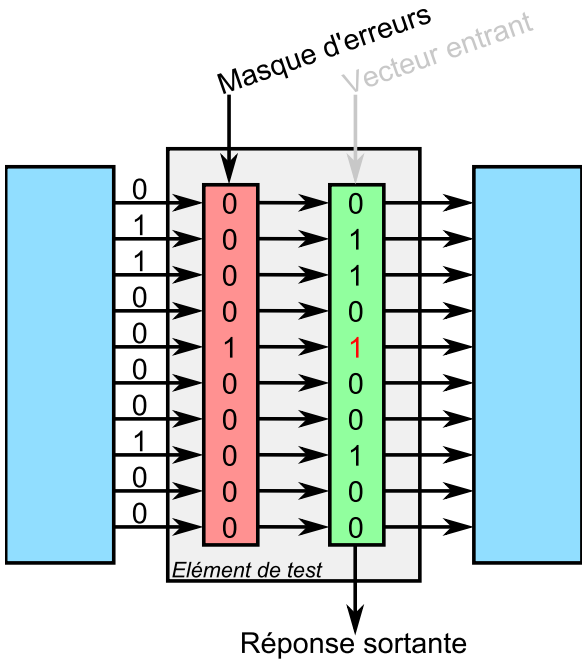


FIGURE 4.28 – Méthode d'injection d'erreurs dans l'architecture.

Parallèlement, chaque processus de génération d'erreurs va, de manière aléatoire ou non, générer un masque d'erreurs dans chaque EdT composant le routeur associé au processus. Ces masques d'erreurs permettent l'injection de fautes lors de la capture de la réponse de chaque EdT. La figure 4.28 illustre le principe d'injection de fautes dans le système proposé à l'aide d'EdT modifiés. Ainsi lors d'une séquence de test et la récupération d'un vecteur réponse, ces masques vont artificiellement émuler une ou plusieurs erreurs par inversion de un ou plusieurs bits.

En définitive, malgré le bon fonctionnement du système, une mesure de taux de localisation d'erreurs est possible. Plus précisément, les fautes générées pour chaque routeur,

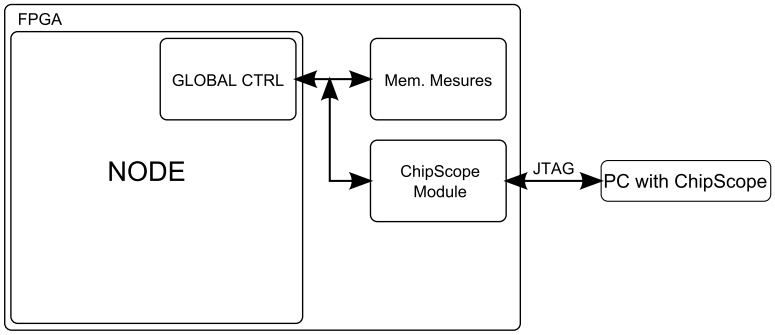


FIGURE 4.29 – Bloc d'interfaçage d'un nœud reconfigurable avec l'outil *ChipScope* pour la prise de mesures.

ainsi que leurs localisations spécifiées dans le vecteur réponse, sont mémorisées par le *test-bench*. Par conséquent, à partir du traitement des vecteurs réponses ou trames d'erreurs envoyées par le mécanisme de test, le processus principal extrait les informations relatives aux tests effectués. Ces trames contenant soit le nombre d'erreurs soit le vecteur réponse intégrale, permettent d'effectuer une comparaison entre les erreurs injectées et les erreurs localisées dans le but d'évaluer la capacité de localisation du mécanisme délocalisé de tests proposé.

En utilisant cette même technique, une mesure temporelle est également effectuée afin de déterminer la durée nécessaire au traitement d'un vecteur en fonction du nombre de fautes introduites. Plus précisément, un comptage en termes de cycles d'horloges pour chaque nouveau vecteur de test envoyé dans le système est effectuée entre le moment de réception du vecteur par le bloc *Wrapper* jusqu'à l'envoi de la dernière trame de réponse.

4.4.2 Auto-organisation

L'évaluation de la rapidité du système d'auto-organisation consiste en une série de mesures réalisées grâce à l'outil d'analyse des signaux internes *ChipScope* de *Xilinx*. Plus précisément, les mesures effectuées sont réalisées physiquement lors de la mise en place d'un système de plusieurs nœuds sur différentes plateformes à base de FPGA. Les données récupérées permettent donc d'évaluer les performances temporelles de ce système dans des conditions réelles. La figure 4.29 représente le schéma bloc de l'implantation d'un module de mémorisation des valeurs ainsi qu'un module permettant de faire l'interfaçage avec l'outil logiciel *ChipScope*.

La figure 4.30 est une capture d'écran du logiciel *ChipScope* lors de la prise des mesures. Ce logiciel permet l'acquisition de signaux spécifiquement mis en place dans le but d'évaluer les performances de l'ensemble des nœuds. Il est donc possible d'évaluer avec une précision finie le temps de traitement de l'architecture et des différentes étapes du mécanisme d'auto-organisation développé.

Les mesures sont prises à partir d'un *contrôleur global* spécifiquement développé. Les

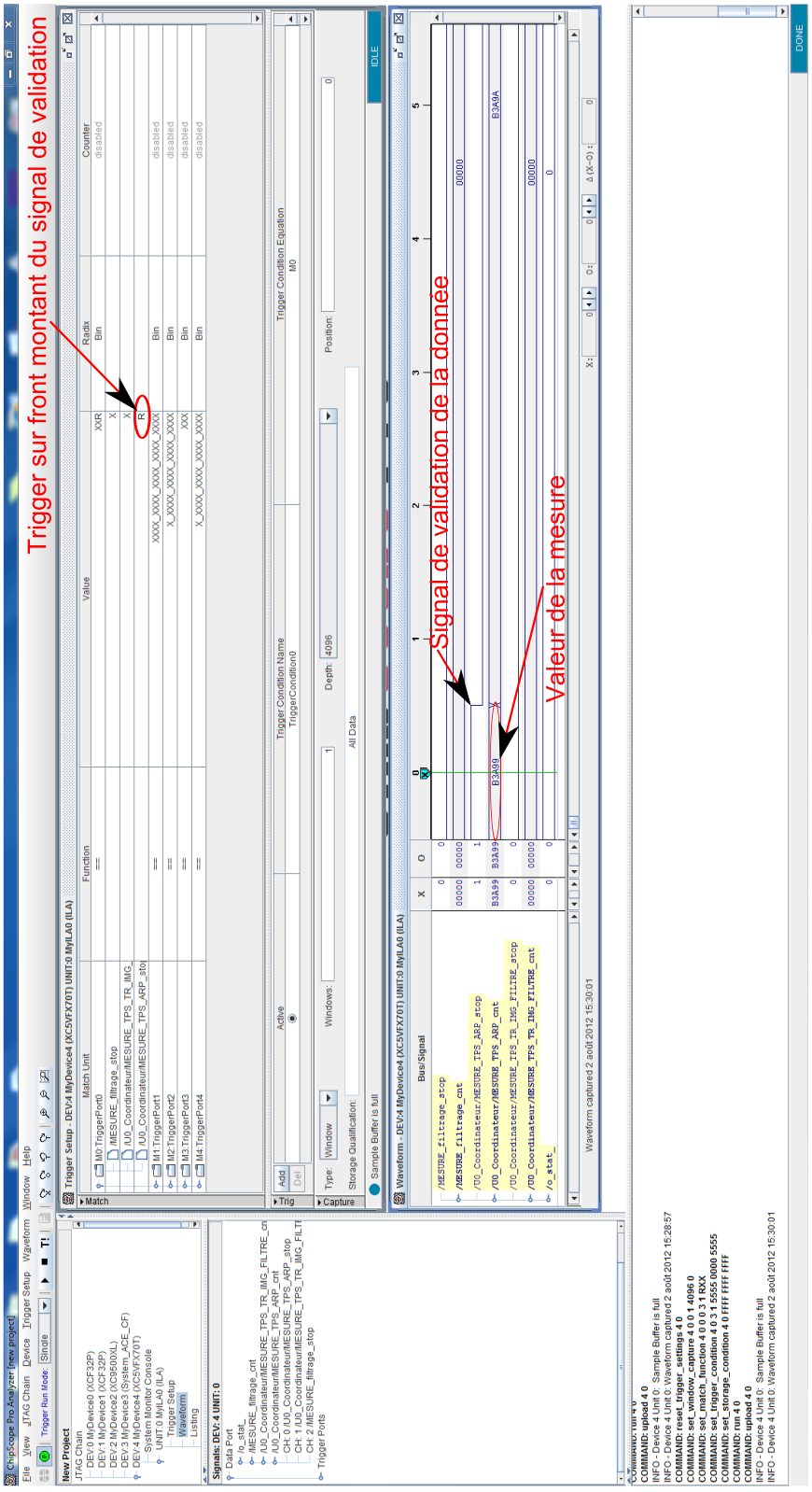


FIGURE 4.30 – Mesures temporelles des signaux à l'aide de *ChipScope*.

Etape	Taille (octets)	Temps estimés (sec)	Temps simulés (sec)	Temps mesurés (sec)	Commentaires
Filtrage NEXYS3	82398	0,041199	0,033378	0,034	sur N3
Envoi du Bitstream	66946	5,811	6,561	6,562	N3 vers ML507
Envoi de l'image	82398	7,153	8,076	8,076	N3 vers ML507
Reconfiguration	66946	0,006696	0,007359	0,007359	sur ML507
Filtrage ML507	82398	0,041199	0,041755	0,042	sur ML507
Envoi de l'image	82398	7,153	8,809	8,81	ML507 vers N3
Temps Total		**20,206**	**23,528**	**23,531**	

TABLE 4.9 – Temps d'exécutions des différentes étapes du processus d'auto-organisation appliquées au filtrage *moyenneur* d'une image couleur RGB de taille 194x141 pixels.

temps sont mesurés entre la commande d'exécution de la tâche et la validation de la fonction sollicitée. Les valeurs sont représentatives et l'unité est choisie en fonction de la pertinence de l'information. Par exemple, dans notre cas d'application d'étude de filtrage *moyenneur*, le temps de transfert d'une image sera mesuré à la milliseconde près, tandis que le temps de reconfiguration sera défini à la microseconde près, car l'un nécessite plusieurs secondes et l'autre quelques millisecondes. Ces mesures permettent de déterminer les différents temps de traitement et de transmission.

Le système mis en place pour ces mesures est une communication entre 3 nœuds hétérogènes. Une erreur est volontairement injectée dans un des nœuds afin de déclencher le mécanisme d'auto-organisation adapté à la gestion de fautes. Concrètement, les tâches implémentées sont identiques à la manipulation de validation du système. Ainsi, une tâche de traitement d'images (une image *RGB* de taille 194x141 dans le test effectué) est mise en place dans un nœud considéré "défaillant" (plateforme *NEXYS3*), puis est délocalisée dans un autre nœud (plateforme *ML507*) afin de maintenir le système global fonctionnel.

Chaque étape de l'auto-organisation est mesurée, en partant du fonctionnement normal jusqu'à la récupération des données provenant de la tâche délocalisée. Le tableau 4.9 reprend l'ensemble des mesures réalisées en cours de traitement, ainsi que celles obtenues par simulations et calculs théoriques. Dans ce tableau figure la quantité de données traitées pour chaque étape (en octets) mais également les temps d'exécution estimés par calculs, simulés et mesurés expérimentalement. On relève par exemple le temps de transfert d'une image de taille environ 80 Ko, sur le réseau *Zigbee*, en un peu plus de 8 secondes.

4.4.3 Évaluation de la capacité de localisation des erreurs

L'objectif principal du mécanisme de test délocalisé proposé étant la localisation précise d'erreurs générées dans l'architecture du réseau sur puce, cette section se focalise sur la capacité du système à identifier les fautes. Pour cela, la méthode d'injection de fautes

par co-simulation présentée précédemment est utilisée. Dans le but d'obtenir des résultats probants, une série de 10 000 vecteurs est envoyée à l'architecture de test. Comme précisé dans la description de la méthode, le *testbench* développé en langage C agit au niveau du système comme étant un IP test. Pour chaque vecteur de test généré, des erreurs pseudo-aléatoires sont injectées dans l'architecture afin d'évaluer le concept proposé. Cependant, le nombre d'erreurs est répartie équitablement entre les différentes catégories présentées dans le chapitre précédent. Plus précisément, en analysant l'ensemble des erreurs de chaque EdT et de chaque routeur de la Zone Globalement Sûre, les vecteurs réponses des EdT présentant aucune, 1, de 2 à 8 et plus de 9 erreurs sont équilibrées. Par conséquent, en obtenant une répartition équitable entre ces 4 types de trames, les résultats associés sont plus représentatifs des possibilités d'erreurs pouvant apparaître.

De plus, chaque vecteur de test injecté par le *testbench* est également généré de manière pseudo-aléatoire. Par conséquent, chaque séquence de test, constituée de la génération d'un nouveau vecteur et de l'injection des différentes erreurs dans l'architecture des routeurs, est différente du précédent. L'objectif est clairement de couvrir le plus grand nombre de possibilités et de contrôler au minimum les informations circulant dans le réseau sur puce. Comme l'illustre le tableau 4.10, sur une série de 10 000 vecteurs tests pseudo-aléatoires générés avec un nombre pseudo-aléatoire d'erreurs injectées dans l'architecture, l'ensemble des bits fautifs ont été localisés par le mécanisme, soit une localisation avec une couverture de 100 %. Dans ce tableau, se trouve le nombre d'erreurs générées (*ErrGen*) par l'ensemble des processus d'injection d'erreurs ainsi que le nombre de ces erreurs localisées par le mécanisme (*ErrLoc*) pour chaque nouveau vecteur de test transmis à l'architecture. De plus, le processus principal fournit le nombre de trames de réponses reçues (Trames), ainsi que leur type (*Type1*, *Type2* et *Type3*). Enfin, la mesure du temps nécessaire à l'exécution du test est donnée en nombre de Cycles d'Horloge (CLK). Lors de cette série de 10K vecteurs, le temps minimal d'exécution est de 18067 CLK pour un total de 2795 erreurs. De même, le temps maximal relevé est de 22457 CLK pour 4445 erreurs localisées permettant un taux de couverture de 100 %.

Le traitement des données ainsi que la composition des trames de réponse étant dynamique, l'injection pseudo-aléatoire d'erreurs ne permet pas de couvrir une plage suffisante du nombre d'erreurs pouvant être présentes dans les routeurs. Par conséquent, une incrémentation du nombre d'erreurs est utilisée comme série de vecteurs. Ainsi lors de la première boucle, aucune erreur n'est injectée, puis à chaque nouveau vecteur une erreur supplémentaire vient s'ajouter dans un même EdT jusqu'à le saturer d'erreurs. Lorsque le premier EdT contient que des erreurs, l'incrémentation s'effectue sur l'EdT suivant et ainsi de suite jusqu'à la saturation complète en erreurs du routeur. Lorsque le routeur est complètement fautif, le schéma se reproduit sur le routeur suivant jusqu'à injecter une erreur sur chaque bit possible. En utilisant cette méthode de gestion des vecteurs tests, l'ensemble de la plage d'erreurs possibles est balayée. Le tableau 4.11 représente la synthèse de l'injection de 22849 vecteurs de test en incrémentant à chaque nouveau vecteur le nombre d'erreurs. Les vecteurs sont, de la même manière que précédemment, générés pseudo-aléatoirement. Malgré l'intérêt d'incrémenter le nombre d'erreurs dans le but d'obtenir une grandeur temporelle, les résultats démontrent à nouveau une couverture

Vecteur	**ErrGen**	**Trame**	**Type1**	**Type2**	**Type3**	**ErrLoc**	**CLK**	**%**
0	3483	264	87	100	77	3483	21378	100
1	3437	248	88	77	83	3437	19109	100
2	3530	250	81	87	82	3530	19961	100
3	3888	266	84	89	93	3888	21107	100
4	3998	262	81	84	97	3998	20845	100
5	3541	258	93	82	83	3541	20042	100
6	3573	259	87	86	86	3573	20483	100
7	3446	250	83	83	84	3446	20025	100
...	...	...	...	...	...	...	...	...
10000	3821	250	65	95	90	3821	21289	100
Moy.	**3550**	**252**	**85**	**85**	**83**	**3550**	**20119**	**100**

TABLE 4.10 – Résultats de l'évaluation des performances de tests après un série de 10000 Vecteurs Tests pseudo-aléatoires.

Vecteur	**ErrGen**	**Trame**	**Type1**	**Type2**	**Type3**	**ErrLoc**	**CLK**	**%**
0	0	0	0	0	0	0	1575	100
1	1	1	1	0	0	1	1599	100
2	2	1	0	1	0	2	1702	100
3	3	1	0	1	0	3	1709	100
4	4	1	0	1	0	4	1716	100
5	5	1	0	1	0	5	1723	100
6	6	1	0	1	0	6	1730	100
7	7	1	0	1	0	7	1737	100
...	...	...	...	...	...	...	...	...
22848	22848	336	0	0	336	22848	33036	100

TABLE 4.11 – Résultats de l'évaluation après un série de 22849 Vecteurs de Test pseudo-aléatoire, avec incrémentation progressive du nombre d'erreurs.

de localisation totale du nombre d'erreurs démontrant l'efficacité de l'approche de test proposée. La figure 4.31 donne la synthèse des résultats obtenus après la saturation complète d'une ZGS. Cette courbe représente le temps de traitement nécessaire, en nombre de cycles d'horloge, à un vecteur de test en fonction du nombre d'erreurs injectées dans l'architecture.

En apparence, dû au nombre important de vecteurs envoyés, la courbe semble linéaire alors qu'en réalité les différents types de trame sont clairement visibles lors de l'incrémentation. La figure 4.32 donne un "zoom" de la courbe principale à l'échelle d'une saturation d'un *side* de routeur. L'allure de la courbe met en évidence les différents types de trames émises par le bloc *Wrapper* après analyse des réponses aux vecteurs tests. En effet, pour l'incrémentation d'erreurs dans chaque vecteur, la pente positive correspond à l'incrémentation pouvant atteindre jusqu'à 8 erreurs par EdT, puis lorsque la pente est nulle, le nombre d'erreurs présentes dans cet même EdT est supérieur à 8. On observe un temps

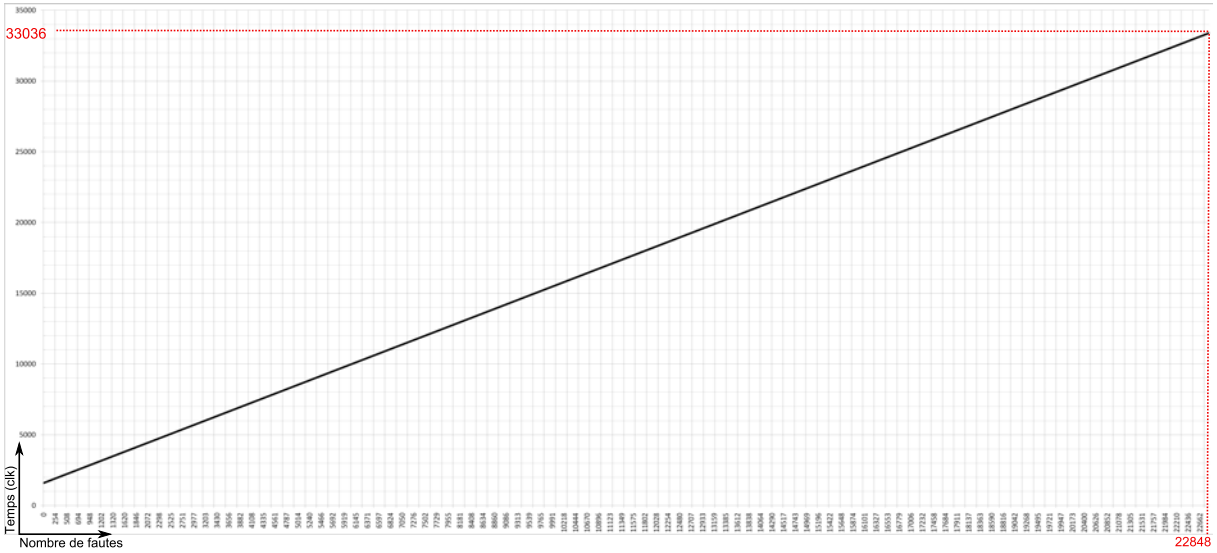


FIGURE 4.31 – Temps de traitement d'un vecteur en fonction du nombre de fautes.

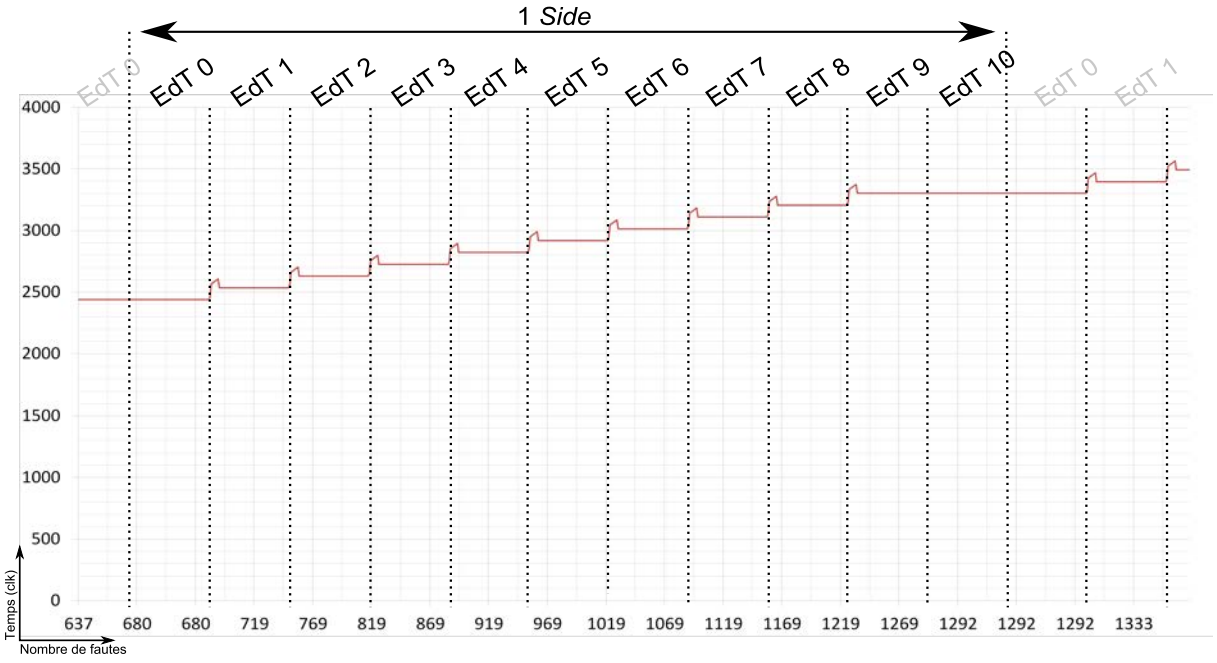


FIGURE 4.32 – Temps de traitement d'un vecteur en fonction du nombre de fautes, "zoomé" à l'échelle d'un *Side*.

de traitement inférieur lorsque ce nombre est supérieur à 8. Ce résultat s'explique par la recherche des positions d'erreurs dans les vecteurs réponses. Cependant, la quantité de données transmises à l'IP test reste supérieure, et ce gain de temps se perd lors de la transmission.

En réduisant encore la plage d'erreurs visible, les trames peuvent être distinctement identifiées. La figure 4.33 est un "zoom" de la courbe de la figure 4.32 à l'échelle d'une sa-

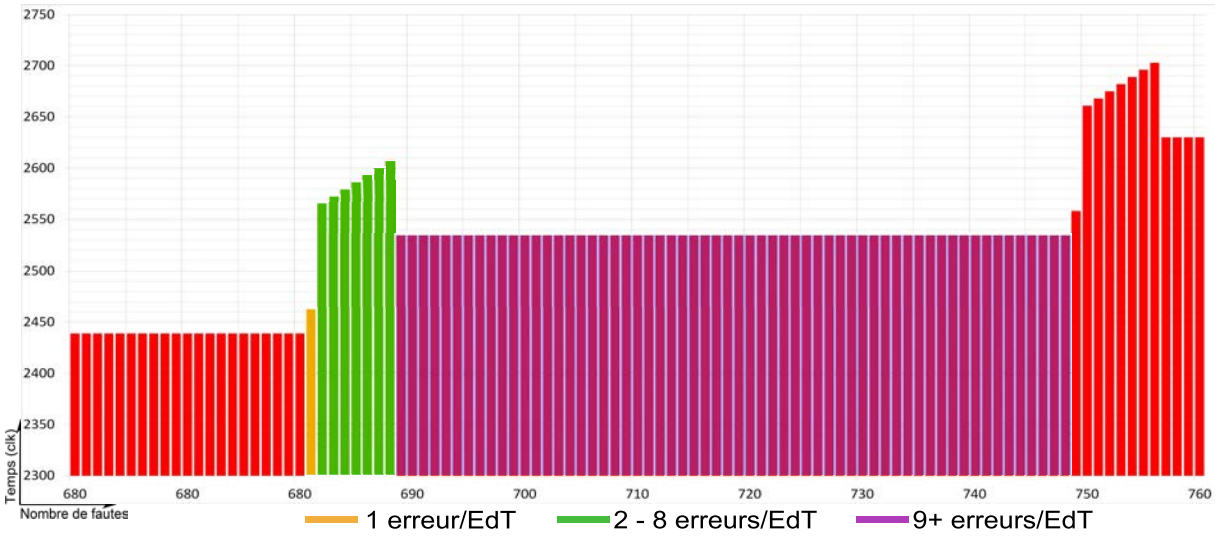


FIGURE 4.33 – Temps de traitement d'un vecteur de test en fonction du nombre de fautes, "zoomé" à l'échelle d'un EdT.

turation d'un EdT. Sur cette courbe, l'incrémentation du nombre d'erreurs est clairement visible. Pour ce cas, une seule erreur est présente dans l'EdT visualisé, et donc une seule localisation est présente dans la trame. Le temps nécessaire correspond à l'échantillon *orange* dans la figure 4.33. Lorsque ce nombre augmente, toujours au sein de ce même EdT, le type de trame change et inclut les champs de localisation des différentes erreurs. L'allure de cette étape correspond aux échantillons en *vert* dans la figure 4.33. Enfin, lorsque 9 erreurs ou plus sont localisées dans le vecteur réponse, l'intégralité de celui-ci est envoyée à l'IP test, d'où l'allure constante de la courbe sur cette période (en *mauve* dans la figure 4.33).

4.5 Limitations et discussions

Après avoir effectué ces évaluations de performances et d'efficacité en termes de localisations d'erreurs, plusieurs limitations sont mises en évidence. Dans un premier temps, concernant le système auto-organisé proposé, la communication inter-nœuds reste limitée. En effet, la proposition de communication par flux de données mise en œuvre reste encore moins évoluée comparé au concept théorique. En effet, cette communication s'effectue actuellement par un système de *broadcast* pour les demandes et en communication directe pour les échanges de données. La communication par flux pour ce type de système nécessite un réflexion plus avancée pour se rapprocher du concept d'interactivité d'un système auto-organisé.

Toujours sur la même partie des travaux, malgré une approche générale, le support de communication au protocole *Zigbee* limite les performances au niveau de la vitesse de transmission des données. Plus précisément, les performances temporelles au niveau système sont insuffisantes par rapport au temps nécessaire à transmettre un fichier de

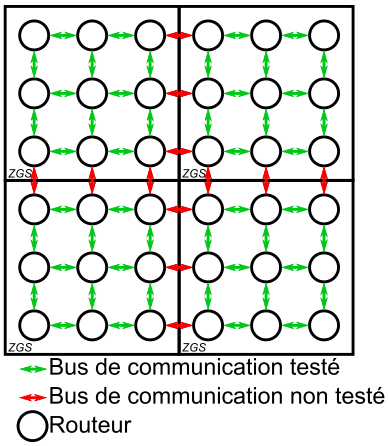


FIGURE 4.34 – Limite de testabilité des bus de communication inter-ZGS.

reconfiguration. C'est d'ailleurs dans ce contexte de limitations que la mise en œuvre d'un système de formatage des trames contenant les vecteurs réponses aux vecteurs de test a été élaboré afin d'optimiser le temps de transfert des données.

Le mécanisme de test développé comporte également des limitations. La première est l'impossibilité, dans la configuration actuelle, de tester les bus de communications inter-ZGS. En effet, l'architecture de test mise en place permet, en plus de localiser les erreurs internes aux routeurs, de tester les bus de communication entre les routeurs d'une même Zone Globalement Sûre. Cependant, elle ne permet pas de vérifier le bon fonctionnement d'un bus situé entre deux ZGS. La figure 4.34 illustre cette limitation.

Enfin, l'architecture de test intégrée dans le NoC permet la localisation des erreurs survenues dans les différents blocs d'un routeur ainsi qu'au niveau des bus de communication inter-routeurs. Cependant, l'identification des types d'erreurs ainsi que des causes associées sont assurées par l'IP test. Ce dernier est en phase de développement et est limité actuellement dans le cadre des tests de performances du mécanisme proposé pour définir cette identification. Des erreurs courantes telles que des inversions temporaires de bits sur un bus de données peuvent être aisément identifiées contrairement aux erreurs sur des signaux de commande, qui peuvent éventuellement provoquer une cascade d'erreurs supplémentaires.

Conclusion

Dans ce chapitre, les concepts proposés d'auto-organisation et de tests délocalisés ont été validés. Ces validations se sont faites par une proposition architecturale d'un système auto-organisé matériel original intégrant un réseau de communication de type NoC, ainsi qu'un mécanisme délocalisé de tests basé sur cette auto-organisation. Les deux concepts permettent la gestion de tâches multiples grâce à la capacité de reconfiguration d'un FPGA intégré dans chacun des nœuds composant le système réseau adopté. Ils permettent égale-

ment les tests partiels des structures de communication NoC RKT à l'aide d'un mécanisme d'injection de vecteurs de test à distance et basé sur le concept d'auto-organisation adopté.

Le premier concept permet donc la gestion avancée de tâches de manière entièrement autonome entre plusieurs nœuds d'un réseau. Cela, dans l'objectif de maintenir le système global en fonctionnement, ou de permettre l'utilisation d'un mode dégradé. A travers les résultats présentés, le fonctionnement de la capacité de maintient du fonctionnement d'un système multi-nœuds en réseau a été validé lors de l'implémentation sur plateforme *FPGA Xilinx* de l'architecture réseau proposée. En effet, lors par exemple d'une simulation d'erreurs dans une des tâches d'un nœud défaillant du réseau, il est possible d'observer l'auto-gestion de ces erreurs par une délocalisation de la tâche défaillante vers un autre nœud tout en maintenant la communication des échanges de données utiles au bon fonctionnement du système global. En plus de la validation de l'architecture, des mesures en termes de ressources logiques consommées et de temps d'exécutions des différentes étapes de l'auto-organisation ont été présentées et analysées, montrant l'intérêt de l'approche proposée.

La seconde validation effectuée est la capacité de localisation d'un mécanisme de tests délocalisé et basé sur le système communicant développé. Des simulations de ce mécanisme ont démontré une capacité de localisation avec un taux de couverture de 100%. En effet, malgré les nombreuses générations de vecteurs aléatoires ainsi que des injections de fautes variables et aléatoirement placées, le mécanisme proposé permet d'une part de localiser systématiquement les erreurs présentent dans l'architecture du réseau de communication. D'autre part, de localiser les blocs logiques défaillants et sources de fautes. Les résultats de synthèse des différentes architectures du système global proposé ont mis en évidence un surcoût acceptable de ressources logiques nécessaires à l'implémentation du mécanisme de tests dans le cas de l'architecture NoC RKT. En effet, l'intégration de la logique de tests intégrée et proposée entraine une augmentation d'environs 30% de la surface logique par routeur.

En résumé, dans ce chapitre, nous avons présenté la mise en œuvre des concepts et des techniques proposées dans les chapitres précédents. En particulier, les techniques de localisation d'erreurs ont été validées aux travers de nombreuses simulations matérielles afin d'estimer leur capacité de détection et de localisation des sources au sein d'un réseau de NoC dynamiques. De même, des synthèses logiques du système intégrant les différentes solutions proposées sont analysées en termes de performances, de ressources logiques et de puissance consommées dans le cas de la technologie FPGA, démontrant l'intérêt d'intégrer de telles structures dans la conception de systèmes MPSoC en réseau. Les mécanismes développés ont permis la conception de la plateforme du réseau auto-adaptatif RKT à base de *FPGA Xilinx* mettant en œuvre les mécanismes architecturaux dédiés aux nœuds reconfigurables pour assurer des communications sûres de fonctionnement, et qui sont primordiaux pour assurer les interactivités des entités constituant un système auto-organisé et permettant de réaliser les principales caractéristiques que sont *l'augmentation de l'ordre* et *l'émergence* d'un système auto-organisé.

Conclusion générale

Avec l'évolution actuelle des systèmes complets intégrant des modules de nature différente sur une même puce (System on Chip - SoC), les technologies reconfigurables deviennent primordiales. En effet, les SoC ayant à l'origine une flexibilité matérielle limitée (structure de type Application-Specific Integrated Circuit (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 termes 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 ont été élaborées. Ces derniers doivent permettre la mise à disposition d'une puissance de calcul suffisante tout en bénéficiant d'une grande flexibilité et d'une grande adaptabilité, cela dans le but de répondre aux évolutions des traitements distribués caractérisant le contexte évolutif du fonctionnement des systèmes. L'aboutissement de la conception de tels systèmes communicants, auto-organisés et auto-adaptatifs repose sur des nœuds de calcul reconfigurables. Ces derniers matérialisent les propriétés d'autonomie, d'intelligence, de capacité de déployer et de redéployer des modules de calcul en temps réel et en fonction de la demande de traitement et puissance de calcul, grâce à la technologie FPGA. De tels architectures auto-adaptatives permettent d'étudier l'impact des systèmes reconfigurables dans une structure distribuée et auto-organisée. Cependant, la faisabilité de tels systèmes complexes de calcul distribués, et dont l'intelligence repose principalement sur les interactions des éléments constitutifs, nécessite de disposer de structures de communication adaptées et fiables. C'est donc tout naturellement que les systèmes auto-organisés ont vu leur évolution vers des structures MPSoC reconfigurables en réseau à base de NoC adaptatifs matérialisés par la technologie FPGA. En effet, la conception de systèmes communicants MPSoC adaptables, dédiés aux applications pour lesquelles les contraintes de la sûreté de fonctionnement (SdF) sont très critiques, nécessite de prendre en compte la fiabilité des communications. Le défi scientifique est donc le développement de nouvelles stratégies de fiabilisation des communications sur puce à partir d'une caractérisation des transmissions et des sources de perturbation possibles au sein de l'ensemble des communications d'un système multi-nœuds communicant.

C'est dans ce contexte que se situent ces travaux de recherche présentés dans ce mémoire de thèse. Plus précisément, ces travaux ont pour objectif de développer un réseau de nœuds reconfigurables à base de FPGA intégrant des communications sur puce (Network on Chip - NoC) adaptatives, capables de s'auto-organiser et de s'auto-tester dans le but d'une maintenabilité du fonctionnement global du système dans un contexte en

réseau (Wireless Sensor Network - WSN). Pour ce faire, des techniques de détection, de localisation et de correction d'erreurs au sein des NoC reconfigurables ont été intégrées dans la conception du réseau considéré. La principale originalité des travaux développés est la conception d'un système multi-nœuds MPSoC (Multi-processeurs sur puce) capable d'échanger et d'interagir, et permettant ainsi une gestion avancée de tâches et une auto-gestion de mécanismes délocalisables de tolérance aux fautes. Des techniques combinées ont donc été proposées et développées permettant d'identifier et de localiser avec précision les éléments défaillants de telles structures dans le but de les corriger ou de les isoler pour prévenir toutes défaillances du système. Ce concept original permet des échanges d'informations entre les modules constituant le système, sans un contrôle centralisé et tout en répondant 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. Ainsi, en termes de validation, nous avons développé et implanté un modèle de gestion de nœuds auto-organisés communicants au sein d'un réseau sans fil au protocole *ZigBee*, afin d'ordonnancer des tâches matérielles selon une auto-distribution sur des cibles FPGA. Ces techniques ont été validées dans la plateforme du réseau auto-adaptatif RKT à base de technologies *Xilinx* et aux travers de nombreuses simulations matérielles permettant d'estimer leur capacité de détection et de localisation des sources d'erreurs au sein du réseau. De même, des synthèses logiques du système intégrant les différentes solutions proposées ont été analysées en termes de performances, de ressources logiques et de puissance consommées dans le cas de la technologie FPGA. Les résultats montrent l'intérêt d'intégrer de telles structures dans la conception de systèmes MPSoC en réseau.

Les travaux menés durant cette thèse sont donc originaux en plusieurs points. D'abord, parce qu'ils présentent 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 particulier sur la mise en œuvre de la stratégie de tolérance aux fautes des structures de communication de multi-nœuds reconfigurables en réseau, permettant la faisabilité du concept SdF par auto-organisation grâce à la flexibilité et l'adaptabilité des interactions permis par ce type de réseaux matériels versatiles.

En perspective à court terme, une suite de ces travaux théoriques et expérimentaux consiste à développer l'implantation simultanée une défaillance d'un module au sein d'un réseau en phase de reconfiguration dynamique partielle. Pour les perspectives à moyen terme, l'exploration des méthodes et outils formels permettrait de contribuer à effectuer la preuve de fiabilité de fonctionnement lors de la conception des mécanismes de tolérance aux fautes par auto-organisation dans les systèmes communicants. L'idée à développer consiste à doter le processus de conception du système de la notion de preuve formelle. Plus précisément, il s'agit de prouver le développement d'un réseau de communications sur puce (NoC) tolérant aux fautes des systèmes MPSoC. Un premier travail dans ce sens a été abordé [DTBH14], avec pour objectif la réalisation d'une ébauche de formalisation d'une architecture NoC en se basant essentiellement sur le langage B et à travers

les spécifications et particularités de l'architecture matérielle structurelle développée au LCOMS-ASEC. L'utilisation d'outil de preuve devrait contribuer à valider l'intégration de la preuve par raffinements successifs lors d'une conception et intégration des mécanismes proposés de tolérance aux fautes des routeurs et des NoC SdF. Dans le cas d'étude actuel, il s'agit de formaliser l'approche de vérification par preuve d'une architecture micro-électronique d'un routeur NoC combinant simultanément, traitement de données, acheminement de paquets de données et sûreté de fonctionnement selon les mécanismes proposés dans ces travaux de thèse. L'objectif premier recherché est la vérification de structures de communication compte tenu de leur importance dans la conception des systèmes auto-organisés actuels, et afin de les valider par preuve les propriétés de sûreté de fonctionnement qu'elles doivent posséder.

En perspective à long terme, un autre aspect fondamental à prendre en compte lors de la conception d'un système reconfigurable auto-organisé est l'aspect d'apprentissage permettant au système l'acquisition de connaissances par expériences comportementales et auto-adaptatives face à des environnements de traitement évolutifs.

Liste des Acronymes

AES Advanced Encryption Standard
ASIC Application-Specific Integrated Circuit
BIST Built-In Self-Test
BRAM Block Ram
CCE Code Correcteur d'Erreurs
CE Chip Enable
CLB Configurable Logic Block
CLK Cycles d'Horloge
CRC Cyclic Redundancy Check
DfT Design for Testability
DRU Dynamic Reconfiguration Unit
DSP Digital Signal Processor
EdT Elément de Test
FIFO First In First Out
FPGA Field-Programmable Gate Array
HDL Hardware Description Language
ICAP Internal Configuration Access Port
IDD Indication Diagonale de Disponibilité
IP Intellectual Property
JTAG Joint Test Action Group
LED Light-Emitting Diode
LR WPAN Low Rate Wireless Personal Area Network
LUT Look-Up-Table
MBU Multiple Bit Upset
MOS Metal Oxide Semiconductor
MOSFET Metal Oxide Semiconductor Field Effect Transistor
MPSoC Multi-Processor System on Chip
NoC Network on Chip

PaP Point à Point
PE Processor Elements
PR Partial Reconfiguration
PRM Partial Reconfiguration Module
PRR Partial Reconfiguration Region
RD Reconfiguration Dynamique
RCSF Réseau de Capteurs Sans Fil
RNoC Reconfigurable Network on Chip
RSoC Reconfigurable System on Chip
RSS Reconfigurable Self-Organized System
RTL Register Transfer Level
SaF Store and Forward
SBU Single Bit Upset
SdF Sûreté de Fonctionnement
SRD Système Reconfigurable Dynamiquement
SEB Single Event Burnout
SEE Single Event Effect
SEFI Single Event Fonctional Interupt
SEGR Single Event Gate Rupture
SEL Single Event Latchup
SET Single Event Transient
SEU Single Event Upset
SoC System on Chip
SRAM Static Random Access Memory
UART Universal Asynchronous Receiver Transmitter
VHDL VHSIC Hardware Description Language
WSN Wireless Sensor Network
ZGS Zone Globalement Sûre

Liste de publications

Revues Internationales

M. Heil, C. Tanougast and C. Diou, "Integrated Hybrid Test Mechanism for Networked Smart Reliable Network-on-Chips", IEEE Transactions on Very Large Scale Integration (VLSI) Systems,September 2015, 16 pages, Submitted.

M. Heil, C. Tanougast and C. Diou, "Networked Reliable Embedded Self-Organized System for Vision Applications", Journal of Real-Time Image Processing, 2015, 13 pages, Under review.

Communications internationales avec comités de lecture et publications des actes

M. Heil and C. Tanougast, "Fault-Tolerant Self-Organized Mechanism for Networked Recongurable MPSoC", 2nd International Conference on Control, Decision and Information Technologies, IEEE Control System Society and IEEE Computer Society, 2014, pp. 774-777. (WoScience)

H. Daoud, C. Tanougast, M. Belarbi and M. Heil, "Formal Specication and Verication of Wireless Networked Self-Organized Systems-on-Chip", 2nd International Conference on Control, Decision and Information Technologies, IEEE Control System Society and IEEE Computer Society, 2014, pp. 730-735. (WoScience)

M. Frihi, M. Boutalbi, C. Tanougast, M. Heil and S. Toumi, "Optimized and Dependable Router Suitable for Dynamic Networks-on-Chip", 2nd International Conference on Control, Decision and Information Technologies, IEEE Control System Society and IEEE Computer Society, 2014, pp. 783-788. (WoScience)

M. Heil and C. Tanougast, "A wireless fault-tolerant mechanism for networked Network-on-Chip", International Conference on Embedded Systems in Telecommunications and Instrumentation, 2014, 4 pages.

M. Frihi, M. Boutalbi, S. Toumi, C. Tanougast and M. Heil, "Reliable router for FPGA based adaptative Network-on-Chip", International Conference on Embedded Systems in

Telecommunications and Instrumentation, 2014, 4 pages.

H. Douad, C. Tanougast, M. Belarbi and M. Heil, "Formal verification of wireless networked Network-on-Chip", International Conference on Embedded Systems in Telecommunications and Instrumentation, 2014, 6 pages.

M. Heil, C. Tanougast, C. Killian, A. Dandache, "Self-Organized Reliability Suitable for Wireless Networked MPSoC ", The 25th International Conference on Microelectronics, IEEE Advancing Technology for Humanity, 2013. pp.1-4.(WoScience)

M. Heil, C. Tanougast, K. Cheng, A. Dandache, "Wireless Network for Self-Reconfigurable Hardware Nodes", International Conference on Computer, Networks and Communication Engineering, Advances in Intelligent Systems Research, Atlantis Press, May 2013, pp. 243-246. (WoScience)

Colloques nationaux avec comité de lecture et publications des actes

M. Heil, C. Tanougast, A. Dandache "Mécanisme Auto-Organisé de Tolérance aux Fautes pour MPSoC Reconfigurable Communicant en Réseau", 17ème Journées Nationales du Réseau Doctoral en Micro-Nano-électronique, 2014.

Bibliographie

[A+72] Philip W Anderson et al., *More is different*, Science **177** (1972), no. 4047, 393–396.

[AB15] A. Pinna-B. Granado F. Pêcheux A. Brière, J. Denoulet, *A rf network-on-chip dynamically reconfigurable for many-cores*, Technique et Science Informatiques (TSI) **vol. 34/1-2** (2015), pp. 11–29.

[AD14] El-Bay Bourennane Abdellatif Mtibaa Atef Dorai, Virginie Fresse, *A novel architecture for inter-fpga traffic collision management*, International Conference on Computational Science and Engineering (2014).

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, *Basic concepts and taxonomy of dependable and secure computing*, IEEE Trans. Dependable Secur. Comput. **1** (2004), no. 1, 11–33.

[Ami11] Mohsin Amin, *Conception d'une architecture journalisé tolé rante aux fautes pour un processeur à pile de données*, Thèse en systèmes électroniques, Université de Metz, 2011.

[ASSC02] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci, *Wireless sensor networks : a survey*, Computer networks **38** (2002), no. 4, 393–422.

[BCT08] Brendan Bridgford, Carl Carmichael, and Chen Wei Tseng, *Single-event upset mitigation selection guide*, Xilinx, March 2008, Application Note : FPGAs.

[BK09] Amit Berman and Idit Keidar, *Low-overhead error detection for networks-on-chip*, Proceedings of the 2009 IEEE international conference on Computer design (Piscataway, NJ, USA), ICCD'09, IEEE Press, 2009, pp. 219–224.

[Bru04] Philippe Brunet, *Exploration multicriteres d'architecturesa reconfiguration dynamique*, These de doctorat, Université Henri Poincaré, France (2004).

[Che11] Kevin Cheng, *Reconfigurable self-organised systems : Study, integration and analysis*, Ph.D. thesis, Université Paul Verlaine de Metz, 2011.

[CTA08] Sesh Commuri, V Tadigotla, and Mohammed Atiquzzaman, *Reconfigurable hardware based dynamic data aggregation in wireless sensor networks*, International Journal of Distributed Sensor Networks **4** (2008), no. 2, 194–212.

[CTBD11] K. Cheng, C. Tanougast, C. Bobda, and A. Dandache, *Peer-to-peer control application for reconfigurable self-organised system*, Signal and Image Processing Conference, vol. 2011, ACTA Press, 2011.

[CZF+07] Yves-Andre Chapuis, Lingfei Zhou, Yamato Fukuta, Yoshio Mita, and Hiroyuki Fujita, *Fpga-based decentralized control of arrayed mems for microrobotic application*, Industrial Electronics, IEEE Transactions on **54** (2007), no. 4, 1926–1936.

[CZM+10] Kevin Cheng, Ali Akbar Zarezadeh, Felix Muhlbauer, Camel Tanougast, and Christophe Bobda, *Auto-reconfiguration on self-organized intelligent platform*, In proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2010) **Volume 2010** (2010), Pages309–316.

[Dig09] Digi, *Xbee-xbee-pro rf modules, product manual v1.xex - 802.15.4 protocol*, Digi International Inc **2009** (2009).

[DTBH14] Hayat Daoud, Camel Tanougast, Mostefa Belarbi, and Mikael Heil, *Formal specification and verification of wireless networked self-organized systems on chip*, Control, Decision and Information Technologies (CoDIT), 2014 International Conference on, IEEE, 2014, pp. 730–735.

[DWH05] Tom De Wolf and Tom Holvoet, *Emergence versus self-organisation : Different concepts but promising when combined*, Engineering self-organising systems, Springer, 2005, pp. 1–15.

[EYPZ09] Ashkan Eghbal, Pooria M. Yaghini, Hossein Pedram, and Hamid R. Zarandi, *Fault injection-based evaluation of a synchronous NoC router*, On-Line Testing Symposium, IEEE International **0** (2009), 212–214.

[FCCK06] Arthur Pereira Frantz, Luigi Carro, Erika Cota, and Fernanda Lima Kastensmidt, *Evaluating SEU and crosstalk effects in network-on-chip routers*, On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International, IEEE, 2006, pp. 2–pp.

[GAP+07] Cristian Grecu, Lorena Anghel, Partha P. Pande, Andre Ivanov, and Resve Saleh, *Essential fault-tolerance metrics for NoC infrastructures*, Proceedings of the 13th IEEE International On-Line Testing Symposium (Washington, DC, USA), IEEE Computer Society, 2007, pp. 37–42.

[Ger01] Carlos Gershenson, *Complex philosophy*, arXiv preprint nlin/0109001 (2001).

[Ger05] ———, *A general methodology for designing self-organizing systems*, arXiv preprint nlin/0505009 (2005).

[GG00] Pierre Guerrier and Alain Greiner, *A generic architecture for on-chip packet-switched interconnections*, Proceedings of the conference on Design, automation and test in Europe, ACM, 2000, pp. 250–256.

[GGRG09] Rafael Garcia, Ann Gordon-Ross, and Alan D George, *Exploiting partially reconfigurable FPGAs for situation-based reconfiguration in wireless sensor networks*, Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on, IEEE, 2009, pp. 243–246.

[GGWT11] Martin Gag, Philipp Gorski, Tim Wegner, and Dirk Timmermann, *Evaluation of switch-to-switch header flit protection schemes in networks-on-chip*, In proceedings of GMM/GI/ITG- Fachtagung Zuverlassigkeit und Entwurf (ZuE 2011), 2011.

[GIS+06] Cristian Grecu, Andre Ivanov, Res Saleh, Egor S Sogomonyan, and Partha Pratim Pande, *On-line fault detection and location for NoC interconnects*, On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International, IEEE, 2006, pp. 6–pp.

[GOR15] Florent De Lamotte Éric Rutten El-Bay Bourennane Diguet Jean-Philippe Gogniat Guy Gilberto Ochoa-Ruiz, Sébastien Guillet, *An mde approach for rapid prototyping and implementation of dynamic reconfigurable systems*, ACM Transactions on Design Automation of Electronic Systems (2015).

[Gra02] Mentor Graphics, *Modelsim foreign language interface, version 5.6d*, August 2002.

[H+01] Francis Heylighen et al., *The science of self-organization and adaptivity*, The encyclopedia of life support systems **5** (2001), no. 3, 253–280.

[Hak06] Hermann Haken, *Information and self-organization : A macroscopic approach to complex systems*, Springer Science & Business Media, 2006.

[Han05] Jie Han, *Toward hardware-redundant, fault-tolerant logic for nanoelectronics*, IEEE Design & Test of Computers **22** (2005), 328–339.

[HBBN06] Mohammad Hosseinabady, Abbas Banaiyan, Mahdi Nazm Bojnordi, and Zainalabedin Navabi, *A concurrent testing method for NoC switches*, Proceedings of the conference on Design, automation and test in Europe : Proceedings (3001 Leuven, Belgium, Belgium), DATE '06, European Design and Automation Association, 2006, pp. 1171–1176.

[Hey89] Francis Heylighen, *Self-organization, emergence and the architecture of complexity*, Proceedings of the 1st European conference on System Science, vol. 18, AFCET Paris, 1989, pp. 23–32.

[HJ01] Francis Heylighen and Cliff Joslyn, *Cybernetics and second order cybernetics*, Encyclopedia of physical science & technology **4** (2001), 155–170.

[HK12] Chin Hau Hoo and Akash Kumar, *An area-efficient partially reconfigurable crossbar SWitch with low reconfiguration delay*, Proceeding of the 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012.

[HTCD13] M. Heil, C. Tanougast, K. Cheng, and Abbas D., *Wireless network for self-reconfigurable hardware nodes*.

[IEE09] IEEE, *802.15.4 standard*, ,2009, Available from http ://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf.

[Jov09] Slavisa Jovanovic, *Architecture reconfigurable de systemes embarqué auto-organisé*, Ph.D. thesis, PhD thesis, Université Henry Poincaré-Nancy 1, Laboratoire d'Instrumentation Electronique de Nancy (LIEN), 2009.

[JS10] Szilárd Jaskó and Gyula Simon, *Reconfigurable sensor network architecture for distributed measurement systems*, Proc. I2MTC (2010), 198–203.

[JTBW09] S. Jovanović, C. Tanougast, C. Bobda, and S. Weber, *Cunoc : A dynamic scalable communication structure for dynamically reconfigurable FPGAs*, Microprocess. Microsyst. **33** (2009), no. 1, 24–36.

[JTW08] S. Jovanovic, C. Tanougast, and S. Weber, *A new high-performance scalable dynamic interconnection for FPGA-based reconfigurable systems.*, ASAP, 2008, pp. 61–66.

[KASN08] Naghmeh Karimi, Armin Alaghi, Mahshid Sedghi, and Zainalabedin Navabi, *Online network-on-chip switch fault detection and diagnosis using functional switch faults*, Journal of Universal Computer Science (J-JUCS) **14** (2008), no. 22, 3716–3736.

[Kil12] Cédric Killian, *Réseaux embarqués sur puce reconfigurable dynamiquement et sûrs de fonctionnement*, Ph.D. thesis, Université de Lorraine, 2012.

[Kon11] Cheick-Tidjane Kone, *Conception de l'architecture d'un réseau de capteurs sans fil de grande dimension*, Ph.D. thesis, Université Henri Poincaré-Nancy I, 2011.

[KPGdlT09] Yana E Krasteva, Jorge Portilla, Félix Tobajas Guerrero, and Eduardo de la Torre, *Using partial reconfiguration for soc design and implementation*, SPIE Europe Microtechnologies for the New Millennium, International Society for Optics and Photonics, 2009, pp. 736306–736306.

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

[Lan90] Chris G Langton, *Computation at the edge of chaos : phase transitions and emergent computation*, Physica D : Nonlinear Phenomena **42** (1990), no. 1, 12–37.

[LLP07] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila, *Online reconfigurable self-timed links for fault tolerant NoC*, VLSI Design **2007** (2007), 13 pages.

[LMS91] J-L LE MOIGNE and Herbert A SIMON, *Sciences des systèmes, sciences de l'artificiel*, Dunod, 1991.

[LR06] Paulo Leitão and Francisco Restivo, *Adacor : A holonic architecture for agile and adaptive manufacturing control*, Computers in industry **57** (2006), no. 2, 121–130.

[LR15] B. Granado L. Rodriguez, B. Miramond, *Toward a sparse self-organizing map for neuromorphic architectures*, ACM Journal on Emerging Technologies in Computing Systems **vol. 11 (4)** (2015), pp. 33.

[LSH+09] S. Lin, W. Shen, C. Hsu, C. Chao, and A. Wu, *Fault-tolerant router with built-in self-test/self-diagnosis and fault-isolation circuits for 2D-mesh based chip multiprocessor systems*, International Journal Of Electrical Engineering **16** (2009), 213–222.

[Luc97] Chris Lucas, *Transient attractors and emergent attractor memory*, CALResCo Group. URL : http ://www. calresco. org/transatr. htm **17** (1997).

[MGC+07] Leandro Möller, Ismael Grehs, Ewerson Carvalho, Rafael Soares, Ney Calazans, and Fernando Moraes, *A NoC-based infrastructure to enable dynamic self reconfigurable systems*, ReCoSoC, 2007, pp. 23–30.

[Mic08] Microchip, *Zena wireless network analyzer, users guide, reference DS51606c*, Microchip Technology **200** (2008).

[Mic11] Microsemi, *FPGA reliability and the sunspot cycle*, September 2011.

[MRF+03] S Mostefaoui, Omer F Rana, Noria Foukia, Salima Hassas, G Di Marzo Serugendo, Chris Van Aart, and Anthony Karageorgos, *Self-organising applications : a survey.*

[MT12] El-Bay Bourennane Abderrezak Guessoum Kamel Messaoudi Maamar Touiza, Gilberto Ochoa-Ruiz, *A novel methodology for accelerating bitstream relocation in partially reconfigurable systems*, Microprocessors and Microsystems (2012), 4.

[MTV+05] Srinivasan Murali, Theocharis Theocharides, N. Vijaykrishnan, Mary Jane Irwin, Luca Benini, and Giovanni De Micheli, *Analysis of error recovery schemes for networks on chips*, IEEE Des. Test **22** (2005), no. 5, 434–442.

[Muk08] Shubu Mukherjee, *Architecture design for soft errors*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[MZ04] Marco Mamei and Franco Zambonelli, *Self-organization in multi agent systems : A middleware approach*, Engineering Self-Organising Systems, Springer, 2004, pp. 233–248.

[N.N12] C.ESCRIBA J.Y.FOURNIOLS N.NASREDDINE, J.L.BOIZARD, *Embedded fpga simulator for wireless sensor network desig*, Journal of Energy and Power Engineering **Vol.6** (2012), pp.984–992.

[Par96] H Van Dyke Parunak, *Applications of distributed artificial intelligence in industry*, Foundations of distributed artificial intelligence **2** (1996).

[PB04] H Van Dyke Parunak and Sven A Brueckner, *Engineering swarming systems*, Methodologies and Software Engineering for Agent Systems, Springer, 2004, pp. 341–376.

[PBEKCRA09] Jorge Portilla Berrueco, Yana Esteves Krasteva, Jose María Carnicer, and Teresa Riesgo Alcaide, *Wireless sensor networks node with remote HW/SW reconfiguration capabilities.*

[PHF07] Giovanni Pezzulo, Joachim Hoffmann, and Rino Falcone, *Anticipation and anticipatory behavior*, Cognitive processing **8** (2007), no. 2, 67–70.

[PO07] Kim Petersén and Johnny Öberg, *Toward a scalable test methodology for 2D-mesh network-on-chips*, Proceedings of the conference on Design, automation and test in Europe (San Jose, CA, USA), DATE '07, EDA Consortium, 2007, pp. 367–372.

[POdlT+10] Jorge Portilla, Andrés Otero, Eduardo de la Torre, Teresa Riesgo, Oliver Stecklina, Steffen Peter, and Peter Langendörfer, *Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors*, International Journal of Distributed Sensor Networks **2010** (2010).

[RAS+10] Vincenzo Rana, David Atienza, MarcoDomenico Santambrogio, Donatella Sciuto, and Giovanni Micheli, *A reconfigurable network-on-chip architecture for optimal multi-processor SoC communication*, VLSI-SoC : Design Methodologies for SoC and SiP (Christian Piguet, Ricardo Reis, and Dimitrios Soudris, eds.), IFIP Advances in Information and Communication Technology, vol. 313, Springer Berlin Heidelberg, 2010, pp. 232–250.

[RGFSC10] M. Raffo, J. Gomes Filho, M. Strum, and Wang Jiang Chau, *A placement tool for a noc-based dynamically reconfigurable system*, Proceedings of the VI Southern Programmable Logic Conference (SPL), IEEE, 2010, pp. 47–52.

[RGU06] Jaan Raik, Vineeth Govind, and Raimund Ubar, *An external test approach for network-on-a-chip switches*, Proceedings of the 15th Asian Test Symposium (Washington, DC, USA), ATS '06, IEEE Computer Society, 2006, pp. 437–442.

[Rob85] Rosen Robert, *Anticipatory systems-philosophical, mathematical and methodological foundations*, P ergamon P ress (1985).

[RUG07] Jaan Raik, Raimund Ubar, and Vineeth Govind, *Test configurations for diagnosing faulty links in NoC switches*, Proceedings of the 12th IEEE European Test Symposium (Washington, DC, USA), ETS '07, IEEE Computer Society, 2007, pp. 29–34.

[RYKO11] Wenjing Rao, Chengmo Yang, Ramesh Karri, and Alex Orailoglu, *Toward future systems with nanoscale devices : Overcoming the reliability challenge*, Computer **44** (2011), no. 2, 46–53.

[S+01] Cosma Rohilla Shalizi et al., *Causal architecture, complexity and self-organization in the time series and cellular automata*, Ph.D. thesis, University of Wisconsin–Madison, 2001.

[SCM+12] Yongxian Song, Ting Chen, Juanli Ma, Yuan Feng, and Xianjin Zhang, *Design and analysis for reliability of wireless sensor network*, Journal of Networks **7** (2012), no. 12, 2003–2010.

[SF89] Norman Sadeh and Mark S Fox, *Cortes : An exploration into micro-opportunistic job-shop scheduling*, Proceedings of Workshop on Manufacturing Production Scheduling, vol. 58, 1989.

[SGC11] Saeed Shamshiri, Amirali Ghofrani, and Kwang-Ting (Tim) Cheng, *End-to-end error correction and online diagnosis for on-chip networks*, International Test Conference, IEEE, IEEE, 09/2011 2011.

[SH10] Jih-Sheng Shen and Pao-Ann Hsiung, *Dynamic reconfigurable network-on-chip design : Innovations for computational processing and communication*, IGI Global, 2010.

[S.K11] H.BOUKABACHE C.ESCRIBA-J.Y.FOURNIOLS S.KSOURI, M.MATMAT, *Damage detection composite laminate aeronautics structures through accelerometers network*, Advances in Materials Sciences **Vol.11** (2011), pp.37–43.

[SKRZ04] Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F Rana, and Franco Zambonelli, *Engineering self-organising systems : nature-inspired approaches to software engineering*, vol. 2977, Springer, 2004.

[SSC08] E. Stott, P. Sedcole, and P. Cheung, *Fault tolerant methods for reliability in FPGAs*, Proc. of the Int. Conf. on Field Programmable Logic and Applications, 2008, pp. 415–420.

[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*, Ph.D. thesis, Nancy 1, 2001.

[Tin08] LIU Ting, *Optimisation par synthese architecturale des methodes de partitionnement temporel pour les circuits reconfigurables*, Ph.D. thesis, These de doctorat, These de doctorat, UHP-Universite Henri Poincare, 2008.

[VDPB01] H Van Dyke Parunak and Sven Brueckner, *Entropy and self-organization in multi-agent systems*, Proceedings of the fifth international conference on Autonomous agents, ACM, 2001, pp. 124–130.

[VWS+12] Markus Völker, Dorothea Wagner, Johannes Schmid, Tobias Gädeke, and Klaus Müller-Glaser, *Force-directed tracking in wireless networks using signal strength and step recognition*, Localization and GNSS (ICL-GNSS), 2012 International Conference on, IEEE, 2012, pp. 1–8.

[WWW06] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen, *VLSI test principles and architectures : Design for testability (systems on silicon)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[Xil] Xilinx, *Integrated software environment (ise), version 13.2*.

[Xil08a] ———, *Early access partial reconfiguration user guide*, 2008.

[Xil08b] ———, *Virtex-5 FPGA configuration user guide*, 2008.

[Xil12] ———, *Virtex 5 FPGA configuration user guide, ug702*, 2012.

[YMG08] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal, *Wireless sensor network survey*, Computer Networks **52** (2008), no. 12, 2292–2330.

[YQG+12] Shenfang Yuan, Lei Qiu, Shang Gao, Yao Tong, and Weiwei Yang, *Providing self-healing ability for wireless sensor node by using reconfigurable hardware*, Sensors **12** (2012), no. 11, 14570–14591.

[YSM09] XU Yuan, QIU Shubo, and HOU Meng, *Reconfigure ZigBee network based on system design*, Wireless Sensor Network **1** (2009), no. 03, 206.