Conception des réseaux sur puce reconfigurables dynamiquement

Rachid Dafali

THÈSE / UNIVERSITÉ DE BRETAGNE SUD

sous le sceau de l'Université Européenne de Bretagne

Pour obtenir le grade de :
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE SUD
Mention : STIC
École Doctorale SICMA

présentée par

Rachid DAFALI

Laboratoire en sciences et technologies de l'information, de la communication et de la connaissance

Conception des réseaux sur puce reconfigurables dynamiquement

Thèse soutenue le 30 Mai 2011,
devant la commission d'examen composée de :

M. Christophe JEGO
Professeur, IPB/ENSEIRB-MATMECA, Talence / Rapporteur

M. Dominique HOUZET
Professeur, INP, Grenoble / Rapporteur

M. Olivier SENTIEYS
Professeur, ENSSAT, Lannion / Examinateur

M. Raphael DAVID
Ingénieur de recherche, CEA LIST, Paris / Examinateur

M. Jean-Philippe DIGUET
Chargé de recherche CNRS, Lab-STICC, Université de Bretagne-Sud, Lorient / Directeur de thèse

M. Marc Sevaux
Professeur, Lab-STICC, Université de Bretagne-Sud, Lorient / Co-directeur de thèse

M. Sylvain FABRE
Ingénieur, InPixal, Rennes / Invité

Résumé

Ce travail de thèse porte sur la problématique des communications entre les unités de traitement ou de stockage d'un système reconfigurable sur puce (RSoC).

Notre approche repose sur l'intégration de mécanismes de reconfiguration dynamique dans les réseaux sur puce afin de répondre aux difficultés croissante de prédiction a priori du trafic au sein des futurs systèmes sur puce. Ainsi, l'objectif est de conférer au NoC des propriétés d'auto-configuration lui permettant de s'adapter en temps réel, aux besoins réels et variables de chaque unité de traitement en termes de qualité de service et de type de transfert.

Cette thèse propose deux approches pour rendre le NoC adaptatif. La première repose sur un concept de mémoires tampons configurables dynamiquement qui permet d'adapter la profondeur des FIFOs dans les interfaces réseau en temps réel et selon les besoins des communications. La seconde approche propose une table TDMA configurable dynamiquement, qui adapte le nombre d'intervalles de temps alloués aux communications selon les besoins tout en conservant la propriété de trafic garanti.

Ce travail a également consisté à développer un nouvel environnement de CAO, μSpider II, pour automatiser le flot de conception. Celui-ci est constitué de plusieurs outils qui permettent l'exploration, l'optimisation, la génération de la description matérielle du NoC, et la simulation de son fonctionnement et ses performances.

L'ensemble des approches ont été validées avec des expériences et implantations sur FPGA qui intègrent les différents versions du NoC μSpider II au sein d'architectures multiprocesseurs.

Mots clés : système reconfigurable sur puce, réseau sur puce, reconfiguration dynamique, administration de la reconfiguration, interface réseau auto-adaptative, adaptateur de protocole, outil de l'aide à la conception, administrateur local, administrateur global, FIFO configurable dynamiquement, table TDMA configurable dynamiquement, topologie.

Abstract

This work addresses the issue of communications between processing or storage units within reconfigurable system on chip.

Our approach relies on the implementation of reconfiguration mechanisms in Network On Chips (NoC) in order to solve the increasing problem of traffic variability in future RSoC. Thus, the objective is to provide the NoC with self-adaptitvity properties so that it can adapt at run-time to real and variable communication requirements of processing and storage units.

This thesis proposes two original and efficient mechanisms of reconfiguration. The first one relies on the concept of dynamically reconfigurable memory buffers that allow for the runtime adaptation of FIFO depths in Network Interfaces according to communication needs. The second one is complementary and controls the TDMA table which is dynamically reconfigurable, it can adapt the number of time slots allocated to different communications according to real bandwidth needs while preserving guaranteed traffic property.

This work also consists in developing a new CAD environment, μSpider II, to automize the design flow. This framework is composed of various associated tools that perform exploration, optimization and VHDL code generation, it also provides material for test and performances evaluation

Both approaches have been validated with experiments and implementations on FPGA with different versions of the μSpider II NoC with multiprocessor architectures.

Keywords : reconfigurable system on chip, network on chip, dynamic reconfiguration, self-adaptivity, reconfiguration management, network interface, protocol adapter, electronic design automation, local manager, global manager, FIFO reconfiguration, TDMA reconfiguration, topology.

n d'ordre : 000000000

Université de Bretagne-Sud

Centre de recherche Christiaan Huygens BP 92116 F-56321 Lorient Cedex FRANCE
Tél : +33 (0)2 97 87 45 60
Fax : +33 (0)2 97 87 45 27

Les savants des temps passés et des nations révolues n'ont cessé de composer des livres. Ils l'ont fait pour léguer leur savoir à ceux qui les suivent. Ainsi demeurera vive la quête de la vérité.

Al-Khwarizmi [783-850]

Table des matières

Introduction générale

Contexte

La première décennie du vingt-et-unième siècle a connu une évolution technologique sans précédent. Cette évolution est due principalement à la démocratisation des télécommunications, par exemple le nombre d'abonnements mobiles recensé par l'ITU [1] est passé de 719 millions à $5,3$ milliards en l'espace de dix ans seulement. Cet engouement pour le mobile est révélateur de l'explosion du marché des systèmes embarqués dont les champs applicatifs sont extrêmement vastes.

Le succès grandissant des nouvelles technologies auprès du grand public a précipité le développement de nouvelles générations de systèmes embarqués plus polyvalents et multi-tâches. Ces nouvelles générations intègrent des architectures matérielles complexes et performantes, afin de réaliser plusieurs fonctions en temps réel. Le meilleure exemple, qui reflète cette multifonctionnalité des systèmes embarqués, est le smartphone. Ce type de mobile intègre des architectures permettant de traiter plusieurs normes réseau, de prendre des photos de haute définition, de naviguer sur le Web, de consulter le courrier électronique, et de réaliser d'autres fonctions facultatives de plus en plus nombreuses.

L'évolution des systèmes embarqués rend leur conception de plus en plus complexe. En effet, les concepteurs doivent exploiter toujours d'avantage le parallélisme et des unités de traitement spécialisées pour atteindre les performances souhaitées, tout en optimisant l'espace occupé et la consommation d'énergie. L'évolution simultanée des technologies d'intégration apporte des solutions pour répondre à ces besoins. En effet les capacités grandissantes des SoC [2] permettent de concevoir des architectures multiprocesseurs (généralistes ou spécialisées) qui offrent le parallélisme requis. Cependant l'accroissement du parallélisme des traitements et des transferts des données impose des schémas de communications de plus en plus complexes.

Ainsi, la problématique d'interconnexion dans les SoCs est devenue une thématique de recherche à part entière, qui soulève un nombre important de questions. Cette thématique a suscité l'intérêt de plusieurs équipes de recherche, ce qui a permis l'émergence de plusieurs approches innovantes. Parmi ces approches, le réseau sur puce (NoC [3]) est apparu comme une solution permettant à la fois d'appréhender la complexité du système d'interconnexion et d'augmenter la bande passante requise.

Le réseau sur puce est un système composé de plusieurs éléments de routage (routeurs) connectés selon une topologie spécifique. Les différents communicants (processeurs, mémoires, IPs, ...) se connectent au réseau via les interfaces réseau. L'architecture du NoC est conçue de manière à intégrer ou exécuter différents protocoles réseau, qui définissent les mécanismes d'échanges des données. Le paradigme réseau sur puce soulève plusieurs problématiques ordonnées selon deux axes de recherches, qui visent la conception d'un réseau répondant globalement aux besoins des applications en termes de communications. Le premier axe adresse les problèmes liés à l'architecture du NoC, tels que le choix de la topologie et le placement des communicants. Le deuxième axe regroupe les problèmes associés aux choix des protocoles à utiliser pour acheminer les données dans le NoC.

La méthodologie de conception des NoCs, utilisée jusqu'à présent, dépend des contraintes définies dans la plupart des cas par le concepteur et selon des estimations qui reposent sur une

1. International Telecommunication Union
2. System on Chip
3. Network on Chip

simple simulation des échanges réalisés par l'application. Cette approche ne tient pas compte du caractère dynamique des applications et se traduit généralement par un surdimensionnement. Un certains nombre de raisons expliquent ces problèmes de dimensionnement, a priori il s'agit par exemple du comportement de caches, des applications dont le comportement dépend des données ou du contexte, de l'interaction au sein du réseau, etc. Il en résulte la nécessité de proposer des solutions d'interconnexion capables de s'adapter en temps réel aux besoins en bande passante.

Ce nouveau concept des réseaux sur puce adaptables a donné lieu au projet ANR [4] AFANA [5], qui vise à concevoir des NoCs en reposant sur une utilisation judicieuse des contraintes. Et en utilisant aussi des métriques qui tiennent compte non seulement des variations du trafic mais également de la nature des données transportées afin de décider en temps réel de la configuration du NoC. Les démonstrateurs développés dans AFANA ont porté sur deux domaines : la vidéo et les communications numériques. Ce projet a réuni 2 partenaires académiques (Telecom bretagne et UBS) et 2 partenaires industriels (Turbo Concept et Thomson).

Objectif

Cette thèse s'est déroulé dans le projet AFANA avec un objectif de conception de réseaux sur puce reconfigurables dynamiquement. Notre rôle dans le projet a été d'étudier les besoins des partenaires industriels, en explorant les contraintes des communications de leur applications. Dans un second temps nous avons développé une méthodologie qui permet de concevoir un réseau garantissant les transferts entre les communicants, en exploitant les mécanismes de configuration dynamique. Pour y parvenir, nous avons commencé dans un premier temps par étudier et concevoir un NoC qui traite principalement des trafics avec une qualité de service qui garantit le trafic. L'étude approfondie de ce type de réseaux, nous a permis d'en montrer les avantages mais aussi les limites en terme de flexibilité dans un contexte dynamique. Ainsi, dans un deuxième temps, nous avons introduit des mécanismes de reconfigurabilité dans le réseau de manière à le rendre adaptatif dynamiquement.

Pour démontrer l'intérêt de notre approche, nous avons conçu une interface réseau qui intègre une architecture pour contrôler plusieurs mémoires tampons avec des tailles reconfigurabes dynamiquement. Cette interface pilote également différentes tables d'ordonnancement, selon les débits demandés par les communicants. Enfin, nous avons développé un environnement CAO permettant de produire automatiquement l'architecture matérielle du réseau en suivant les différentes étapes du flot de conception.

Organisation de la thèse

Nous avons organisé la thèse en trois parties. La première partie présente un état de l'art sur les réseaux sur puce, elle décrit également l'architecture des différents composants de notre NoC et son flot de conception. La deuxième partie décrit le concept des réseaux sur puce reconfigurables dynamiquement, en étudiant les travaux réalisés dans ce domaine, en définissant un nouveau modèle de conception pour ce type de réseau, et en expliquant les mécanismes de reconfiguration intégrés pour rendre le NoC adaptatif. Enfin, La troisième partie détaille les environnements de conception, de simulation et d'implantation que nous avons développés pour tester notre réseau,

4. Agence Nationale de la Recherche
5. Application-Field-Aware Adaptive Network on chip Architecture

elle explique également les expériences que nous avons menées pour démontrer l'intérêt de notre approche.

Ce manuscrit est composé de 7 chapitres, nous distinguons :

1. *réseau sur puce : définitions et état de l'art*, ce chapitre a pour objectif d'identifier les différents paramètres pour concevoir un réseau, de les classifier selon le modèle de référence OSI, et de décrire l'impact des choix sur les performances et la fonction de coût ;
2. *paramétrage de l'architecture et des protocoles du NoC μSpider II*, ce chapitre définit les paramétrages que nous avons réalisés dans les différentes couches du modèle OSI pour concevoir un réseau sur puce, qui garantit le trafic. Il présente aussi l'architecture des différents composants du réseau μSpider II ;
3. *Dimensionnement et flot de conception du NoC μSpider II*, présente les cinq étapes du flot de conception, ainsi que les algorithmes développés pour dimensionner les mémoires tampons, pour calculer les intervalles de temps des tables TDMAs, et pour construire les chemins spatio-temporels ;
4. *les réseaux sur puce adaptatifs*, ce chapitre explique le modèle de conception que nous avons développé pour les NoCs reconfigurables dynamiquement ;
5. *architecture et protocoles de la version reconfigurable du réseau μSpider II*, ce chapitre définit les mécanismes de reconfiguration et la méthode utilisée pour les intégrer au NoC μSpider II. Il détaille aussi la nouvelle architecture de l'interface réseau auto-adaptative ;
6. *environnements de conception, de simulation, et d'implantation du réseau μSpider II*, ce chapitre décrit les outils développés pour aider le concepteur à modéliser et générer la description matérielle de NoC. Il présente également les environnements de test, qui permettent de vérifier le fonctionnement et les performances du NoC ;
7. *résultats d'implantation du NoC μSpider II*, ce chapitre présente les différentes expériences qui ont permis de comparer les performances du NoC statique et le NoC reconfigurable dynamiquement.

Partie I : Paradigme d'interconnexion des systèmes sur puce

1

Réseau sur puce : définitions et état de l'art

L'espace de conception d'une architecture de communication sur puce se définit par plusieurs paramètres, tels que la topologie, l'algorithme de routage, et la politique d'arbitrage. Par conséquent, le choix des valeurs des différents paramètres pour réaliser un réseau de communication simple et efficace est difficile et complexe. Puisque, chaque décision a pour but d'optimiser la fonction de coût du réseau et de satisfaire les contraintes.

La meilleure manière de maitriser la complexité de l'espace de conception d'un réseau est de le diviser en sous-problèmes indépendants. Ainsi, chacun des sous-problèmes contrôle un nombre limité de paramètres. Cette division est soumise à deux conditions majeures. La première impose que les paramètres dépendants doivent appartenir au même sous-problème. La deuxième condition consiste à choisir un ordre de traitement des différents sous-problèmes, cet ordre doit minimiser la fonction de coût du réseau. De ce fait, plus l'impact d'un sous-problème sur la fonction de coût est important, plus sa priorité de traitement augmente.

Le modèle de référence OSI introduit dans la section 1.1 est la solution évidente pour segmenter l'espace de conception d'un réseau sur puce en sous problèmes, puisqu'il classifie les paramètres dépendants dans des couches indépendantes.

L'objet de ce chapitre est d'identifier les différents paramètres pour concevoir un réseau, de les classifier selon le modèle de référence OSI, et de décrire l'impact des choix sur les performances et la fonction de coût.

Lors de notre investigation, nous avons identifié beaucoup de réseaux sur puce, différentes études [BM06, SKH08, AIS09] et plusieurs états de l'art complets et précis, tels que, l'état de l'art de la thèse d'Anthony Leroy [Ler06]. Ce travail nous a permis d'identifier les paramètres des réseaux sur puce, et de les classifier selon le modèle OSI. Pour notre état de l'art, nous avons sélectionné quelques réseaux sur puce pour les comparer tout au long de ce chapitre, ces réseaux sont :

- le réseau Æthereal [RGR$^+$03, GDR05] développé par *Philips* ;
- le réseau MANGO (*Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces*) [BS04, BS05] de la *Technical University of Denmark* ;
- le réseau Proteo [AN05] qui est le fruit d'un consortium entre les universités finlandaises (*Tampere University of Technology* et *University of Turku*) et l'institut suédois (*Stockholm Royal Institute of Technology*) ;
- le réseau QNoC [BCGK04] développé par le *Technion-Israel Institute of Technology* ;
- le réseau SoCBus [WL03] de l'univérsité *Linköpings* ;
- le réseau SPIN (*Scalable Programmable Integrated Network*)[GG00, ACG$^+$03] crée au sein de l'université Pierre et Marie Curie ;
- le réseau STNoC ou Spidergon [STM] conçu par *STMicroelectronics* ;
- et le réseau XPIPES [DBG$^+$03, BB04, BJM$^+$05] élaboré par l'université de Bologne.

Le choix de ces réseaux n'est pas le fruit du hasard, mais il permet de créer une sélection qui met la lumière sur le maximum de valeurs que les paramètres des réseaux sur puce peuvent avoir.

1.1 Le modèle de référence OSI

En 1977, l'organisation internationale de standardisation ISO [6] crée une nouvelle sous-commission pour développer une architecture d'interconnexion pour systèmes ouverts qui servira comme structure pour normaliser les protocoles de communications. En 1979, la commission OSI [7] adopte une architecture intitulée modèle de référence OSI [Zim80, DZ83], ce modèle est divisé en sept couches : physique, liaison de donnée, réseau, transport, session, présentation, et application (Fig.1.1). Ces couches sont décrites de la façon suivante :

- la *couche application* est le point d'accès de l'utilisateur aux différents services offerts par le réseau, comme l'établissement de connexion, le transfert de fichiers, et la configuration du réseau ;
- la *couche présentation* contrôle la syntaxe et la sémantique des données transmises, elle est chargée du codage, du chiffrement et de la compression des données ;
- la *couche session* s'occupe de l'organisation et de la synchronisation des échanges et des transactions ;
- la *couche transport* traite les communications de bout en bout entre processus ;
- la *couche réseau* contrôle les communications de bout en bout entre routeurs. Cette gestion change selon les mécanismes de routages, la topologie du réseau, et les techniques de commutations utilisées ;
- la *couche liaison de données* décrit le comportement d'échange entre deux composants adjacents reliés par un support physique ;
- et la *couche physique* qui prend en charge la transmission effective des signaux entre les composants du réseau.

Le modèle de référence OSI détermine le rôle de chaque couche en définissant des normes, mais sans préciser véritablement les services et les protocoles intégrés dans chacune. De plus, la segmentation du modèle OSI est flexible car le regroupement de certaines couches en une seule est envisageable. D'ailleurs, le modèle de référence utilisé pour segmenter l'espace de conception d'un réseau sur puce simplifie le modèle OSI en regroupant les couches hautes (5, 6 et 7) en une seule. Nous considérons en effet qu'elles ont la même responsabilité, qui se résume au traitement de l'information relative à la gestion des échanges entre les éléments connectés au réseau. Ainsi, le nouveau modèle de référence adopté comporte seulement cinq couches :

1. la *couche application* (section 1.2) joue le rôle d'une interface entre les blocs fonctionnels complexes IP [8] et le réseau, elle offre plusieurs services pour contrôler les échanges ;
2. la *couche transport* (section 1.3) repose sur un protocole qui garantit le transfert de données de bout en bout entre la source et la destination. Ainsi, elle contrôle le flux de données en utilisant des mécanismes pour la gestion de la qualité de service, et des stratégies pour le contrôle du flux de données afin de minimiser les risques de contention ;

6. International Organization for Standardization
7. Open Systems Interconnection
8. Intellectual Property

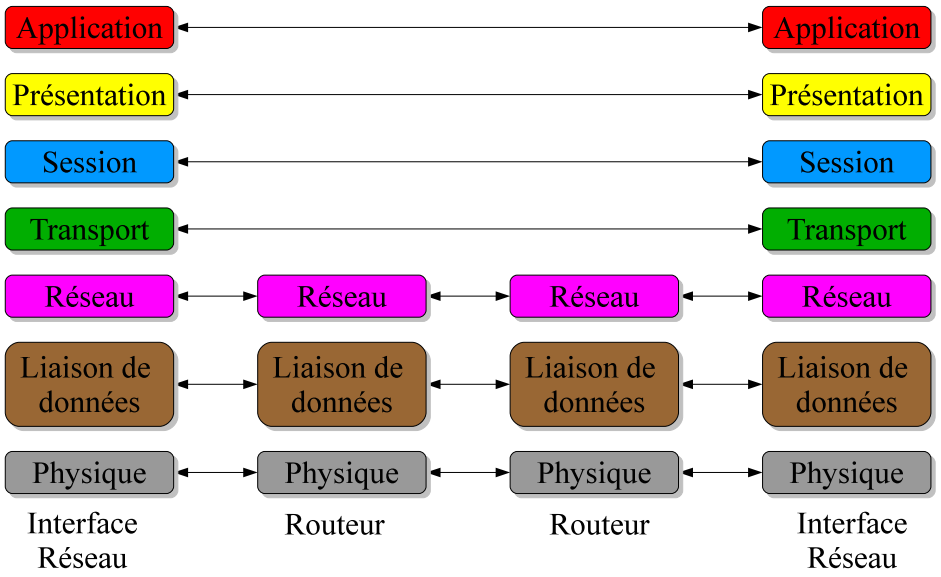


FIGURE 1.1 – Représentation des protocoles d'échanges des composants réseau selon les sept couches du modèle de référence OSI.

3. la *couche réseau* (section 1.4) permet l'acheminement des paquets de la source à la destination. Cette couche définit la topologie, la technique de routage, la technique de commutation, la politique de mémorisation, et le codage des instructions ;
4. la *couche liaison de données* (section 1.5) contrôle les ressources réseau pour l'acheminement d'un paquet, en définissant le protocole d'échange de données entre les routeurs, la technique de multiplexage, et la technique de détection et correction d'erreurs ;
5. et la *couche physique* (section 1.6) définit la largeur des mots transportés sur le support physique, et le mode de synchronisation des transferts.

Nous décrivons plus en détails le fonctionnement et les paramètres de ces cinq couches dans les section suivantes.

1.2 La couche application

La couche application définit l'interface entre le support de communication et le cœur de traitement qui exécute l'application. Le rôle de cette couche est de fournir un canal de communication abstrait entre les différents cœurs de traitement connectés au réseau, en administrant la frontière entre l'application et le réseau. Cette administration est intégrée dans l'adaptateur réseau, qui offre aux cœurs de traitement des services de communications de haut niveau. Puis, il transforme ces services de haut niveau en procédures de transmission ou de réception adaptées aux protocoles du réseau. Par conséquent, la couche application définit la qualité de service des communications que nous expliquons dans la section 1.2.1, et le protocole d'échange de l'adaptateur réseau que nous détaillons dans la section 1.2.2.

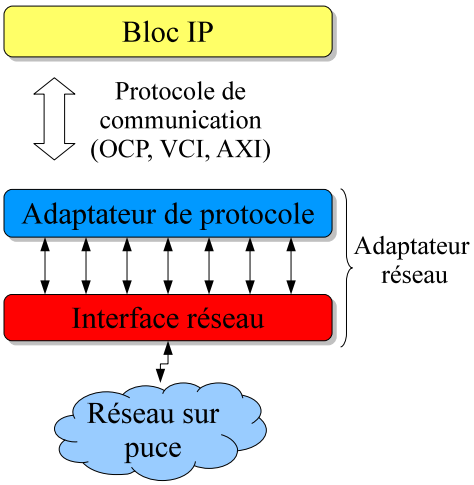


FIGURE 1.2 – La connexion d'un bloc IP avec le réseau via l'adaptateur réseau

1.2.1 Qualité de service

Dans le contexte des communications sur puce, le but de la qualité de service (QoS [9]) est de résoudre le non déterminisme intrinsèque de l'occupation des ressources d'un réseau sur puce. Ainsi, la qualité de service consiste à fournir aux communications un niveau de garantie plus ou moins élevé. Ce niveau de garantie repose généralement sur l'incorporation ou pas de différents services, tels que :

- l'intégrité des données qui garantit un transfert sans perte et sans altération des données ;
- l'ordonnancement des données qui garantit un ordre de réception des données identique à celui de l'émission ;
- et le respect des contraintes de débit et de latence.

Pour intégrer ces services, nous identifions deux classes de QoS. Le trafic garanti (GT [10]) qui réserve certaines ressources du réseau à une communication pour garantir un débit ou/et une latence quelque soit la charge du réseau. Et le trafic au mieux (BE [11]) qui ne réserve aucune ressource ce qui, par conséquent, n'offre aucune garantie de latence ou de débit.

1.2.2 Adaptateur réseau

Un adaptateur réseau réalise l'interfaçage entre le protocole de communication du réseau et celui des cœurs de traitement. Son rôle est de séparer le traitement effectué dans le bloc IP et les communications acheminées par le réseau. Comme le montre la figure 1.2, l'adaptateur réseau est composé d'une interface réseau connectée directement au réseau et d'un adaptateur de protocole. L'adaptateur de protocole intègre souvent un standard de communication, tel que OCP [12], VCI [13], AXI [14] pour s'interfacer avec le bloc IP.

9. Quality of Service
10. Guaranteed Traffic
11. Best Effort
12. Open Core Protocol
13. Virtual Component Interface
14. Advance eXtensible Interface

	Couche application	
Réseau	*Qualité de service*	*Adaptateur de protocole*
XPIPES	BE	OCP
SPIN	BE	VCI
Proteo	BE	OCP et VCI
STNoC	GT	AXI, OCP et STBus
SoCBus	GT	Dédié
MANGO	GT et BE	OCP et VCI
Æthereal	GT et BE	AXI, OCP et VCI
QNoC	Quatre services	dédié

TABLE 1.1 – Comparaison d'une sélection de réseaux sur puce selon les paramètres de la couche application

1.2.3 État de l'art

Le tableau 1.1 montre que la plupart des réseaux sur puce adoptent une interface réseau utilisant le standard OCP, afin d'assurer une large utilisation. Nous constatons aussi que les réseaux sur puce intègrent principalement deux types de trafics BE et GT, sauf le réseau QNoC [BCGK04] qui adapte la qualité de service aux besoins réels. Ainsi, il identifie quatre services, un service pour les messages urgents et courts, un service temps réel, un service d'accès à la mémoire pour les messages courts, et un service pour le transfert des blocs de données réservé pour les messages très longs.

Nous passons maintenant à la définition des paramètres de la couche transport.

1.3 La couche transport

La couche transport est construite sur un protocole qui garantit le transfert des données de bout en bout entre la source et la destination. Elle propose plusieurs services pour gérer la fiabilité des communications, tels que, le contrôle de flux, la fragmentation et l'ordonnancement dans le cas où les paquets du même message empruntent des chemins différents. La section suivante présente et explique l'utilisation de ces différents protocoles.

1.3.1 Contrôle de flux

Le contrôle de flux est un mécanisme qui asservit le débit de l'émetteur par rapport à la capacité de réception du destinataire, ainsi la source règle son débit pour ne pas submerger le récepteur. Nous distinguons deux types de contrôle de flux, le contrôle de flux de bout en bout entre la source et la destination, et le contrôle de flux aux extrémités d'un lien.

La méthode la plus utilisée pour le contrôle de flux de bout en bout est basée sur les crédits d'émission. À l'initialisation, l'émetteur dispose d'une quantité de crédits, et à chaque fois qu'il envoit une donnée au récepteur, il décrémente son nombre de crédits. Une fois la donnée reçue et consommée par le récepteur, il l'indique à l'émetteur, en lui envoyant le nombre de crédits correspondant à l'espace vide de sa mémoire de réception.
Il existe différents protocoles pour intégrer le contrôle de flux aux extrémités d'un lien, on distingue principalement les deux suivants :

	Couche transport	
Réseau	Qualité de service	*Contrôle de flux*
XPIPES	BE	ACK/NOACK
SPIN	BE	crédits d'émission
Proteo	BE	non spécifié
STNoC	GT	-
SoCBus	GT	-
MANGO	GT et BE	-
Æthereal	GT et BE	crédits d'émission
QNoC	Quatre services	crédits d'émission

TABLE 1.2 – Comparaison d'une sélection de réseaux sur puce selon les paramètres de la couche transport

- Le protocole *ACK/NOACK* est un mécanisme d'accusé de réception, où l'émetteur stocke la donnée envoyée jusqu'à la réception d'un acquittement du destinataire. Habituellement, l'acquittement est matérialisé par deux signaux *ACK* et *NOACK*. Ainsi, si la donnée est reçue sans altération, le signal *ACK* passe de 0 à 1 pendant un cycle, sinon le signal *NOACK* passe de 0 à 1 pendant un cycle ;
- Le protocole *STALL/GO* utilise deux liens entre chaque paire d'émetteur et de récepteur. Quand le tampon mémoire du récepteur a suffisamment de place pour recevoir une donnée, il fait passer le signal *GO* à 1. Et dès que le tampon mémoire de réception n'a plus assez de place, le récepteur fait passer le signal *STALL* à 1.

La section suivante présente une comparaison des réseaux sur puce de notre état de l'art selon le contrôle de flux utilisé.

1.3.2 État de l'art

Le tableau 1.2 montre que les réseaux gérant un trafic au mieux (BE), intègrent un contrôle de flux, sauf le réseau MANGO [BS04, BS05] qui par son principe de réseau asynchrone inclut implicitement le contrôle de flux. Nous observons aussi que tous les réseaux qui garantissent le trafic n'ont pas l'obligation d'inclure le contrôle de flux car ces réseaux réservent, à chaque communication, des ressources pour acheminer ces paquets sans altération.

1.4 La couche réseau

La couche réseau gère les communications de bout en bout entre les routeurs. Cette gestion change selon les mécanismes de routage, la topologie du réseau et les techniques de commutations utilisées. Pour assurer le bon acheminement des paquets, la couche réseau définit la topologie du réseau, l'algorithme de routage, et la technique de commutation.

1.4.1 Topologie

La topologie d'un réseau désigne sa configuration spatiale c'est-à-dire la manière dont les différents IPs sont disposés et interconnectés. La topologie est généralement représentée par un

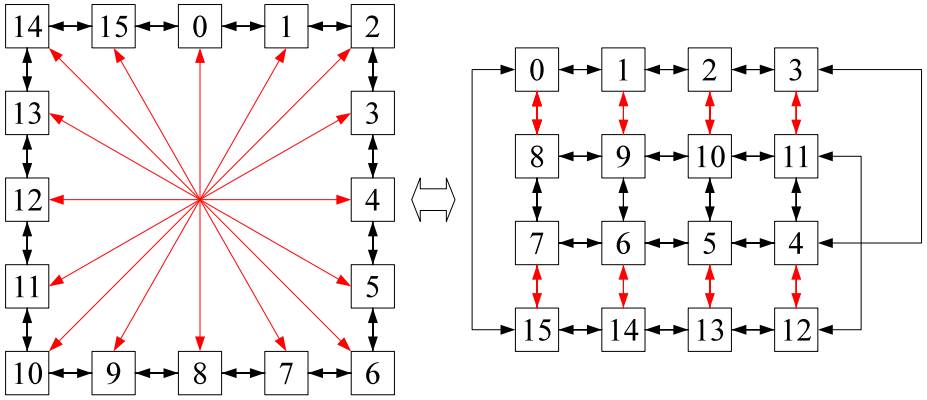


FIGURE 1.3 – La structure de connectivité de la topologie Spidergon

graphe $G(V, E)$, où V représente les sommets c'est à dire l'ensemble des nœuds du réseau, et E représente l'ensemble des liens qui connectent les nœuds.

La structure de connectivité d'un réseau joue un rôle primordial en ce qui concerne le bon fonctionnement des algorithmes de routage, la fiabilité des transmissions, l'efficacité du réseau à répondre aux performance exigées, et la tolérance aux pannes et aux fautes. Ainsi, le choix d'une topologie est une étape critique car elle doit répondre aux exigences en introduisant un impact minimum sur la surface et la consommation d'énergie du réseau.

Plusieurs états de l'art classifient les topologies selon le type d'interconnexion. Ils considèrent que la topologie est régulière si les composants sont raccordés d'une manière uniforme, sinon elle est irrégulière. Nous préférons répertorier les topologies selon la manière de sélection, ainsi si la topologie est construite ou choisie par rapport aux caractéristiques de l'application, nous l'appellerons *topologie spécifique*, sinon si le choix est arbitraire, c'est une *topologie générique*. Parmi les topologies génériques, nous distinguons :

- La *topologie en arbre* : c'est une extension de la topologie en bus. Dans une structure en arbre les éléments sont connectés sur l'un des bus uniques, les bus sont reliés à des ponts, et les ponts sont reliés les uns aux autres et d'une manière hiérarchique jusqu'au pont racine. La topologie en arbre présente plusieurs avantages, la hiérarchie de la structure permet de connecter un nombre limité d'IP sur le même bus unique, ceci diminue la consommation et offre la possibilité d'utiliser une fréquence plus élevée. Elle offre aussi un parallélisme en communication, donc les transactions peuvent se faire en parallèle sur les différents segments des bus. Cependant, l'utilisation des ponts augmente la latence des communications ;
- La *topologie en gille à deux dimensions* (mesh 2D) : c'est la topologie la plus employée, les nœuds sont organisés en maille. Cette topologie est la plus utilisée car les algorithmes de routage assortis sont simples, elle est évolutive, et elle procure en théorie des bandes passantes élevées, Mais c'est la topologie la plus gourmande en terme de surface et de consommation d'énergie car elle se compose d'un nombre élevé de ports et de liens ;
- La topologie Spidergon [CLM+04] représentée sur la figure 1.3 : elle raccorde les blocs IP en anneau et chaque bloc est connecté à son voisin immédiat comme dans une simple architecture polygonale en anneau. Mais chaque bloc IP est également relié directement à son correspondant disposé en diagonal dans le réseau, ce qui permet à l'algorithme de routage de minimiser le nombre de nœuds qu'un paquet doit traverser avant de parvenir

à destination. L'avantage majeur de l'architecture Spidergon est de fournir un compromis coût/performance par rapport à d'autres topologies, comme la topologie en grille 2D ou la topologie en arbre.

Afin d'affecter le moins possible les performances du réseau, son extensibilité, sa surface, et sa consommation. Le choix d'une topologie se doit d'être fondé sur le type de transfert, l'emplacement des communicants sur le réseau, et la qualité de service que requièrent les communications.

La plupart des outils de conception des réseaux sur puce imposent l'utilisation d'une topologie générique unique, généralement la topologie en grille 2D. Ainsi le problème de conception se réduit au placement des blocs IPs sur l'architecture, tout en explorant différentes options de routage [MDM04a, JM05]. Une telle approche a des conséquences sur les performances du réseau en négligeant le choix de la topologie par rapport à la spécificité et à la particularité des communications d'une application donnée.

Pour intégrer et automatiser le choix de la topologie dans le flot de conception des réseaux sur puce. Certains travaux [MDM04b, KCZS01, EMEKG07] sélectionnent une topologie parmi plusieurs topologies génériques. Cette approche s'appuie sur le placement des IPs sur différentes topologies, ensuite une comparaison des performances est effectuée pour choisir la meilleure architecture. L'efficacité de cette méthode dépend de nombreux facteurs, tels que les critères de comparaison, la librairie des topologies, le nombre de topologies comparées, et l'algorithme de placement utilisé.

Parmi les solutions proposées pour automatiser le choix de la topologie, une équipe de l'université de Stanford présente dans [MDM04b] un outil pour la sélection automatique de la meilleure topologie et la génération des réseaux sur puce appelé SUNMAP. Cet outil effectue un placement des IPs sur plusieurs topologies standard (mesh 2D, torus, Hypercube, Butterfly, et clos), puis sélectionne la meilleure architecture en évaluant les modèles en termes de performances et de surface. Le placement est réalisé grâce à une approche heuristique formulée dans [MDM04a], avec une fonction objectif qui vise à réduire le temps moyen des communications, la surface et l'énergie consommée, ainsi qu'à satisfaire les contraintes de bande passante.

De même, l'outil de simulation *Simoo* conçu par l'équipe de l'université fédérale de *Rio Grande do Sul* choisit automatiquement une topologie en analysant les communications d'une application donnée [KCZS01]. Cet outil commence par générer un graphe dynamique de communication en utilisant les traces de simulation de l'application, ensuite il place arbitrairement les IPs sur plusieurs topologies, puis il sélectionne celle qui offre le meilleur temps moyen pour réaliser les différents transferts.

Une autre façon d'aborder la sélection de la meilleure topologie du réseau est le partitionnement [OM05, ASTBN04]. Dans cette approche, le réseau est divisé en deux partitions ou plus puis chaque partition est modélisée par une topologie. Ensuite, les modèles sont regroupés pour former la topologie finale. L'efficacité de cette approche repose sur le choix du modèle pour une partition donnée et sur l'algorithme de partitionnement.

La personnalisation de la topologie du réseau pour une application donnée reste la meilleure solution pour atteindre le plus haut degré de flexibilité dans la conception des réseaux sur puce. Cette personnalisation peut être réalisée grâce à l'exploitation de la spécificité des communications de l'application pour concevoir le réseau qui répond aux exigences du concepteur en terme de performance, surface et énergie consommée.

1.4.2 Granularité du routage

La granularité du routage précise de quelle manière les données seront transmises sur le réseau. Nous distinguons deux types de granularité de routage ou de commutation :

- la **commutation de paquets** (paquet switching) : elle consiste à acheminer les messages en les découpant sous forme de paquets, chaque paquet comporte un entête contenant les adresses nécessaires pour son routage dans les nœuds du réseau. À l'arrivée les messages sont reconstitués à partir des paquets reçus. Ce type de commutation offre une utilisation plus rationnelle des ressources mais demande un contrôle de flux et de congestion ;
- et la **commutation de circuits** (circuit switching) : elle repose sur la réservation d'un ensemble de ressources du réseau pour établir un chemin entre la source et la destination pendant le transfert de tout le message. À la fin de l'envoi, le chemin est dissous.

1.4.3 Les règles de routage

Les règles de routage sont des mécanismes qui élaborent le chemin à emprunter par un message d'un point à un autre. La technique de routage fait partie d'une algorithmique complexe, de part la distribution des décisions à prendre, qui relèvent à la fois de l'espace et du temps. L'algorithme de routage permet de renseigner les tables ou les vecteurs de routage de la fonction de commutation. Il n'est pas exécuté pour chaque message, mais il est invoqué périodiquement pour élaborer les routes ou chemins disponibles. Les principales considérations que l'algorithme de routage prend en compte pour déterminer les meilleures routes dans le réseau, sont :

- le coût des liaisons ;
- le coût de passage dans un nœud (routeur) ;
- le nombre de nœuds à traverser pour arriver à la destination ;
- le débit et la latence demandés ;
- la gestion des conflits et des blocages ;
- et la gestion des contentions en trouvant des chemins alternatifs.

Parmi les algorithmes de routage qui s'adaptent à différentes topologies, il existe l'algorithme déterministe où le chemin du paquet est défini à la source, et l'algorithme adaptatif où tous les chemins sont potentiellement autorisés.

1.4.3.1 Le routage déterministe

Les algorithmes de routage déterministes établissent les chemins des paquets en fonction de l'adresse de destination utilisant toujours le même chemin entre chaque paire de nœuds. Le routage ordonné par dimensions X-Y est un algorithme de routage déterministe utilisé sur les topologies en grille 2D. Cet algorithme impose que les routeurs acheminent le paquet sur la dimension X puis sur la dimension Y, ainsi le paquet ne peut effectuer qu'un seul changement de direction. La décision du routage des paquets se fait grâce à l'entête des paquets qui contient l'adresse du cluster destinataire. Tout le paquet est retransmis de routeur à routeur jusqu'au destinataire. Cette technique de routage est simple à mettre en œuvre, permettant ainsi une intégration compacte et rapide. De plus, elle exclut le risque de blocage, car les chemins sont définis par la source.

	Couche réseau		
Réseau	*Topologie*	*Granularité de routage*	*Règle de routage*
XPIPES	Grille, torus, hypercube, Clos, ou butterfly	Paquet	Source/Routage Street-Sign/Chemin le plus court
SPIN	Arbre élargi	Paquet	Adaptatif et distribué
Proteo	Anneau variable	Paquet	Source
STNoC	Spécifique (Fig.1.3)	Paquet	non spécifié
SoCBus	Grille 2D	Circuit	Distribué
MANGO	Grille 2D	Circuit et Paquet	Source
Æthereal	Grille 2D	Paquet	Source
QNoC	Grille 2D	Paquet	Source/Routage X-Y

TABLE 1.3 – Comparaison d'une sélection de réseaux sur puce selon les paramètres de la couche réseau

1.4.3.2 Le routage adaptatif

Les algorithmes de routage adaptatifs s'adaptent facilement à toutes les topologies. Les paquets peuvent y circuler librement sans restriction. Un paquet peut contourner tous les conflits qu'il croise. En revanche, cette technique de routage est complexe à mettre en place et elle engendre des conflits. En effet, les paquets peuvent tourner dans le réseau en occupant les ressources sans trouver leur destination ou tout au moins cela entraîne une incertitude sur la durée de transmission.

1.4.4 État de l'art

La comparaison des réseaux sur puce représentée dans le tableau 1.3 montre que la plupart de ces réseaux adoptent une topologie générique, surtout la topologie grille à deux dimensions. Nous observons aussi, que la majorité de ces réseaux intègre une granularité de routage par paquet et un algorithme de routage déterministe.

1.5 La couche liaison de données

La couche liaison de données définit les protocoles d'échange d'un flux de données entre les composants du réseau, le choix des techniques concerne les points suivant :

- la granularité de transmission et de contrôle du flux de données entre les composants ;
- la politique de mémorisation du flux de données dans les composants ;
- et la technique d'arbitrage en cas de contention.

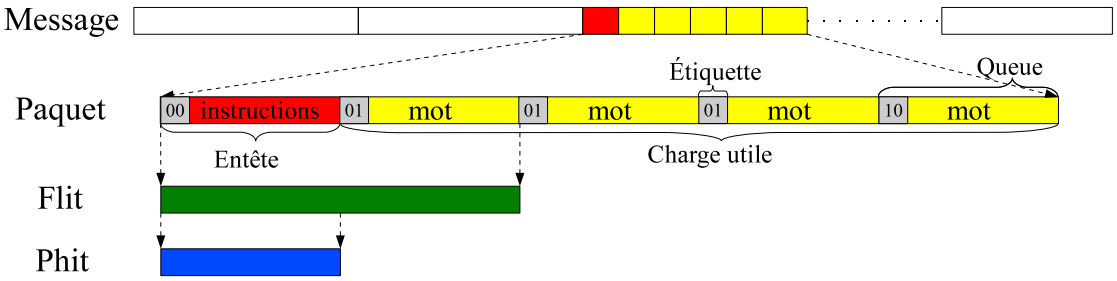


FIGURE 1.4 – Le découpage d'un message en paquets, et la composition d'un paquet

1.5.1 Granularité de transmission/contrôle du flux de données

La quantité de données transmises simultanément entre deux composants du réseaux est limitée à la capacité du lien, donc à sa largeur. Ainsi, un message doit être découpé en plusieurs morceaux quelque soit le protocole de commutation adopté dans la couche réseau. La granularité des données envoyées sur le réseau correspond à l'unité physique *phit*[15], qui détermine la quantité de bits transportables sur le lien en un cycle. En revanche, le contrôle de flux sur un lien peut être réalisé avec une granularité de un ou plusieurs mots (*phits*). Alors, nous définissons un deuxième niveau de granularité mesurable avec l'unité de contrôle de flux *flit*[16]. Le contrôle se fait donc à la fréquence des *flits*, alors que les données circulent à la fréquence des *phits*.

L'utilisation du protocole de commutation de paquets impose un découpage du message en plusieurs paquets, chaque paquet est composé de *flits* eux même formés de *phits* (Sur l'exemple de la figure Fig.1.4, un *flit* contient deux *phits*). Le paquet débute avec un entête qui contient les informations nécessaires au routage de ce paquet vers la destination. Son corps transporte la charge utile, et possède également une queue qui indique sa fin.

1.5.2 Politique de mémorisation

La politique de mémorisation est choisie selon la granularité de transmission/contrôle du flux de données (*flit*/*phits*), et la taille des tampons mémoires à l'entrée et/ou à la sortie des composants du réseau. Parmi les politiques de mémorisation, on distingue :

- *Store-and-Forward (SAF)* : chaque composant du réseau attend la réception de la totalité du paquet, le stocke entièrement avant de le transmettre au suivant ;
- *Virtual Cut Through (VCT)* : cette technique a été introduite pour réduire la latence du SAF. Un composant réseau fait suivre les *flits* d'un paquet au plus vite, mais il peut stocker tout le message s'il y a une contention ;
- *Wormwhole* : cette politique consiste à acheminer le paquet sous forme de *flits*, sans possibilité de stocker la totalité du paquet. Ainsi on réduit la taille des tampons mémoires avec le risque d'interblocage en cas de contention de l'un des *flits* [NM93] ;
- *Mad Postman* [IBJ94] : cette technique a le même procédé que la politique *wormhole*, la différence est que la granularité du contrôle du flux de données *flit* est égale à la granularité de transmission *phits*.

15. PHysical unIT
16. FLow control unIT

	Couche liaison de données	
Réseau	*Politique de mémorisation*	*Arbitrage*
XPIPES	Wormhole	priorité calculée ou tournante
SPIN	Wormhole	priorité tournante
Proteo	Wormhole	non spécifié
STNoC	Wormhole	-
SoCBus	Circuit virtuel	-
MANGO	Circuit virtuel	non spécifié
Æthereal	Wormhole (BE)/ Store-and-forward (GT)	non spécifié
QNoC	Wormhole	non spécifié

TABLE 1.4 – Comparaison d'une sélection de réseaux sur puce selon les paramètres de la couche liaison de données

1.5.3 Arbitrage

L'arbitrage résout le problème de contention, qui se présente lors d'un conflit entre différentes données, qui veulent accéder à la même ressource. Le choix de la technique d'arbitrage est influencé par la politique de mémorisation, donc par la taille des tampons mémoires des composants du réseau. En plus, la technique d'arbitrage choisie doit être équitable, en fournissant le même temps d'accès à toutes les requêtes qui ont la même priorité. Par conséquent, les conflits sont résolus en appliquant l'une des techniques suivantes :

- Sans arbitrage : cette technique est utilisée dans le cas du trafic GT uniquement, puisque les conflits sont résolus en amont par le préordonnancement des intervalles de temps de toutes les communications ;
- Avec arbitrage à priorité tournante (*round-robin*) : cette technique donne l'accès à la communication qui détient le jeton. Et à chaque cycle le jeton change de propriétaire ;
- Avec arbitrage à priorité calculée : cette technique permet d'affecter à chaque paquet un niveau de priorité en fonction de l'importance et/ou de l'urgence des données transportées. Cette priorité est stockée dans l'entête du paquet.

1.5.4 État de l'art

Le tableau 1.4 montre que les réseaux sur puce transférant les messages sous forme de paquets, adoptent une politique de mémorisation *Wormhole*.Nous observons aussi, que la politique d'arbitrage n'est pas obligatoire dans les réseau GT, car les chemins alloués aux communications sont calculés d'une manière à empêcher les conflits aux niveaux des routeurs.

1.6 La couche physique

La couche physique inclut deux paramètres :

- Le dimensionnement de la largeur du lien est une étape critique car l'augmentation de la largeur permet certainement une augmentation considérable des performances du réseau. Mais d'un autre côté, cette augmentation a un effet négatif sur la taille du réseau, car elle augmente la taille des tampons mémoires des entrées/sorties des routeurs et des interfaces réseau.

	Couche physique	
Réseau	*Type de synchronisation*	*Largeur du lien*
XPIPES	Synchrone	variable
SPIN	Synchrone	32 bits de données et 4 bis de contrôle
Proteo	Synchrone	32 bits
STNoC	Synchrone	Non spécifiée
SoCBus	Synchrone	Non spécifiée
MANGO	Asynchrone	33 bits
Æthereal	Synchrone	32 bits
QNoC	Synchrone	16 bits de données et 10 bits de contrôle

TABLE 1.5 – Comparaison d'une sélection de réseaux sur puce selon les paramètres de la couche physique

- La synchronisation dans les réseaux sur puce est introduite avec différents types d'approches, il existe :
  - des réseaux entièrement synchrones avec une horloge commune utilisée par tous les composants du réseau. L'approche synchrone est la méthode la plus utilisée par les NoCs, comme le montre le tableau 1.5 ;
  - des réseaux entièrement asynchrones, comme le réseau MANGO[BS04, BS05] ;
  - et des réseaux utilisant l'approche GALS [9] [EDH06], avec des parties du réseau synchrone qui échangent des données d'une manière asynchrone.

1.7 Conclusion

Ce chapitre a permis d'étudier les réseaux sur puce et de définir un espace de conception, qui se structure autour de deux axes fondamentaux. Le premier vise à définir une architecture, en se basant sur les contraintes des communications de l'application. La conception de cette architecture peut être divisée en trois problèmes, nous distinguons :

- Le choix de la topologie est très important car la capacité du réseau à diffuser efficacement l'information dépend essentiellement de la géométrie de sa structure. Outre un impact direct sur la latence, le débit, la surface, et la consommation d'énergie du réseau. la topologie joue un rôle déterminant dans le choix des politiques de routage et le placement des blocs IP. Jusqu'à présent, aucune solution théorique n'a proposé une méthode pour déterminer la topologie optimale à mettre en œuvre pour une application donnée. Pourtant, la personnalisation de l'architecture est cruciale pour améliorer les performances, et réduire la consommation d'énergie et la surface du réseau ;
- Le dimensionnement de la largeur des canaux du réseau n'a pas été adressé à ce jour, tandis que la détermination de la meilleure largeur des canaux pour une application donnée, est nécessaire pour optimiser la surface, les flux au sein du réseau, ainsi que le choix du calibrage et l'espacement des fils qui déterminent la fréquence de fonctionnement du canal ;
- La taille de la mémoire tampon de chaque entrée d'un routeur ou d'une interface réseau, a des répercussions lourdes sur la surface et la consommation d'énergie. Ainsi, l'utilisation globale des ressources en mémoire tampon doit être réduite au maximum.

9. Globally Asynchronous Locally Synchronous

La définition de l'architecture ne permet pas à elle seule, de décrire le comportement du réseau sur puce. Ceci est déterminé dans le deuxième axe par le choix de la granularité de transmission, l'algorithme de routage, la politique de mémorisation, et l'arbitrage. Le choix de ces protocoles est très important car, nous avons observé à l'aide de l'état de l'art qu'ils déterminent la qualité de service du réseau.

Dans le chapitre suivant, nous définissons l'architecture matérielle des différents composants de notre réseau sur puce μSpider II. Nous abordons également le choix des protocoles que nous avons implantés, pour concevoir un réseau avec une qualité de service qui garantit le trafic.

2

Paramétrage de l'architecture et des protocoles du NoC μSpider II

Dans le chapitre 1, nous avons organisé l'espace de conception des réseaux sur puce selon une réduction du modèle OSI, qui arrange les paramètres des NoCs [17] en 5 couches. À partir de cette classification, nous avons identifié trois types de NoCs selon leur qualité de service. Les réseaux de type GT, les réseaux de type BE, et les réseaux mélangeant les deux types GT et BE.

Notre réseau μSpider II gère principalement des trafics de type GT. Nous avons choisi ce type de trafic, car il permet de satisfaire les contraintes du concepteur, en garantissant l'intégrité et le débit des communications. Mais en réalité ce choix vise plutôt l'étude approfondie des réseaux de type GT, ce qui nous permettra d'en montrer les avantages mais aussi les limites en termes de flexibilité dans un contexte dynamique. C'est pour cette raison que notre étude est divisée en deux parties, la première partie étudie et conçoit un réseau de type GT performant et la deuxième partie introduit des mécanismes de reconfigurabilité sur ce type de réseau pour le rendre adaptatif dynamiquement.

Ce chapitre identifie dans la section 2.1 les paramétrages que nous avons réalisé dans les différentes couches du modèle OSI pour concevoir un réseau sur puce gérant des communications de type GT. Il présente aussi l'architecture des différents composants du réseau μSpider II. Ainsi, la section 2.2 présente l'architecture du routeur et les mécanismes de routage et d'arbitrage adoptés. La section 2.3 développe l'architecture de l'interface réseau, l'ordonnancement des envois et le protocole de mise en paquets. Et la section 2.4 introduit l'architecture de l'adaptateur de protocole et ses modes de fonctionnement.

2.1 Paramétrage des couches OSI du NoC μSpider II

Les couches OSI du réseau μSpider II sont paramétrées de manière à satisfaire des communications d'une qualité de service de type GT, ainsi :

– La couche application gère des trafics garantis, et elle utilise le protocole du bus PLB [18] de *Xilinx* pour l'adaptateur réseau. Nous avons choisi ce protocole car notre cible d'implantation est un FPGA [19] de *Xilinx*. Ce choix spécifique n'affecte pas la généralité de l'approche et l'architecture de l'adaptateur réseau qui peut s'interfacer avec des composants utilisant d'autres standards ;

17. Network on Chip
18. Processor Local Bus
19. Field Programmable Gate Array

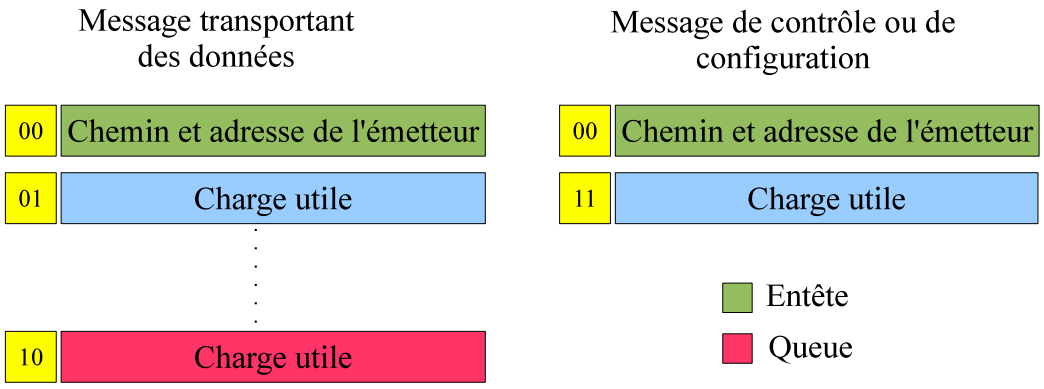


FIGURE 2.1 – Le découpage des messages en paquets, et l'identification de la nature du paquet par une étiquette

- La couche transport contrôle le flux dans notre réseau, en utilisant une méthode nouvelle basée sur l'effet domino, cette méthode est expliquée en détails dans la section 2.1.2 de ce chapitre ;
- La couche réseau adopte une topologie irrégulière construite suivant le type de l'application, l'adressage mémoire, et les protocoles d'échanges appliqués. Elle utilise une commutation par paquets et un routage déterministe. Les chemins sont calculés en amont puis stockés à la source (interfaces réseau). La mise en paquets est détaillée dans la section 2.1.1 et l'algorithme de calcul des chemins est présenté dans la section 3.6 du chapitre 3 ;
- La couche liaison de données adopte une granularité de transmission du flux de données (*Phit*) égale à la granularité de contrôle du flux de données (*Flit*). Cette égalité implique l'utilisation d'une politique de mémorisation *wormhole*. L'application de l'arbitrage n'est pas obligatoire dans cette couche car, la qualité de service du réseau µSpider II est de type GT. Néanmoins, nous adoptons un arbitrage avec priorité calculée en cas de besoin pour permettre l'implantation de trafics de type BE ;
- La couche physique utilise une largeur de lien paramétrable, le calcul de la largeur dépend principalement de la fréquence du système et de la bande passante demandée. Elle applique une horloge commune à tous les composants du réseau pour le synchroniser.

2.1.1 Mise en paquets d'un message

Un message est un ensemble de paquets, le premier paquet est l'entête et les suivants contiennent la charge utile. L'entête renferme les informations nécessaires pour le routage de l'ensemble des paquets qui constituent le message. Les deux premiers bits de chaque paquet permettent l'identification de sa nature, nous distinguons :

- 00 pour l'entête ;
- 01 pour la charge utile ;
- 10 pour la queue du message ;
- 11 représente les paquets de contrôle et de configuration.

Le réseau µSpider II transporte deux types de messages, le premier type contient des données échangées entre les communicants connectés au réseau. Le deuxième type est un message de contrôle ou de configuration transmis par et vers un composant du réseau.

Comme le montre la figure 2.1, le découpage des deux types de messages en paquets est similaire, à la différence que la fin du message contenant des données est représentée par un paquet de type queue, alors que le message de contrôle contient un seul paquet qui représente le début et la fin du message.

L'entête est constitué du chemin que le message emprunte pour arriver à sa destination et l'adresse de l'émetteur. Une partie de l'entête est consommée au niveau de chaque routeur pour trouver l'adresse du port de sortie que le message doit emprunter, le reste identifie l'émetteur auprès de l'interface réseau de réception.

2.1.2 Contrôle de flux par effet domino

Les composants du réseau μSpider II s'échangent des données sur un lien de largeur paramétrable, chaque lien d'échange est associé à un lien de contrôle de flux. Ainsi, un composant qu'il soit un routeur ou une interface réseau, peut envoyer des données à un autre composant si et seulement si, le récepteur autorise la transmission en mettant le lien de contrôle de flux à 0. Comme le montre la figure 2.2, le routeur R0 ne peut transmettre des données vers le routeur R1, que si le port d'entrée IP0 du routeur R1 met à 0 le lien de contrôle de flux relié au port de sortie OP0 du routeur R0. Cette technique permet de contrôler le flux aux extrémités des composants du réseau.

Le contrôle de flux de bout en bout dans le réseau μSpider II permet de contrôler la transmission d'un flux de données entre les interfaces réseaux. Ainsi, l'interface réseau émettrice n'envoie des données que si l'interface réseau réceptrice peut les consommer ou les stocker. Notre réseau n'utilise pas de crédits d'émission pour le contrôle de flux de bout en bout, en revanche il se base sur le contrôle de flux aux extrémités des composants et l'effet domino. Ce choix est essentiellement dû à l'utilisation importante des ressources matérielles par la technique de contrôle de flux par crédits d'émission, car pour contrôler le flux d'une seule communication il faut intégrer un compteur/décompteur sur l'interface réseau émettrice. De plus, le renvoi des crédits d'émission par l'interface réseau réceptrice augmente considérablement la charge du réseau.

Pour expliquer cette technique, nous allons nous référer à l'exemple de la figure 2.2. Sur cette illustration, nous établissons une liaison de l'interface réseau NI0 vers l'interface réseau NI1 en passant par les routeurs R0 et R1. Dans le cas où l'interface réseau NI1 ne peut plus consommer ni stocker les paquets, elle interdit la réception de paquets supplémentaires en mettant à 1 le lien de contrôle de flux relié au routeur R1. Ensuite, le routeur R1 met à 1 le lien de contrôle de flux relié au routeur R0. Puis, le routeur R0 met à 1 le lien de contrôle de flux relié à l'interface réseau NI0, ce qui implique l'arrêt de la transmission des données vers l'interface réseau NI1. Cet enchaînement de blocages de transmission de la destination à la source permet de contrôler le flux de bout en bout dans le réseau μSpider II, et c'est pour cette raison que nous le nommons contrôle de flux par effet domino.

Cette technique permet de contrôler le flux de bout en bout en introduisant le minimum de complexité car, nous avons besoin d'un seul lien de contrôle de flux de 1 bit par lien de transmission. Ainsi, en remplaçant la méthode de contrôle de flux par crédits d'émission par la nôtre, nous nous débarrassons des compteurs/décompteurs de crédits d'émission dans les interfaces réseaux, et nous diminuons le trafic dans le réseau car il ne transmet plus aucun crédit d'émission.

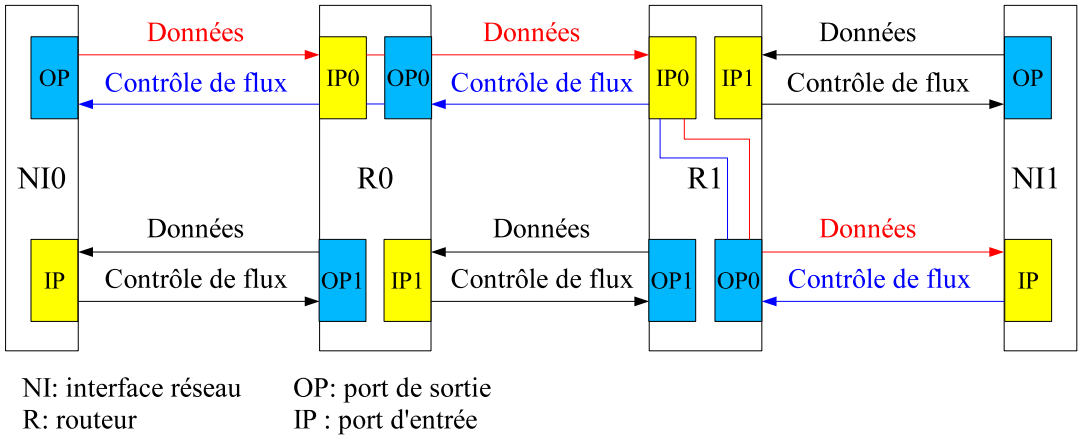


FIGURE 2.2 – Le mécanisme de contrôle de flux aux extrémités des composants du réseau μSpider II et le contrôle de flux de bout en bout

2.2 Routeur μSpider II

Un routeur est un composant intermédiaire dans le NoC, assurant le routage des paquets. Son rôle est de faire transiter des paquets d'une interface réseau ou d'un routeur vers une autre interface réseau ou un autre routeur, selon les règles de l'algorithme de routage adopté par le réseau.

Un routeur assure la liaison entre les ports de sortie et les ports d'entrée. Nous établissons une liaison entre un port de sortie m et un port d'entrée n selon l'adresse fournie par l'entête du message. Cette adresse est décodée puis transmise à l'arbitre qui prend la décision d'établir ou pas la liaison. Ainsi, l'architecture du routeur μSpider II représentée sur la figure 2.3 est composée de plusieurs entités, nous distinguons :

- **Des ports d'entrée** qui lient différents composants du réseau au routeur pour recevoir des données. Chaque composant est connecté à un seul port d'entrée, ce port est constitué de trois signaux. Le premier signal transporte les données (paquets). Le deuxième est un signal de validation, il passe à 1 pendant un cycle d'horloge, ceci indique au décodeur la validité des données transportées par le premier signal. Le troisième signal du port d'entrée contrôle le flux, ainsi quand ce signal est à 0, il autorise la transmission des données par l'entité précédente connectée au routeur sur ce port. Un routeur doit avoir au moins deux ports d'entrée, sinon il est inutile ;
- **Des ports de sortie** qui permettent au routeur de transmettre les paquets vers d'autres composants du réseau. Chaque port de sortie est composé aussi de trois signaux. Le premier transmet les données vers l'entité suivante (interface réseau ou routeur). Le deuxième signal valide les données transmises sur le premier signal. Et le troisième contrôle le flux du port de sortie, ainsi il n'autorise au routeur d'envoyer des données à l'entité suivante, seulement si ce signal est à 0. Un routeur doit inclure au minimum un port de sortie ;
- **Des décodeurs** connectés aux ports d'entrée, leurs fonction est de déchiffrer la nature du paquet (entête, charge utile, ou queue) selon son étiquette. Lors du décodage, les paquets de type entête sont traités pour trouver l'étiquette du port de sortie, puis cette étiquette est transmise à l'entité nommée table de routage. En revanche les autres paquets sont transmis directement à l'arbitre sans traitement ;

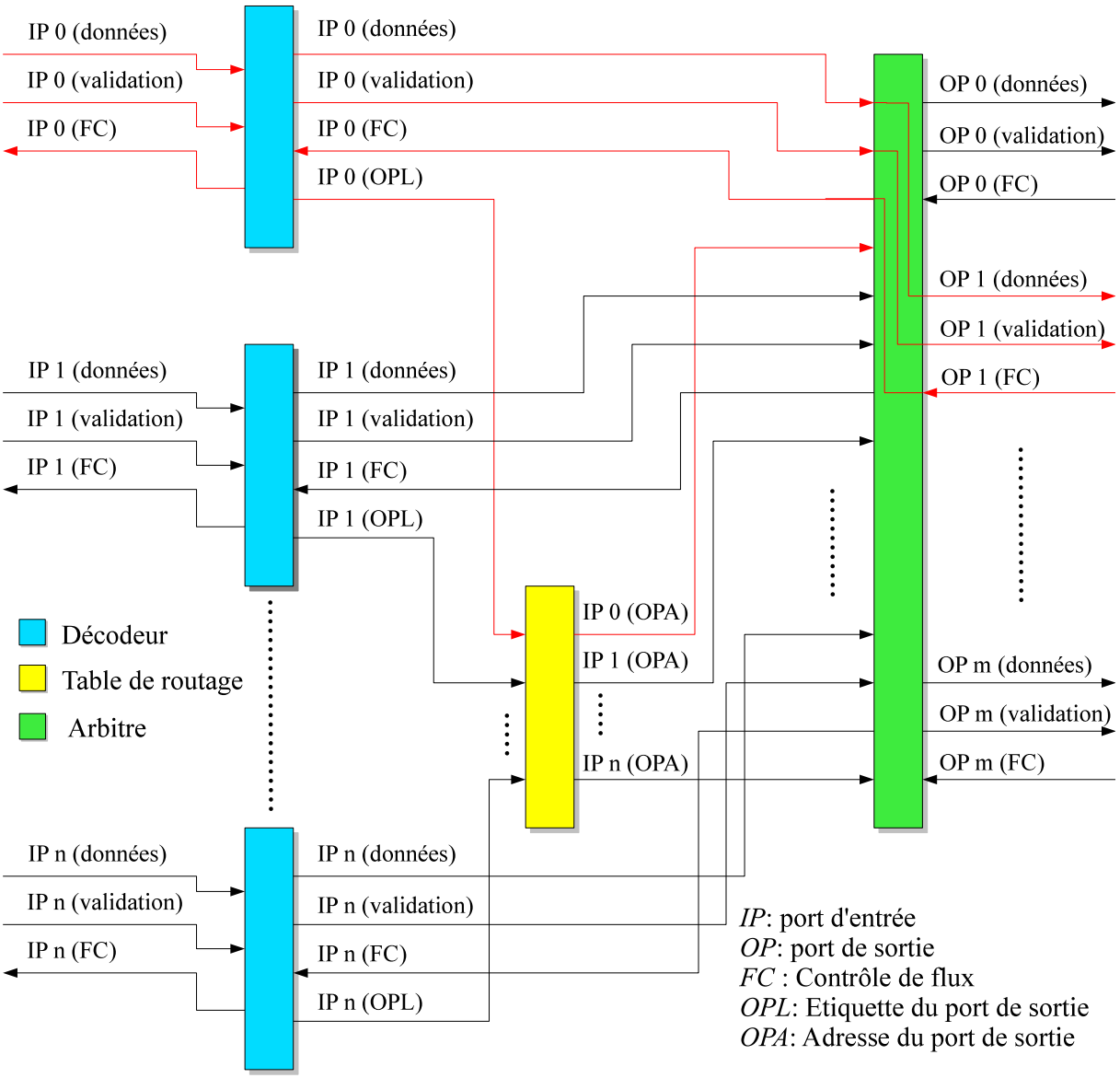


FIGURE 2.3 – Architecture du routeur µSpider II est composée d'un nombre de décodeurs égal au nombre de ports d'entrée du routeur, d'une table de routage et d'un arbitre

- **D'une table de routage** qui donne l'adresse du port de sortie à l'arbitre, selon la valeur de l'étiquette du port de sortie fournie par le décodeur ;
- **D'un arbitre** qui établit la liaison entre le port d'entrée et le port de sortie selon l'adresse calculée par la table de routage. En cas de conflit, où deux ports d'entrée demandent d'établir une liaison avec le même port de sortie, l'arbitre choisit l'un des deux ports d'entrée selon sa priorité.
  Notre NoC garantit le trafic, ainsi les chemins calculés en amont assurent un routage sans conflits. Par conséquent, il est inutile d'intégrer un arbitrage dans les routeurs. Néanmoins, le routeur μSpider II ajoute un arbitrage basé sur la notion de priorité calculée en cas de besoin dans les stratégies de conception futures.

2.2.1 Transfert des paquets dans le routeur

La fonction du routeur μSpider II est de recevoir des paquets sur ses ports d'entrée puis de les aiguiller vers ses ports de sortie. Ce transfert de paquets dans le routeur demande l'établissement d'une liaison entre un port de sortie et un port d'entrée, ainsi nous distinguons trois phases pour transférer un message dans le routeur :

1. **L'initialisation** de la liaison : à l'identification d'un paquet de type entête par le décodeur, ce dernier consomme une partie du poids fort de l'entête codé en binaire puis décale le reste du chemin à gauche. Le nombre de bits consommé dépend du nombre de ports de sortie du routeur. Par exemple, le décodeur consomme 3 bits de l'entête si le nombre de ports de sortie du routeur est comprise entre 5 et 8.
   Ensuite, le décodeur transmet l'étiquette du port de sortie à la table de routage qui identifie le numéro du port de sortie par rapport à la valeur de l'étiquette, ce numéro est transmit par la suite à l'arbitre.
   À la réception de ce numéro, l'arbitre prend la requête en charge, et établie une liaison entre le port de sortie identifié par le décodeur et le port d'entrée. Cependant, la liaison n'est établie que si le port de sortie n'est pas déjà connecté à un autre port d'entrée.
   Sur la figure 2.3, la liaison entre le port de sortie numéro 1 et le port d'entrée numéro 0 est illustré par des liens de couleur rouge ;
2. **Le transfert** des paquets avec une charge utile : tant que le décodeur identifie des paquets de type charge utile, la liaison est maintenue ;
3. **La déconnexion** de la liaison : à la réception du paquet qui représente la queue du message, le décodeur coupe la liaison entre le port d'entrée et le port de sortie.

Le routeur μSpider peut établir au même instant autant de liaisons que de ports de sortie inutilisables. L'établissement de ces liaisons est asynchrone, mais le transfert d'un paquet est synchrone et il dure 1 cycle d'horloge. Par conséquent, la latence de la traversée d'un routeur par un paquet est d'un cycle d'horloge car, sur les trois éléments qui composent le routeur, il n'y a que le décodeur qui traite le paquet sur le front montant de l'horloge pour extraire son type.

2.2.2 Contrôle de flux aux extrémités du routeur

Le rôle d'un port de sortie d'un routeur est de le connecter à un autre composant (routeur ou interface réseau), si ce composant ne peut pas recevoir de paquets, il met le signal FC du port de sortie à 1.

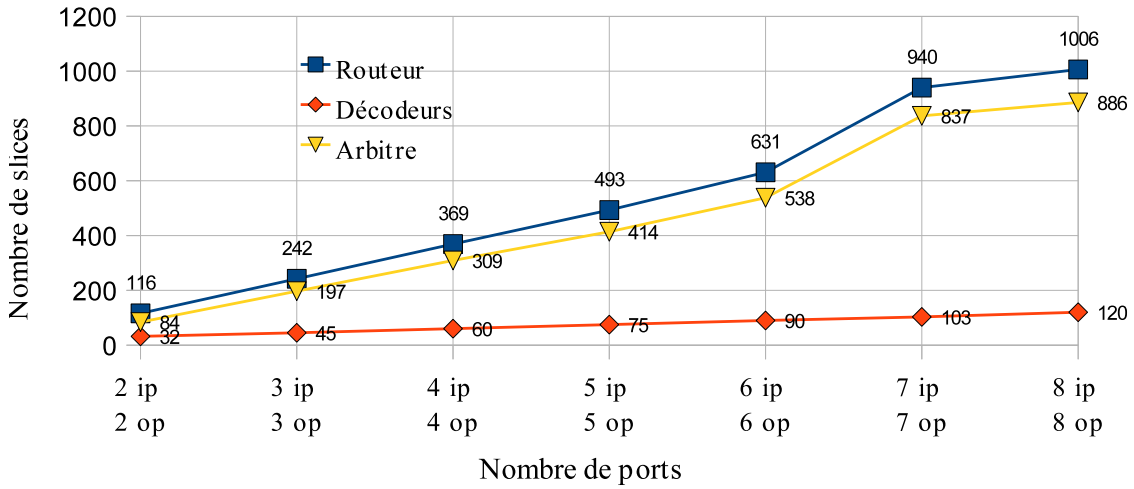


FIGURE 2.4 – Évolution de la taille du routeur, de l'arbitre et des décodeurs, par rapport au nombre de ports (critère 1 : le nombre de ports d'entrée est égale au nombre de ports de sortie

Quand le routeur établit une liaison entre un port d'entrée et un port de sortie, il connecte implicitement leurs signaux de contrôle de flux. Alors si le signal FC est à 1 sur le port de sortie, le signal FC du port d'entrée a lui aussi la valeur 1. Ce transfert de la valeur du signal FC entraîne un blocage de transmission dans le routeur, et grâce à l'effet domino, ceci entraîne un blocage de transfert depuis la source du message jusqu'au port de sortie concerné.

Le système de contrôle de flux par effet domino permet d'économiser les mémoires tampons sur les ports d'entrée et de sorties du routeur. Ainsi, le routeur µSpider II n'utilise qu'un seul registre par port d'entrée, pour stocker un paquet en cas de conflit ou de blocage.

2.2.3 Résultats d'implantation du routeur

Cette partie présente les résultats que nous avons obtenus lors de l'implantation de différentes configurations du routeur µSpider II sur une cible FPGA. Pour nos expériences, nous avons choisi un FPGA Xilinx de la famille Virtex-5 LXT et nous avons utilisé l'environnement ISE version 11.5 pour la synthèse. L'unité que nous avons utilisé pour quantifier la surface des différentes architectures est l'unité "Slice". Cette appellation désigne le bloc logique, élémentaire et programmable qui constitue les FPGAs de Xilinx. L'architecture logique d'un "Slice" de la famille Virtex-5 est donnée dans l'annexe A.

Nos expériences consistent donc à étudier l'évolution de la surface du routeur par rapport au nombre de ses ports d'entrée/sortie. Pour cela, nous avons fait varier le nombre de ports d'entrée/sortie selon trois critères :

1. Le premier critère change le nombre de ports d'entrée (IP [20]) et de sortie (OP [21]) d'une façon uniforme, en faisant varier le nombre de ports de 2 à 8, tout en respectant l'égalité entre le nombre de ports d'entrée et le nombre de ports de sortie ;

20. Input Port
21. Output Port

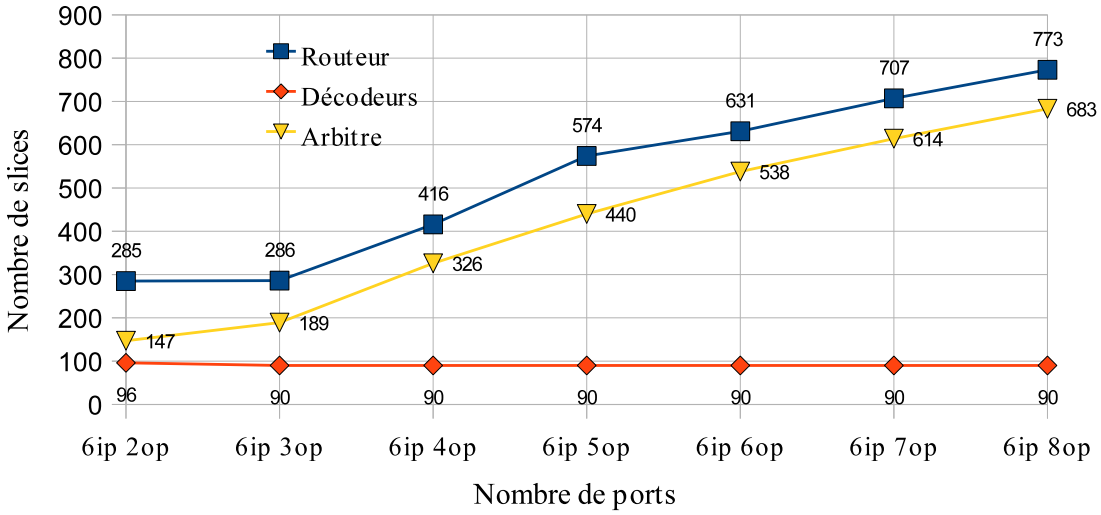


FIGURE 2.5 – Évolution de la taille du routeur, de l'arbitre et des décodeurs, par rapport au nombre de ports (critère 2 : le nombre de ports d'entrée constant)

2. Le deuxième critère fixe le nombre de ports d'entrée à 6 et fait varier le nombre de ports de sortie de 2 à 8 ;
3. Le troisième critère fixe le nombre de ports de sortie à 6 et fait varie le nombre de ports d'entrée de 2 à 8.

Les résultats de ces expériences sont présentés sur trois graphiques, chaque graphique illustre l'évolution de la taille du routeur (courbe bleu), de la taille de l'arbitre (courbe jaune) et de la taille des décodeurs (courbe rouge). La figure 2.4 décrit cette évolution selon le premier critère, alors que les figures 2.5 et 2.6 utilisent respectivement le deuxième et le troisième critère.

Nous observons sur les trois graphiques que la courbe de la taille du routeur et la courbe de la taille de l'arbitre ont des évolutions similaires, alors que la courbe de la taille des décodeurs évolue peu. Ainsi, nous concluons que le nombre de décodeurs dans un routeur n'a pas un grand impact sur la taille du routeur. En revanche, plus l'arbitre est complexe plus la taille du routeur augmente.

Nous observons aussi que la complexité de l'arbitre est proportionnelle au nombre de ports du routeur. Les graphiques représentés sur les figures 2.5 et 2.6 démontrent que la nature du port qu'il soit en entrée ou en sortie n'a pas d'impact sur la complexité de l'arbitre donc du routeur. Par contre, plus le nombre de ports d'entrée augmente plus la taille des décodeurs augmente d'une manière proportionnelle. Car chaque port d'entrée est relié à un décodeur qui a une complexité de l'ordre de $7,5$ *Slices*.

Pour conclure, la taille du routeur dépend de la taille de l'arbitre, et cette dernière dépend du nombre de ports d'entrée/sortie du routeur. Pour construire un réseau performant et optimisé en terme de surface, le concepteur doit prendre en compte lors du choix des paramètres et de la conception de la topologie, le nombre de ports d'entrée/sortie qu'il met par routeur et le nombre de routeurs qui constituent la topologie. Pour confirmer ce résultat, nous avons comparé la surface de deux topologies illustrées sur la figure 2.7, la première est constituée d'un seul routeur avec 8 ports d'entrée et 8 ports de sortie. Et la deuxième offre le même nombre de ports d'entrée/sortie, en revanche elle est constituée de 3 routeurs.

Nous observons que le remplacement du routeur de l'architecture 1 par 3 petits routeurs dans

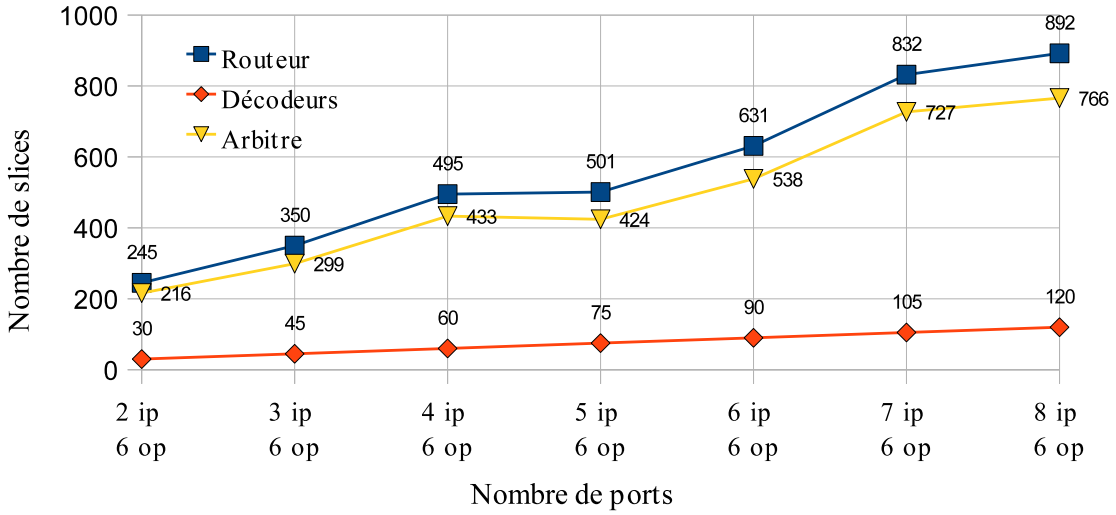


FIGURE 2.6 – Évolution de la taille du routeur, de l'arbitre et des décodeurs, par rapport au nombre de ports (critère 3 : le nombre de ports de sortie constant)

l'architecture 2, nous permet de réduire la surface de 15, 85%. Par contre, nous augmentons la latence des messages qui empruntent les chemins utilisant les ports de sorties 3 et 4, en passant d'un cycle d'horloge pour la première topologie à deux cycles d'horloge pour la deuxième.

Cette comparaison confirme que ce n'est pas le nombre de routeurs qui influe sur la taille du réseau mais le nombre de ports d'entrée/sortie par routeur. Ainsi, une méthodologie reposant sur une topologie générique qui ne prend pas en compte ce paramètre peut conduire à un surcoût très important. En revanche, la conception d'une topologie spécifique qui considère le nombre de routeurs et le nombre de ports dans les routeurs devient incontournable dans le flot de conception d'un réseau sur puce.

2.3 Interface réseau µSpider II

L'interface réseau est l'élément organisateur de la distribution des données sur le réseau. Nous attribuons une interface réseau à chaque communicant connecté au réseau. Cette interface remplit deux rôles simultanément. le premier rôle consiste à recevoir les paquets du réseau, les trier par correspondant puis les transmettre à l'adaptateur réseau. Son deuxième rôle est de mettre en paquets les données à transmettre sur le réseau en ajoutant les chemins pour arriver aux bonnes destinations. Ainsi nous pouvons considérer l'interface réseau comme un bureau de poste, qui récolte et distribue les messages vers les autres bureaux de poste connectés à son réseau de distribution.

L'interface réseau µSpider II est constituée principalement de mémoires tampons de type FIFO [22]. Ces mémoires tampons servent à stocker les données des communications établies par l'interface réseau en émission ou en réception. Ainsi, chaque mémoire tampon est réservée à une communication unique, par exemple, si une interface échange des données en émission et en réception avec 5 autres interfaces réseaux, elle doit intégrer 5 mémoires tampons en réception et 5 autres mémoires tampons en émission.

22. First In First Out

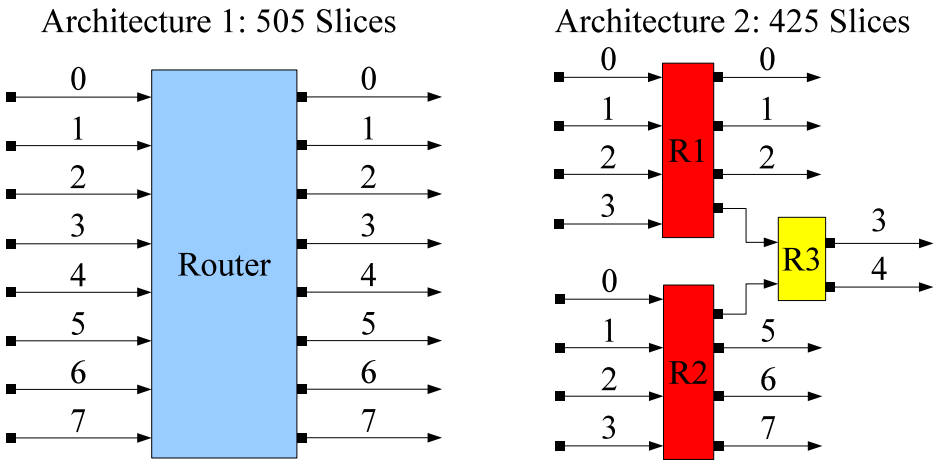


FIGURE 2.7 – Compromis entre le nombre de routeurs d'une topologie et le nombre de ports dans chaque routeur

Les mémoires tampons sont gérées par un contrôleur qui fournit l'accès en écriture et en lecture à l'une des mémoires tampons et il permet également de connaître leurs statuts (pleines ou vides).

La taille d'une mémoire tampon dépend de la bande passante de la communication, de la fréquence du réseau et de la largeur des liens. La section 3.5 du chapitre 3 détaille la méthode de calcul de la taille des mémoires tampons en réception et en émission.

L'architecture de l'interface réseau μSpider II représentée sur la figure 2.8 est composée de deux parties. La première partie contient un port d'entrée connecté à l'adaptateur réseau, un contrôleur de mémoires tampons et un port de sortie réseau. Cette partie gère les paquets envoyés sur le réseau. En revanche, la deuxième partie gère les paquets reçus du réseau, son architecture est constituée d'un port d'entrée réseau, d'un contrôleur de mémoires de tampons, et d'un port de sortie connecté à l'adaptateur réseau.

2.3.1 Méthode de traitement des paquets émis

L'envoi des données sur le réseau à travers l'interface réseau passe par deux procédés. Le premier est présenté dans la section 2.3.1.1, ce procédé stocke et trie les données transmises par l'adaptateur de protocole dans des mémoires tampons, le tri est réalisé en fonction du destinataire. Le deuxième procédé est détaillé dans la section 2.3.1.1. Il récupère les données à tour de rôle dans ses mémoires tampons et à des instants précis, puis il les empaquette avec les informations nécessaires pour qu'elles puissent être acheminées à la bonne destination.

2.3.1.1 Procédure de stockage

Pour envoyer des données à l'interface réseau, l'adaptateur de protocole vérifie que la mémoire tampon réservée au destinataire dans l'interface réseau n'est pas pleine. Cette vérification est réalisée grâce au signal "FIFOs pleines" comme le montre la figure 2.8. Ce signal est composé d'autant de bits que de mémoires tampons en émission. Par conséquent, quand l'adaptateur réseau veut envoyer des données à un destinataire, il vérifie que le bit du signal "FIFOs pleines" correspondant à la FIFO réservée à ce destinataire est à 0.

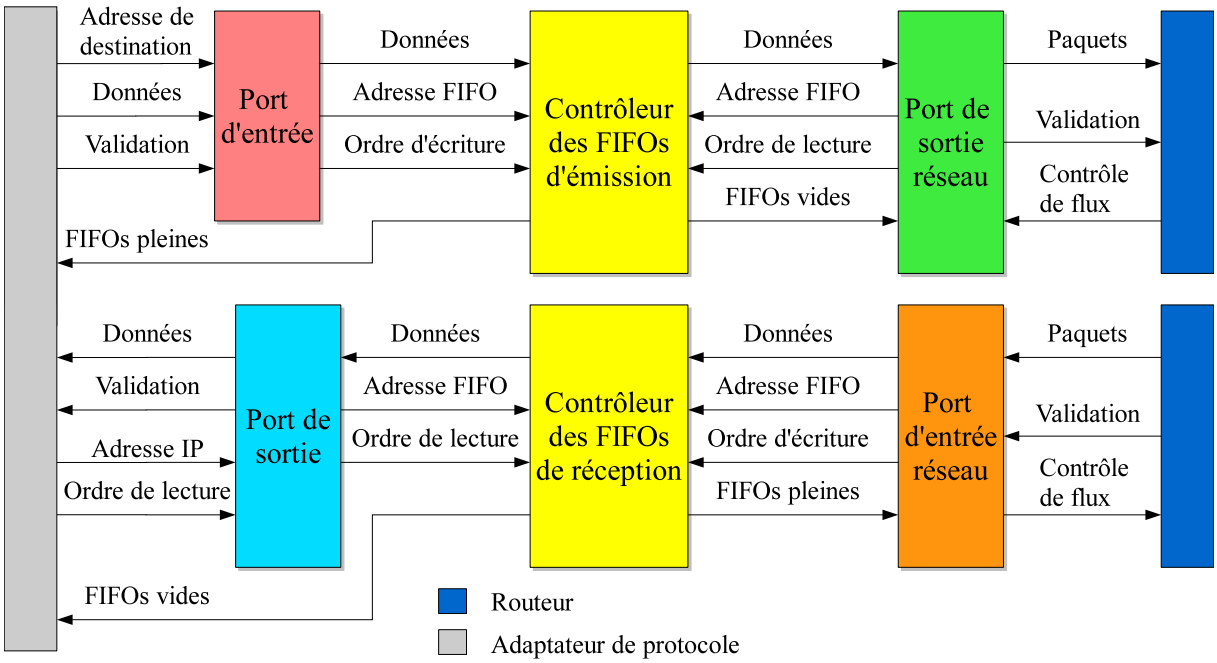


FIGURE 2.8 – Architecture de l'interface réseau μSpider II

Après cette vérification, l'adaptateur de protocole transmet la donnée au port d'entrée en spécifiant l'adresse du destinataire et en mettant à 1 et pendant un cycle le signal de validation. L'adresse du destinataire est un chiffre compris entre 0 et $n-1$, avec n le nombre de composants connectés au réseau.

À la réception de la donnée de l'adaptateur de protocole, le port d'entrée la transmet au contrôleur des FIFOs d'émission, en calculant l'adresse de la FIFO à partir de l'adresse du destinataire et en mettant à 1 et pendant un cycle le signal "ordre d'écriture". Par conséquent, le rôle du port d'entrée se résume à convertir l'adresse de destination à une adresse de mémoire tampon.

2.3.1.2 Procédure de mise en paquets

La mise en paquets des données dans l'interface réseau est réalisée par le port de sortie réseau. La figure 2.9 illustre l'architecture de ce port qui se constitue d'un élément d'empaquetage, d'un ordonnanceur, d'un étiqueteur et d'une partie qui fournit les entêtes.

Le réseau μSpider II gère des communications de type GT. Ainsi, pour garantir les contraintes de ces communications, l'interface réseau doit envoyer les paquets à des instants précis. Cette synchronisation d'expédition est gérée par l'ordonnanceur. L'ordonnanceur est une table TDMA [23] composée de plusieurs intervalles de temps, chaque intervalle dure un cycle d'horloge et indique la communication (le numéro de la FIFO) qui a le droit d'émettre pendant ce cycle. Les intervalles de temps d'une même communication sont successives, et leur nombre est calculé selon la bande passante de la communication (voir la section 3.4).

L'étiqueteur fournit à l'entité d'empaquetage plusieurs étiquettes pour identifier les différents

23. Time division multiple access

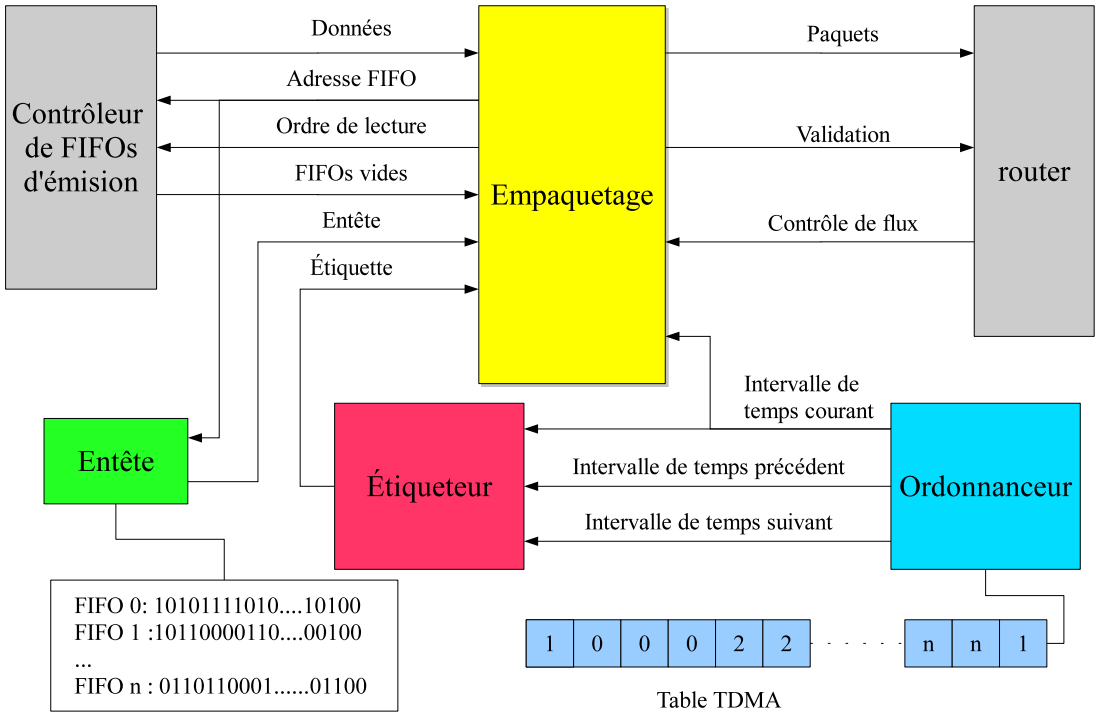


FIGURE 2.9 – Architecture du port de sortie réseau de l'interface réseau μSpider II

paquets. La valeur de l'étiquette est calculée selon les valeurs de trois intervalles de temps, le précédent, le suivant et le courant. Ainsi l'étiqueteur identifie :

- l'entête, si la valeur de l'intervalle de temps précédent est différente de la valeur de l'intervalle courant ;
- la charge utile, si la valeur de l'intervalle de temps courant est égale à la valeur de l'intervalle de temps précédent et à la valeur de l'intervalle de temps suivant ;
- et la queue, si la valeur de l'intervalle de temps courant est différente de la valeur de l'intervalle de temps suivant.

L'entité d'empaquetage reçoit à chaque cycle une valeur de la table TDMA identifiant la FIFO qui doit émettre, et une étiquette qui détermine la nature du paquet à envoyer. Elle envoie des paquets sur le réseau seulement si le signal de contrôle de flux est à 0, et si le signal "FIFOs vides" indique que le bit de la FIFO qui a la permission d'envoyer est à 0 également. L'entité de mise en paquets applique des procédés d'expédition différents, le procédé exécuté dépend de la nature du paquet.

Par conséquent, si le paquet est un entête, elle récupère le chemin de la communication grâce à l'entité "entête", en lui fournissant le numéro de la FIFO, ce qui correspond à l'adresse du destinataire. À la récupération du chemin, elle rajoute au chemin l'étiquette d'entête puis l'envoie au composant suivant (routeur) sur le signal "paquets" tout en mettant le signal de validation à 1 pendant un cycle d'horloge.

Sinon, si le paquet est une charge utile ou une queue, elle récupère une donnée de la FIFO, en fournissant l'adresse de celle-ci au contrôleur des FIFOs en émission, et en mettant à 1 le signal 'ordre de lecture' pendant une cycle d'horloge. Quand elle récupère la donnée du contrôleur, elle l'empaquette avec l'étiquette puis l'envoie au composant suivant de la même manière que l'entête.

2.3.2 Protocole de réception des paquets

La réception des paquets par l'interface réseau est contrôlée par le port d'entrée réseau comme le montre la figure 2.8. Le rôle de ce port est de récupérer les paquets puis les stocker dans les mémoires tampons de réception en les triant selon leurs expéditeurs. Ainsi, quand le port d'entrée réseau reçoit un paquet de type entête, il récupère l'adresse de l'expéditeur et il vérifie l'état de la FIFO qui stockera les données qui suivent l'entête. Si le bit du signal "FIFOs pleines" qui correspond à la FIFO de réception réservée pour cet expéditeur est à 1, le port d'entrée réseau arrête la réception des paquets en mettant le signal de contrôle de flux à 1. Sinon, si la FIFO de réception n'est pas pleine, le port d'entrée réseau récupère les paquets du réseau puis les transmet au contrôleur de FIFOs de réception jusqu'à ce qu'il identifie la queue du message.

Après le tri et le stockage des données dans les différentes mémoires tampons de réception, l'adaptateur de protocole récupère ces données selon la disponibilité du communicant connecté à ce dernier. Cette récupération des données stockées dans les différentes FIFOs de réception est réalisée d'une manière équitable et itérative. Ainsi l'adaptateur de protocole récupère la même quantité de données dans les FIFOs en passant d'une FIFO à une autre ce qui empêche la monopolisation de la récupération des données sur une seule FIFO.

Quand une FIFO est pleine, son bit correspondant dans le signal "FIFOs vides" est mis à 1. Ainsi, à travers ce signal l'adaptateur de protocole reconnaît les FIFOs pleines, et transmet l'ordre de lecture d'une donnée au port de sortie en spécifiant l'adresse de l'IP. Quand le port de sortie reçoit l'ordre, il calcule l'adresse de la FIFO par rapport à l'adresse de l'IP et il récupère la donnée du contrôleur de FIFOs de réception pour la retransmettre à l'adaptateur de protocole.

2.4 Adaptateur de protocole μSpider II

Le réseau μSpider II connecte plusieurs communicants. Ces communicants peuvent être des blocs fonctionnels complexes (IP), des processeurs avec ou sans mémoire locale, ou des architectures complexes composées d'un processeur, de mémoires, d'accélérateurs matériels et d'un bus. La diversité de la nature et des protocoles d'échanges des communicants, nous oblige à concevoir un élément qui permet de les connecter aux interfaces réseau quelque soit leur nature et leur protocole de communication. Cet élément a pour mission d'adapter les échanges entre le communicant et son interface réseau, d'où le nom d'adaptateur de protocole. La conception d'un adaptateur de protocole avec une architecture qui facilite la transmission des données au communicant est essentielle, car elle contribue à la réduction des temps d'accès aux données, et par conséquent à l'augmentation des performances du réseau.

La plupart des réseaux sur puce se contentent de mentionner le protocole accepté par leurs réseaux sans donner les détails de l'architecture et le fonctionnement de l'adaptateur de protocole. Aussi, l'optimisation de l'architecture de l'adaptateur de protocole est souvent ignorée par les concepteurs des NoCs, alors qu'elle est essentielle pour la conception d'un réseau performant. Le but de cette section est de présenter les modes de fonctionnement de l'adaptateur de protocole μSpider II, en utilisant une architecture indépendante du protocole. Ainsi, notre adaptateur peut intégrer n'importe quel protocole en ajoutant, ou en modifiant quelques aspects d'échanges sans changer les modes de fonctionnement et les blocs qui composent son architecture.

L'architecture de l'adaptateur de protocole représentée sur la figure 2.10 est composée d'une mémoire cache, d'un DMA [24], et de deux contrôleurs d'émission et de réception. Le contrôleur

24. Direct Memory Access

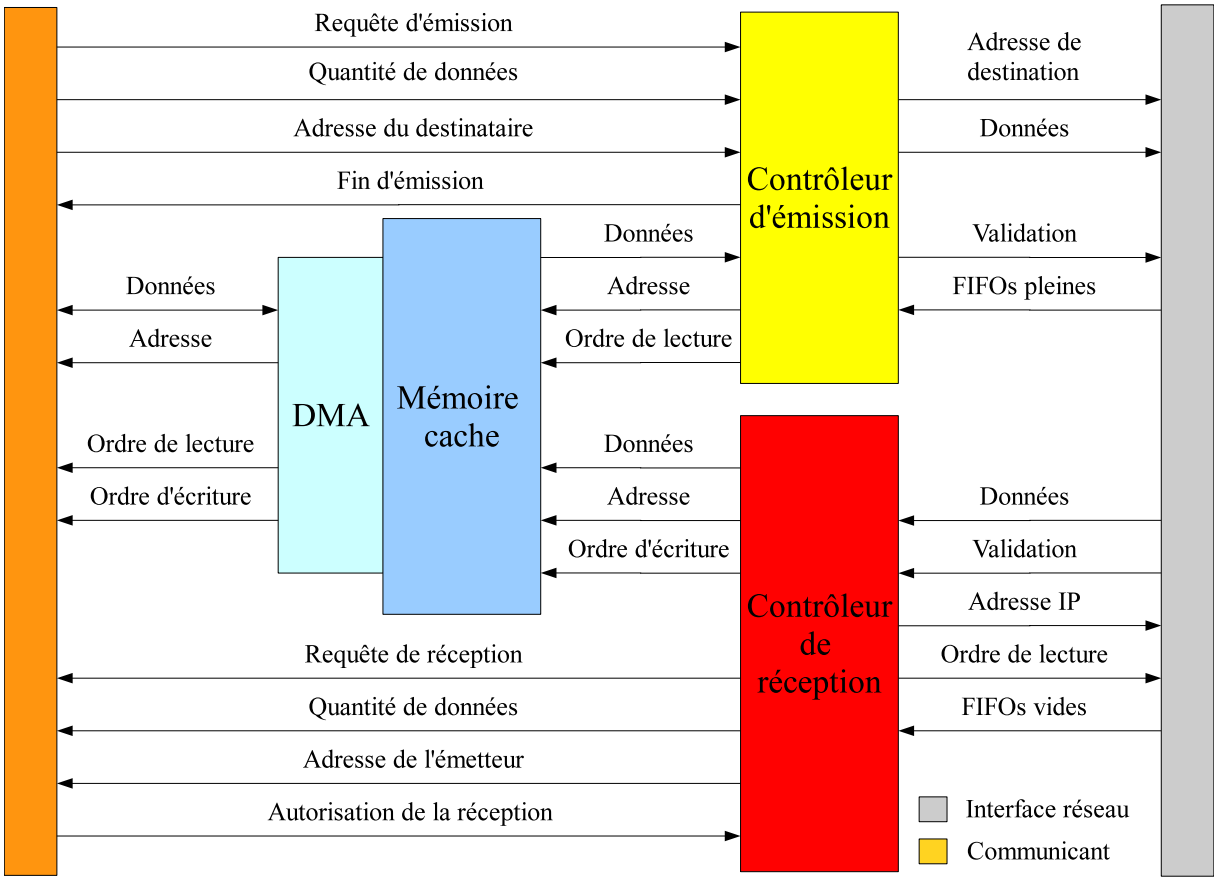


FIGURE 2.10 – Architecture de l'adaptateur de protocole µSpider II

d'émission gère les requêtes d'envoi de messages par le communicant. Le contrôleur de réception informe le communicant des nouveaux messages reçus par l'interface réseau, et il transfert ces messages selon la disponibilité du communicant. La mémoire cache stocke les messages en émission ou en réception. Son utilisation diminue le nombre d'accès à la mémoire locale du communicant. Ainsi, l'envoi ou la réception d'une quantité de données d'un message se fait par bloc dont la taille est égale à la taille de la mémoire cache. Cette taille ne dépend pas de la taille des données d'une communication. Elle est calculée selon le nombre des FIFOs en émission et en réception de l'interface réseau et selon le temps d'accès à la mémoire locale du communicant. La procédure de ce calcul est détaillée dans la section 3.5. Le DMA permet l'accès direct à la mémoire du communicant sans passer par son unité de traitement. Ceci permet d'alléger l'unité de traitement des différentes tâches d'accès à la mémoire. L'utilisation du DMA est justifié seulement si l'accès à la mémoire locale du communicant est autorisé.

Un communicant dans les nouveaux systèmes multiprocesseurs est souvent constitué d'un processeur, d'une mémoire et de plusieurs périphériques, le tout étant connecté par un bus. L'introduction du DMA et d'une mémoire cache dans l'adaptateur de protocole nous semble incontournable afin de mieux répondre aux attentes de ces nouvelles architectures. Ce procédé permet à l'adaptateur de protocole de se connecter au bus du communicant et d'accéder aux données des différentes entités connectées au bus sans passer par le processeur.

2.4.1 Mode d'émission

L'émission des données par un communicant vers divers destinataires doit respecter le mode d'émission de l'adaptateur de protocole quelque soit le protocole du communicant. Ce mode applique une procédure composée de trois étapes :

- la première étape est l'envoi par le communicant d'une requête d'émission à l'adaptateur de protocole, en spécifiant l'adresse du destinataire, la quantité de données à transmettre, et l'adresse de la première donnée stockée dans la mémoire locale connectée au bus du composant. Cette requête est traitée par le contrôleur démission de l'adaptateur de protocole ;
- la deuxième étape consiste à lire les données par blocs et les stocker dans la mémoire cache. Puis, à la fin de lecture de chaque bloc, les données lues sont transmises par le contrôleur d'émission à l'interface réseau. Ainsi l'adaptateur de protocole réalise plusieurs transferts entre le communicant et l'interface réseau afin d'envoyer la quantité de données spécifiée dans la requête d'émission. Le nombre de transferts dépend de la taille de la mémoire cache, et chaque transfert est réalisé en deux phases :
  1. le contrôleur d'émission spécifie l'adresse de la première donnée du bloc à lire et sa quantité au DMA. La quantité de données du bloc est paramétrable mais elle doit être inférieure ou égale à la taille de la mémoire cache. Ensuite, le DMA transfert ce bloc de données de la mémoire du communicant à la mémoire cache sans passer par l'unité de traitement du communicant,
  2. et à la fin de lecture du bloc de données, le contrôleur d'émission transmet les données de la mémoire cache vers l'interface réseau, en respectant le protocole d'émission des données vers l'interface réseau (voir section 2.3.1) ;
- la troisième étape commence quand le contrôleur d'émission envoie toute la quantité de données spécifié dans la requête d'émission du communicant. Il informe le communicant de la fin de la requête d'envoi en utilisant l'interruption "fin d'émission".

Cette technique de transfert permet de récupérer les données de la mémoire locale du communicant par blocs avant de les envoyer à l'interface réseau. Ceci permet d'alléger le travail de l'unité de traitement du communicant et de réduire le temps de transfert en accédant aux données par le biais du DMA. Notre adaptateur de protocole peut gérer simultanément plusieurs requêtes d'émissions avec des destinataires différents.

2.4.2 Mode de réception

Le mode de réception des données par l'adaptateur de protocole est composé de quatre étapes :

1. Quand le contrôleur de réception de l'adaptateur de protocole est averti que l'une des FIFOs de l'interface réseau n'est pas vide. Il informe par interruption le communiquant que l'un de ses interlocuteurs a envoyé des données, en spécifiant l'adresse de l'émetteur ;
2. Le communicant autorise le contrôleur de réception à récupérer ces données et à les stocker dans la mémoire cache. L'autorisation n'est émise que si la mémoire cache est vide et qu'aucune autre requête d'émission ou de réception n'est en cours ;
3. À la réception de l'autorisation le contrôleur de réception récupère une quantité de données inférieure ou égale à la taille de la mémoire cache de l'interface réseau en utilisant le protocole abordé à la section 2.3.2 ;

4. Ensuite le DMA envoi directement les données sur la mémoire du communicant, puis il informe le contrôleur de réception de la fin du transfert et que la mémoire cache est de nouveau vide.

2.5 Conclusion

Le réseau μspider II connecte plusieurs communicants. Chaque communicant a besoin d'un adaptateur de protocole et d'une interface réseau pour accéder au réseau en émission et en réception. Les interfaces réseau sont connectées par une topologie composée de routeurs. Ce chapitre nous a permis de définir les valeurs des paramètres du modèle OSI afin de concevoir un réseau sur puce garantissant les trafics. Nous distinguons plusieurs choix qui différencies notre réseau μSpider II des autres NoCs :

- le premier concerne le choix d'une topologie irrégulière construite selon le type de l'application, le type d'adressage mémoire et le protocole d'échange appliqué ;
- l'utilisation d'un nouveau contrôle de flux par effet domino qui diminue la complexité et le débit du réseau si on le compare à un contrôle de flux par crédit d'émission ;
- et l'introduction d'une architecture pour l'adaptateur de protocole indépendante du standard utilisé par le communicant, cette architecture a la particularité d'utiliser un DMA pour accéder directement à la mémoire locale du communicant.

Ce chapitre définit également les architectures des différents composants du réseau μspider II, qui répondent aux choix faits dans les différentes couches OSI. Nous présentons dans le chapitre suivant le flot de conception du réseau μspider II, qui permet le dimensionnement des différents éléments du NoC. Ainsi, ce flot de conception définit la méthode de conception de la topologie, la procédure de calcul des intervalles de temps pour satisfaire les contraintes des communications. Il introduit également l'algorithme pour calculer les tailles des FIFOs en réception et en émission de l'interface réseau, et la taille de la mémoire cache de l'adaptateur de protocole, ainsi qu'une heuristique pour calculer les chemins spatio-temporelles.

3

Dimensionnement et flot de conception du NoC μSpider II

Le rôle d'un réseau sur puce est d'interconnecter plusieurs communicants qui peuvent être des blocs fonctionnels complexes (IP), des mémoires, ou des processeurs, mais ils peuvent être également des systèmes complexes constitués d'un processeur, de mémoires, et d'accélérateurs, le tout relié par un bus. Donc, le réseau sur puce raccorde les différentes unités de traitement d'un système multiprocesseur afin qu'elles puissent échanger des données.

La nature des échanges effectués dans une architecture multiprocesseur dépend principalement du type de l'application, et du partitionnement de ses tâches sur les différentes unités de traitement. Par conséquent, la conception d'un réseau sur puce est liée à l'application, et elle doit suivre un flot de conception afin de dimensionner certains éléments des composants du réseau, pour répondre aux exigences de n'importe quel type d'application.

Ce chapitre présente le flot de conception du réseau μSpider II, qui est composé de cinq étapes :

1. L'étape de **spécification** permet au concepteur de définir la fréquence du réseau, le nombre de communicants connectés au réseau, le partitionnement des tâches de l'application sur les communicants, ainsi que les communications attribuées à chaque tâche. Cette définition est exprimée sous forme de graphes de dépendances, où chaque graphe représente une tâche avec les communications qui la compose. Ce graphe de dépendance des communications CDG [25] est définit dans la section 3.2 de ce chapitre ;
2. L'étape de **construction** définit la bande passante maximale et réelle des différentes communications en se basant sur les graphes CDG. Cette méthode de calcul prouve que la bande passante déterminée par le concepteur est souvent surestimée. Cette étape conçoit également la topologie du réseau en appliquant l'un des algorithmes de conception, selon la nature de l'application et le type d'adressage des mémoires dans l'architecture multiprocesseur. La section 3.3 de ce chapitre présente la méthodologie et les algorithmes de conception de la topologie du réseau μspider II ;
3. L'étape de **dimensionnement de la table TDMA** détaillée dans la section 3.4 de ce chapitre, consiste à calculer le nombre d'intervalles de temps à réserver pour chaque communication selon sa bande passante ;
4. L'étape de **dimensionnement des mémoires** détermine les tailles des mémoires tampons des interfaces réseau, et les tailles des mémoires caches des adaptateurs de protocole. Ce calcul est détaillé dans la section 3.5 de ce chapitre ;

25. Communication Dependency Graph

5. Et l'étape d'**allocation des chemins** explore les chemins dans l'espace (topologie) et dans le temps (TDMA) pour trouver une solution de routage à toutes les communications tout en respectant leur contraintes en terme de bande passante. La méthodologie d'exploration et d'allocation des chemins spatiotemporels est expliquée dans la section 3.6 de ce chapitre.

Les différents algorithmes utilisés dans ce chapitre considèrent la variable temporelle. Nous exprimons ce temps avec le nombre de cycles.

3.1 Notation des composants et des variables

Dans ce chapitre nous utilisons une notation spécifique pour identifier les différents composants du réseau, l'ensemble des communications, ainsi que les variables utilisées dans les différentes méthodes de calcul.

La notation des composants et des variables du réseau est la suivante :

- NI_i représente l'interface réseau i, avec $i = \{0, ..., M-1\}$ et M est le nombre de NI dans le réseau
- S_i est le nombre de slots de la table TDMA attribué à la NI_i
- R_i identifie le routeur i, avec $i = \{0, ..., N-1\}$ et N est le nombre de routeurs qui composent le réseau
- PA_i est l'adaptateur de protocole i, avec $i = \{0, ..., M-1\}$. En sachant que chaque interface réseau doit être connectée à un adaptateur de protocole, le nombre de PA doit être égale au nombre de NI
- LW est la largeur des liens du réseau, sa valeur est exprimée en $(bits)$
- F est la fréquence d'horloge du réseau, sa valeur est exprimée en (Hz)

Les communications et leur variables sont identifiées avec les symboles suivants :

- C_{ij} symbolise la communication j émise par l'interface réseau NI_i, avec $j = \{0, ..., L_i - 1\}$ et L_i est le nombre de communications administrées et émies par la NI_i ;
- $Sour_{ij}$ et $Dest_{ij}$ représentent respectivement la NI source et la NI destination de la communication C_{ij} ;
- BW_{ij} est la bande passante de la communication C_{ij} ;
- BW_{ij}^{S} est la bande passante de la communication C_{ij} exprimée avec le nombre de slots de temps alloué à la communication C_{ij} ;
- MQ_{ij} est la quantité maximale de données que la communication C_{ij} peut transporter, elle est exprimée en $(bits)$;
- S_{ij} ou $S^{'}ij$ est le nombre de slots alloué à la communication C_{ij} dans la table TDMA de la NI_i, il représente aussi le nombre de paquets de cette communication ;
- $Path_{ij}$ est une suite orientée de liens qui composent le chemin que la communication C_{ij} doit prendre pour arriver à sa destination ;
- $Length_{ij}$ est la longueur du chemin $Path_{ij}$;
- $Tdep_{ij}$ symbolise l'instant de départ de l'entête de la communication C_{ij} ;
- P_{ij} est la priorité calculée de la communication C_{ij} ;
- FBS_{ij} symbolise la taille de la mémoire tampon d'émission utilisée pour stocker les données de la communication C_{ij} dans l'interface réseau NI_i, elle est exprimée en $bits$;
- C_{ik} est la communication k reçue par l'interface réseau NI_i, avec $k = \{0, ..., R_i - 1\}$ et R_i est le nombre de communications administrées et reçues par la NI_i ;
- RBS_{ik} identifie la taille de la mémoire tampon de réception utilisée pour stocker les données de la communication C_{ik} dans l'interface réseau NI_i, elle est exprimée en $bits$;

– NF_i est le nombre de mémoires tampons de réception gérées par l'interface réseau NI_i.

Les variables d'un adaptateur de protocole PA_i sont symbolisées par :
– WM_i : la taille de la mémoire cache de l'adaptateur de protocole, elle est exprimée en $(bits)$;
– WD_i : le délai du DMA pour écrire ou lire un bloc de données de taille WM_i, il est exprimé en cycle d'horloge.

Pour la procédure de calcul des chemins dans le NoC, on considère les définitions suivantes :
– La topologie du réseau est défini par un graphe orienté $G(V, E)$, où V représente les sommets c'est à dire l'ensemble des routeurs du réseau, et E représente l'ensemble des arcs réalisant les connexions entre les sommets, ainsi l'arc $E^{V}_{V'}$ a V comme source et V' comme destination ;
– La période T représente la durée de rotation des tables TDMA, T est aussi appelée la période de la table TDMA.

3.2 Graphe de dépendance des communications

Habituellement, l'application est décrite avec le graphe des communications des tâches CTG [26] [MHO05]. Le CTG noté $G(T, D)$ est un graphe orienté acyclique, où chaque sommet $t_i \in T$ représente une tâche de l'application et renferme plusieurs informations sur celle-ci, tel que le temps et le délai d'exécution, l'identifiant de l'unité de traitement qui l'exécute, sa périodicité dans le graphe. Chaque arc d_{ij} entre la tâche t_i et la tâche t_j caractérise une communication et spécifie la quantité de données $v(d_{ij})$ envoyées de la tâche t_i à la tâche t_j.

Le graphe CTG est suffisant pour décrire une application, en donnant les informations nécessaires sur les tâches qui la composent, et les quantités de données échangées entre elles. Dans notre flot de conception, nous avons défini une nouvelle description de l'application, cette description est réalisée avec plusieurs graphes de dépendance des communications (CDG), où chaque graphe représente une tâche.

Une tâche est un ensemble de processus exécuté d'une manière séquentielle. Chaque processus est soit un traitement, soit un échange avec une autre tâche. Ainsi, une tâche peut être décrite avec un graphe orienté cyclique $G(P, D)$, où chaque sommet $P_k \in P$ représente un processus et chaque arc D_{lm} représente la dépendance entre le processus P_l et le processus P_m.

La figure 3.1 représente un exemple de description d'une tâche avec le graphe CDG. Cette tâche est composée de 10 processus, trois processus de traitement et sept processus d'échange. Chaque processus d'échange est représenté par un sommet $P_k \in P$. Ce sommet définit la tâche émettrice des données T_i, la tâche réceptrice T_j, la quantité de données transmises Q_{ij} en nombre de paquets, et la nature de l'échange. Ainsi le sommet $Com1$ représente une communication entre la tâche T_1 et T_2 de type écriture W avec une quantité de données Q_{12} égale à 16 paquets. Et le sommet $Com2$ représente une communication entre la tâche T_4 et T_1 de type lecture R avec une quantité de données Q_{41} égale à 50 paquets. Chaque processus de traitement est représenté aussi par un sommet $P_k \in P$ qui définit le temps d'exécution de ce traitement $t(P_i)$.

Le concepteur spécifie, pour chaque graphe CDG, la taille des paquets envoyés et reçus par la tâche, le temps d'exécution pour un cycle, et l'identifiant de l'élément de l'architecture multiprocesseur qui exécutera cette tâche. Le graphe CDG permet de modéliser l'exécution en parallèle de plusieurs processus, comme le montre la figure 3.1. Les processus d'échange $Com6$

26. Communication Task Graph

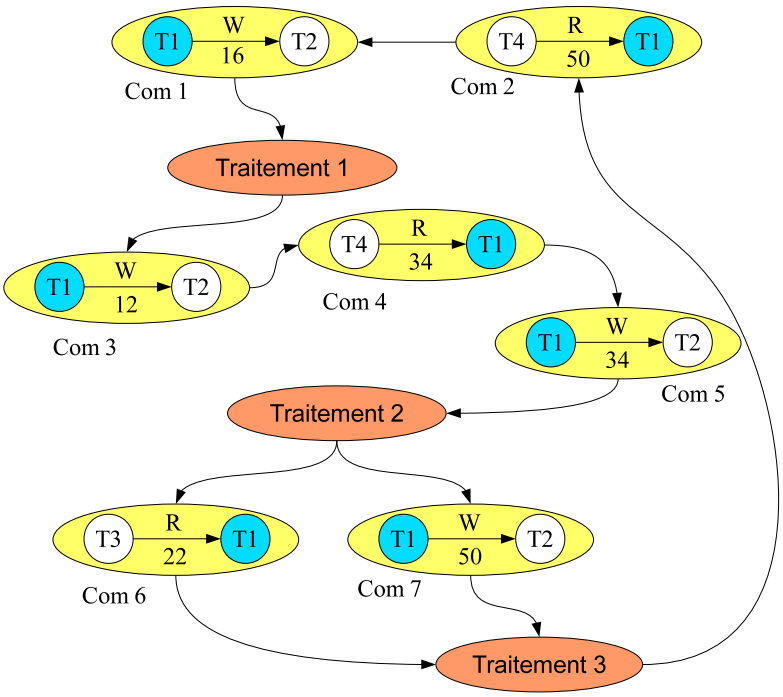


FIGURE 3.1 – Graphe représentant une tâche avec 3 processus de traitement et 7 processus de communication

et $Com7$ sont exécutés en parallèle et ils sont lancés au même instant à la fin du processus de traitement 2. Il faut préciser aussi que le processus de traitement 3 n'est lancé qu'à la fin de l'exécution des deux processus d'échange $Com6$ et $Com7$.

La représentation de l'application avec des graphes CDG permet de calculer la bande passante réelle, et la taille des données maximale des communications échangées par les composants d'une architecture multiprocesseur. Le parallélisme est exprimé de manière explicite avec les graphes CDG, cela permet donc de calculer de manière précise les bornes minimum et maximum de la bande passante des différentes communications. La borne minimum est identifiée en considérant l'exécution séquentielle des processus d'échange de la même communication. En revanche, la borne maximum est obtenue en considérant les processus d'échange exécutés en parallèle pour une même communication. Ceci repose sur l'exhibition du parallélisme entre les processus d'échange, rendu possible par les arcs de dépendance au sein des graphes CDG.

3.3 Méthodologie de conception de la topologie

L'hétérogénéité des applications et des architectures des systèmes embarqués multiprocesseur empêche la définition d'une méthode générale pour la conception de la topologie du NoC car, la nature des communications et l'architecture du système varie d'un domaine d'application à un autre. Notre approche classifie les systèmes multiprocesseurs selon différentes classes, puis, elle conçoit la topologie du NoC selon la classe du système multiprocesseur. Elle place également les différents communicants sur cette topologie selon les contraintes, et les dépendances de leurs communications, en exploitant les graphes de dépendance CDG.

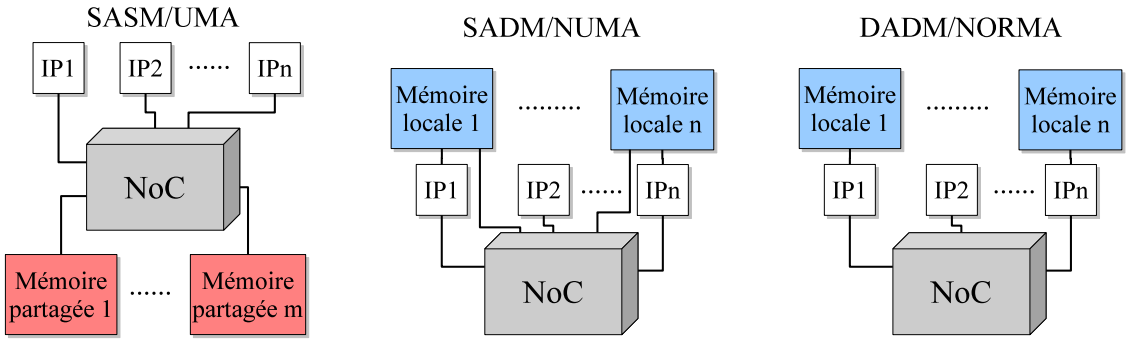


FIGURE 3.2 – Classification de Raina des architectures multiprocesseurs

La classification des architectures de machines parallèles la plus connue est celle de Flynn [Fly72], qui caractérise les machines suivant leurs flots de données et d'instructions, cette classification distingue quatre catégories :

1. SISD [27] correspond aux machines séquentielles, où chaque opération traite une seule donnée à la fois ;
2. MISD [28] réalise un traitement sur la même donnée par plusieurs processeurs en parallèle ;
3. SIMD [29] est une architecture composée de processeurs exécutant au même instant la même instruction, chacun sur des données différentes ;
4. MIMD [30] correspond à une architecture multiprocesseur, où chaque processeur exécute ses propres instructions de manière indépendante des autres.

La classification de Flynn a été enrichie par Raina [Rai92], en créant une sous-classification de la classe MIMD. Cette sous-classification illustrée dans la figure 3.2 prend en compte l'architecture mémoire et le type d'accès à la mémoire, ainsi nous distinguons :

- SASM [31]/UMA [32] est une architecture multiprocesseur utilisant des mémoires partagées, avec un accès symétrique à la mémoire et de coût identique pour tous les processeurs ;
- DADM [33]/NORMA [34] est une architecture multiprocesseur avec une mémoire distribuée, sans accès aux données distantes, et nécessitant le passage de messages ;
- SADM [35]/NUMA [36] est une architecture multiprocesseur à mémoire distribuée, mais avec un espace d'adressage global, qui autorise l'accès aux données situées sur d'autres mémoires.

Cette section présente deux méthodes de conception de topologies. La première méthode utilisée pour les classes SASM et SADM, et détaillée dans la section 3.3.1. La deuxième génère une topologie pour l'architecture multiprocesseur appartenant à la classe DADM. Elle est expliquée dans la section 3.3.2.

27. Single Instruction Single Data
28. Multiple Instruction Single Data
29. Single Instruction Multiple Data
30. Multiple Instruction Multiple Data
31. Single Address space, Shared Memory
32. Uniform Memory Access
33. Distributed Address space, Distributed Memory
34. No Remote Memory Access
35. Single Address space, Distributed Memory
36. Non-Uniform Memory Access

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
M1	50	20	30	0	0	0	0	20	50	10
M2	0	0	20	30	50	40	65	10	0	0
M3	0	0	0	0	0	0	10	20	24	100
M4	30	20	15	0	50	20	0	0	0	0

TABLE 3.1 – Bandes passantes en Mbits/s des échanges entre les processeurs et les mémoires de l'exemple utilisé pour expliquer le méthode de conception de la topologie pour les classes SASM et SADM

3.3.1 Classes SASM et SADM

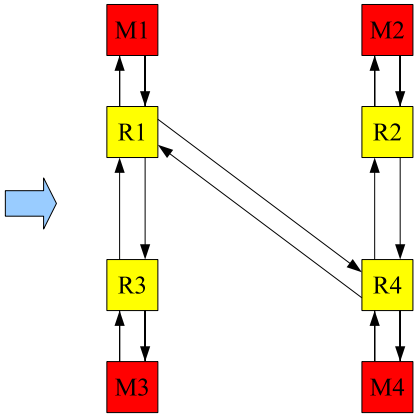
Les architectures multiprocesseur appartenant aux classes SASM et SADM centralisent les communications sur les mémoires car, la majorité des échanges sont des communications entre les processeurs et les mémoires. Par conséquent, la méthode de conception de la topologie du NoC pour les architectures de types SASM et SADM commence par connecter et placer les mémoires. Puis, elle connecte les processeurs à ces mémoires d'une façon hiérarchique en se basant sur les contraintes de bande passante. Donc, la méthode de conception de la topologie pour les classes SASM et SADM contient deux étapes :

- la première étape consiste à connecter et placer les mémoires dans une topologie irrégulière. Cette topologie doit contenir un nombre de routeurs égal au nombre de mémoires, ainsi chaque mémoire est connectée à un seul routeur principal, qui sera le lien exclusif avec un ensemble de processeurs. Afin de permettre à ces processeurs d'accéder aux autres mémoires qui constituent le système, nous relions les routeurs principaux avec un nombre de connexions égal au nombre de mémoires moins une. Et pour que les accès aux mémoires soient en lecture et en écriture, nous composons chaque connexion de deux liens unidirectionnels.
  Après la définition du nombre de routeurs et du nombre de connexions qui constituent la topologie connectant les mémoires. Nous relions les différents routeurs, donc les différentes mémoires selon un classement établi, en fonction du nombre de communicants que les mémoires ont en commun et en fonction de leur bande passante ;
- la deuxième étape contient deux phases :

  1. la première phase crée un nombre de groupes égal au nombre de mémoires, afin d'attacher à chaque mémoire un groupe distinct. Puis, nous plaçons chaque processeur du système dans le groupe de la mémoire avec lequel il échange le plus de données. Au cas où le processeur échange la même quantité de données avec différentes mémoires, nous le plaçons dans le groupe qui contient le moins de processeurs.
     Pour la classe SADM, la réalisation de ce groupement doit intégrer une contrainte supplémentaire. Cette contrainte exige le placement de certains processeurs dans les groupes de leur mémoire locale même s'ils n'échangent aucune donnée avec ces mémoires.
  2. la deuxième phase conçoit une topologie pour connecter les processeurs d'un même groupe avec la mémoire attachée à ce groupe. Cette topologie est constituée d'un nombre de routeurs égal au nombre de processeurs divisé par deux. Nous relions chaque routeur à deux processeurs et au routeur principal lui même connecté à la mémoire du groupe. Puis, nous connectons les routeurs du groupe avec une structure grille deux dimensions. Ceci permet d'augmenter le nombre de chemins possibles entre

Première étape

Classement	Connexion	Les communicants en commun	Bande passante en commun
1	M1 ↔ M3	P8, P9, et P10	224
2	M2 ↔ M4	P3, P5, et P6	195
3	M1 ↔ M4	P1, P2, et P3	165
4	M2 ↔ M3	P7 et P8	105
5	M1 ↔ M2	P3 et P8	80



Deuxième étape

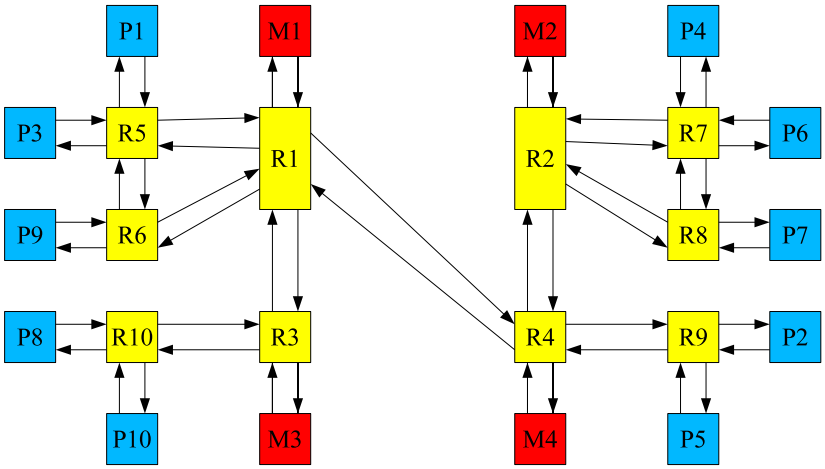


FIGURE 3.3 – Méthode de conception de la topologie pour la classes SASM

les processeurs de ce groupe et le routeur principal.

Pour mieux comprendre la méthode de conception de la topologie pour les classe SASM et SADM, nous considérons l'exemple d'une architecture multiprocesseur composée de 10 processeurs (P_1, $P_2, \ldots,$ *et* P_{10}) qui échangent des données via 4 mémoires partagées (M_1, M_2, M_3, *et* M_4). Les bandes passantes des échanges avec les mémoires sont définies dans le tableau 3.1.

Pour cet exemple, la première étape de la procédure de conception de la topologie génère une topologie représentée sur la partie haute de la figure 3.3. Cette topologie connecte les quatre mémoires de l'exemple et elle est constituée de quatre routeurs et trois connexions. Les liaisons entres les différents routeurs sont réalisés selon le classement des connexions. Ce classement est établi à partir des contraintes de communications du tableau 3.1. Le résultat de ce classement est représenté sur le tableau de la partie haute de la figure 3.3.

L'exécution de la deuxième étape de la procédure complète la topologie connectant les mémoires, en classant les processeurs dans différents groupes, puis en connectant les processeurs de chaque groupe au routeur principal lié à la mémoire attachée à ce groupe. La topologie finale générée pour l'exemple est représentée sur la partie basse de la figure 3.3. Cette topologie reflète

	P1	P2	P3	P4	P5	P6	P7	Total en émission
P1	0	10	0	10	100	0	50	170
P2	0	0	0	20	0	80	90	190
P3	15	10	0	80	30	100	0	235
P4	30	0	0	0	80	10	40	160
P5	50	0	100	100	0	10	100	360
P6	0	100	0	40	0	0	0	140
P7	20	0	20	0	50	10	0	100
Total en réception	115	120	120	250	260	210	280	

TABLE 3.2 – Bandes passantes en Mbits/s des échanges entre les processeurs de l'exemple utilisé pour expliquer le méthode de conception de la topologie pour la classe DADM

l'efficacité de notre méthode. Nous observons que tous les processeurs peuvent échanger avec les différentes mémoires. De plus les processeurs sont placés de manière à minimiser le temps d'accès aux mémoires avec lesquelles ils échangent le plus de données.

3.3.2 Classe DADM

Une architecture multiprocesseur appartenant à la classe DADM est composée exclusivement de cœurs de traitements. Pour cette classe, nous avons développé un procédé de conception de topologie fondé sur le principe de proximité, qui consiste à rapprocher les communicants qui échangent le plus de données.

Le méthode de conception place chaque communicant dans une topologie de type étoile, et elle connecte aux branches de l'étoile les cœurs de traitement qui échangent le plus de données avec ce communicant. Ensuite, elle connecte les différentes topologies étoiles selon des règles qui appliquent le principe de proximité. On notera que ce principe vise également la réduction du temps de réponse.

Pour expliquer cette méthode, nous utilisons un exemple d'architecture multiprocesseur constitué de 7 IPs. Les valeurs des bandes passantes des communications émises et reçues par les IPs sont regroupées dans le tableau 3.2.

La première étape de la méthode de conception de la topologie pour la classe DADM consiste à placer chaque communicant au centre d'une topologie de type étoile. Nous utilisons cette topologie car sa structure offre la même distance entre l'IP placé au centre et les différentes IPs placés sur les extrémités. Le choix des IPs qui seront connectés aux extrémités de la topologie étoile dépend de la bande passante des communications échangées avec l'IP placé au centre de l'étoile. Par conséquent, pour placer un IP sur l'une des extrémités de l'étoile, il faut que le pourcentage de la bande passante de ses communications échangées avec l'IP placé au centre dépasse un certain seuil. Ce seuil est paramétrable, et il est défini par l'utilisateur lors de la conception du NoC. Ce seuil est un paramètre de réglage de la méthode, plus sa valeur est faible, plus le principe de proximité sera déterminant.

L'application de cette étape sur l'exemple du tableau 3.2 se traduit par la conception de 7 topologies distinctes et de type étoile, puis le placement au centre de chaque étoile d'un IP parmi les 7 qui constituent l'architecture multiprocesseur. Ensuite, le placement des différents IPs sur les extrémités des différentes étoiles est réalisé, suivant le pourcentage des bandes passantes des communications échangées avec l'IP placé au centre de l'étoile. Par exemple, l'IP_5 échange

des données avec l'IP_1, l'IP_3, l'IP_4, l'IP_6 et l'IP_7, en sachant que la bande passante totale en émission et en réception de l'IP_5 est égal à 620 $Mbits/s$. Les pourcentages des bandes passantes des communications échangées entre l'IP_5 et les IP_1, IP_3, IP_4, IP_6, IP_7 sont respectivement 24%, 21%, 29%, 1,6% et 24%. Pour cet exemple nous avons fixé le seuil à 10%. Par conséquent, nous connectons les IP_1, IP_3, IP_4, et IP_7 aux extrémités de la topologie étoile où l'IP_5 est placé au centre. La figure 3.4 représente les différentes topologies étoiles que nous avons obtenu pour les 7 IPs.

La deuxième étape de la méthode repose sur la fusion des différentes topologies de type étoile générées par la première étape. La fusion de ces étoiles est réalisée en plusieurs phases, la première phase combine deux étoiles distinctes, ensuite à chaque nouvelle phase on fusionne la topologie résultante de la phase précédente avec une nouvelle étoile. Le choix des étoiles à fusionner de chaque phase dépend de la bande passante totale en émission et en réception de l'IP placé au centre de l'étoile, ainsi la priorité des étoiles à fusionner est proportionnelle aux valeurs des bandes passantes totales des IPs placés à leurs centres. Pour respecter le principe de proximité, la fusion de deux topologies doit considérer les règles suivantes :

- tous les IPs connectés sur les deux topologies doivent être connectés à la topologie résultante de la fusion ;
- chaque IP ne doit être présente qu'une seule fois sur la topologie résultante de la fusion ;
- chaque IP est connecté à un seul routeur ;
- et la topologie résultante doit inclure tous les liens entre les IPs (routeurs) trouvés sur les deux topologies fusionnées.

La réalisation de la deuxième étape sur l'exemple 3.2 permet de composer une topologie en fusionnant les différentes topologies de types étoiles. Dans cet exemple, la mise en œuvre de cette fusion comporte six phases, car il y a sept IPs. Comme le montre la figure 3.4, la première phase combine les étoiles des IPs qui possède la bande passante totale la plus élevée. Dans cet exemple il s'agit de l'IP_5 et l'IP_4. Ensuite, les phases suivantes fusionnent la topologie résultante de la phase précédente et une étoile parmi celles qui n'ont pas été encore fusionnées, et en choisissant toujours celle qui connecte à son centre l'IP qui possède la bande passante la plus élevée.

3.3.3 Algorithme de génération des topologies

L'algorithme de génération des topologies pour notre réseau µSpider II est donc composée de trois étapes majeures :

1. L'initialisation consiste à créer une topologie initiale en utilisant les deux méthodes de conception décrites dans les sections précédentes. Le choix de la méthode à utiliser dépend entièrement de la classe de l'architecture multiprocesseur ;
2. La deuxième étape permet d'améliorer la solution initiale. Cette amélioration concerne le nombre de ports d'entrée/sortie des routeurs. Nous avons montré dans la section 2.2.3 du chapitre précédent, que ce n'est pas le nombre de routeurs qui influe sur la taille du réseau mais le nombre de ports d'entrée/sortie par routeur. Ainsi, nous remplaçons les routeurs de la topologie initiale dans le cas où ils ont un nombre de ports important, par plusieurs routeurs avec moins de ports dans le but de réduire la taille de la topologie.
   Puis, nous exécutons l'algorithme de recherche des chemins spatio-temporels, de la section 3.6, en utilisant la topologie résultante afin de savoir si celle-ci permet de router toutes les communications. En cas d'échec, nous rajoutons à la topologie des liens et des routeurs supplémentaires. Cette technique augmente le nombre de routes à utiliser par l'algorithme de routage.

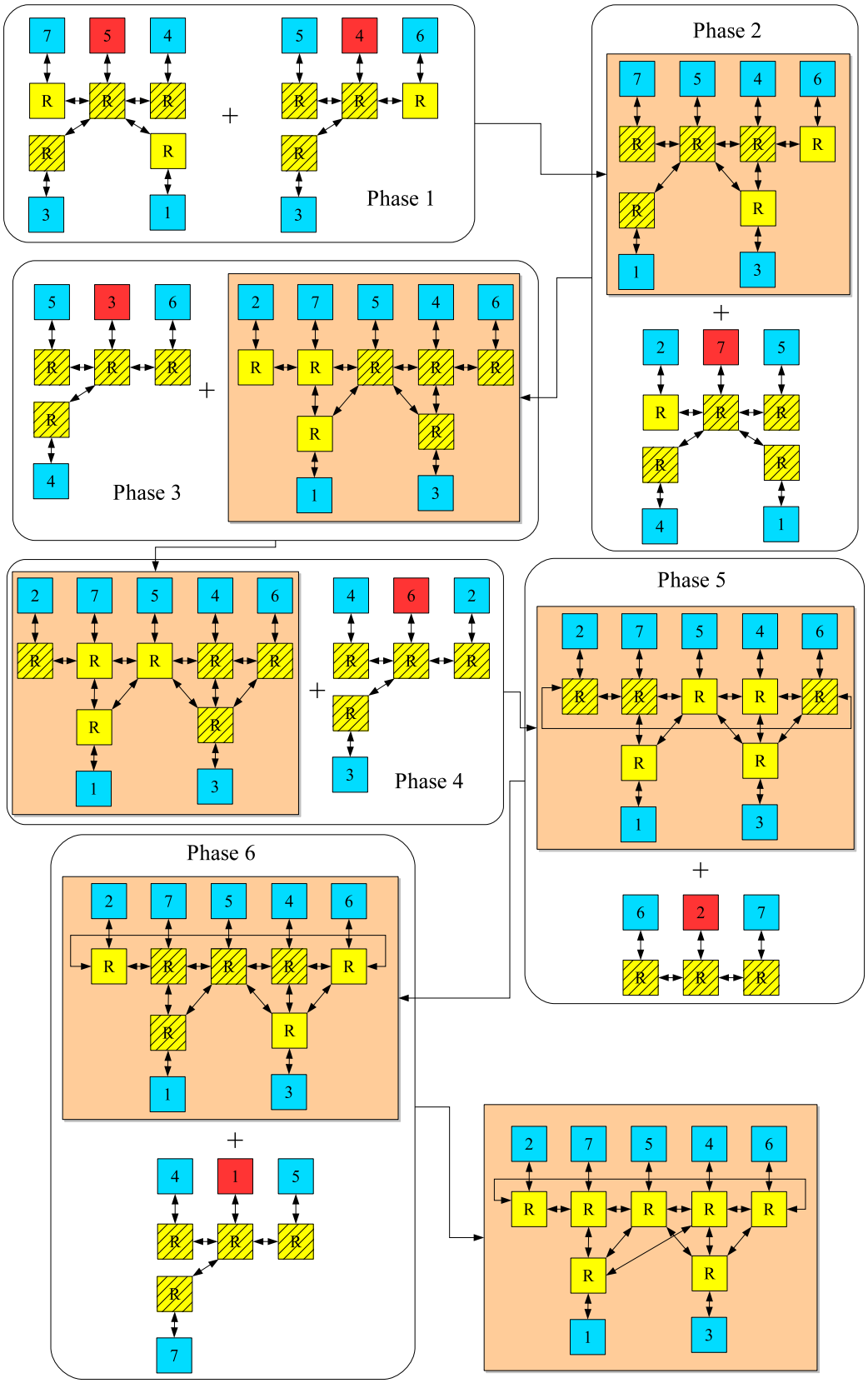


FIGURE 3.4 – Les différentes étapes de conception d'une topologie pour la classe DADM

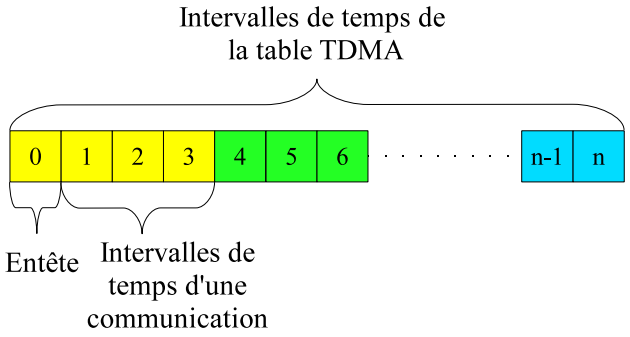


FIGURE 3.5 – Table d'ordonnancement de type TDMA

Nous réitérons la deuxième étape jusqu'à ce que l'algorithme de recherche de chemins s'exécute avec succès ;

3. La troisième étape optimise la solution de la deuxième étape, en supprimant les liens et les routeurs non utilisés ou sous-utilisés par les chemins des différentes communications. Cette optimisation est réalisée de manière itérative. À chaque itération, nous supprimons un seul élément. Nous exécutons l'algorithme de recherche de chemins. En cas d'échec, nous remettons l'élément supprimé, sinon nous continuons l'optimisation. Il s'agit donc d'une approche gloutonne avec un critère de choix qui repose sur le taux d'utilisation des routeurs et des liens de la topologie.

3.4 Calcul des intervalles de temps

Le réseau sur puce µSpider II est constitué de n interfaces réseau NI [37]. Le rôle principal de l'interface réseau est d'ordonnancer l'envoi des données vers le réseau afin de garantir une qualité de service conforme aux contraintes de bande passante. Cet ordonnancement est mis en œuvre avec l'intégration du mécanisme d'accès multiple à répartition dans le temps (table TDMA), dont le principe est de découper le temps disponible entre les différentes communications de la NI (Figure 3.5).

Ainsi, la table TDMA est divisée en un ensemble d'intervalles de temps (appelés "slots" par la suite). L'attribution, à chaque communication de la NI, d'un certain nombre de slots est la première étape du processus permettant de garantir les contraintes en terme de bande passante. Cette section détaille la méthode de calcul du nombre de slots réservé à chaque communication.

3.4.1 Procédure de calcul du nombre d'intervalles de temps

La bande passante d'une communication, selon le principe de la table TDMA, est le nombre de paquets envoyé par seconde sur le réseau. Ainsi elle peut être exprimée par le produit de la fréquence du réseau et du rapport entre le nombre de slots alloué à cette communication et le nombre de slots de la table TDMA (Eq.3.1).

$$BW_{ij}^{S}(paquets/s) = \frac{S_{ij}}{S_i} * F \tag{3.1}$$

37. Network Interface

La valeur de la bande passante spécifiée par le concepteur est souvent exprimée avec l'unité $bits/s$. Pour l'exprimer avec le nombre de paquets envoyé par seconde, nous la divisons par la largeur du lien (Eq.3.2).

$$BW_{ij}(paquets/s) = \frac{BW_{ij}(bits/s)}{LW} \tag{3.2}$$

Les équations (3.1) et (3.2) permettent d'établir la relation entre le nombre de slots nécessaires à une communication et sa bande passante. Pour un bon dimensionnement, la bande passante BW_{ij} spécifiée par le concepteur doit être inférieure ou égale à la bande passante BW_{ij}^{S} calculée selon le nombre de slots de temps. Ainsi, l'équation (3.3) obtenue, calcule le nombre de slots d'une communication selon sa bande passante, le nombre de slots de la table TDMA, la fréquence du réseau et la largeur des liens.

$$S_{ij} = \frac{BW_{ij} * S_i}{F * LW} \tag{3.3}$$

L'envoi d'un paquet ou de plusieurs paquets successifs d'une même communication dans le réseau impose l'envoi d'un paquet d'identification appelé entête. Alors, pour chaque communication, nous allouons un nombre de slots qui satisfait la bande passante plus un slot de temps pour l'entête. Par conséquent, une communication a au moins deux slots, un pour l'entête et l'autre pour transporter des données. Ainsi, nous obtenons une nouvelle équation (3.4) pour calculer le nombre de slots d'une communication, en ajoutant un slot pour l'entête à l'équation (3.3).

$$\begin{aligned} S_{ij}^{'} &= S_{ij} + 1 \\ &= \frac{BW_{ij} * S_i}{F * LW} + 1 \end{aligned} \tag{3.4}$$

Le nombre de slots S_i d'une table TDMA est la somme des slots de toutes les communications contrôlées par l'interface réseau NI_i. Nous l'exprimons avec l'équation (3.5).

$$S_i = \sum_{j=0}^{L_i-1} S_{ij}^{'} \tag{3.5}$$

Nous remplaçons l'inconnue $S_{ij}^{'}$ dans l'équation (3.5) par sa valeur formulée dans l'équation (3.4). La simplification permet de trouver l'équation exprimant le nombre de slots de la table TDMA.

$$\begin{aligned} (3.4)\ dans\ (3.5) \quad &\Leftrightarrow \quad S_i = \sum_{j=0}^{L_i-1} \left(\frac{BW_{ij} * S_i}{F * LW} + 1 \right) \\ &\Leftrightarrow \quad S_i - L_i = \frac{S_i}{F * LW} \sum_{j=0}^{L_i-1} BW_{ij} \\ &\Leftrightarrow \quad \frac{L_i}{S_i} * (F * LW) = (F * LW) - \sum_{j=0}^{L_i-1} BW_{ij} \\ &\Leftrightarrow \quad S_i = \left\lceil \frac{L_i}{1 - \sum_{j=0}^{L_i-1} \frac{BW_{ij}}{F * LW}} \right\rceil \end{aligned} \tag{3.6}$$

Afin de trouver le nombre de slots de la communication C_{ij}, nous remplaçons l'inconnue S_i dans l'équation (3.4) par sa valeur exprimée par l'équation (3.6).

$$(3.6)\ dans\ (3.4) \quad \Rightarrow \quad S_{ij} = \left(\frac{BW_{ij}}{F * LW} * \left\lceil \frac{L_i}{1 - \sum_{k=0}^{L_i-1} \frac{BW_{ik}}{F*LW}} \right\rceil \right) + 1$$

$$\Rightarrow \quad S_{ij} = \left\lceil \frac{L_i * BW_{ij}}{(F * LW) - \sum_{k=0}^{L_i-1} BW_{ik}} \right\rceil + 1 \tag{3.7}$$

Le calcul des équations (3.6) et (3.7) est correct si la condition suivante est vérifiée :

$$F * LW > \sum_{j=0}^{L_i-1} BW_{ij}$$

Cette condition démontre que la somme des bandes passantes des communications contrôlées par la même interface réseau ne doit pas dépasser la bande passante du réseau. Par conséquent, si la valeur de la fréquence est faible, la valeur de la largeur du lien sera grande selon la formule (3.8).

La largeur des liens a un effet négatif sur la fonction du coût, car en augmentant la taille des liens, nous augmentons la taille des mémoires tampons dans les interfaces réseau, et la taille des registres affectés aux ports d'entré/sortie des routeurs. Ainsi lors de la conception, il faut toujours ajuster la fréquence du réseau par rapport aux exigences demandées en terme de bande passante afin d'éviter l'augmentation de la taille du réseau.

$$LW = \left\lceil \frac{\sum_{j=0}^{L_i-1} BW_{ij}}{F} \right\rceil \tag{3.8}$$

3.4.2 Procédure d'allocation des intervalles de temps

La procédure de calcul des slots de temps des tables TDMA et des communications comporte trois étapes. La première étape, décrite avec l'algorithme 1, consiste à calculer pour chaque communication le nombre de slots en utilisant l'équation (3.7). Cette étape permet aussi de calculer le nombre de slots de temps de chaque table TDMA. Ainsi elle additionne les slots de temps alloués aux communications transitant par la même interface réseau, afin de trouver le nombre de slots de temps minimum de la table TDMA attribuée à cette interface réseau.

À la fin de la première étape, chaque table TDMA a un nombre de slots de temps proportionnel aux bandes passantes de ses communications. Mais pour synchroniser l'envoi des données dans le NoC, toutes les tables TDMA doivent avoir le même nombre d'intervalles de temps. Ainsi, la deuxième étape scrute l'ensemble des tables TDMA pour connaitre le nombre d'intervalles de temps maximal qui sera alloué à toutes les tables TDMA.

En revanche chaque interface réseau reçoit un nombre de paquets des autres interfaces. Pour qu'elle puisse consommer tous les paquets reçus pendant un tour de la table TDMA, il faut que le nombre de slots maximal des tables TDMA soit supérieur ou égal au nombre de paquets reçus par chaque interface réseau. La recherche du nombre de slots maximal est définit avec l'algorithme 2.

La synchronisation des tables TDMA impose une taille commune, ce qui signifie que certaines interfaces réseaux peuvent voir la taille de leur table TDMA augmenter. Ainsi l'uniformisation

Algorithme 1: Calcul des slots de temps pour les communications et les tables TDMA

pour *i=0* **a** M **faire**
  $L[i] \leftarrow NI[i].getNbrCom()$ *On récupère le nombre de communication gérées par la NI*
  $S[i] \leftarrow 0$
  $TotalBW \leftarrow 0$
  pour *k=0* **a** *L[i]* **faire**
    $TotalBW \leftarrow TotalBW + BW[i][k]$
    $k \leftarrow k + 1$
  fin
  pour *j=0* **a** *L[i]* **faire**
    si $TotalBW < F * LW$ **alors**
      $S[i][j] \leftarrow \left\lceil \frac{BW[i][j]*L[i]}{(F*LW)-TotalBW} \right\rceil$
    sinon
      $S[i][j] \leftarrow 0$
    fin
    $S[i] \leftarrow S[i] + S[i][j]$
    $j \leftarrow j + 1$
  fin
  $i \leftarrow i + 1$
fin

des tables conduit généralement à une augmentation de la bande passante disponible. Afin de profiter de cette augmentation, nous distribuons les slots de temps vacants sur les différentes communications. Cette distribution décrite avec l'algorithme 3, est réalisée en deux phases :

- la première phase consiste à classer dans une liste de priorité toutes les communications de toutes les interfaces réseau selon leur priorité. La priorité d'une communication formulée par l'équation (3.9) est la différence entre la bande passante obtenue et la bande passante demandée, divisée par la bande passante de la table TDMA de l'interface réseau émettrice de cette communication ;
- dans la deuxième phase, nous attribuons aux communications par ordre croissant de priorité, un nouveau slot de temps si deux conditions sont satisfaites. La première condition vérifie que la table TDMA de l'interface réseau émettrice de la communication a un slot de temps vacant à attribuer. La deuxième condition s'assure que le slot de temps ajouté à la communication, peut être consommé par la NI réceptrice.
  Si l'une des deux conditions n'est pas remplie, la communication est enlevée de la liste de priorité, sinon elle se voit attribuer un nouveau slot de temps.

Les deux phases de classement et d'attribution des slots vacants sont répétées jusqu'à ce que la liste de priorité soit vide.

$$P_{ij} = \frac{\frac{S_{ij}*F*LW}{S_i} - BWij}{\sum_{k=0}^{L_i-1} BW_{ik}} \tag{3.9}$$

Cette section décrit la procédure de calcul du nombre de slots de temps des différentes

Algorithme 2: Recherche du nombre de slots maximal

```
MaxSlotTdma ← S[0]
pour i=1 a M faire
    si MaxSlotTdma < S[i] alors
        MaxSlotTdma ← S[i]
    fin
    i ← i + 1
fin
pour i=1 a M faire
    NbSlotReceived ← 0
    pour j=1 a NI[i].getNbrComReceived() faire
        NbSlotReceived ← NbSlotReceived + NI[i].ComReceived[j].getNbSlot
        j ← j + 1
    fin
    si MaxSlotTdma < NbSlotReceived alors
        MaxSlotTdma ← NbSlotReceived
    fin
    i ← i + 1
fin
```

communications et le nombre de slots de temps de la table TDMA. En revanche la répartition des slots de temps des communications sur les tables TDMA des différentes interfaces réseau est réalisée en même temps que le calcul des chemins spatio-temporels, cette répartition est détaillée dans la section 3.6.

3.5 Calcul de la taille des mémoires tampons

Nous attribuons deux mémoires tampons à chaque communication C entre deux communicants A et B connectés au réseau. La première mémoire tampon est associée à la partie émettrice de l'interface réseau connectée à A, la seconde est incorporée dans la partie réceptrice de l'interface réseau connectée à B. L'utilisation de ces mémoires tampons est essentielle, car la première régule le débit d'entrée imposé par l'adaptateur de protocole par rapport au débit de sortie contrôlé par la table TDMA. Alors que la seconde mémoire tampon régule le débit d'entrée déterminé par le réseau et le débit de sortie contraint par la procédure de lecture de l'adaptateur de protocole.

L'utilisation des mémoires tampons est nécessaire pour empêcher la monopolisation de l'émetteur ou du récepteur, et donc une perte de performances. Par conséquent, le calcul de la taille des mémoires tampons est très important, car une sous-estimation a un effet négatif sur les performances et une surestimation engendre une augmentation inutile de la taille globale du réseau.

Algorithme 3: Allocation de slots de temps vacants

pour *i=1* **a** *M* **faire**
  $TotalBW \leftarrow 0$
  pour *j=1* **a** *L[i]* **faire**
    $TotalBW \leftarrow TotalBW + BW[i][j]$
    $j \leftarrow j + 1$
  fin
  pour *j=1* **a** *L[i]* **faire**
    $PrioCom[i][j] \leftarrow \frac{\frac{S[i][j]*F*LW}{S[j]-BW[i][j]}}{TotalBW}$
    $j \leftarrow j + 1$
  fin
  $i \leftarrow i + 1$
fin

3.5.1 Méthode de calcul de la taille maximale des FIFOs d'émission

La taille de la FIFO en émission est la différence entre le nombre de paquets IP_{ij} transmis à l'interface réseau par l'adaptateur de protocole et le nombre de paquets OP_{ij} envoyés sur le réseau par l'interface réseau pendant une période T. Afin de calculer la taille de la FIFO au pire cas, nous considérons la quantité maximale de données MQ_{ij} susceptible d'être transmise lors de la communication C_{ij}.

La période T dépend de la procédure d'émission administrée par l'adaptateur de protocole, quand ce dernier reçoit une requête du communicant spécifiant la quantité de données à envoyer et l'adresse de destination, il se configure en mode d'émission. Ce mode d'émission expliqué dans la section 2.4.1 du chapitre 2 consiste à effectuer plusieurs transferts de données de la mémoire locale du communicant vers la mémoire cache de l'adaptateur de protocole, en utilisant un DMA. La quantité de données récupérée pendant un transfert est inférieure ou égale à la taille de la mémoire cache WM_i, et le temps de transfert dépend du délai d'accès à la mémoire locale et du temps de récupération d'un bloc de données WD_i. Ainsi la période T exprimée par l'équation (3.10) est le nombre de transferts TR multiplié par le délai d'un transfert WD_i.

$$\begin{aligned} T &= TR * WDi \\ &= \frac{MQ_{ij} * WDi}{WM_i} \end{aligned} \tag{3.10}$$

Le nombre de paquets envoyés lors de la communication C_{ij} est la quantité maximale de données divisée par la largeur du lien Eq (3.11).

$$IP_{ij} = \frac{MQ_{ij}}{LW} \tag{3.11}$$

Le nombre de paquets envoyés sur le réseau OP_{ij} lors de la communication C_{ij} pendant une scrutation de la table TDMA, est le rapport entre le nombre de slots de temps $S_{ij} - 1$ attribués à cette communication et le nombre de slots de temps de la table TDMA S_i. Pour rapporter OP_{ij} à toute la période du transfert, nous le multiplions par la durée T Eq (3.12).

$$\begin{aligned} OP_{ij} &= \left(\frac{S_{ij}-1}{S_i}\right) * T \\ &= \left(\frac{S_{ij}-1}{S_i}\right) * \left(\frac{MQ_{ij} * WDi}{WM_i}\right) \\ &= \frac{(S_{ij}-1) * MQ_{ij} * WD_i}{S_i * WM_i} \end{aligned} \tag{3.12}$$

Ainsi, la taille de la FIFO en émission exprimée par l'équation (3.13) est la différence entre le nombre de paquets IP_{ij} transmis à l'interface réseau par l'adaptateur de protocole, et le nombre de paquets OP_{ij} envoyés sur le réseau par l'interface réseau. La multiplication de cette différence par la largeur de lien LW permet d'exprimer la valeur de la taille de la FIFO en *bits*.

$$\begin{aligned} FBS_{ij} &= (IP_{ij} - OP_{ij}) * LW \\ &= \left(\frac{MQ_{ij}}{LW} - \left(\frac{(S_{ij}-1) * MQ_{ij} * WD_i}{S_i * WM_i}\right)\right) * LW \\ &= MQ_{ij} * \left(1 - \left(\frac{(S_{ij}-1) * WD_i * LW}{S_i * WM_i}\right)\right) \end{aligned} \tag{3.13}$$

Le calcul de la taille de la mémoire tampon d'émission n'est possible que si la taille de la mémoire cache WM_i de l'adaptateur réseau respecte la condition de l'équation (3.14).

$$WM_i > \frac{(S_{ij}-1) * WD_i * LW}{S_i} \tag{3.14}$$

3.5.2 Méthode de calcul de la taille maximale des FIFOs de réception

Chaque communicant connecté au NoC reçoit différentes communications via l'interface réseau. Les données de chaque communication sont stockées dans l'interface réseau sur une FIFO de réception unique. Ces données, stockées dans les différentes FIFOs, sont transférées à la mémoire locale du communicant par l'adaptateur de protocole, en utilisant le mode de réception expliqué dans la section 2.4.2 du chapitre 2.

Dans ce mode, l'adaptateur de protocole transfère une quantité de données de l'une des FIFOs de l'interface réseau vers sa mémoire cache. Puis il transmet les données de sa mémoire cache à la mémoire locale du communicant en utilisant un DMA. La spécificité de ce mode de réception, impose à l'adaptateur de protocole de récupérer les données des FIFOs de l'interface réseau d'une manière répartie dans le temps. Ainsi l'adaptateur récupère les données de l'interface réseau en passant d'une FIFO de réception à une autre. À chaque passage, il lit une quantité de données équivalente ou inférieure à la taille de sa mémoire

La taille d'une FIFO de réception RBS_{ik}, attribuée à la communication C_{ik} dans une interface réseau NI_i, est la différence entre le nombre de paquets IP_k reçus pendant cette communication par l'interface réseau, et le nombre de paquets OP_k transmis de cette FIFO à l'adaptateur de protocole pendant une période D.

La durée D représente le temps que l'adaptateur de protocole met pour scruter toutes les FIFOs de réception, et envoyer les données à la mémoire locale. Cette durée, calculée avec l'équation (3.15), est le produit du nombre de FIFOs de réception NF_i, et le temps de transmission

WD_i d'une quantité de données équivalente à la taille de la mémoire cache WM_i de l'adaptateur réseau.

$$D = WD_i * NF_i \tag{3.15}$$

Le nombre de paquets OP_k, lus d'une FIFO de réception pendant la période D, est égale au nombre de paquets que nous pouvons stocker dans la mémoire cache (Eq.3.16).

$$OP_k = \frac{WM_i}{LW} \tag{3.16}$$

Par contre, le nombre de paquets IP_k, écrits dans cette FIFO, est proportionnel au nombre de slots de temps $S_{i'j} - 1$ réservés à la communication $C_{i'j}/C_{ik}$ dans la table TDMA de l'interface réseau d'émission $NI_{i'}$ par rapport au nombre de slots de temps $S_{i'}$ de cette table TDMA rapporté à la durée D. Ainsi l'équation (3.17) calcule le nombre de paquets reçus par une FIFO de réception pendant la période D.

$$\begin{aligned} IP_k &= \frac{S_{i'j} - 1}{S_{i'}} * D \\ &= \frac{(S_{i'j} - 1) * WD_i * NF_i}{S_{i'}} \end{aligned} \tag{3.17}$$

Ainsi, la taille de la FIFO de réception est la différence (Eq.3.18) entre le nombre de paquets écrits IP_k dans la FIFO et le nombre de paquets lus OP_k.

$$\begin{aligned} RBS_{ik} &= IP_k - OP_k \\ &= \frac{(S_{i'j} - 1) * WD_i * NF_i}{S_{i'}} - \frac{WM_i}{LW} \end{aligned} \tag{3.18}$$

L'équation (3.18) n'est possible que si la valeur de la taille de la mémoire cache WM_k respecte la condition de l'équation (3.19).

$$WM_k < \frac{WD_i * NF_i * LW * (S_{i'j} - 1)}{S_{i'}} \tag{3.19}$$

3.6 Méthodologie d'allocation des chemins spatio-temporels

Le NoC μSpider II véhicule des communications de type GT, en utilisant le principe de TDMA. Afin de satisfaire les contraintes de ces communications en bande passante, nous devons calculer pour chacune d'elle un chemin spatio-temporel. Le chemin spatio-temporel comporte le plus court chemin spatial reliant l'émetteur et le destinataire et l'instant de départ de la communication dans la table TDMA de l'interface réseau qui la gère.

Cette section présente la solution que nous avons élaboré avec Jean-Charles Créput, maître de conférence au laboratoire SET, lors de son séjour au Lab-STICC [CDRS10]. Cette approche repose sur un algorithme heuristique de type itératif expliqué dans la section 3.6.2. Cet algorithme utilise une construction parallèle gloutonne pour calculer les chemins spatio-temporels (section 3.6.1).

Algorithme 4: Procédure de la construction parallèle gloutonne

Fonction ConstructionGloutonne(S : **solution**) : **solution**
tant que *tous les chemins ne sont pas "construits" ou "bloqués"* **faire**
  Pour toutes les communications de toutes les interfaces réseaux
  pour *i=1* **a** M **faire**
    pour *j=1* **a** L_i **faire**
      Étape 1 :
      $V \leftarrow Path_{ij}\,[Length_{ij} - 1]$
      $t_{ij} \leftarrow Tdep_{ij} + Length_{ij} - 1$
      Étape 2 :
      si $E^V_{V'}[t_{ij}, \ldots, (t_{ij} + S_{ij})]$ *est libre et minimal* **alors**
        $Path_{ij}\,[Length_{ij}] \leftarrow V'$
        $Length_{ij} \leftarrow Length_{ij} + 1$
        $E^V_{V'}[t_{ij}, \ldots, (t_{ij} + S_{ij})] \leftarrow$ occupé
        si $V' = Dest_{ij}$ **alors**
          $Path_{ij} \leftarrow$ *"construit"*
        fin
      sinon
        $Path_{ij} \leftarrow$ *"bloqué"*
      fin
    fin
  fin
fin
Retourner S

3.6.1 Algorithme de construction des chemins spatio-temporels

La construction parallèle gloutonne permet de construire simultanément les chemins de toutes les communications, en suivant la cadence de la table d'ordonnancement TDMA. Ainsi, Cette construction ajoute un seul sommet dans les chemins de toutes les communications à un instant t, en prenant en compte la disponibilité temporelle des arcs et la contrainte du plus court chemin.

La gestion de la disponibilité des arcs dans la construction gloutonne est réalisée grâce à un graphe de marquage nécessaire pour mémoriser et présenter l'occupation des arcs à chaque slot de temps. Ce graphe de marquage consiste à associer un tableau de booléen de taille T à chaque arc du graphe. Ce tableau spécifie l'état d'occupation de l'arc à n'importe quel instant $t \; mod \; T$.

De plus, pour assurer une construction gloutonne avec les chemins les plus courts, nous allouons à chaque sommet du graphe $G(V, E)$ un tableau qui spécifie les distances minimales à parcourir pour atteindre toutes les destinations possibles. Ces tableaux sont construits une seule fois à l'initialisation, par l'application de l'algorithme du plus court chemin de *Ford-Fulkerson* [FF56].

La procédure de la construction gloutonne est présentée par l'algorithme 4. Cette procédure construit les chemins de toutes les communications en parallèle. Ainsi à chaque itération, tous les chemins vont inclure un seul nouveau sommet pour se rapprocher le plus rapidement possible de leurs destinations. Au cas où il est impossible d'inclure un nouveau sommet dans un chemin, ce

Algorithme 5: Procédures de modification des dates d'émission dans une table TDMA

Fonction PermutationMessages(S : **solution**) : **solution**
$i \leftarrow random(0, M)$ *Choix aléatoire de la NI*
$j \leftarrow random(0, L_i)$ *Choix aléatoire de la première communication*
$k \leftarrow random(0, L_i)$ *Choix aléatoire de la deuxième communication*
$permuter(Tdep_{ij}, Tdep_{ik})$ *Permutation des dates de départ prenant en compte la différence entre le nombre de paquets de chacune des deux communications*
Retourner S

Fonction TranslationDates(S : **solution**) : **solution**
$i \leftarrow random(0, M)$ *Choix aléatoire de la NI*
pour *j=1* **a** L_i **faire**
  $Tdep_{ij} \leftarrow (Tdep_{ij} + 1)\ mod\ T$ *Translation de 1 des dates de départ de toutes les communications de la même table TDMA*
fin
Retourner S

dernier est déclaré "bloqué". En revanche si le dernier sommet ajouté correspond à la destination, le chemin est déclaré "construit".

L'algorithme 4 s'exécute tant que les chemins de toutes les communications ne sont pas déclarés "bloqués" ou "construits". Ainsi, à chaque itération qui correspond à une unité de temps (slot), nous réalisons sur toutes les communications de chaque interface réseau du NoC un traitement qui inclut deux étapes :

1. La première étape consiste à se placer sur le dernier sommet V ajouté dans le chemin de la communication, et sur l'instant effectif courant t_{ij} de l'exécution de la fonction ;
2. La deuxième étape permet d'ajouter un nouveau sommet V' dans le chemin pour se rapprocher de la destination. Ainsi, elle vérifie pour tous les successeurs potentiellement utilisables, par lecture du graphe d'occupation, que l'arc $E^V_{V'}$ qui relie les sommets V et V' n'est pas occupé pendant l'intervalle de temps $[t_{ij}, \ldots, (t_{ij}+S_{ij})]$. Cet intervalle est calculé selon le nombre de paquets qui constitue la communication.
   Après la vérification, elle choisit l'un des successeurs selon la contrainte du plus court chemin. Au cas où il y a plusieurs sommets qui remplissent les même conditions, elle en choisit un au hasard. Ensuite, elle met à jour le statut de l'arc $E^V_{V'}$ dans le graphe d'occupation aux instants de passage des différents paquets du message $[t_{ij}, \ldots, (t_{ij} + S_{ij})]$.
   Si aucun successeur n'est choisi, le chemin est déclaré "bloqué". Par contre si le sommet successeur est identique à la destination on déclare le chemin "construit".

3.6.2 Algorithme heuristique de recherche des chemins spatio-temporels

La recherche des chemins avec la construction parallèle gloutonne nous contraint à attribuer aléatoirement les instants de départ aux différentes communications. Ces instants de départ sont compris dans l'intervalle de la table TDMA $[0, T-1]$, et l'écart entre les instants de départ de

deux communications successives de la même interface réseau dépend du nombre de paquets de la communication qui précède. Cette distribution aléatoire des instants de départ ne garantit pas la résolution du problème de recherche car certains chemins peuvent se bloquer lors de l'exécution de la construction gloutonne. Ainsi pour optimiser la recherche des chemins, nous introduisons une heuristique itérative qui exécute à chaque itération la construction gloutonne en changeant les instants de départ de certaines communications, puis elle compare la solution courante à la solution calculée dans l'itération précédente, et garde la meilleure.

Les deux opérateurs décrits par l'algorithme 5 sont utilisés pour modifier les instants de départ avant chaque nouvelle construction gloutonne des chemins. Le premier opérateur utilise la permutation de l'ordre de départ des messages de la même interface réseau. En revanche, le deuxième opérateur réalise une translation des dates de départ de la même table TDMA.

L'algorithme heuristique décrit dans le pseudo-code 6 consiste à exécuter deux étapes d'une manière répétée. La première étape dite de construction sert à trouver la meilleure solution en réitérant la procédure 4 de construction gloutonne avec une nouvelle initialisation. Cette initialisation effectue une permutation et une translation aléatoire des dates de départ de certaines communications. Ainsi, à chaque itération de cette première étape, nous construisons une nouvelle solution et nous la retenons comme meilleure solution si elle possède plus de chemins déclarés "construits" que la solution précédente.

Dans la deuxième étape, la meilleure solution obtenue lors de la première étape devient une source d'amélioration. Cette amélioration consiste à réitérer la procédure de réparation en choisissant un nombre de chemins à modifier tiré au hasard entre 0 et $NbrMg$, $NbrMg$ étant le nombre total de messages. Cette réparation consiste à supprimer un nombre déterminé de chemins déclarés "construits" ainsi que tous les chemins déclarés "bloqués", et elle met à jour le graphe d'occupation selon les suppressions effectuées, puis elle reconstruit une nouvelle solution. Cette étape a pour but de modifier localement la solution courante considérant un voisinage plus au moins large de cette solution.

Le nombre d'itérations de l'heuristique, et des deux étapes de construction et d'amélioration sont configurables et il sont choisis par le concepteur du NoC. Ainsi, nous offrons à l'architecte la possibilité de modifier les paramètres de l'algorithme afin de lui permettre de trouver un compromis entre le temps d'exécution et la qualité de la solution proposée.

3.7 Conclusion

La conception d'un réseau qui garanti le trafic nécessite le dimensionnement des différents éléments du NoC. Dans ce chapitre nous avons défini le flot de conception, ainsi que les différents algorithmes et méthodes qui permettent un bon dimensionnement de notre réseau μSpiderII.

Notre flot de conception introduit plusieurs nouvelles méthodes, nous distinguons :

- l'utilisation des graphes de dépendance des communications, pour modéliser les contraintes de l'application en terme de bande passante, et pour représenter d'une manière claire les dépendances entre les processus de traitements et d'échanges. Cette représentation nous a permis de différencier les échanges exécutés séquentiellement et ceux exécutés en parallèles. Ceci nous permet de calculer avec précision la valeur de la bande passante des différentes communication ;
- l'introduction d'un nouvel algorithme pour construire des topologies dédiées. Cette construction repose essentiellement sur le principe de proximité qui permet de réduire la latence. De plus, nous avons développé plusieurs méthodes de construction. Ainsi l'algorithme choisit l'une de ces méthodes selon la classe de l'architecture multiprocesseur. Ensuite, il optimise

Algorithme 6: Fonctions de construction et de réparation d'une solution et l'heuristique itérative

Fonction Construction(S : **solution**) : **solution**
Initialisation du graphe d'occupation
$S \leftarrow Init(S)$
Permutation et translation aléatoire des dates de départ $S \leftarrow PermutationMessages(S)$
$S \leftarrow TranslationDates(S)$
Lancement d'une construction parallèle gloutonne
$S \leftarrow ConstructionGloutonne(S)$
Retourner S

Fonction Réparation(S : **solution**, NbrMg : **entier**) : **solution**
Suppression de $NbrMg$ *chemins aléatoirement et mise à jour du graphe d'occupation selon les chemins supprimés*
$S \leftarrow RemovePaths(S, NbrMg)$
Suppression des chemins déclarés "bloqués" dans la solution S *et mise à jour du graphe d'occupation selon les chemins supprimés*
$S \leftarrow RemovePathsBlocked(S)$
$S \leftarrow ConstructionGloutonne(S)$
Retourner S

Fonction Heuristique() : Solution
$Best_1, Best_2, Best_3, S : Solution$
$i \leftarrow 0$
$S \leftarrow Init(S)$
$Best_1 \leftarrow null$
tant que $i < NbIteration1$ **faire**
  Étape de construction
  $j \leftarrow 0$
  $Best_2 \leftarrow null$
  tant que $j < NbIteration2$ **faire**
    $S \leftarrow$ **Construction**(S)
    si S *est meilleur que* $Best_2$ **alors**
      $Best_2 \leftarrow S$
    fin
    $j \leftarrow j + 1$
  fin
  Étape d'amélioration
  $j \leftarrow 0$
  $S \leftarrow Best_2$
  $Best_3 \leftarrow null$
  tant que $j < NbIteration3$ **faire**
    $S \leftarrow$ **Réparation**$(S, Random(0, NbrMg))$
    si S *est meilleur que* $Best_3$ **alors**
      $Best_3 \leftarrow S$
    fin
    $j \leftarrow j + 1$
  fin
  si $Best_3$ *est meilleur que* $Best_1$ **alors**
    $Best_1 \leftarrow Best_3$
  fin
  $i \leftarrow i + 1$
fin
Retourner $Best_1$

la topologie générée par l'une des méthodes en utilisant une approche gloutonne avec un critère de choix qui est le taux d'utilisation des liens et des routeurs ;
- le calcul de la taille maximale des mémoires tampons en émission et en réception ;
- le calcul du nombre de slots de temps nécessaire pour garantir la bande passante de chaque communication ;
- et le développement d'un algorithme heuristique pour la recherche des chemins spatio-temporels. Cet algorithme repose sur une méthode de construction des chemins parallèle et gloutonne.

Partie II : Réseaux sur puce reconfigurables : définitions, protocoles et architectures

4

Les réseaux sur puce adaptatifs

L'évolution technologique des ces dernières années a permis aux concepteurs d'élaborer des systèmes embarqués dynamiques, en exploitant notamment la flexibilité offerte par les FPGAs. Ce nouveau niveau d'abstraction qui introduit la reconfigurabilité dans les systèmes sur puce présente de nouvelles contraintes pour les réseaux d'interconnexions. Par ailleurs, différentes évolutions des systèmes embarqués imposent un comportement dynamique difficile à traiter hors ligne sans surdimensionnement. Il s'agit principalement de la multiplicité des applications, des différents modes de fonctionnement (ex : codecs multimédia, radio-logicielle) ou de l'optimisation des applications dont les traitements dépendent des données. Ainsi, pour interconnecter les communicants d'un système sur puce reconfigurable (RSoC [38]), une solution consiste à proposer un nouveau type de NoC, qui prend en compte les changements des contraintes des communications en utilisant le principe de la reconfiguration dynamique.

Cette nouvelle génération de NoC dits réseaux sur puce reconfigurables (RNoC [39]), a déjà été utilisée par certaines équipes de recherches pour résoudre d'autres problèmes. Par exemple, l'article [LV05] explique que l'utilisation d'un RNoC permet de gérer le replacement des différents IPs afin de distribuer efficacement l'échauffement thermique sur toute la surface du composant. D'autres équipes utilisent les RNoCs pour améliorer l'usage des ressources [NMV04, PKA06, HKHT05], ou changer en temps réel les protocoles et les mécanismes du RNoC [BHB+07].

Dans ce chapitre, nous définissions un modèle de conception pour les RNoCs qui repose sur un mécanisme de reconfiguration dynamique expliqué dans la section 4.1. Ce modèle met en lumière trois axes de recherche fondamentaux pour le développement des RNoCs, l'administration 4.2.1, la reconfiguration de l'architecture 4.2.2, et la reconfiguration des protocoles 4.2.3.

4.1 Le modèle de reconfiguration dynamique

La reconfiguration dynamique est un mécanisme de contrôle qui transforme et adapte les paramètres matériels et/ou logiciels d'un environnement à de nouvelles conditions de fonctionnement, sans aucune interruption. Aussi, la reconfiguration dynamique offre aux concepteurs un nouveau niveau d'abstraction, en passant d'un niveau de disponibilité à un niveau de service par le biais de la mise en œuvre en ligne. La figure 4.1 illustre le principe de la reconfiguration dynamique, définit par Kramer [KM85, KM90], ce principe s'organise autour de trois procédés complémentaires :

1. *La validation :* chaque système comporte des spécifications composées de paramètres lo-

38. Reconfigurable System-on-Chip
39. Reconfigurable Network on Chip

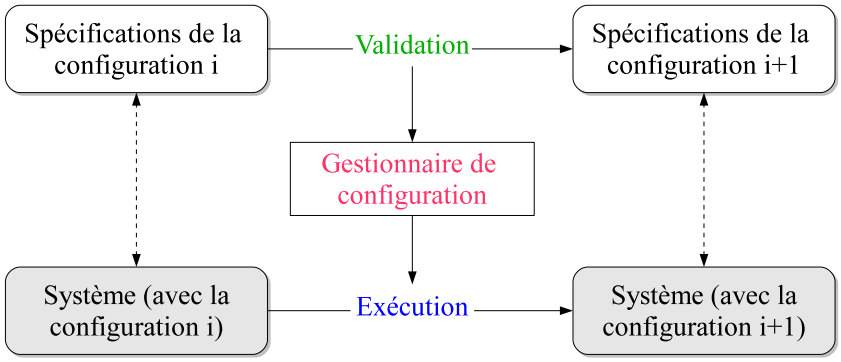


FIGURE 4.1 – Le mécanisme de reconfiguration dynamique selon Kramer [KM85, KM90]

giques et d'une structure physique. Par conséquent, sa reconfiguration se présente sous forme de changements des spécifications, tels que l'ajout de nouveaux composants, la modification des composants existants, et la suppression d'anciens éléments. L'étape de validation assure que les modifications apportées par les nouvelles spécifications sont compatibles avec l'architecture logique et physique du système ;

2. *La configuration :* le gestionnaire de configuration traduit les changements valides de la nouvelle spécification en commandes, puis il transfert ces commandes au système d'exploitation pour modifier le système en cours. Cette traduction nécessite la connaissance de l'état du système, par conséquent les informations nécessaires peuvent être obtenues à partir d'une base de données gérée par le gestionnaire de configuration, et alimentée en permanence par les nouveaux états des différents composants du système ;
3. *L'exécution :* le système d'exploitation compile, installe, et configure la nouvelle spécification sur le système, selon les commandes envoyées par le gestionnaire de configuration.

Ce mécanisme présente le processus de la reconfiguration dans une approche qui n'intègre pas la prise de décision, car il considère que la décision dépend de la stratégie du concepteur pour la mise en œuvre de la reconfiguration. Alors, pour compléter le précèdent mécanisme de la reconfiguration dynamique nous lui ajoutons un nouveau procédé en intégrant la décision.

Ainsi, le mécanisme que nous utilisons pour l'intégration de la reconfiguration dynamique dans les RNoC repose sur une approche où le gestionnaire de configuration décide, valide, et construit une nouvelle configuration, en utilisant des métriques choisies par le concepteur, et/ou les variations de l'état du système.

4.2 Méthodologie de conception des réseaux sur puce reconfigurables

La partie gauche de la figure 4.2 présente la segmentation de l'espace de conception d'un NoC en cinq couches, la couche application, la couche transport, la couche réseau, la couche liaison de données, et la couche physique. Cette segmentation est basée sur le modèle OSI que nous avons expliqué dans le chapitre 1. Le modèle OSI ne précise pas réellement les services et les protocoles à utiliser pour chaque couche, mais il décrit néanmoins ce que les niveaux d'abstraction doivent réaliser.

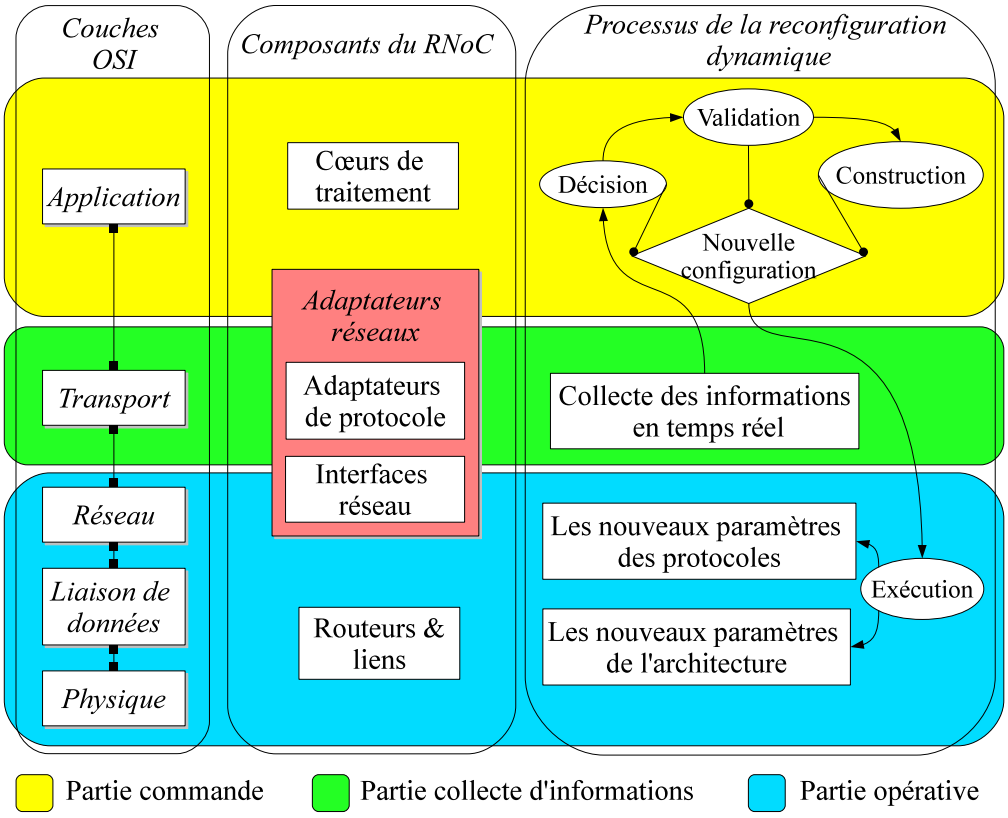


FIGURE 4.2 – Modélisation du mécanisme de la reconfiguration dynamique dans les RNoCs en corrélation avec les couches du modèle OSI et les composants du réseau

La figure 4.2 illustre notre répartition des procédés du mécanisme de la reconfiguration dynamique sur les composants d'un RNoC. Cette répartition réalisée en corrélation avec les couches du modèle OSI, nous a permis de définir trois parties essentielles pour la conception d'un RNoC.

La partie commande de la reconfiguration des RNoCs est définie dans la couche application. Cette partie doit avoir une connaissance exacte de la structure de l'architecture et des paramètres des protocoles du réseau, et elle doit posséder également les outils nécessaires pour prendre la décision de changer une configuration, de valider cette décision, puis de construire les paramètres de la nouvelle configuration. Cette partie est souvent implantée comme un bloc logiciel et exécutée par l'un des cœurs de traitement connectés au RNoC. Mais elle peut aussi être implantée comme un bloc matériel sur les adaptateurs réseaux.

La partie collecte d'informations réunit en temps réel des statistiques sur le nombre de messages transmis, consommés, bloqués ou rejetés, puis elle transmet ces informations à la partie commande pour qu'elle puisse décider de la mise en œuvre d'une reconfiguration. Cette partie est définie dans la couche transport, et elle est implantée exclusivement sur les interfaces réseaux, car elles représentent les points d'entrée et de sortie des paquets transmis sur le RNoC.

La partie opérative réalise la nouvelle configuration en modifiant les paramètres des protocoles et de l'architecture des différents composants (interfaces réseaux, routeurs, et liens) du RNoC. Ces modifications agissent sur trois couches du modèle OSI. Au niveau de la couche réseau, la partie opérative réalise des modifications sur la structure de la topologie, les mécanismes de routage, et les techniques de commutation. Au niveau de la couche liaison de données, elle

modifie le protocole d'échange de données entre les routeurs, la technique de multiplexage et la technique de détection et correction d'erreurs.

Notre modélisation de l'espace de conception des RNoCs est fondée sur les relations entre les couches du modèle OSI, les composants du RNoC, et le mécanisme de reconfiguration dynamique. Ce nouveau modèle révèle trois axes de recherche fondamentaux pour la conception des RNoC. L'axe d'administration intégrant les parties commande et collecte d'informations, cet axe détermine les stratégies de décision et de validation des configurations. Et les axes de reconfiguration de l'architecture et des protocoles du RNoC, qui déterminent les métriques et les méthodes utilisées pour réaliser des changements dynamiques sur la structure ou les protocoles du RNoC selon l'évolution des contraintes des communications.

4.2.1 L'administration de la reconfiguration dynamique

L'administration consiste à récolter des informations qui peuvent être des statistiques, comme le nombre de paquets bloqués au sein des différentes interfaces réseaux du RNoC. Grâce à ces données, elle décide et valide la nouvelle configuration qui permet de résoudre un problème dans le réseau, comme la congestion par exemple. Cette administration est managée par le gestionnaire de la reconfiguration dynamique.

Pour garantir une mise en œuvre optimale et efficace de l'administration, le gestionnaire doit s'assurer que son déploiement respecte les points suivants :

- **la garantie de la cohérence et de la qualité de service** du RNoC. Ainsi le gestionnaire doit s'assurer que la nouvelle configuration est compatible avec l'architecture et les protocoles du RNoC, et que les changements introduits par celle-ci n'ont pas d'impact négatif sur les performances du NoC ;
- **la décentralisation de l'administration**. Ceci implique une délégation d'une partie de l'administration à certains composants du réseau, afin de réduire les infrastructures utilisées pour la gestion de la reconfiguration et les temps nécessaires pour le déploiement d'une configuration ;
- **la gestion de l'évolution matérielle ou logicielle** du RNoC. Ainsi le gestionnaire doit avoir une connaissance exacte de l'infrastructure du réseau ;
- et **la minimisation du temps de la reconfiguration**. Ainsi le déploiement de la nouvelle configuration doit monopoliser le minimum d'éléments pour assurer la continuité des transferts qui ne sont pas concernés par cette nouvelle configuration.

La plupart des modèles de conception des RNoC n'intègrent aucune stratégie d'administration, à l'exception de l'approche définie par Nollet [NMV04, NMA+05]. Cette approche ajoute au NoC un système d'exploitation (OS [40]) qui optimise l'allocation des ressources et minimise le blocage des paquets dans le réseau. La stratégie de reconfiguration de cette approche repose sur les statistiques calculées selon le nombre des paquets bloqués dans toutes les interfaces réseaux. Et grâce à ces statistiques, l'OS adapte le temps d'émission des différents communicants, et configure dynamiquement le routage de certains paquets. L'application de cette stratégie demande l'utilisation de deux réseaux, un réseau de données pour transporter les données entre les différentes communicants et un réseau de contrôle qui permet à l'OS de récolter les informations sur l'état de chaque interface réseau, et d'envoyer les nouvelles configurations sans qu'elles soient bloquées par les paquets de données.

Cette solution présente des limites car l'adaptation dynamique du routage n'est pas réalisée en temps réel et elle exige l'arrêt total de toutes communications, ce qui rend le réseau inexploitable

40. Operating System

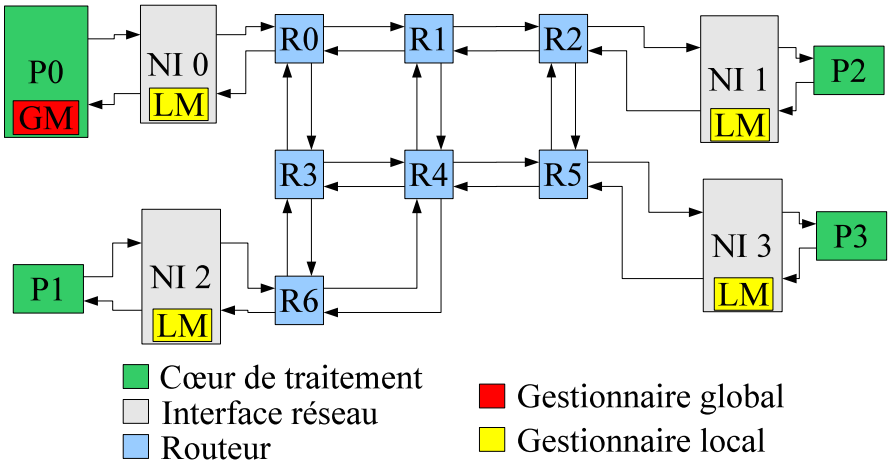


FIGURE 4.3 – Distribution de l'administration sur le gestionnaire global (GM) et les gestionnaires locaux (LMs)

pendant toute la période de reconfiguration. De plus, l'intégration de cette solution implique un surcoût significatif du réseau, car elle utilise deux réseaux différents, un réseau pour transporter les données, et un autre pour le contrôle.

L'administration de la reconfiguration dynamique est efficace quand elle respecte les règles de cohérence et de délégation, gère l'évolution de l'infrastructure et réduit le temps d'exécution. Notre approche intègre l'administration en la distribuant entre un seul gestionnaire global (GM [41]) et plusieurs gestionnaires locaux (LM [42]). Cette délégation ou répartition de l'administration permet aux gestionnaires locaux de décider, valider, construire et exécuter une reconfiguration partielle selon les règles établies par le gestionnaire global. En plus, la délégation de l'administration permet de définir deux niveaux de reconfiguration, le premier que nous intégrons dans le LM réalise des modifications locales simples et très rapides à exécuter. Et le deuxième niveau que nous incorporons dans le GM, effectue des changements globaux nécessitant une connaissance exacte des paramètres du RNoC et un algorithme de décision complexe et couteux.

Comme le montre la figure 4.3, le gestionnaire local est un module matériel implanté sur chacune des interfaces réseaux du RNoC, son rôle est d'administrer la reconfiguration de l'infrastructure et des protocoles d'une seule interface réseau. La plupart des décisions à ce niveau concerne la reconfiguration de la profondeur des mémoires tampons et la redistribution des slots de temps de la table TDMA sur les différentes communications. Cette gestion locale a pour but une prise de décision et une exécution rapide de la nouvelle configuration sans introduire une augmentation de la surface de l'interface réseau.

En revanche, le gestionnaire global a une vue complète sur le réseau grâce aux messages reçus des différents gestionnaires locaux. Ces messages contiennent des informations sur l'état des communications gérées par les différentes interfaces réseaux. À partir de ces informations, le GM peut redéfinir certaines règles des gestionnaires locaux ou décider de réinitialiser le réseau, en déployant une nouvelle configuration qui changera par exemple la répartition spatio-temporelle des communications. Le GM intègre des stratégies de reconfiguration avec des algorithmes complexes, alors son intégration dans le RNoC comme module matériel aura un impact considérable

41. Global Manager
42. Local Manager

sur la surface du réseau. Pour ces raisons nous intégrons le GM comme un module logiciel sur l'un des cœurs de traitements connectés au RNoC, où on peut le considérer comme un nouveau service du système d'exploitation gérant la reconfiguration dynamique du RNoC.

4.2.2 La reconfiguration dynamique de l'architecture

L'ajout ou la suppression de modules matériels dans un système multicœur reconfigurable, et l'évolution des bandes passantes de communications durant l'exécution, ajoute de nouvelles contraintes à supporter par le réseau d'interconnexion. Ainsi, le réseau sur puce doit se conformer à ces nouvelles contraintes en utilisant entre autres une architecture dynamique qui offre une adaptabilité au niveau de la structure de sa topologie et la profondeur de ces mémoires tampons.

La majorité des travaux qui traitent de la reconfiguration de l'architecture des RNoC, tels que [BAM+05, PKA06, **?**], ciblent l'adaptation de la topologie du réseau. Parmi les solutions proposées, Pionteck [PKA06, PAK06] définit une méthodologie d'implantation et d'adaptation en temps réel de la structure du réseau sur une architecture FPGA partiellement reconfigurable. En effet le réseau CoNoChi [43] est conçu pour adapter, structurer, et ajouter ou supprimer des routeurs dans la topologie selon les modifications apportées à l'architecture du RSoC. Cependant, cette solution est limitée, car la reconfiguration du réseau dépend fortement de la structure de la plateforme FPGA et de ses capacités à la reconfiguration partielle. Et elle exige l'utilisation d'un hôte pour le calcul des nouvelles tables de routage ainsi que l'arrêt de toutes communications pendant la reconfiguration.

La stratégie d'adaptation de l'architecture des RNoCs en ajoutant ou supprimant des routeurs dans la topologie du réseau, requiert l'utilisation de plateforme adaptée à la reconfiguration partielle, tel que les FPGAs de Xilinx. De plus, son implantation génère une augmentation considérable des ressources.

D'autres travaux, comme l'approche proposée par Stensgaard [SS08] utilisent un routeur programmable qui combine la commutation de paquets et la commutation de circuits pour modifier la structure de la topologie.

De notre point de vue, l'adaptation de la topologie du RNoC ne doit pas rester la seule voie d'investigation à explorer pour reconfigurer dynamiquement l'architecture du RNoC. La reconfiguration de la profondeur des mémoires tampons dans les différents composants du RNoC représente une opportunité pour réduire la surface et la consommation du RNoC. Nous constatons aussi que si plusieurs méthodes existent pour le calcul de la taille optimale des mémoires tampons pendant le flot de conception du réseau, aucune solution n'a été proposée pour l'adaptation dynamique de cette taille pendant l'exécution.

4.2.3 La reconfiguration dynamique des protocoles

Le but d'un RNoC est d'échanger des données entre différents cœurs de traitement en utilisant des communications avec différents niveaux de qualité de service. Pour chaque niveau de qualité de service, le RNoC doit définir et intégrer des protocoles de réseau spécifiques garantissant les besoins de ce niveau. Ces protocoles déterminent entre autres la technique de commutation, le mécanisme de routage, et la méthode de gestion du flot de données. Ainsi la reconfiguration dynamique des protocoles dans les RNoCs devient capitale, dans le cas où la qualité de service ou bien les contraintes des communications évoluent dans le temps. En pratique, la reconfigu-

43. Configurable Network-on-Chip

ration des protocoles se traduit par une nouvelle distribution des slots de temps dans les tables d'ordonnancement et le calcul des nouveaux chemins pour éviter les conflits dans le réseau.

Plusieurs études s'intéressent à cette problématique, telles que l'approche proposée par Hansson [HCG07] qui définit un modèle pour la reconfiguration partielle du réseau en utilisant un algorithme qui calcule les paramètres de plusieurs versions du RNoC, et chaque version correspond aux contraintes d'une application donnée. Certes, cette méthode permet de reconfigurer les paramètres du RNoC selon l'application exécutée, pour obtenir le niveau de qualité de service souhaité. Mais elle reste limitée, car les applications sont statiques, leurs contraintes de communications sont connues, les différentes versions qui reconfigurent le RNoC sont pré-calculées et ne prennent pas en compte les changements en temps réel. Une autre approche développée par Marescaux [MBD+05] est basée sur une heuristique calculant des nouvelles distributions spatio-temporelles des slots de temps et des chemins pour les différentes communications. Cette heuristique satisfait les contraintes temps réel mais elle n'est applicable que pour les RNoC avec une topologie régulière de type grille deux dimensions.

Pour appliquer une adaptation dynamique sur les protocoles d'un RNoC, presque toutes les solutions étudiées requièrent une connaissance exacte des contraintes applicatives. Actuellement, nous déterminons deux types de scénarios pour réaliser une reconfiguration au niveau des protocoles. Le premier correspond à une prise de décision basée sur le mode de fonctionnement. Dans ce cas, le passage d'une application à une autre se traduit par la reconfiguration du RNoC car, chaque application définit un ensemble de communications avec un niveau de qualité de service spécifique à la nature de cette application. Ce changement de contraintes impose le calcul des nouveaux paramètres du RNoC ainsi que l'arrêt de toutes communications pendant la reconfiguration.

Le deuxième scénario correspond aux changements des contraintes de communications pour la même application. Ces changements sont répétitifs et aléatoires. Ce nouvel aspect ouvre d'autres voies dans le domaine de reconfiguration dynamique des protocoles, et il demande le développement de nouvelles méthodologies pour adapter certains paramètres du RNoC en limitant le surcoût des ressources utilisées et en évitant l'arrêt complet du réseau.

4.3 Conclusion

Dans ce chapitre, nous avons commencé par définir un espace de conception pour les RNoCs en créant un nouveau modèle qui lie le mécanisme de reconfiguration dynamique, les paramètres des composants réseau, et les protocoles des couches OSI. Ce modèle a révélé trois parties essentielles pour l'intégration de la reconfiguration dans les NoCs. La partie commande, la partie récolte d'informations, et la partie opérative. De plus, grâce à la corrélation définie dans ce modèle, nous avons mis en lumière trois axes de recherches fondamentaux dans le domaine de la conception des RNoCs : l'axe d'administration, et les axes de reconfiguration de l'architecture et des protocoles. Nous avons étudié chacun des axes en définissant son usage et sa place dans le mécanisme de la reconfiguration dynamique dans les RNoC, en réalisant une analyse préalable des travaux qui traitent certaines de ces problématiques, et en révélant plusieurs points pertinents dont le potentiel reste à explorer.

Ce chapitre met l'accent sur la gestion de l'administration de la reconfiguration, car sa mise en œuvre conditionne le bon fonctionnement et les performances du RNoC. L'administration que nous avons défini repose sur une gestion partagée entre un gestionnaire global (GM) et plusieurs gestionnaires locaux (LMs). Nous intégrons le LM à chacune des interfaces réseau, dans le but de réaliser des modifications locales simples et très rapides à exécuter. Et nous incorporons le GM

à l'un des cœurs de traitement connecté au RNoC, son rôle consiste à effectuer des changements globaux en mettant à jour les configurations gérées par les LMs.

5

Architecture et protocoles de la version reconfigurable du réseau μSpider II

Le chapitre précédent étudie les réseaux sur puce reconfigurables dynamiquement en réalisant un état de l'art sur les différentes méthodes, qui conçoivent ce type de NoC pour satisfaire de nouvelles contraintes dues aux systèmes sur puce adaptatifs. Cette étude nous a permis de définir un modèle de reconfiguration dynamique pour les NoCs, et de mettre la lumière sur des nouveaux mécanismes que nous pouvons intégrer à notre NoC μSpider II pour le rendre adaptatif, et améliorer ses performances. Dans ce chapitre, nous définissons ces mécanismes et la procédure que nous appliquons pour les intégrer au NoC μSpider II. La section 5.1 présente l'approche des mémoires tampons configurables dynamiquement qui permet d'adapter la profondeur des FIFOs dans les interfaces réseau en temps réel et selon les besoins des communications. Et la section 5.2 introduit un nouveau mécanisme qui permet de configurer dynamiquement la table d'ordonnancement "TDMA", en adaptant le nombre d'intervalles de temps alloué aux communications selon les besoins.

Ce chapitre explique également la méthode que nous utilisons pour administrer et gérer ces mécanismes de reconfiguration dynamique. Cette administration repose sur l'approche que nous avons développé dans le chapitre précédent, et qui permet de distribuer la gestion de la RD [44] sur un gestionnaire global et plusieurs gestionnaires locaux. La section 5.3 présente ainsi le gestionnaire local que nous intégrons aux interfaces réseau du NoC, et la section 5.4 définit le rôle de gestionnaire global et sa mise en œuvre.

5.1 Mémoires tampons configurables dynamiquement

Le rôle principal des mémoires tampons dans le réseau μSpider II est de stocker les paquets des différentes communications émises ou reçues par les interfaces réseaux afin de répondre aux contraintes de bande passante sans perte de paquets. Ceci rend l'utilisation des mémoires tampons inévitable pour notre réseau car il est conçu pour véhiculer des trafics de type GT.

Le nombre des mémoires tampons dans le réseau dépend du nombre de communications, car chaque communication a besoin de deux mémoires tampons, une dans l'interface réseau d'émission et une autre dans l'interface réseau de réception. Et la taille des tampons mémoires dépend de plusieurs paramètres mais principalement de la bande passante des communications GT, les procédures de dimensionnement des tailles des mémoires tampons en émission et en réception sont respectivement détaillées dans les sections 3.5.1 et 3.5.2 du chapitre 3.

44. Reconfiguration Dynamique

La taille de notre réseau µSpider reste importante malgré les optimisations réalisées au niveau des routeurs. Cette surface est due principalement au dimensionnement au pire cas des tampons mémoires imposés par les trafics de type GT, et à l'effet proportionnel de la taille des mémoires tampons sur la surface du NoC. Afin de relever le challenge que constitue la réduction de la surface de notre NoC sans détériorer les performances, nous avons introduit le principe de la reconfiguration dynamique sur les mémoires tampons dans les interfaces réseau.

5.1.1 Principe

L'introduction du mécanisme de la reconfiguration dynamique au niveau des mémoires tampons réside dans le changement en temps réel de leurs profondeurs. Cette idée a émergé de l'analyse que nous avons réalisé sur le taux de remplissage des mémoires tampons dans les NoCs de type GT. Cette étude a montré que la majorité des mémoires tampons ne sont jamais pleines au même moment. Elles sont en effet dimensionnées au pire cas, en considérant un schéma de communications parallèle où le débit d'écriture sur les mémoires tampons est égal au débit de lecture. Or, les mémoires tampons dans les NIs ont un rôle de découplage nécessaire pour masquer les dépendances qui en pratique interdisent ce schéma parfait. Donc, la taille allouée est supérieure ou égale au besoin, ce qui implique systématiquement des emplacements non occupés.

Par conséquent, nous utilisons le principe d'allocation dynamique de l'espace mémoire soit pour réduire la taille globale des mémoires tampons, soit pour améliorer les performances en optimisant la disponibilité de l'espace mémoire. Ce principe d'allocation consiste à placer les mémoires tampons d'une même interface réseau dans le même espace mémoire. Ainsi chaque mémoire tampon peut s'étendre sur les mémoires tampons adjacentes et de cette manière elle augmente sa capacité de stockage.

Pour mieux comprendre ce principe, nous l'illustrons sur la figure 5.1. La partie supérieure de cette figure représente l'organisation classique des mémoires tampons dans une interface réseau. Chaque mémoire tampon est une FIFO circulaire utilisant deux pointeurs, le premier *"Push"* indique l'adresse de l'emplacement qui recevra la prochaine donnée, il est incrémenté à chaque écriture, et il est remis à 0 lorsque $Push = Taille$. Et le deuxième pointeur *"Pop"* spécifie l'adresse de l'emplacement où est stockée la donnée à lire, il est incrémenté à chaque lecture et il est remis à 0 si $Pop = Taille$. Chaque FIFO possède aussi deux variables pour la gestion d'accès, la première définit la taille de la FIFO, et la deuxième *"count"* caractérise le nombre de données stockées dans la FIFO. Par conséquent si $count = 0$ la FIFO est vide, en revanche si $count = Taille - 1$ la FIFO est pleine.

Avec cette organisation classique, chaque mémoire tampon est gérée séparément, et les adresses des emplacements de toutes les FIFOs sont indexées distinctement. En revanche, dans la nouvelle organisation des tampons mémoires qui permet d'introduire le principe d'allocation dynamique illustré dans la partie inférieure de la figure 5.1 , nous plaçons les FIFOs dans le même espace mémoire et nous utilisons un indexage unique pour adresser les emplacements de toutes les FIFOs. Pour délimiter la zone de chaque FIFO, nous avons créé deux nouvelles variables, la variable *"min"* indique l'adresse du premier emplacement de la FIFO, et la variable *"max"* spécifie l'adresse du dernier emplacement de la FIFO. La gestion des FIFOs dans la nouvelle organisation reste la même, ainsi, une FIFO est pleine lorsque $count > max - min$, en revanche elle est vide lorsque $count = 0$. Comme les FIFO sont circulaires, les pointeurs d'écriture et de lecture sont initialisés en prenant la valeur de la variable min lorqu'ils sont égaux à $max + 1$.

Grâce à cette nouvelle organisation, chaque FIFO peut augmenter sa capacité de stockage en s'étendant sur les FIFOs adjacentes. Cette capacité d'adaptation est réalisée à l'aide des variables

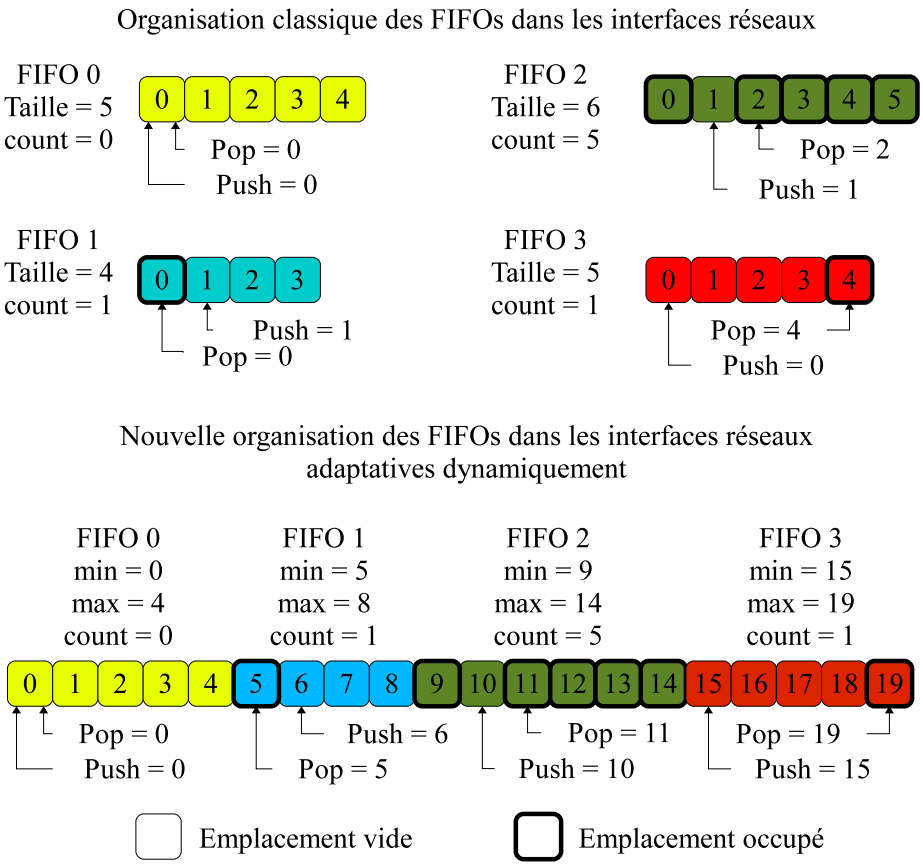


FIGURE 5.1 – Organisation des mémoires tampons dans les interfaces réseaux adaptatives dynamiquement

"min" et *"max"* ajoutées pour délimiter l'espace des FIFOs. Par exemple, pour que la $FIFO_0$ s'étende à droite sur la $FIFO_1$ en augmentant son nombre d'emplacements de 1, on incrémente simultanément de 1 le max_0 de la $FIFO_0$, et min_1 de la $FIFO_1$

La nouvelle configuration des FIFOs dans les interfaces réseau permet d'utiliser une seule mémoire physique. Dans cette mémoire chaque FIFO occupe une place selon sa taille, et elle s'étend sur les FIFOs voisines. Le succès de ce mécanisme dépend fortement de l'emplacement des FIFOs les unes par rapport aux autres. Ce placement est déterminé pendant la phase du dimensionnement des FIFOs en utilisant une méthode de déduction. Cette méthode décide que deux FIFOs sont voisines quand le taux de remplissage de l'une des deux FIFOs ne dépasse pas 25% et à n'importe quel instant de l'exécution, ainsi on garantit l'extension de l'une des deux FIFOs sur l'autre en cas de besoins. Pour réaliser ce placement, la méthode de déduction utilise les dépendances des communications modélisées par les graphes CDGs.

Afin de comparer ces deux organisations des mémoires tampons, nous avons réalisé différentes implantations du contrôleur des FIFOs dans une interface réseau en faisant varier deux paramètres. Le premier paramètre concerne la taille totale $T * 32$ *bits* de l'espace mémoire utilisé pour les différentes FIFOs, où T est le nombre de mots qui constituent cet espace mémoire. Le deuxième paramètre correspond au nombre de FIFOs N qui se partagent équitablement cet espace mémoire. Pour cette expérience, nous utilisons le même environnement paramétré précé-

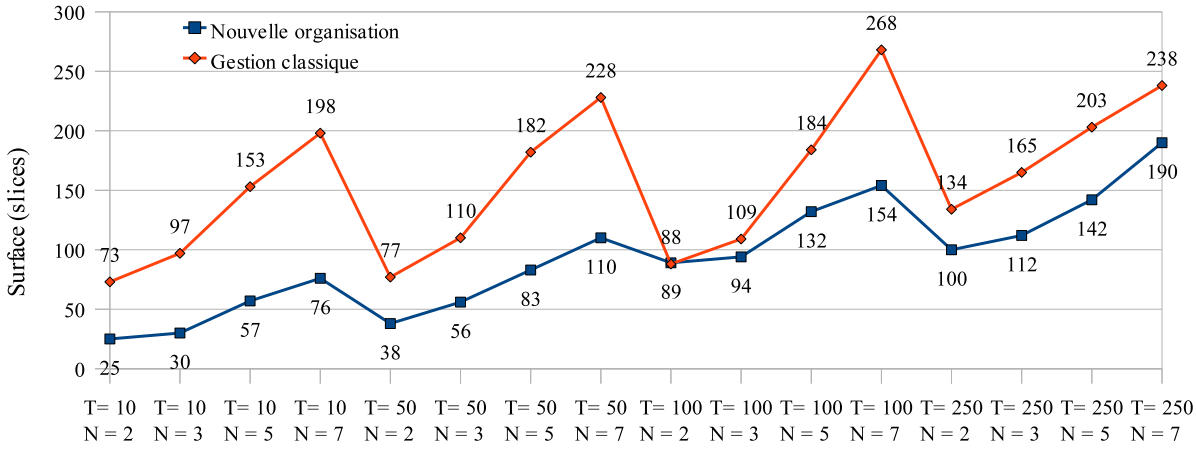


FIGURE 5.2 – Surface du contrôleur des FIFOs pour les deux méthodes d'organisation de la mémoire dans les NIs (avec "N" le nombre de FIFOs gérées et "T" la taille de l'espace mémoire)

demment dans la section 2.2.3 du chapitre 2.

La courbe de couleur rouge illustrée sur la figure 5.2 représente les résultats de la taille du contrôleur des FIFOs en utilisant la méthode d'organisation classique des FIFOs. En revanche, la courbe de couleur bleu décrit les résultats de la taille du nouveau contrôleur utilisant la nouvelle méthode d'organisation des FIFOs. Grâce à ces résultats, nous pouvons affirmer que l'utilisation de la nouvelle organisation des FIFOs réduit la surface du contrôleur des mémoires tampons avec une moyenne de 41%. Le gain obtenu dépend principalement de la taille de l'espace mémoire et du nombre de FIFOs. Nous remarquons aussi que le gain est plus grand quand le nombre de FIFOs à gérer est important. En revanche, quand la taille de l'espace mémoire augmente, le gain obtenu pour le même nombre de FIFOs diminue.

La nouvelle organisation des FIFOs dans les interfaces réseaux nous permet d'introduire le mécanisme d'allocation dynamique de la mémoire en reconfigurant la tailles des différentes FIFOs. Mais elle permet également de réduire la surface du contrôleur des mémoires tampons.

5.1.2 Règles et méthode de fonctionnement

L'allocation dynamique des FIFOs dans les interfaces réseau est fondée sur la reconfiguration de leurs tailles, ainsi chaque FIFO peut changer sa profondeur en profitant de l'espace offert par les FIFOs adjacentes. Cette allocation dynamique doit respecter les sept règles suivantes :

1. Chaque FIFO ne doit gagner ou perdre qu'un seul emplacement à la fois ;
2. La durée de la reconfiguration ne doit pas dépasser un cycle ;
3. Une FIFO peut s'étendre à gauche ou à droite, sauf la première FIFO qui ne peut s'étendre qu'à droite, et la dernière FIFO qui ne peut s'étendre qu'à gauche ;
4. Une FIFO doit toujours garder un minimum d'emplacements, ce minimum est identifié par la variable *"MSpace"*, sa valeur peut être modifiée au cours de l'exécution par le gestionnaire de la reconfiguration ;

5. Une $FIFO_i$ ne peut s'étendre ni à droite ni à gauche si la valeur du pointeur de lecture pop_i est supérieure à la valeur du pointeur d'écriture $push_i$. Par exemple, sur la figure 5.1 la $FIFO_2$ ne peut pas s'étendre ni à droite ni à gauche car la valeur de son pointeur *"pop"* est supérieure à la valeur de son pointeur *"push"*. Cette règle garantit le mécanisme circulaire de la FIFO et empêche d'ajouter des emplacements qui seront lus avant d'être remplis par des données ;
6. Une $FIFO_i$ ne peut s'étendre à droite que si le premier emplacement de la $FIFO_{i+1}$ est vide, ou si la $FIFO_{i+1}$ est vide dans ce cas il faut initialiser les pointeurs pop_{i+1} et $push_{i+1}$ à la valeur de l'indice min_{i+1} après l'exécution de la reconfiguration ;
7. Une $FIFO_i$ ne peut s'étendre à gauche que si le dernier emplacement de la $FIFO_{i-1}$ est vide.

L'allocation dynamique des FIFOs doit être réalisée en adéquation avec le mécanisme de la reconfiguration dynamique expliqué dans la section 4.1 du chapitre 4. Ce mécanisme intègre quatre étapes successives, l'étape de prise de décision déclenche le processus de reconfiguration, l'étape de validation vérifie la possibilité d'exécution de la nouvelle configuration, l'étape de construction détermine les paramètres qui seront modifiés, et l'étape d'exécution modifie les valeurs des paramètres choisis.

5.1.2.1 La prise de décision

Le gestionnaire de la reconfiguration des FIFOs prend la décision d'augmenter la taille de l'une des FIFOs quand le taux de remplissage de celle-ci atteint un certain seuil appelé "Rthreshold". Ainsi, la reconfiguration de la $FIFO_i$ est demandée lorsque la condition 5.1 est vraie. Chaque $FIFO_i$ possède son propre seuil de reconfiguration "$Rthreshold_i$", et sa valeur peut être modifiée par le gestionnaire de la reconfiguration lors de l'exécution.

$$count_i > max_i - min_i - Rthreshold_i \tag{5.1}$$

Lorsque plusieurs FIFOs demandent une reconfiguration au même moment, le gestionnaire de reconfiguration choisit celle qui a la priorité la plus élevée, cette priorité dépend de l'urgence et de l'importance des données de la communication. Il choisit aussi le sens de la reconfiguration de la FIFO selon le taux de remplissage des FIFOs adjacentes.

5.1.2.2 La validation

La prise de décision pour reconfigurer une $FIFO_i$ est suivie par l'étape de validation. Par conséquent, quand le gestionnaire de la reconfiguration transmet une requête de reconfiguration en spécifiant qu'elle FIFO récupérera un emplacement et dans quel sens, le contrôleur des mémoires tampons vérifie la possibilité de la réalisation de cette requête. Cette vérification dépend du sens de la reconfiguration, ainsi, si la $FIFO_i$ doit s'étendre à droite sur la $FIFO_{i+1}$, la condition 5.2 doit être vraie. Sinon, si la $FIFO_i$ doit s'étendre à gauche sur la $FIFO_{i-1}$, la condition 5.3 doit être vraie. La condition 5.2 obéit aux règles ", 4, 5, et 6 de l'allocation dynamique des emplacements, citées précédemment dans cette section. En revanche, la condition 5.3 doit satisfaire les règles 3, 4, 5, et 7.

Algorithme 7: Reconfiguration de la taille d'une FIFO

Données : n *le numéro de la FIFO à reconfigurer*
Données : s *le sens de la reconfiguration*
si s = *"Right"* **alors**
  $max_n \leftarrow max_n + 1$
  $min_{n+1} \leftarrow min_{n+1} + 1$
sinon si s = *"Left"* **alors**
  $min_n \leftarrow min_n - 1$
  $max_{n-1} \leftarrow max_{n-1} - 1$
fin

$$(i \neq N-1)\ and\ (pop_i < push_i)\ and\ (max_{i+1} - min_{i+1} > MSpace_{i+1})\ and \\ ((push_{i+1} > min_{i+1}\ and\ pop_{i+1} > min_{i+1})\ or\ count_{i+1} = 0) \tag{5.2}$$

$$(i \neq 0)\ and\ (pop_i < push_i)\ and\ (max_{n-1} - min_{n-1} > MSpace_{i-1})\ and \\ (pop_{i-1} < max_{i-1}\ and\ push_{i-1} < max_{i-1}) \tag{5.3}$$

5.1.2.3 La construction et l'exécution

Après la validation de la reconfiguration par le contrôleur des mémoires tampons, il construit et exécute le procédé de modification de l'une des FIFOs. Cette modification décrite dans l'algorithme 7 consiste à ajouter un emplacement pour la FIFO qui doit augmenter sa taille, et supprimer un emplacement pour l'une des FIFOs adjacentes selon le sens spécifié par le gestionnaire de la reconfiguration. Le procédé d'exécution dure un cycle d'horloge, ainsi nous pouvons ajouter ou enlever un emplacement pour une FIFO à chaque cycle. Pour réaliser cette exécution avec une durée d'un cycle, nous avons développé un gestionnaire de reconfiguration avec des processus qui s'exécutent sur le front montant de l'horloge et d'autres sur le front descendant. Cette spécificité est rendue possible grâce à l'architecture du gestionnaire de reconfiguration détaillée dans la section 5.3 de ce chapitre.

5.2 Table TDMA configurable dynamiquement

La transmission des paquets par les interfaces réseau est gérée par le port de sortie réseau en utilisant un ordonnanceur de type TDMA. La table TDMA est constituée d'un ensemble d'intervalles de temps appelés "slot", chaque intervalle dure un cycle d'horloge et indique la communication qui a le droit d'émettre durant cet intervalle de temps. Ainsi, chaque interface

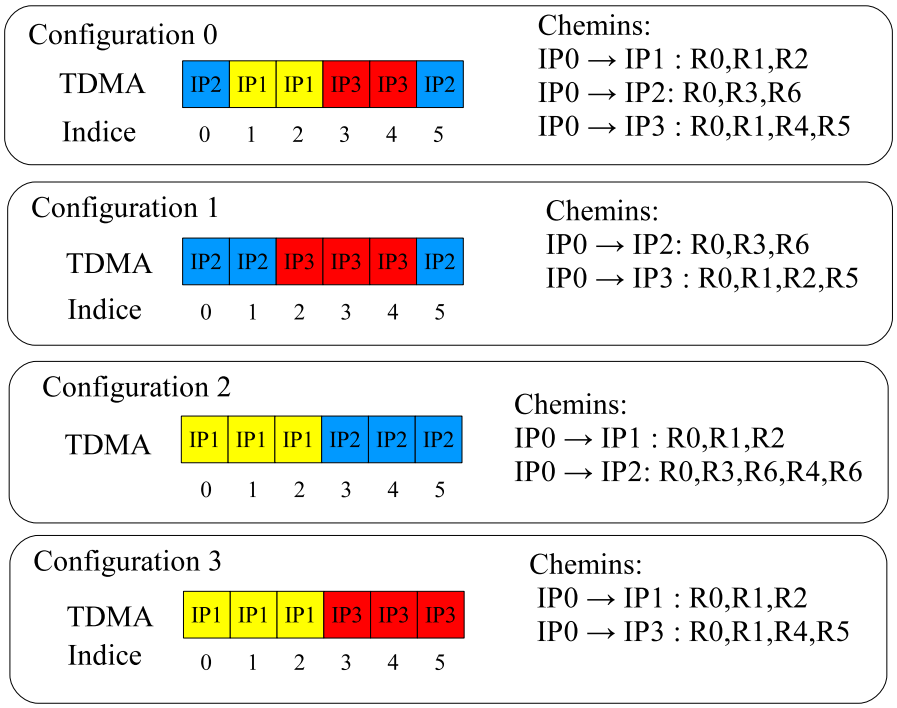


FIGURE 5.3 – Principe de la reconfiguration dynamique d'une table TDMA

réseau organise les transmissions selon une distribution rigoureuse des instants de départ des paquets des différentes communications, cette distribution est calculée conjointement lors de la recherche des chemins de routage expliquée dans la section 3.6 du chapitre 3.

Le mécanisme d'accès multiple à répartition dans le temps permet de garantir une bande passante minimum à toutes les communications. Ce principe de répartition est certes incontournable pour concevoir un NoC de qualité de service GT, mais son utilisation rend le réseau statique. Pour s'affranchir de cette contrainte, nous présentons dans cette section une nouvelle méthode qui permet de configurer la table TDMA dans les interfaces réseau en allouant dynamiquement les slots de temps aux communications.

5.2.1 Principe et fonctionnement

L'introduction de la reconfiguration dynamique sur un ordonnanceur de type TDMA consiste à redistribuer les intervalles de temps sur les différentes communications organisées dans la table TDMA. Cette redistribution permet d'augmenter la bande passante de certaines communications, en revanche elle diminue également la bande passante des autres car le nombre d'intervalles de temps de la table TDMA ne change pas. Elle permet aussi de partager les mêmes slots de temps entre les communications mutuellement exclusives.

Pour réaliser cette technique de reconfiguration sur la table TDMA, nous utilisons plusieurs tables TDMA avec des distributions des slots différentes. Ces distributions sont compatibles entre elles, car chaque distribution organise les instants de départ des paquets avec des chemins précalculés et différents des autres distributions. Ainsi, on peut changer de distribution sans perte de paquets car le calcul des chemins spatio-temporels est réalisé de manière à éviter les conflits et les blocages des paquets. Comme l'illustre la figure 5.3, une interface réseau qui transmet des

paquets vers trois IPs différents, peut reconfigurer la distribution des slots de temps en passant d'une configuration de la table TDMA à une autre. Chaque configuration de la table TDMA utilise des chemins calculés en amont, et qui permettent d'éviter les conflits de routage.

Les différentes distributions de la table TDMA d'une interface réseau sont stockées et administrées par le gestionnaire de la reconfiguration. Cette administration intégre les quatre étapes successives du mécanisme de reconfiguration dynamique décrit dans la section 4.1 du chapitre 4. Ces étapes sont la prise de décision, la validation, la construction et l'exécution.

5.2.1.1 La décision

La prise de décision par le gestionnaire de RD pour passer d'une configuration de TDMA à une autre, repose sur le taux de remplissage des tampons mémoires de l'interface réseau. Par exemple, sur la figure 5.3, le gestionnaire de la RD passe de la configuration 0 à la configuration 1, si et seulement si, la FIFO qui contient les données pour l'IP1 est vide, et que l'une des FIFOs qui contient des données pour l'IP3 ou l'IP2 est pleine. En plus, le gestionnaire de la RD revient à la configuration 0 dès que la FIFO qui contient des données pour l'IP1 commence à se remplir. De cette manière, les communications détiennent un nombre de slots initial qui permet de satisfaire une certaine bande passante, et elles peuvent augmenter leur capitale de slots de temps pour une certaine durée, à condition de ne pas avoir de conséquences négatives sur les performances des autres communications.

5.2.1.2 La validation

Quand le gestionnaire de reconfiguration décide de changer de distribution, il valide cette reconfiguration en vérifiant qu'aucune communication n'est en cours d'émission. Alors, si l'indice de la table TDMA en cours est sur un slot de temps où l'on transmet un paquet de type entête ou données, le gestionnaire de la RD n'autorise pas la reconfiguration. En revanche, dès que cet indice atteint un slot où on transmet un paquet de type queue, il valide le changement de distribution après l'émission du paquet queue. De cette manière, nous préservons la structure des messages transmis sur le RNoC, et qui doivent être constitués obligatoirement d'un paquet entête et d'un paquet queue.

5.2.1.3 La construction et l'exécution

Dans un réseau sur puce de type GT, la synchronisation des tables TDMA est primordiale, car les chemins spatio-temporels prédéfinis pour éviter les conflits au niveau des routeurs dépendent de la distribution des slots de temps des différentes communications dans les tables TDMA. Ainsi, le changement de distribution impose au gestionnaire de la RD de passer la valeur de l'indice de la table actuelle à l'indice de la table qui est choisie pour la remplacer. Par exemple, dans le cas où on remplace la table TDMA de la configuration 0 (Fig.5.3) par la table TDMA de la configuration 2 à un instant où l'indice de la table 0 est égale à 3, il faut que l'indice de la table 2 au moment de la reconfiguration commence avec une valeur égale à 3.

De plus, après l'exécution de la reconfiguration, le gestionnaire de la RD doit s'assurer que les paquets envoyés sont précédés par un paquet de type entête. Alors, si l'indice de la nouvelle table TDMA indique après la reconfiguration un slot où on transmet un paquet de type donnée ou queue, le paquet n'est pas envoyé au réseau. En revanche, la nouvelle table TDMA reprend un cours d'exécution normal, dès que son indice indique un slot où l'on transmet un paquet de type entête.

Les différentes distributions de la table TDMA sont pré-calculées lors de la conception du RNoC. Mais, nous pouvons les calculer également en temps réel avec le gestionnaire global dans le cas où nous l'intégrons à l'un des communicants connectés au RNoC.

5.2.2 Construction des chemins spatio-temporels dans le cas des tables multi-TDMA compatibles

Pour calculer les chemins spatio-temporels de toutes les communications, nous avons utilisé une heuristique expliquée dans la section 3.6 du chapitre 3. Cette heuristique itérative exécute à chaque itération un algorithme de construction glouton, en changeant les instants de départ de certaines communications, puis il compare la solution courante à la solution calculée dans l'itération précédente, et il garde la meilleure solution.

La version actuelle de l'algorithme de construction glouton considère que chaque interface réseau est associée à une seule table TDMA. Afin d'étendre cet algorithme à notre nouveau problème qui considère l'association de plusieurs tables TDMA au sein de la même interface réseau. Nous gardons le même principe de construction parallèle, en revanche, nous modifions la gestion du graphe d'occupation des arcs (liens du RNoC).

L'algorithme glouton parallèle répète deux étapes sur tous les chemins, jusqu'à ce qu'ils soient tous déclarés "construit" ou "bloqué", ces étapes sont :

1. La première étape consiste à se placer sur le dernier sommet V ajouté dans le chemin de la communication, et sur l'instant effectif courant t_{ij} de l'exécution de la fonction ;
2. La deuxième étape permet d'ajouter un nouveau sommet V' dans le chemin pour se rapprocher de la destination. Ainsi, elle vérifie pour tous les successeurs potentiellement utilisables, par lecture du graphe d'occupation, que l'arc $E^{V}_{V'}$ qui relie les sommets V et V' n'est pas occupé pendant l'intervalle de temps $[t_{ij}, \ldots, (t_{ij} + S_{ij})]$. Cet intervalle est calculé selon le nombre de paquets qui constitue la communication.
   Après la vérification, elle choisit l'un des successeurs selon la contrainte du plus court chemin. Dans le cas où il y a plusieurs sommets qui remplissent les mêmes conditions, elle en choisit un au hasard. Ensuite, elle met à jour le statut de l'arc $E^{V}_{V'}$ dans le graphe d'occupation aux instants de passage des différents paquets du message $[t_{ij}, \ldots, (t_{ij} + S_{ij})]$. Si aucun successeur n'est choisi, le chemin est déclaré "bloqué". Par contre si le sommet successeur est identique à la destination on déclare le chemin "construit".

Pour comprendre la démarche de cet algorithme, nous illustrons sur la figure 5.4 un exemple de construction de chemins pour un réseau qui connecte quatre IPs. Chaque IP communique avec les autres en organisant ses communications dans une table TDMA. Cet exemple intègre aussi l'approche des tables multi-TDMA compatibles, en affectant à l'interface réseau connectée à l'IP1 quatre tables TDMAs, avec différentes distributions de slots de temps. Nous différencions les communications en utilisant plusieurs couleurs, ainsi les paquets de chaque communication sont représentés par une couleur unique. Nous attribuons aussi à chaque arc du réseau reliant deux routeurs une table TDMA pour représenter les slots de temps occupés par les différentes communications.

Prenons l'exemple la communication représentée par la couleur rouge, envoyée par l'IP2 à destination de l'IP1. Cette communication contient deux paquets, un entête placé à l'instant 2 de la table TDMA de l'IP2 et un paquet de données placé à l'instant 3. Pour trouver le chemin spatio-temporel de cette communication, l'algorithme de construction se place sur le sommet $V = R6$, et sur l'instant effectif courant $t_{ij} = 3$. Pour ajouter un nouveau sommet V' au chemin de cet

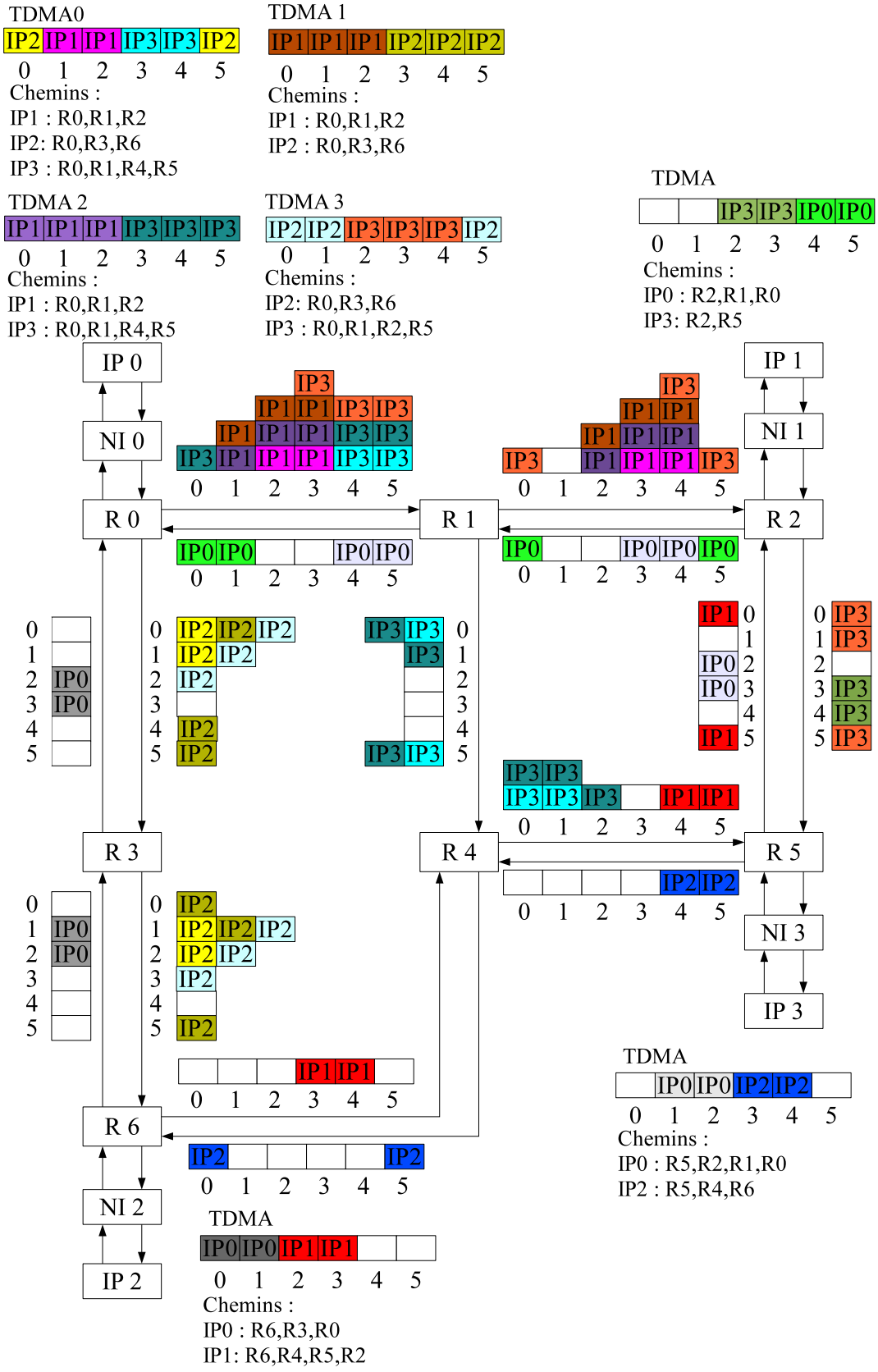


FIGURE 5.4 – Exemple d'allocation de chemins pour les tables multi-TDMAs compatibles

communication, l'algorithme de construction vérifie la disponibilité des sommets qui permettent de parvenir à la destination le plus tôt possible, dans notre cas c'est le sommet $R4$. Comme l'arc ou le lien E_{R6}^{R4} entre les routeurs $R6$ et $R4$ est disponible pendant l'intervalle de temps $[3, 4]$, le sommet $R4$ est ajouté au chemin. De la même manière, l'algorithme de construction ajoute le sommet $R5$ parce que le lien E_{R5}^{R6} est disponible pendant l'intervalle de temps $[4, 5]$. Et il ajoute aussi le sommet $R2$ car le lien E_{R2}^{R6} est libre pendant l'intervalle $[5, 0]$. Ainsi, cette communication envoie ses paquets à chaque tour de table TDMA pendant l'intervalle de temps $[2, 3]$, avec un chemin spatiale $IP2 \longrightarrow R6 \longrightarrow R4 \longrightarrow R5 \longrightarrow R2 \longrightarrow IP1$ sans conflit.

Afin que l'algorithme de construction glouton prenne en compte la gestion des tables multi-TDMA compatibles, nous avons développé une nouvelle version de l'algorithme qui permet à un même arc d'être partagé entre différentes tables TDMAs d'une même interface réseau sans risque de conflits. Ainsi, nous autorisons notre algorithme, lors de la vérification des successeurs potentiellement utilisables, à sélectionner même ceux qui ont des arcs utilisés par d'autres communications. À condition que la communication qui occupe l'arc appartient à la même source (NI) que celle qu'il cherche à l'utiliser. De plus elles doivent être mutuellement exclusives, c'est à dire, elles sont ordonnancées dans deux tables TDMAs différentes mais qui appartiennent à la même interface réseau.

Dans l'exemple de la figure 5.4, certains chemins calculés par l'algorithme de construction partagent le même slot de temps du même arc (lien). Par exemple, l'arc E_{R1}^{R0} partagent l'instant 3 de la table d'occupation sur quatre communications. Ce partage est possible car ces communications appartiennent à des tables TDMA différentes gérées par la même interface réseau de l'IP0, donc ces communications sont mutuellement exclusives car elles sont ordonnancées dans des tables TDMA mutuellement exclusives.

5.3 Gestionnaire local de la reconfiguration dynamique

L'intégration de la reconfiguration dynamique dans le NoC μSpider II consiste à modifier l'architecture des interfaces réseau afin d'inclure les mécanismes de reconfiguration des mémoires tampons et des tables TDMA compatibles. Ainsi, nous ajoutons à l'architecture initiale des interfaces réseau détaillée dans la section 2.3 du chapitre 2, le nouvel élément appelé gestionnaire local LM de la reconfiguration dynamique. Son rôle est de contrôler les mécanismes de reconfiguration selon des paramètres gérés par le gestionnaire global GM.

Cette intégration du LM a permis de définir une nouvelle génération d'interface réseau, appelée interface réseau auto-adaptative (SANI [45]). Nous introduisons dans cette section l'architecture de la SANI 5.3.1, et les procédures utilisées pour administrer les mécanismes de reconfiguration des mémoires tampons (voir section5.3.2) et des tables TDMA compatibles (section 5.3.3).

5.3.1 L'architecture de l'interface réseau auto-adaptative (SANI)

L'architecture de la SANI représentée sur la figure 5.5 est composée de 7 éléments :

1. Le **port d'entrée** connecté à l'adaptateur réseau permet de récupérer les données puis de les transmettre au contrôleur des FIFOs d'émission selon une table de correspondance. Cette table permet de trouver l'adresse de la FIFO qui stockera la donnée courante suivant l'adresse de destination ;

45. Self Adaptive Network Interface

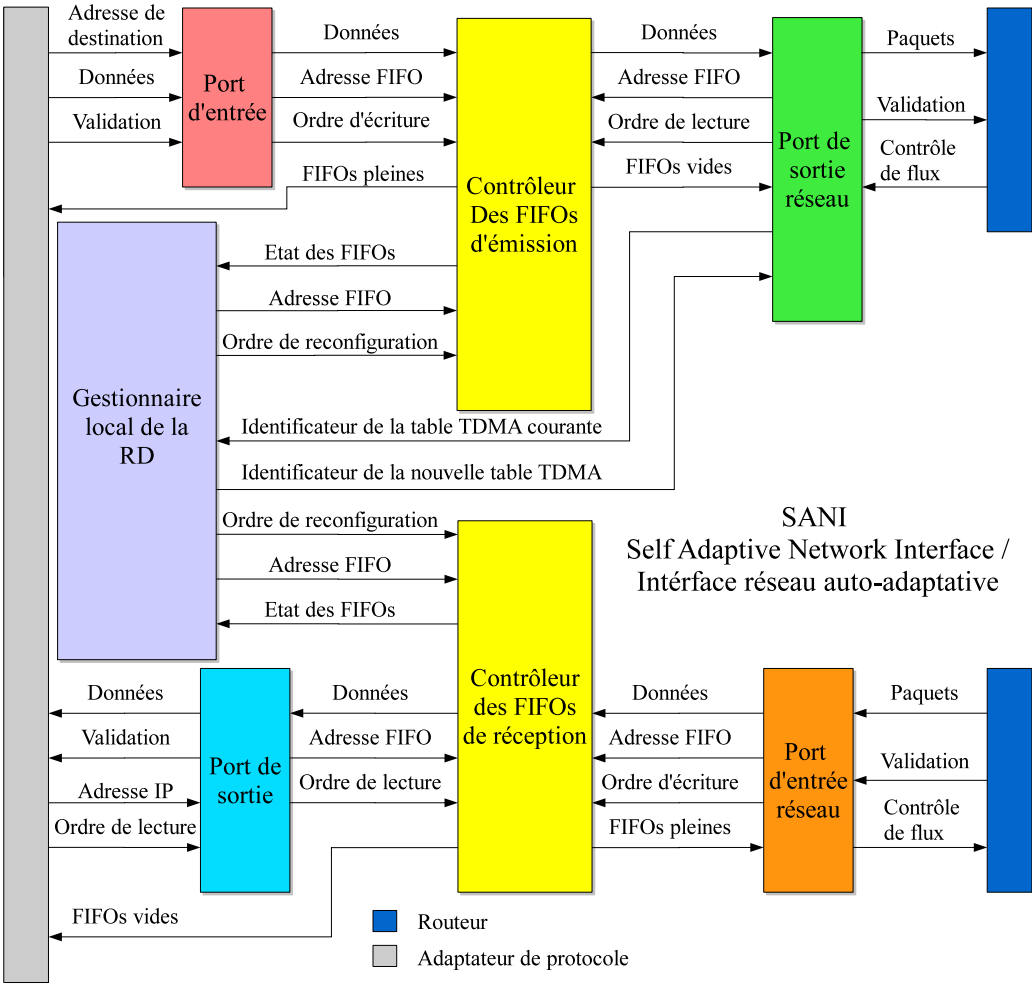


FIGURE 5.5 – Architecture de l'interface réseau auto-adaptative μSpider II

2. Le **port de sortie** transmet à l'adaptateur réseau les paquets reçus par la SANI et stockés dans les différentes FIFOs de réception. Cette transmission est réalisée seulement si l'adaptateur réseau l'autorise en envoyant un ordre de lecture ;
3. Le **port d'entrée réseau** reçoit différents paquets du réseau, puis il les transmet au contrôleur des FIFOs de réception en spécifiant l'adresse de la FIFO qui stockera ces paquets. L'adresse de la FIFO est calculée selon l'adresse de l'émetteur ;
4. Le **port de sortie réseau** gère la transmission des données stockées dans les FIFOs d'émission vers le réseau selon un ordonnancement imposé par la table TDMA. Nous avons modifié l'architecture de ce port afin de gérer différentes tables TDMA compatibles, et par conséquent pouvoir intégrer le mécanisme de reconfiguration dynamique de la table TDMA dans la SANI ;
5. Les **contrôleurs des FIFOs** d'émission et de réception gèrent les écritures et les lectures dans les différentes FIFOs. Pour intégrer le principe des mémoires tampons configurables, nous avons modifié le comportement de ces contrôleurs afin de gérer des FIFOs appartenant à la même mémoire physique, et organisées de manière à appliquer le principe d'allocation dynamique de la mémoire. Cette nouvelle organisation expliquée dans la section 5.1 de ce

chapitre est gérée par les contrôleurs de FIFOs en utilisant trois vecteurs de taille égale au nombre de FIFOs à administrer. Les deux vecteurs "MIN" et "MAX" délimitent l'espace réservé à chaque FIFOs et le vecteur "COUNT" indique le taux de remplissage des FIFOs ;

6. Le **gestionnaire local** administre la reconfiguration dynamique des FIFOs en émission et en réception, ainsi que la configuration des tables TDMA compatibles.

5.3.2 Administration des mémoires tampons reconfigurables

Le mécanisme de la reconfiguration dynamique des FIFOs expliqué dans la section 5.1.2 est intégré matériellement à l'architecture de la SANI afin d'administrer l'allocation dynamique de la mémoire en temps réel. Cette administration est réalisée par le gestionnaire local (LM) grâce aux échanges qu'il effectue avec les contrôleurs des FIFOs d'émission et de réception.

Le LM connait en temps réel le taux de remplissage des FIFOs grâce au signal "Etat des FIFOs", ainsi il peut décider à quel moment et quelle FIFO doit être reconfigurée selon les procédures de prise de décision et de validation, détaillées précédemment dans la section 5.1.2.1. Pour reconfigurer la taille d'une FIFO spécifique, le LM indique l'adresse de la FIFO sur le signal "Adresse FIFO"', puis il met le signal "Ordre de reconfiguration" à 1 sur le front montant de l'horloge et pendant un cycle. Et quand le contrôleur des FIFOs reçoit cet ordre, il lance la procédure de construction et d'exécution détaillée précédemment dans la section 5.1.2.3. La procédure de construction et d'exécution est réalisée sur le front descendant de l'horloge. Ainsi, on obtient une durée de reconfiguration d'un cycle entre le moment où l'ordre de reconfiguration est transmis par le LM et le moment où la nouvelle configuration est exécutée. Par conséquent, on peut reconfigurer la taille de l'une des FIFOs à chaque cycle.

5.3.3 Administration des tables TDMA reconfigurables

L'intégration du principe de la table TDMA reconfigurable est réalisée matériellement et dans les SANIs. Ainsi, chaque SANI administre les différentes tables TDMA compatibles qui lui sont allouées selon les taux de remplissage des FIFOs. Cette administration est effectuée également par le LM, car il a une connaissance exacte des taux de remplissage des différentes FIFOs d'émission à n'importe quel instant grâce au signal "Etat des FIFOs". Le LM connait aussi les caractéristiques des différentes tables TDMA stockées dans le port de sortie réseau, et quelle table TDMA est en cours d'utilisation à l'aide du signal "Identificateur de la table TDMA courante". Grâce à toutes ces informations le LM décide d'utiliser une autre table TDMA selon la procédure de prise de décision expliquée dans la section 5.2.1.1 et il transmet ce choix au port de sortie réseau en lui spécifiant l'identificateur de la nouvelle table TDMA. Quand le port de sortie réseau reçoit l'identificateur de la nouvelle table TDMA, il valide et exécute cette nouvelle configuration selon les procédés décrits dans la section 5.2.1.3. À la fin de la reconfiguration le port de sortie réseau actualise la valeur de l'identificateur de la table TDMA courante.

Il est possible que le LM change de décision au cours de la configuration en choisissant une autre table TDMA tant que le port de sortie réseau n'a pas actualisé la valeur de la table TDMA courante.

5.4 Gestionnaire global de la reconfiguration dynamique

Notre modèle de reconfiguration dynamique instaure une administration partagée entre les gestionnaires locaux (LMs) implantés matériellement sur les différentes interfaces réseau et un gestionnaire global (GM) implanté comme une tâche logicielle sur l'un des communicants connecté au réseau. Ce principe de délégation permet d'administrer les mécanismes de reconfiguration avec les LMs en modifiant les paramètres de certains éléments selon des règles et des configurations qui peuvent être modifiées par le GM.

Le gestionnaire global a pour mission d'adapter les jeux de configurations utilisés par les LMs selon des nouveaux besoins ou pour optimiser la gestion des communications. Ainsi, le GM peut recevoir des requêtes pour ajouter, modifier, ou supprimer des communications. Mais, il peut aussi récolter des statistiques sur le nombre de paquets transmis et bloqués dans les différentes interfaces réseaux. Cette connaissance du réseau permet au GM de redimensionner certaines configurations afin d'optimiser le fonctionnement global du NoC.

Le GM que nous avons développé pour notre RNoC μSpider II gère les paramètres de deux mécanismes de reconfiguration dynamique, la configuration des mémoires tampons et la configuration des tables TDMAs. Notre GM décide de redimensionner certains paramètres du RNoC selon les différentes requêtes envoyées par les SANIs, parmi ces requêtes nous distinguons :

- la modification de la bande passante de certaines communications. Cette requête a pour but de redistribuer les intervalles de temps de la table TDMA d'une SANI en procédant à une nouvelle allocation, et en calculant les nouveaux chemins spatio-temporels ;
- et le changement du destinataire d'une communication. Dans ce cas le GM recalcule les nouveaux chemins spatio-temporels en prenant en compte la nouvelle destination.

Nous remarquons que ces requêtes entraînent le calcul de nouveaux chemins spatio-temporels afin de satisfaire les nouvelles contraintes, pour cette raison nous avons développé un algorithme fondé sur une approche évolutionnaire. Cet algorithme, expliqué dans la section 5.4.1, a pour vocation de diminuer le temps de calcul. Cette diminution permet ainsi à l'algorithme de construction des chemins spatio-temporels d'être exécuter en temps réel et par conséquent de satisfaire toutes les requêtes de reconfiguration que le GM peut recevoir.

Les différentes requêtes et les nouvelles configurations sont transmises par les paquets de type contrôle. Le traitement de ces paquets est réalisé au niveau des SANIs, ainsi quand une SANI reçoit un paquet de contrôle, elle déclenche une procédure spécifique pour le traiter.

5.4.1 Approche évolutionnaire

La première approche adoptée pour rechercher les chemins spatio-temporels utilise un algorithme heuristique itératif simple. Cet algorithme réitère l'exécution de la procédure de construction gloutonne de tous les chemins, en changeant à chaque itération les instants de départ de plusieurs communications dans les différentes tables TDMAs. De cette manière, l'algorithme permet de balayer un ensemble de solutions à partir de configurations différentes, puis il sélectionne la meilleure solution.

La solution heuristique offre des résultats satisfaisants avec des temps de calcul faibles, mais dans une démarche d'optimisation, nous avons développé une métaheuristique qui repose sur un algorithme évolutionnaire. L'algorithme évolutionnaire s'inspire de l'évolution des êtres vivants vers des populations plus adaptées à leur environnement.

L'algorithme évolutionnaire fait évoluer un ensemble (une "population") de solutions (les "individus"). Chaque individu est une solution unique à laquelle est associée une valeur appelée

"*fitness*" mesurant la qualité de la solution. Cette valeur permet l'ordonnancement des individus et de procéder à leur sélection. Les "meilleures" solutions ont la plus grande valeur de *fitness*, tandis que les "mauvaises" solutions ont une *fitness* plus faible. À chaque étape de l'algorithme est associé un "opérateur", qui décrit la façon de manipuler les individus. On regroupe parfois les différents opérateurs sous des termes génériques :

– les opérateurs de sélection pour la sélection et le remplacement d'un individu ;
– et les opérateurs de variation pour la mutation et le croisement.

Pour notre problème, nous ajoutons un opérateur d'amélioration réalisant une recherche locale, ainsi nous inscrivons notre algorithme évolutionnaire dans la catégorie des algorithmes mémétiques [MC03]. En revanche, nous n'utilisons aucun opérateur de croisement.

Dans les sections suivantes, nous définissons l'algorithme mémétique qui résout notre problème de recherche de chemins spatio-temporels, ainsi que les opérateurs utilisés par celui-ci et la méthode de calcul de la valeur de *fitness*.

5.4.1.1 Le calcul de la valeur de *fitness*

La valeur de la fonction de *fitness* est calculée selon deux objectifs, le premier tient compte du nombre de chemins construits, et le deuxième considère la longueur totale des chemins. Ainsi, pour un individu I de la population P qui représente une solution S, sa valeur de *fitness* est définit par l'équation 5.4, avec $NBuilt$ le nombre de chemins construits et $Length$ la somme des longueurs des chemins construits.

$$Fitness(I) = NBuilt - (10^{-\alpha} * Length) \tag{5.4}$$

La fonction de *fitness* est une valeur scalaire qui combine le maximisation du nombre de chemins construits et la minimisation de la longueur totale des chemins. On notera que la prise en compte de cette valeur est en principe équivalente à une comparaison lexicographique directe portant sur les deux objectifs. Par conséquent, pour pouvoir réaliser cette comparaison, on doit choisir une valeur de α suffisamment grande.

Par exemple, en considérant que $\alpha = 3$. Si la solution $S1$ trouve $NBuilt1 = 11$ et $Length1 = 88$, et la solution $S2$ trouve $NBuilt2 = 12$ et $Length2 = 89$. Alors $Fitness1 = 11 - (88 * 10^{-3}) = 10.12$ et $Fitness1 = 12 - (89 * 10^{-3}) = 11.11$. Ainsi, on prouve que la solution $S2$ est meilleure que la solution $S1$. Ceci est normal car la solution $S2$ a trouvé plus de chemins que la solution $S1$, ainsi l'objectif du nombre des chemins construits domine dans le calcul de la valeur de la fonction *fitness*. En revanche, si on considère que $Nbuilt1 = Nbuilt2 = 11$, on trouve que la solution $S1$ est meilleure que la solution $S2$. Car on départage les deux solutions par rapport au deuxième objectif qui considère la longueur totale des chemins construits.

5.4.1.2 Algorithme mémétique

L'algorithme 9 présente l'implémentation de notre algorithme mémétique. Il utilise cinq opérateurs, certains de ces opérateurs sont détaillés dans l'algorithme 8, nous distinguons :

1. deux opérateurs de sélection par rang, le premier remplace les k plus mauvais individus de la population par les k meilleurs individus de la population. Le second est un opérateur élitiste qui remplace les k plus mauvais individus de la population par le meilleur individu rencontré au cours de la recherche ($Best1$) ;
2. un opérateur de variation de type mutation qui permet d'initialiser le graphe d'occupation de certains individus de la population, le choix de ces individus est aléatoire ;

Algorithme 8: Fonctions de construction et de réparation d'une solution et fonctions de génération, de mutation et de recherche locale d'une population de solution

Fonction Construction(S : **solution**) : **solution**
Initialisation du graphe d'occupation
$S \leftarrow Init(S)$
Permutation et translation aléatoire des dates de départ
$S \leftarrow PermutationMessages(S)$
$S \leftarrow TranslationDates(S)$
Lancement d'une construction parallèle gloutonne
$S \leftarrow ConstructionGloutonne(S)$
Retourner S

Fonction Réparation(S : **solution**, NbrMg : **entier**) : **solution**
Suppression de $NbrMg$ chemins aléatoirement et mise à jour du graphe d'occupation selon les chemins supprimés
$S \leftarrow RemovePaths(S, NbrMg)$
Suppression des chemins déclarés "bloqués" dans la solution S et mise à jour du graphe d'occupation selon les chemins supprimés
$S \leftarrow RemovePathsBlocked(S)$
$S \leftarrow ConstructionGloutonne(S)$
Retourner S

Fonction Génération(P : **Population**) : **Population**
pour *Chaque Individu* $I \in P$ **faire**
  $I \leftarrow$ **Construction**(I)
fin
Retourner P

Fonction Mutation(P : **Population**) : **Population**
pour *Chaque Individu* $I \in P$ **faire**
  si $random(0, 1) > 0.5$ **alors**
    Initialisation du graphe d'occupation
    $I \leftarrow$ **Init**(I)
  fin
fin
Retourner P

Fonction RechercheLocale(P : **Population**) : **Population**
pour *Chaque Individu* $I \in P$ **faire**
  $I \leftarrow$ **Réparation**$(S, Random(0, NbrMg))$
fin
Retourner P

Algorithme 9: Algorithme mémétique

```
Déclaration d'une population constituée de 100 individus
P : Population(100)
Gen : Entier
Best1, Best2 : Individu
Génération des solutions pour l'ensemble des individus de la population P
P ⟵ Génération(P)
Best1 ⟵ getBest(P) Sélection de la meilleure solution
Gen ⟵ 0
tant que ((Gen < MaxGen) et (solution non construite)) faire
    Gen ⟵ Gen + 1
    P ⟵ RechercheLocale(P)
    Best1 ⟵ getBest(P, Best1)
    Best2 ⟵ getBest(Best1, Best2)
    Remplacement des taille(P)/5 plus mauvais individus par les taille(P)/5
    meilleurs individus
    P ⟵ Sélection(P,taille(P)/5)
    Remplacement des taille(P)/10 plus mauvais individus par le meilleur individu
    Best1
    P ⟵ SélectionElitiste(P, Best1,taille(P)/10)
    P ⟵ Mutation(P)
    si random(0, 1) > 0.99 alors
        Régénération des individus et sélection du meilleur
        P ⟵ Génération(P)
        Best1 ⟵ getBest(P, Best1)
    fin
fin
```

3. un opérateur de recherche locale qui exécute la procédure de réparation. Cette procédure permet de modifier un nombre de chemins tiré au hasard entre 0 et M, M étant le nombre total de communications ;
4. et l'opérateur de régénération qui reconstruit de temps à autre les solutions de tous les individus de la population. Cet opérateur remédie à la convergence rapide vers un optimum local.

Cet algorithme est réalisé sous forme d'une boucle évolutionnaire, cet boucle est arrêtée dès que l'on atteint un nombre d'itérations fixé par l'utilisateur $MaxGen$ et que l'on trouve une solution au problème. À chaque itération nous exécutons successivement l'opérateur de recherche locale, les opérateurs de sélection, puis l'opérateur de mutation, et enfin l'opérateur de régénération.

5.4.1.3 Performances de l'algorithme mémétique

Nous comparons dans cette section les performances du nouveau algorithme mémétique par rapport aux performances de l'heuristique itérative et selon deux critères. Le premier est le temps

Jeu de test	Heuristique itérative	Algorithme mémétique
A B.1		
Temps CPU (s)	0,3	0,16
Longueur moyenne	113,4	128,24
B B.2		
Temps CPU (s)	-	11,74
Longueur moyenne	-	152,65
C B.3		
Temps CPU (s)	17	73
Longueur moyenne	1499,52	1748,84

TABLE 5.1 – Comparaison des performances de l'heuristique et l'algorithme mémétique

d'exécution de chacun des algorithmes pour trouver une solution au problème, et le second est la valeur de la longueur totale des chemins. Nous avons utilisé différentes instances ou jeux de test pour la comparaison. Ces jeux de test sont détaillés dans l'annexe B. Toutes les expérimentations ont été réalisées sur un PC *Intel Core Duo* $2,6\ GHz$, avec l'utilisation d'un seul processeur.

Le tableau 5.1 présente une synthèse des résultats obtenus sur les trois jeux de test. Pour chaque jeu de test, nous rapportons le temps d'exécution minimum permettant d'obtenir une solution admissible et la longueur totale des chemins obtenus, la longueur d'un chemin correspond au nombre de liens (arcs) qui le constitue . Le résultat présenté est une moyenne réalisée sur 100 exécutions des algorithmes sur un même jeu de données.

Les résultats du tableau 5.1 montrent que l'algorithme mémétique trouve une solution pour les trois jeux de test "A", "B", et "C". En revanche le l'algorithme heuristique n'arrive pas à générer une solution pour le jeu de test "B". Ainsi, on peut en déduire que l'heuristique itérative est utilisée pour résoudre les problèmes qui n'intègre pas le principe des multi-TDMAs compatibles. Alors que l'algorithme mémétique permet de résoudre n'importe quel type de problème.

Nous observons aussi que les performances de l'heuristique sont un peu plus élevées surtout en terme de temps d'exécution, par exemple, pour le jeu de test "C" l'heuristique est quatre fois plus rapide que l'algorithme mémétique. De plus, l'heuristique offre toujours une meilleure solution que celle générée par l'algorithme mémétique par rapport à la longueur total des chemins.

Par conséquent, nous privilégierons l'utilisation de l'algorithme heuristique pour résoudre des problèmes complexes où le nombre de communications à traiter est très élevé comme le cas du jeu de test "C". En revanche, l'algorithme mémétique sera systématiquement utilisé pour les problèmes qui incluent le principe des multi-TDMA compatibles comme l'exemple du jeu de test "B" et pour les problèmes simples comme le cas du jeu de test "A".

5.5 Conclusion

La première section 5.1 de ce chapitre explique la méthode que nous avons développé pour intégrer le principe des FIFOs configurables dynamiquement dans les interfaces réseau du NoC. Cette méthode consiste à organiser les FIFOs dans la même mémoire physique, ainsi, chacune des FIFOs peut augmenter sa profondeur en s'étendant sur ces voisines. La mise en œuvre de ce mécanisme de reconfiguration dynamique a permis d'adapter la taille des FIFOs selon les besoins et d'augmenter les performances en diminuant le temps d'attente des communications pour accéder au réseau. Ce mécanisme a permis aussi de diminuer la complexité du contrôleur

des FIFOs, car les implantations réalisées ont montré que la surface occupée par le contrôleur baisse en moyenne de 41% en utilisant la nouvelle organisation des FIFOs.

Ce chapitre détaille aussi dans la section 5.2 une nouvelle approche qui introduit de la reconfiguration dynamique sur le protocole qui nous permet de concevoir un NoC avec une qualité de service qui garantit le trafic. Ainsi, nous avons développé une tables TDMA configurables, où le nombre d'intervalles de temps alloué à chaque communication peut changer en temps réel selon les besoins.

Pour administrer la reconfiguration dynamique de ces deux mécanismes, nous avons modifié l'architecture de l'interface réseau en lui ajoutant un module, nommé gestionnaire local de la RD et qui permet de commander les changements réalisés sur la taille des différentes FIFOs et la distribution des slots de temps dans la table TDMA. Nous avons défini également le rôle du gestionnaire global qui permet de recalculer les chemins spatio-temporels du NoC dans le cas où certaines communications changent leurs caractéristiques.

Partie III : Mise en œuvre, expériences et résultats

6

Environnements de conception, de simulation, et d'implantation du réseau μSpider II

Les deux premières parties de ce manuscrit relatent l'architecture du NoC et du RNoC et les algorithmes exploités par le flot de conception pour dimensionner les différents paramètres du réseau. L'exploitation de ces architectures et de ce flot de conception est une tâche très complexe, car elle demande une connaissance parfaite des paramètres à dimensionner, et des temps de développement et de corrections d'erreurs (debug) importants. Afin de maitriser cette complexité, nous avons développé un environnement pour aider à la conception des NoCs et des RNoCs. Cet environnement que nous décrivons dans la section 6.1 est constitué de plusieurs outils qui permettent de décrire, de dimensionner, et de générer le réseau.

Ce chapitre présente aussi les environnements que nous avons conçus pour tester les performances du réseau généré. L'environnement de simulation que nous décrivons dans la section 6.2 offre un premier niveau de test, et l'environnement d'implantation expliqué dans la section 6.3 permet de vérifier avec certitude le fonctionnement et les performances du réseau, par son implantation sur une plateforme FPGA.

6.1 Environnement de conception

Nous classons notre environnement de conception μSpider II (Fig.6.1) dans la catégorie de CAO [46] électronique, nommée également en anglais EDA [47]. Car, il permet de générer automatiquement la description matériel du réseau en partant d'une représentation de haut niveau des contraintes des communications. Cet environnement est composé de deux outils. Le premier développé dans la section 6.1.1 sert à éditer, et à analyser les graphes de dépendances des communications (CDGs). Et le deuxième outil décrit dans la section 6.1.2, calcule les différents paramètres du réseau et génère sa description matérielle.

Les outils μSpider II sont conçus avec l'environnement de développement *Eclipse* et en utilisant principalement le langage de programmation orienté objets *JAVA*. Les interfaces graphiques de nos outils sont créées grâce à la bibliothèque graphique SWT [48]. De plus, nous utilisons les interfaces de programmation (API [49]) suivantes :

– l'interface *JDOM* pour manipuler des documents de type XML [50] ;

46. Conception Assistée par Ordinateur
47. Electronic Design Automation
48. Standard Widget Toolkit
49. Application Programming Interface
50. eXtensible Markup Language

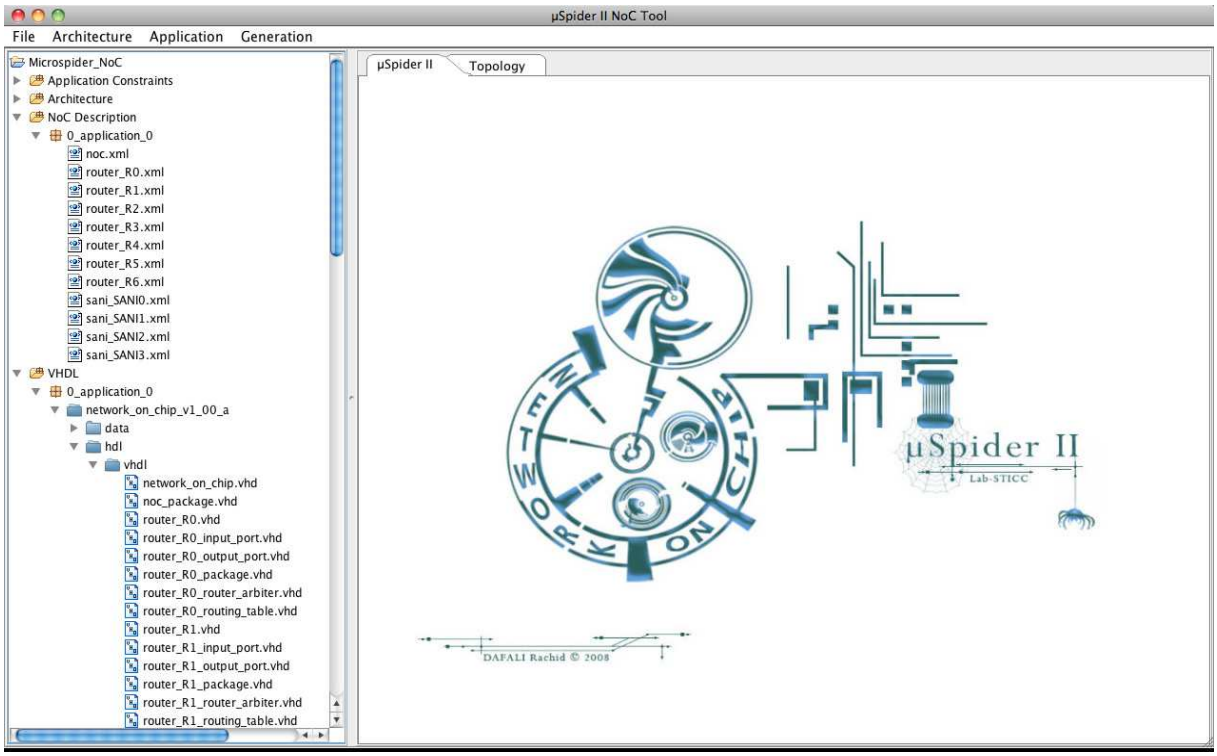


FIGURE 6.1 – Environnement de conception μSpider II

– l'interface *JGraph* pour la gestion des graphes ;
– l'interface *JXL* pour administrer les tableurs ;
– et l'interface *JFreechart* pour créer des graphiques et des diagrammes.

Dans ce chapitre, nous expliquons le fonctionnement de l'environnement μSpider II en utilisant plusieurs captures d'écran. Cette méthode facilite l'explication, mais elle alourdit le contenu du chapitre. Pour y remédier, nous rassemblons ces captures d'écran dans l'annexe C.

6.1.1 Outil d'exploration

L'outil d'exploration analyse l'application à partir de sa description donnée par l'utilisateur sous forme de graphes CDGs. À partir de cette analyse, l'outil génère les contraintes réelles des communications, et conçoit une topologie spécifique en utilisant la méthodologie expliquée précédemment dans le chapitre 3. Cet outil aide le concepteur à modéliser le réseau et ses contraintes en intégrant les étapes de spécification et de construction du flot de conception. Mais, son utilisation est optionnelle, car le concepteur peut définir sa propre topologie et les contraintes des communications sans l'aide de cet outil.

6.1.1.1 Édition des graphes CDGs

L'édition des graphes CDGs est réalisée grâce à l'interface illustrée sur la figure C.1 de l'annexe C. Cette interface permet à l'utilisateur de définir les caractéristiques du graphe en spécifiant son identifiant, le temps d'exécution, et la taille des paquets. Elle permet également

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:schemLocation="http://graphml.graphdrawing.org/xmlns"
   xmlns:y="http://www.yworks.com/xml/graphml">
   <graph xmlns="" edgedefault="directed" ExecutionTime="0.000008" WordSize="64">
     <node id="Com1">
       <data key="CommunicationName">Com1</data>
       <data key="CommunicationKind">R</data>
       <data key="CommunicationSource">I1</data>
       <data key="CommunicationTarget">DDR_O1</data>
       <data key="CommunicationBurstSize">8</data>
     </node>
     <node id="Com2">
     <node id="Syn1">
       <data key="CommunicationName">Syn1</data>
       <data key="SynTime">0.02</data>
     </node>
     <node id="Com3">
     <node id="Com4">
     <node id="Syn2">
     <node id="Com5">
     <edge source="Com1" target="Syn1" />
     <edge source="Com2" target="Syn1" />
     <edge source="Syn1" target="Com4" />
     <edge source="Syn1" target="Com3" />
     <edge source="Com3" target="Syn2" />
     <edge source="Com4" target="Syn2" />
     <edge source="Syn2" target="Com5" />
   </graph>
</graphml>
```

FIGURE 6.2 – Exemple d'enregistrement d'un graphe CDG avec le format GraphML

de créer différents types de nœuds, des nœuds de traitements ou d'attentes, et des nœuds de communications qui identifient l'émetteur, le destinataire et la taille des données transmises. En plus, elle permet d'éditer les liens qui relient les différents nœuds.

Après l'édition du graphe, l'outil l'enregistre dans un fichier qui porte un nom identique à l'identifiant du graphe et il attribue au fichier l'extension "xml". L'enregistrement des données du graphe est réalisé avec le format *GraphML*. Nous avons choisi ce format car il est fondé sur une description XML, et il structure les données d'une manière hiérarchique qui facilite la gestion des différents éléments du graphe. La figure 6.2 représente un graphe CDG enregistré sous le format *GraphML*.

Cette technique d'édition s'avère fastidieuse car elle demande beaucoup d'efforts de la part de l'utilisateur. Pour y remédier, nous avons développé une deuxième interface (Fig.C.2 de l'annexe C) pour générer automatiquement le graphe CDG à partir d'un tableur qui regroupe les informations sur les différents processus de l'application et l'ordre de leur exécution.

Nous avons conçu également une interface pour afficher la représentation graphique du graphe CDG à partir de sa description enregistrée dans le document XML. Cette interface est représentée dans la figure C.3 de l'annexe C.

6.1.1.2 Calcul des contraintes réelles

L'analyse des dépendances représentées par les graphes CDGs permet à l'outil d'exploration de trouver les communications mutuellement exclusives, et les communications transmises en parallèle. Ainsi, il peut calculer avec précision les contraintes réelles des communications en terme

```
<?xml version="1.0" encoding="UTF-8"?>
<application numberOfIP="4" numberOfSlot="6">
  <ip id="IP0" address="00">
    <forwardCom numberOfCom="3" NbrOfTdmaSlot="6" sharedMemoryDepth="108">
      <com id="0" bandwidth="100" quantityMax="1000" receiverId="IP1" receiverAddress="01"
        NbrOfSlot="2" path="00011000000000000000000000000000" fifoDepth="36" />
      <com id="1" bandwidth="100" quantityMax="1000" receiverId="IP2" receiverAddress="10"
        NbrOfSlot="2" path="01001101000000000000000000000000" fifoDepth="36" />
      <com id="2" bandwidth="100" quantityMax="1000" receiverId="IP3" receiverAddress="11"
        NbrOfSlot="2" path="01001010011000000000000000000000" fifoDepth="36" />
    </forwardCom>
    <reverseCom numberOfCom="3" sharedMemoryDepth="108">
      <com id="0" bandwidth="100" transmitterId="IP1" transmitterAddress="01" fifoDepth="36" />
      <com id="1" bandwidth="100" transmitterId="IP2" transmitterAddress="10" fifoDepth="36" />
      <com id="2" bandwidth="100" transmitterId="IP3" transmitterAddress="11" fifoDepth="36" />
    </reverseCom>
    <wrapperDMA burstSize="10" readDelay="3" />
    <TDMA table="0-0-2-2-1-1-" />
  </ip>
  <ip id="IP1" address="01">
  <ip id="IP2" address="10">
  <ip id="IP3" address="11">
</application>
```

FIGURE 6.3 – Description XML du fichier des contraintes

de bande passante. L'outil représente les résultats de ce calcul en utilisant un tableur comme le montre la figure C.4 de l'annexe C, ce tableur donne le détail des bandes passantes requises pour chaque communication. Il permet également de représenter graphiquement les bandes passantes en émission et en réception de chaque communicant en utilisant l'interface représentée sur la figure C.5 de l'annexe C.

6.1.2 Outil de génération

L'outil de génération permet de concevoir la description matérielle du NoC. Pour y parvenir, il calcule les valeurs des différents paramètres du NoC, en s'appuyant sur des éléments qui décrivent la structure de la topologie et qui définissent les contraintes des différentes communications. Ces éléments peuvent être générés automatiquement avec l'outil d'exploration en utilisant la méthodologie que nous avons expliquée dans la section précédente. Mais ces éléments peuvent aussi être définis par le concepteur du réseau.

Pour décrire manuellement la topologie du NoC, nous avons développé une interface graphique (Fig. C.6 de l'annexe C) qui permet au concepteur du réseau de construire graphiquement le NoC grâce à une palette qui contient les éléments principaux (routeur, interface réseau, IP, ports d'entrée/sortie, et liens), et plusieurs topologies génériques avec des dimensions paramétrables.

Après la description de la structure du NoC, le concepteur définit grâce à l'outil, les communications établies par les IPs et leurs contraintes en terme de bande passante et de quantité de données maximale. L'outil structure et enregistre ces données dans un fichier de contraintes en utilisant la description XML. Comme le montre la figure 6.3, le fichier des contraintes organise les communications selon l'émetteur, ainsi chaque IP est représenté par une balise principale qui porte son nom. Puis, dans cette balise, les communications émises et reçues sont structurées dans des sous-balises séparées.

Ces balises renferment plusieurs informations qui évoluent durant l'exécution des différentes

étapes du flot de conception. Ainsi, lors de la création du fichier des contraintes, l'outil ne spécifie dans les balises des communications émises et reçues que la bande passante, la quantité de données maximale, et l'adresse de l'émetteur ou du récepteur.

Après la définition des paramètres des communications et l'architecture du NoC. Nous dimensionnons les différents éléments du NoC, en exécutant avec l'outil le reste des étapes du flot de conception. Pour accomplir ce dimensionnement, le concepteur du réseau réalise grâce à l'outil les procédures suivantes :

1. la **procédure du dimensionnement de la table TDMA** calcule le nombre de slots de temps alloué à chaque communication émise, et le nombre de slots de temps total alloué aux différentes interfaces réseau du NoC. Les résultats du calcul sont enregistrés dans le fichier des contraintes en ajoutant aux différentes sous-balises des communications émises, le nombre de slots de temps alloué à ces communications ;
2. la **procédure d'allocation des chemins** calcule les chemins spatio-temporels en allouant à chaque communication un chemin spatial et un instant de départ. Le chemin spatial est enregistré dans le fichier des contraintes en l'ajoutant au contenu de la sous-balise qui décrit la communication émise. En revanche, l'instant de départ est enregistré dans une nouvelle sous-balise "TDMA" utilisée pour organiser les instants de départ des différentes communications émises. Dans le cas où le principe des tables multi-TDMA compatibles est introduit, chaque table TDMA est enregistrée dans une sous-balise "TDMA" avec un identifiant unique ;
3. et la **procédure du dimensionnement des mémoires tampons** calcule les tailles des FIFOs utilisées dans les interfaces réseau pour stocker les données des différentes communications. Les résultats de ce calcul sont également enregistrés dans les balises qui décrivent les communications émises et reçues.

À la fin du dimensionnement, l'outil génère la description matérielle du NoC ou du RNoC selon la version choisie par le concepteur du réseau. Cette génération consiste à remplacer les variables non définies dans les modèles (*templates*) avec les valeurs calculées durant le dimensionnement. Ces modèles sont des gabarits qui décrivent l'architecture des différents composants du NoC avec le langage de description matérielle VHDL.

6.2 Environnement de simulation

La simulation du NoC ou du RNoC consiste à définir un modèle comportemental pour tester le fonctionnement et les performances du réseau. Ce modèle décrit le comportement des IPs connectés au réseau, en représentant les différents processus exécutés sur l'IP et leurs dépendances. Pour réaliser cette simulation, nous modélisons chacun des IPs avec deux entités matérielles (Fig. 6.4), le générateur de trafic et le récepteur de trafic.

Le rôle de ces deux entités est d'exécuter les processus de traitement, et de communication en suivant les dépendances décrites dans les graphes CDGs. Ainsi, le générateur de trafic simule les processus de traitement en réalisant une attente d'une durée égale à celle du traitement. En revanche, il exécute les communications en transférant la quantité de données spécifiée. Le récepteur de trafic vérifie la provenance des communications ainsi que la quantité de données qu'il doit recevoir pour chaque communication.

L'outil de génération permet de concevoir l'architecture de l'environnement de simulation, en modélisant le trafic de chaque IP à partir des graphes CDGs que le concepteur du réseau définit à l'aide de l'outil d'exploration. Après la modélisation des trafics de l'environnement de

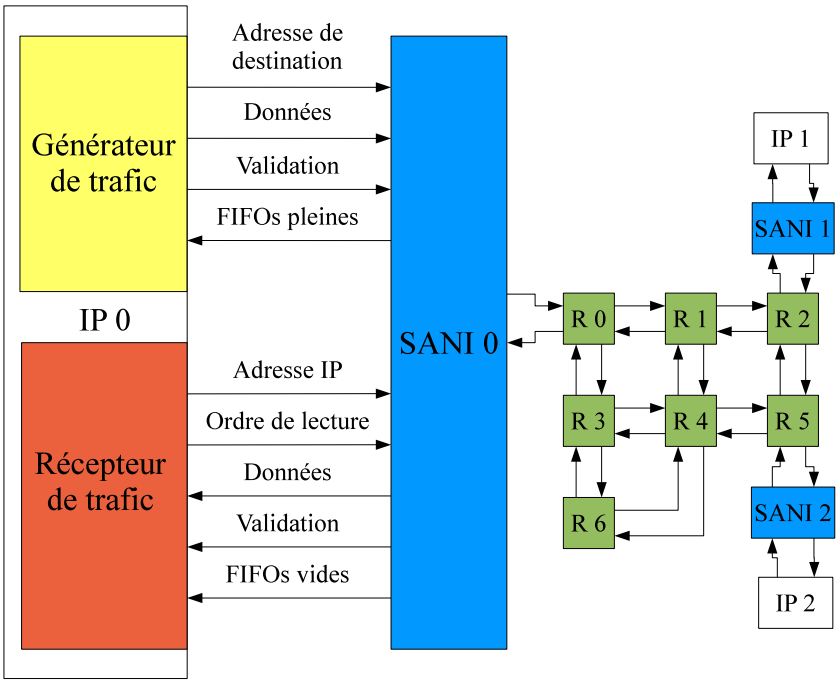


FIGURE 6.4 – Architecture du simulateur de trafic pour le NoC μSpider II

simulation, l'outil de génération construit la description matérielle de celui-ci à l'aide de plusieurs fichiers VHDL spécifiés au niveau RTL. Il génère également le *testbench* qui permet de tester le fonctionnement du réseau, et de calculer les bandes passantes des différentes communications. Ces bandes passantes sont déterminées à partir des durées de transferts des communications ; la durée de transfert est calculée à l'aide de prélèvements effectués au début et à la fin de chaque transfert sur un compteur de cycles synchrone.

6.3 Environnement d'implantation

La conception d'un environnement d'implantation a pour but de tester les performances réelles du NoC et du RNoC en prenant en compte certains phénomènes physiques que l'environnement de simulation ne peut pas inclure. Cette mise en œuvre vise à implanter sur une cible FPGA Xilinx, une architecture multiprocesseur qui connecte plusieurs unités de traitement avec le NoC μSpider II. Comme l'illustre la figure 6.5, chaque unité de traitement est composée d'un processeur de type μBlaze, d'un bloc mémoire BRAM [51] de 64 ko [52], d'un bus PLB [53], et d'un adaptateur de protocole PLB pour connecter l'unité de traitement au NoC. Cet adaptateur permet au NoC de contrôler directement le bloc BRAM en respectant le protocole du bus PLB. Il offre également au processeur μBlaze plusieurs méthodes pour exécuter les requêtes d'émission et de réception des données.

La figure 6.5 illustre également l'architecture de l'adaptateur de protocole qui se compose de plusieurs éléments, nous distinguons :

51. Block Random Access Memory
52. kilooctet
53. Processor Local Bus

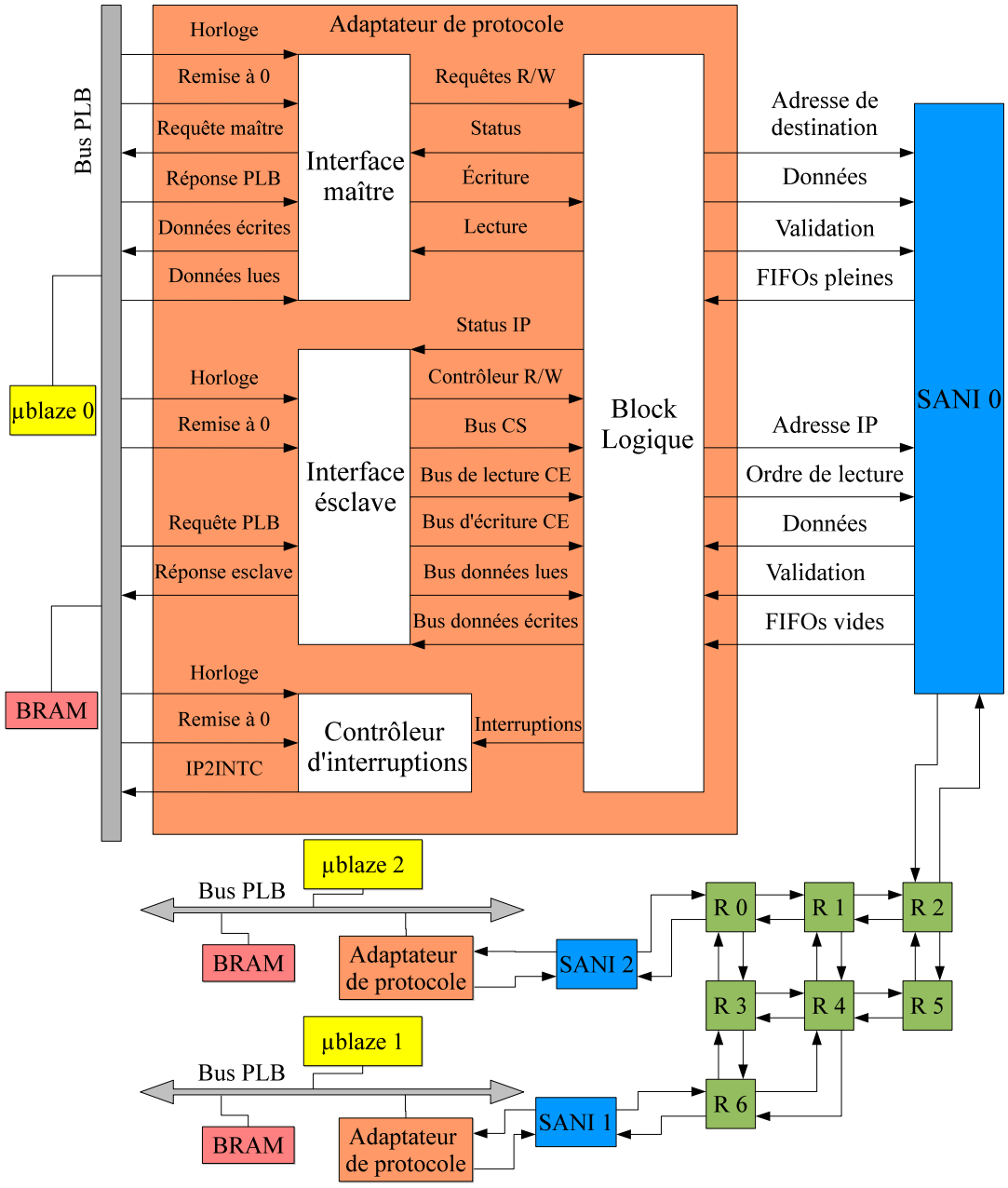


FIGURE 6.5 – Architecture multiprocesseur de l'environnement d'implantation, et la structure de l'adaptateur de protocole PLB

- l'interface esclave qui connecte l'adaptateur de protocole au bus PLB en tant que périphérique esclave. Ainsi, elle permet au processeur μBlaze de contrôler l'adaptateur et de lui transmettre des requêtes d'émission et de réception ;
- l'interface maître qui connecte l'adaptateur de protocole au bus PLB en tant que périphérique maître. Cette interface permet à l'adaptateur d'accéder directement au bloc BRAM, pour lire les données à transmettre sur le réseau, ou d'écrire celles que le réseau reçoit ;
- le contrôleur d'interruptions gère les interruptions que l'adaptateur de protocoles émet vers le processeur μBlaze à travers le bus PLB ;
- et le bloc "user logic" qui décrit le fonctionnement de l'adaptateur de protocole. Ce fonctionnement que nous avons détaillé dans la section 2.4 du chapitre 2 repose sur l'utilisation d'un DMA et d'une mémoire cache pour la gestion des communications en émission ou en réception.

Pour contrôler l'adaptateur de protocole, nous avons développé un driver qui regroupe différentes routines, ainsi que des fonctions écrites avec le langage de programmation C pour réaliser les requêtes d'émission et de réception des données. Ainsi, pour programmer l'envoi d'une quantité de données par le μBlaze, nous ajoutons à son programme C, la fonction "transmit-data-to-noc" 6.1. Cette fonction contient la quantité de données à transmettre "data-size", l'adresse où la première donnée est stockée dans le bloc BRAM "bram-addr", et l'adresse de destination "dest-addr". Quand l'adaptateur reçoit cette requête, il transmet la quantité de données spécifiée au destinataire à travers le réseau. Puis, il envoie une interruption au processeur μBlaze pour l'informer de la fin de la requête. Il lui transmet également à l'aide d'un registre le nombre de cycles utilisés pour transmettre la quantité de données. La valeur de ce registre est ensuite exploitée par le programme du processeur pour calculer la bande passante de la communication.

$$void\ transmit-data-to-noc(Xuint32\ bram-addr,\ int\ data-size,\ Xuint32\ dest-addr) \quad (6.1)$$

Dans le cas où l'adaptateur de protocole reçoit des données du réseau, il informe le processeur μBlaze avec une interruption en indiquant l'identifiant de l'émetteur. À la réception de cette interruption le processeur autorise l'adaptateur à écrire les données reçues dans la zone mémoire réservée à cet émetteur, si l'accès au bloc mémoire BRAM est possible à ce moment.

6.4 Conclusion

Dans ce chapitre, nous avons décrit les différents environnements que nous avons développés pour concevoir, simuler et implanter le réseau μSpider II. L'environnement de conception aide le concepteur du réseau à définir la structure du NoC, et les contraintes des communications, il permet également d'automatiser la génération de la description matérielle du NoC pour empêcher les erreurs dues à l'édition manuelle.

Ce chapitre détaille également les environnements de simulation et d'implantation qui permettent de tester le fonctionnement et les performances du réseau.

7

Résultats d'implantation du NoC μSpider II

Nous avons présenté, dans les chapitres précédents, l'architecture et le flot de conception du NoC μSpider II, ainsi qu'une nouvelle génération de réseaux sur puce reconfigurables dynamiquement (RNoC). Le but de ce dernier chapitre est de vérifier la pertinence de l'intégration de la reconfiguration dynamique dans les NoCs, en comparant les performances du NoC statique et des RNoCs au sein de différents systèmes multiprocesseurs.

La première expérience détaillée dans la section 7.1 consiste à évaluer les performances des différentes versions du NoC dans un système multiprocesseur qui traite une application de détection de mouvements et suivi d'objets. Nous avons choisi ce type d'application car elle crée des d'échanges entre les processeurs avec des tailles de données importantes. La deuxième expérience décrite dans la section 7.2 utilise aussi une architecture multiprocesseur, en revanche, elle teste les différentes versions du NoC en réalisant un trafic simple et aléatoire avec des échanges de taille moyenne.

Pour implanter les différentes architectures des deux expériences, nous avons choisi la cible FPGA Xilinx de la famille Virtex-5 LXT et nous avons utilisé l'environnement ISE version 11.5 pour la synthèse. L'unité utilisée pour quantifier la surface des différentes architectures est l'unité "Slice". Cette appellation désigne le bloc logique, élémentaire et programmable qui constitue les FPGAs de Xilinx. L'architecture logique d'un "Slice" de la famille Virtex-5 est donnée dans l'annexe A.

7.1 Application détection et suivi d'objets

L'application détection de mouvements et suivi d'objets a été développée durant la thèse de Yvan Eustache [Eus08], et afin d'augmenter les performances du traitement de cette application, celle-ci a été partitionnée en différentes tâches pour profiter du parallélisme. Nous distinguons cinq tâches principales :

1. **L'extraction des objets**, cette étape réalise une moyenne pixel par pixel sur des images successives, puis elle effectue la différence avec une image de fond précédemment acquise, ensuite elle exécute un traitement de binarisation afin d'extraire les objets non présents dans l'image de fond ;
2. **L'ouverture**, cette tâche filtre les bruits de binarisation en réalisant deux traitements successifs, l'érosion qui élimine le bruit dû à la binarisation, et la dilatation qui permet de retrouver la forme approximative des objets en mouvement ;
3. **La reconstruction**, cette étape reconstitue la forme initiale des objets restants ;

4. **L'étiquetage** permet l'attribution d'une étiquette spécifique à chacun des objets présents dans l'image ;
5. **L'enveloppe**, cette tâche définit les coordonnées de l'enveloppe rectangulaire englobant chacun des objets, puis elle calcule les coordonnées du centre de gravité des objets, ces coordonnées permettent de mettre à jour l'image de fond en ajoutant à celle-ci les objets qui deviennent statiques.

Le tableau 7.1 regroupe les temps moyens d'exécution de ces tâches qui sont implantées en matériel. Ces temps sont calculés pour le traitement d'une image avec une résolution QVGA [54] ($320 * 240$ pixels), et chaque pixel est codé avec un bit car le traitement est réalisé sur des images binaires, ainsi la taille des images est 74 752 kbits. La mesure des temps d'exécution a été réalisée à l'aide d'un compteur matériel cadencé par l'horloge du système (50 MHz).

Tâches	Nombres de cycles
T_1	237 098 (15%)
T_2	21 054 (2%)
T_3	476 920 (31%)
T_4	376 036 (24%)
T_5	424 419 (28%)

TABLE 7.1 – Temps d'exécution moyens des tâches implantées en matériel de l'application "détection de mouvements et suivi d'objets"

L'implantation de cette application a pour but de tester les performances de la version statique et de la version reconfigurable du réseau μSpider II. Pour y parvenir, nous utilisons une architecture multiprocesseur afin d'exécuter les tâches de l'application en parallèle et de créer des échanges entre les processeurs. La section 7.1.1 décrit cette architecture, elle présente le partitionnement de l'application et le calcul des contraintes des communications. La section 7.1.2 présente le dimensionnement du NoC en se basant sur le flot de conception du réseau μSpider II et en utilisant les valeurs des bandes passantes des communications définies lors du partitionnement. La section 7.1.3 introduit l'utilisation de la reconfiguration dynamique sur le NoC statique, et elle présente l'architecture du RNoC, et le dimensionnement réalisé pour satisfaire les contraintes des communications. Enfin, la section 7.1.4 compare les performances des deux versions NoC et RNoC.

7.1.1 Architecture multiprocesseur

Nous avons partitionné les différentes tâches de l'application détection de mouvements et suivi d'objets sur neuf unités de traitement. Chaque unité de traitement est composée d'un processeur, d'un accélérateur matériel exécutant une ou plusieurs tâches de l'application, une mémoire de données locale pour stocker deux images et un bus qui connecte les différents éléments. Le processeur de chaque unité gère les communications, l'accès à la mémoire locale et contrôle l'accélérateur.

7.1.1.1 Partitionnement logiciel/matériel

Le partitionnement logiciel/matériel que nous avons retenu est le suivant :

54. Quarter Video Graphics Array

- les processeurs P_1 et P_2 prennent en charge la lecture des images, l'exécution de la tâche T_1 et l'affichage du résultat ;
- les processeurs P_3, P_4 et P_5 exécutent les tâches T_2 et T_3 ;
- et les processeurs P_6, P_7, P_8, et P_9 exécutent les tâches T_4 et T_5.

Le but principal de ce partitionnement est de partager les ressources de traitement selon le temps d'exécution des différentes tâches, Ainsi, nous allouons quatre processeurs pour exécuter en parallèle les tâches T_4 et T_5, car ces deux tâches représentent 52% du temps total d'exécution de l'application. En revanche, nous n'allouons que deux processeurs à la tâche T_1, car elle ne représente que 15% du temps total de l'exécution.

Ce partitionnement crée trois étages au niveau traitement. Le premier lit l'image, exécute la tâche T_1, et affiche l'image résultat. Le deuxième étage réalise les tâches T_2 et T_3, et le troisième exécute les tâches T_4 et T_5. De cette manière, l'architecture multiprocesseur réalise simultanément une partie de traitement sur trois images différentes.

Le premier étage comporte deux processeurs qui se partagent la lecture des différentes images, avec une règle qui impose la lecture des images paires par le processeur P_1, et les images impaires par le processeur P_2. L'association de cette règle avec les trois étages de traitement permet au système d'augmenter son débit, en réalisant en parallèle une partie du traitement sur six images différentes.

Ce parallélisme impose la définition de quelques règles afin d'afficher les images dans le bon ordre. Pour y parvenir, nous définissons les combinaisons suivantes entre le numéro de l'image traitée et les processeurs qui seront chargés de son traitement :

- les images $I_{(i*8)}$, $I_{(i*8)+2}$, $I_{(i*8)+4}$, et $I_{(i*8)+6}$ sont lues et traitées par le processeur P_1, puis elles sont respectivement envoyées aux processeurs P_3, P_5, P_5, et P_3 du deuxième étage. Elles sont ensuite traitées par ces processeurs et envoyées respectivement aux processeurs P_6, P_9, P_6, et P_9. Après leur traitement par les processeurs du troisième étage, elles sont renvoyées au processeur P_1 pour être affichées ;
- les images $I_{(i*8)+1}$, $I_{(i*8)+3}$, $I_{(i*8)+5}$, et $I_{(i*8)+7}$ sont lues et traitées par le processeur P_2, puis elles sont envoyées respectivement aux processeurs P_4, P_3, P_4, et P_5 du deuxième étage. Elles sont ensuite traitées par ces processeurs et envoyées respectivement aux processeurs P_8, P_7, P_7, et P_8. Après leur traitement par les processeurs du troisième étage, elles sont renvoyées au processeur P_2 pour être affichées.

En considérant une numérotation des images de 0 à n et $i \in \left\{0, \ldots, \lfloor \frac{n}{7} - 1 \rfloor\right\}$, la distribution précédente oblige les images $I_{(i*8)}$, $I_{(i*8)+1}$, $I_{(i*8)+2}$, $I_{(i*8)+3}$, $I_{(i*8)+4}$, $I_{(i*8)+5}$, $I_{(i*8)+6}$ et $I_{(i*8)+7}$ à parcourir les trois étages de traitement de manière différentes, car elles ne sont pas traitées par les mêmes processeurs. Le but de cette distribution est de garantir le bon ordre des images résultats lors de l'affichage, et la création d'un parallélisme afin d'augmenter la qualité de service de l'application.

Le tableau 7.2 décrit cette distribution pour les onze premières images, chaque colonne représente une image I_n et ses étapes de traitement. Par exemple, l'image I_0 est traitée en quatre étapes, la première permet au processeur P_1 de lire cette image, de la traitée en exécutant la tâche T_1 puis de la transmettre au processeurs P_3. À la deuxième étape l'image I_0 est traitée par le processeur P_3 et envoyée au processeur P_6. Ensuite, le processeur P_6 exécute les tâches T_4 et T_5 sur l'image I_0 puis il la renvoie au processeur P_1. La quatrième étape et la dernière permet d'afficher le résultat de l'image I_0 en parallèle de la lecture de l'image I_6.

Étapes \ Images	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}
1	$A(I_{-6})$, L $P_1(T_1)$ $C(P_1, P_3)$	$A(I_{-5})$, L $P_2(T_1)$ $C(P_2, P_4)$										
2	$P_3(T_{2,3})$ $C(P_3, P_6)$	$P_4(T_{2,3})$ $C(P_4, P_8)$	$A(I_{-4})$, L $P_1(T_1)$ $C(P_1, P_5)$	$A(I_{-3})$, L $P_2(T_1)$ $C(P_2, P_3)$								
3	$P_6(T_{4,5})$ $C(P_6, P_1)$	$P_8(T_{4,5})$ $C(P_8, P_2)$	$P_5(T_{2,3})$, $C(P_5, P_9)$	$P_3(T_{2,3})$ $C(P_3, P_7)$	$A(I_{-2})$, L $P_1(T_1)$ $C(P_1, P_5)$	$A(I_{-1})$, L $P_2(T_1)$ $C(P_2, P_4)$						
4			$P_9(T_{4,5})$ $C(P_9, P_1)$	$P_7(T_{4,5})$ $C(P_7, P_2)$	$P_5(T_{2,3})$ $C(P_5, P_6)$	$P_4(T_{2,3})$ $C(P_4, P_7)$	$A(I_0)$, L $P_1(T_1)$ $C(P_1, P_3)$	$A(I_1)$, L $P_2(T_1)$ $C(P_2, P_5)$				
5					$P_6(T_{4,5})$ $C(P_6, P_1)$	$P_7(T_{4,5})$, $C(P_7, P_2)$	$P_3(T_{2,3})$ $C(P_3, P_9)$	$P_5(T_{2,3})$ $C(P_5, P_8)$	$A(I_2)$, L $P_1(T_1)$ $C(P_1, P_3)$	$A(I_3)$, L $P_2(T_1)$ $C(P_2, P_4)$		
6							$P_9(T_{4,5})$, $C(P_9, P_1)$	$P_8(T_{4,5})$ $C(P_8, P_2)$	$P_3(T_{2,3})$ $C(P_3, P_6)$	$P_4(T_{2,3})$ $C(P_4, P_8)$	$A(I_4)$, L $P_1(T_1)$ $C(P_1, P_5)$	$A(I_5)$, L $P_2(T_1)$ $C(P_2, P_3)$

L : la lecture de l'image courante
$P_x(T_a, ..., T_b)$: la réalisation des tâches T_a,...,T_b sur l'image courante par le processeur P_x
$C(P_x, P_y)$: la transmission de l'image courante du processeur P_x vers les processeurs P_y
$A(I_n)$: affichage de l'image résultat I_n

TABLE 7.2 – Description des parcours des images dans l'architecture multiprocesseur, et explication du parallélisme réalisé au niveau des différentes étapes

Communications	Équation	Durée (s)	bande passante (Mbits/s)
$C(P_1, P_3)$, $C(P_2, P_4)$, $C(P_1, P_5)$, $C(P_2, P_3)$, $C(P_2, P_5)$	$D_{com} = \frac{2}{QoS} - (D_A + D_L + D_{T_1})$	0,019	3,85
$C(P_3, P_6)$, $C(P_4, P_8)$,$C(P_5, P_9)$, $C(P_3, P_7)$ $C(P_4, P_7)$, $C(P_3, P_9)$, $C(P_5, P_8)$, $C(P_5, P_6)$	$D_{com} = \frac{2}{QoS} - (D_{T_2} + D_{T_3})$	0,015	4,87
$C(P_6, P_1)$, $C(P_8, P_2)$,$C(P_9, P_1)$, $C(P_7, P_2)$	$D_{com} = \frac{2}{QoS} - (D_{T_4} + D_{T_5})$	0,009	8,11
QoS : qualité de service, D_X : temps d'exécution d'un traitement T_x ou la durée de transmission d'une communication C_x D_L : temps de lecture d'une image, et D_A : temps d'affichage d'une image			

TABLE 7.3 – Calcul des contraintes des communications en terme de bande passante pour une qualité de service égale à 80 images/s

Chaque ligne de ce tableau représente les images traitées, les processeurs actifs, et les communications réalisées entre les processeurs. Ce tableau permet de définir une chronologie du traitement, où chaque ligne est une étape qui réalise un ensemble de processus en parallèle selon le partitionnement logiciel/matériel que nous avons définit pour l'application. Par exemple l'étape 4 permet de réaliser une partie du traitement sur six images en parallèle I_2, I_3, I_4, I_5, I_6, et I_7. Chaque image est traitée par un processeur différent, ceci est vérifiable sur le tableau grâce au code couleurs que nous utilisons pour représenter les processeurs, en attribuant à chacun d'eux une couleur spécifique. Chaque étape permet également d'afficher le résultat de deux images, ainsi, l'étape 4 par exemple affiche le résultat des images I_0, I_1.

Le tableau7.2 illustre le partitionnement logiciel/matériel de l'application, en définissant à chaque étape les images traitées et les processeurs actifs. Cette organisation permet à l'architecture multiprocesseur de réaliser simultanément une partie du traitement sur six images différentes et d'afficher le résultat de deux images à chaque étape.

7.1.1.2 Calcul des contraintes des communications

La qualité de service de notre application dépend du nombre d'images traitées pas seconde. Dans notre cas, nous affichons le résultat de deux images à chaque étape. Donc c'est la durée de l'étape qui permettra de définir la qualité de service de notre application (Eq.7.1). En sachant que chaque étape est constituée de plusieurs processus exécutés en parallèle, alors la durée de l'étape formulée avec l'équation 7.2 est égale à la durée du processus qui a le temps d'exécution le plus long. Dans notre cas, chaque processus (cellule du tableau 7.2) exécute d'une manière séquentielle certaines tâches de l'application sur une image, puis il transmet celle-ci à un autre processeur. Ainsi, la durée d'un processus exprimée par l'équation 7.3 est égale à la somme des temps d'exécution des tâches qui le composent, à laquelle s'ajoute le temps de la transmission d'une image.

Par conséquent, l'équation de la qualité de service dépend principalement de la bande passante des communications car les temps d'exécution des tâches sont imposés. Alors, si la bande passante des communications augmente, nous améliorons la qualité de service de l'application.

$$QoS = \frac{2}{D_{step}} \tag{7.1}$$

$$D_{step} = Max(D_{process_0}, \ldots, D_{process_n}) \tag{7.2}$$

$$D_{process_i} = (D_{T_x} + \ldots + D_{T_y}) + D_{Com} \tag{7.3}$$

La bande passante des communications dépend de la qualité de service imposée par l'application et du temps d'exécution des tâches qui précédent la transmission de l'image. Le tableau 7.3 regroupe les équations qui permettent de calculer les durées de transmission ainsi que les bandes passantes des communications, en fixant la qualité de service de l'application détection de mouvements et suivi d'objets à 80 images/s.

Dans cette section, nous avons défini les ressources matérielles de l'architecture multiprocesseur que nous utiliserons pour exécuter l'application détection de mouvements et suivi d'objets. Nous avons distribué les différentes tâches de l'application sur ces ressources afin d'accélérer le traitement et de créer des communications entre les processeurs de l'architecture.

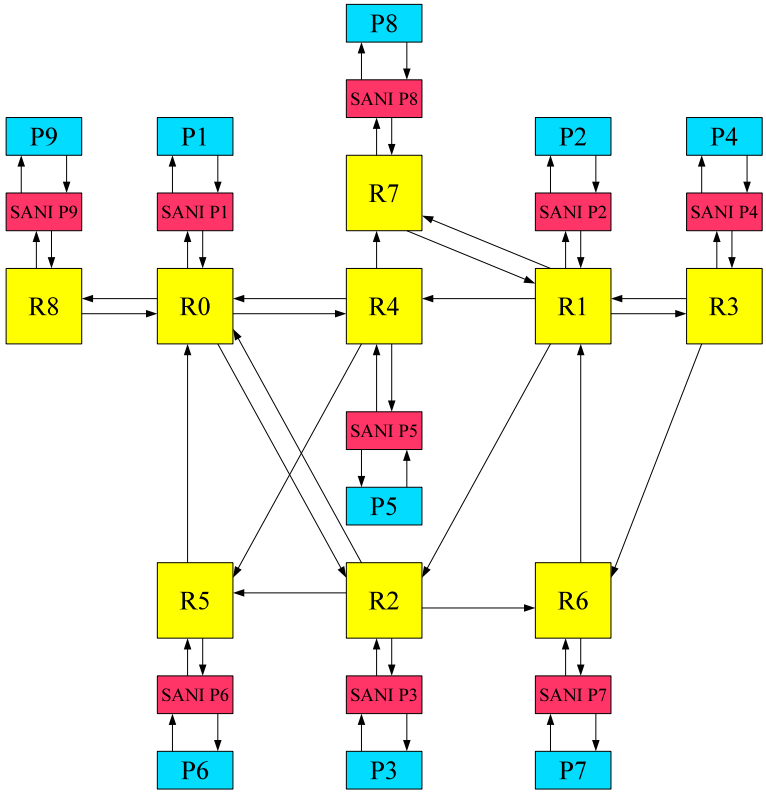


FIGURE 7.1 – Topologie spécifique du NoC générée par l'outil μSpider II

7.1.2 Dimensionnement du NoC

Le dimensionnement du NoC pour l'architecture multiprocesseur de l'application détection de mouvements et suivi d'objets dépend du flot de conception que nous avons présenté dans le chapitre 3, ce flot de conception est composé de cinq étapes :

1. l'étape de **spécification** définit la fréquence du NoC et les graphes CDGs de l'application. Nous fixons la fréquence du réseau à 50 Mhz, car les différentes tâches de l'application s'exécutent à cette fréquence. Nous définissons les graphes CDGs selon le partitionnement que nous avons spécifié dans la section précédente. Ainsi, nous allouons à chaque processeur de l'architecture un graphe CDG qui représente les dépendances entre ses tâches de

Topologie	Description	Surface (slices)	Longueur totale des chemins
Grille 2D	9 routeurs 42 liens	4168	45
Spidergon	10 routeurs 52 liens	5120	44
μSpider II	9 routeurs 36 liens	3760	37

TABLE 7.4 – Comparaison de deux topologies génériques avec la topologie spécifique générée par μSpider II

traitement et les communications qu'il établit avec les autres processeurs. Les différents graphes CDGs sont représentés dans l'annexe D ;

2. l'étape de **construction** calcule la bande passante réel des communication et elle conçoit la topologie du réseau. L'exécution de cette étape a permis la génération automatique d'une topologie spécifique pour notre NoC (Fig.7.1), en se basant sur la méthode de conception de topologie détaillée dans la section 3.3 du chapitre 3.
   Nous avons comparé cette topologie avec deux topologies génériques, la topologie grille 2D car elle est la plus utilisée dans les NoCs, et la topologie Spidergon, car elle est considérée parmi les plus performantes. Cette comparaison est réalisée selon deux critères, le premier est la surface occupée par la topologie, et le deuxième critère considère la longueur totale des chemins spatio-temporels. Le deuxième critère permet de comparer les performances des topologies, car la structure de la topologie influence les décisions prises par l'algorithme de calcul des chemins spatio-temporels, et la longueur des chemins a un impact sur la latence des communications, donc sur les performances du réseau. Ainsi, une topologie est considérée performante, si elle permet à l'algorithme de recherche des chemins spatio-temporels de calculer les chemins les plus courts.
   Le tableau de résultats 7.4 confirme que la topologie générée par l'outil µSpider II est la plus performante, car les chemins calculés sont plus courts. En plus, la surface occupée par la topologie spécifique est inférieure respectivement de $9,79\%$ et de $26,59\%$ par rapport à la surface occupée par la topologie grille 2D et la topologie Spidergon ;

3. l'étape de **dimensionnement de la table TDMA** détaillée dans la section 3.4 du chapitre 3 calcule le nombre d'intervalles de temps à réserver pour chaque communication selon sa bande passante. Pour réaliser ce calcul, nous avons spécifié à l'outil la largeur des liens ($LW = 32\ bits$).
   Le nombre d'intervalles calculé pour chacune des communications est détaillé dans la colonne "Table TDMA" du tableau 7.5. Cette table est constituée de 6 slots de temps, chaque interface réseau distribue ses slots de temps entre les communications émises, selon le nombre de slots alloués à chacune d'elles. L'emplacement de ces slots de temps dans la table TDMA est déterminé dans la dernière étape, conjointement avec la recherche des chemins spatio-temporels ;

4. L'étape de **dimensionnement des mémoires** détermine les tailles des mémoires tampons des interfaces réseau, ce calcul est détaillé dans la section 3.5 du chapitre 3. Afin que ce calcul soit réalisé, nous avons défini la valeur de la taille de la mémoire cache de l'adaptateur de protocole ($WM_i = 640\ bits$), et la valeur du temps d'accès du DMA pour lire ou écrire une quantité de données égale à la taille de la mémoire cache ($WD_i = 42\ cycles$). Les résultats de cette étape sont présentés dans la colonne "profondeur de la FIFO" du tableau 7.5 ;

5. L'étape d'**allocation des chemins** explore les chemins dans l'espace (topologie) et dans le temps (TDMA) pour trouver une solution de routage à toutes les communications. Pour ce problème l'heuristique itérative expliquée dans la section 3.6 du chapitre 3, a suffi pour trouver une solution au problème de routage. Nous avons représenté sur le tableau 7.5 et pour chaque interface réseau, la distribution des communication dans la table TDMA qui leurs impose un temps de départ précis, ainsi que les chemins spatio-temporels calculés pour garantir le trafic.

Interfaces réseau	Table TDMA	Type de communication	Communication	Profondeur de la FIFO	Chemin
SANI P1	$\|C_0\|C_0\|C_0\|C_1\|C_1\|C_1\|$	En émission	$C_0(P_1, P_3)$	720	P_1, R_0, R_2, IP_3
			$C_1(P_1, P_5)$	720	P_1, R_0, R_4, IP_5
		En réception	$C_2(P_6, P_1)$	8	
			$C_3(P_9, P_1)$	8	
SANI P2	$\|C_1\|C_2\|C_2\|C_0\|C_0\|C_1\|$	En émission	$C_0(P_2, P_3)$	1560	P_2, R_1, R_2, IP_3
			$C_1(P_2, P_4)$	1560	P_2, R_1, R_3, IP_4
			$C_2(P_2, P_5)$	1560	P_2, R_1, R_4, IP_5
		En réception	$C_3(P_7, P_2)$	8	
			$C_4(P_8, P_2)$	8	
SANI P3	$\|C_2\|C_2\|C_0\|C_0\|C_1\|C_1\|$	En émission	$C_0(P_3, P_6)$	1560	P_3, R_2, R_5, IP_6
			$C_1(P_3, P_7)$	1560	P_3, R_2, R_6, IP_7
			$C_2(P_3, P_9)$	1560	P_3, R_2, R_0, R_8, IP_9
		En réception	$C_3(P_1, P_3)$	8	
			$C_4(P_2, P_3)$	4	
SANI P4	$\|C_1\|C_0\|C_0\|C_0\|C_1\|C_1\|$	En émission	$C_0(P_4, P_7)$	720	P_4, R_3, R_6, IP_7
			$C_1(P_4, P_8)$	720	P_4, R_3, R_1, R_7, IP_8
		En réception	$C_2(P_2, P_4)$	2	
SANI P5	$\|C_0\|C_0\|C_1\|C_1\|C_2\|C_2\|$	En émission	$C_0(P_5, P_6)$	1560	P_5, R_4, R_5, IP_6
			$C_1(P_5, P_8)$	1560	P_5, R_4, R_7, IP_8
			$C_2(P_5, P_9)$	1560	R_4, R_0, R_8, IP_9
		En réception	$C_3(P_1, P_5)$	8	
			$C_4(P_2, P_5)$	4	
SANI P6	$\|X\|X\|X\|C_0\|C_0\|C_0\|$	En émission	$C_0(P_6, P_1)$	720	P_6, R_5, R_0, IP_1
		En réception	$C_1(P_3, P_6)$	4	
			$C_2(P_5, P_6)$	4	
SANI P7	$\|C_0\|C_0\|C_0\|X\|X\|X\|$	En émission	$C_0(P_7, P_2)$	720	P_7, R_6, R_1, IP_2
		En réception	$C_1(P_3, P_7)$	4	
			$C_2(P_4, P_7)$	8	
SANI P8	$\|X\|X\|X\|C_0\|C_0\|C_0\|$	En émission	$C_0(P_8, P_2)$	720	P_8, R_7, R_1, IP_2
		En réception	$C_1(P_4, P_8)$	8	
			$C_2(P_5, P_8)$	4	
SANI P9	$\|C_0\|C_0\|C_0\|X\|X\|X\|$	En émission	$C_0(P_9, P_1)$	720	P_9, R_8, R_0, IP_2
		En réception	$C_1(P_3, P_9)$	4	
			$C_2(P_5, P_9)$	4	

$C_i(P_x, P_y)$: la communications i entre le processeur P_x et le processeurs P_y
X : le slot de temps n'est pas attribué

TABLE 7.5 – Paramétrage des différentes interfaces réseau du NoC pour l'application détection de mouvements et suivi d'objets

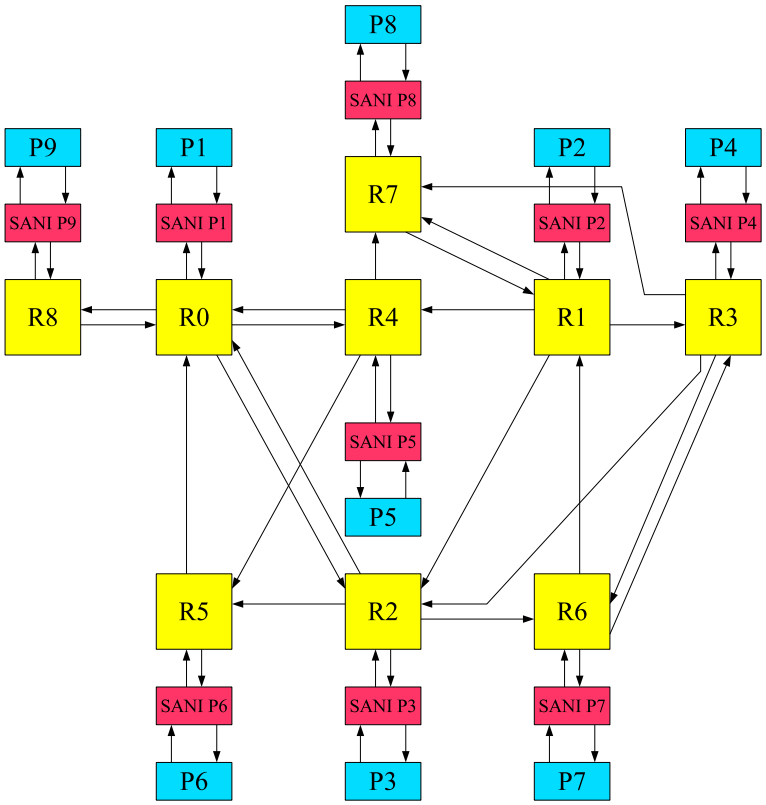


FIGURE 7.2 – Topologie spécifique du RNoC générée par l'outil µSpider II

7.1.3 Migration d'un NoC vers un RNoC

Le dimensionnement du NoC pour l'application détection de mouvements et suivi d'objets révèle que certaines communications ont un nombre de slots de temps plus élevé que le reste. Cet écart est dû aux règles imposées par la méthode de calcul des slots de temps, ainsi certaines communications profitent plus des tables TDMA, en utilisant 3 slots de temps au lieu de 2 slots, alors que les autres communications ne peuvent profiter d'aucune augmentation avec la configuration actuelle du NoC.

L'augmentation du nombre de slots de temps de quelques communications n'a pas d'impact sur la qualité de service de l'application réglée à 80 images/s, il permet seulement de diminuer le temps de transmission de l'image entre certains processeurs. Ceci est dû au fait que le partitionnement de l'application que l'on a défini dans la section précédente impose un parallélisme, notamment au niveau des transmissions. Par conséquent, pour augmenter la qualité de service de l'application, il faut augmenter la bande passante de toutes les communications réalisées au même instant.

Pour augmenter la bande passante des différentes communications, nous avons ajouter à certaines interfaces réseau le mécanisme de la reconfiguration dynamique des tables TDMA. L'introduction de ce mécanisme permet d'augmenter les slots de temps de toutes les communications en passant de 2 slots à 3, tout en respectant les règles de la méthode de calcul et de distribution des slots de temps.

Le tableau 7.6 détaille la nouvelle distribution des slots de temps et les chemins spatio-temporels des différentes communications. Ce nouveau dimensionnement prend en compte l'in-

troduction de la reconfiguration dynamique sur les interfaces réseau qui utilisent plusieurs tables TDMA.

Nous remarquons que les nouvelles contraintes introduites avec la reconfiguration ont modifié la topologie du réseau, la figure 7.2 illustre la nouvelle topologie générée par l'environnement μSpider II, qui permet à l'algorithme de calcul des chemins spatio-temporels de trouver une solution de routage pour le RNoC.

Cette nouvelle topologie contient plus de liens que la topologie utilisée pour le NoC. Le lien qui relie le routeur $R3$ au routeur $R7$, le lien qui relie le routeur $R6$ au routeur $R3$, et le lien qui relie le routeur $R3$ au routeur $R2$ ont été ajoutés. En revanche, le lien qui relie le routeur $R3$ au routeur $R1$ est supprimé dans la nouvelle topologie. L'augmentation du nombre de liens n'a pas eu d'impact sur la surface occupée par la topologie, comme le montre le tableau de comparaison 7.8. Ceci est dû à la suppression du lien qui relie le routeur $R3$ au routeur $R1$. Cette suppression diminue le nombre de ports du routeur $R1$, ce qui permet de diminuer le surface de celui-ci de $23,58\%$. Ce gain a compensé largement l'augmentation de la surface des autres routeurs, parce que, leur nombre de ports même après l'ajout des nouveaux liens reste inférieur au nombre de ports du routeur $R1$. Cette expérience justifie encore une fois que la surface du NoC ne dépend pas seulement du nombre de liens et du nombre de routeurs. Mais, elle dépend aussi du nombre de ports par routeur.

7.1.4 Comparaison des performances du NoC et du RNoC

Pour comparer les performances du réseau statique "NoC" et le réseau reconfigurable "RNoC". Nous avons utilisé l'environnement de simulation détaillé dans la section 6.2 du chapitre précédent. Cet environnement est composé de plusieurs générateurs de trafics qui simulent les transmissions entre les différents processeurs, en prenant compte les dépendances entre les processus de traitement et des communications que l'on a décrit dans les graphes CDGs. L'environnement de simulation permet également de relever pour chaque communication l'instant où la transmission de l'image commence et l'instant de fin de réception de l'image à l'aide d'un compteur synchrone. Grâce aux instants prélevés, nous calculons la durée de transmission et la bande passante de la communication. Le tableau 7.7 compare les durées de transmission des images que nous avons obtenues avec la version "NoC et la version "RNoC".

La comparaison des performances des deux versions du réseau prouve que l'introduction du mécanisme de configuration dynamique sur les tables multi-TDMAs compatibles diminue le temps de transmission de l'image entre les différents processeurs. Ce mécanisme augmente le nombre de slots de temps alloués aux différentes communications. Par exemple, l'interface réseau $SANI_2$ a 3 tables TDMAs compatibles, les deux tables $TDMA_1$ et $TDMA_2$ qui n'existent pas dans la version NoC, permettent aux communications gérées par la $SANI_2$ d'avoir 3 slots de temps au lieu de 2, ce qui permet d'augmenter la bande passante de ces communications de $33,33\%$. Par conséquent, l'utilisation du RNoC permet d'augmenter la qualité de service de l'application de $21,25\%$, en passant de 80 à 97 $images/s$.

De plus le mécanisme de configuration dynamique n'augmente pas la surface occupée par le réseau, comme le montre le tableau 7.8, la surface occupée par le RNoC est légèrement inférieure à celle occupée par le NoC. En effet, même si la configuration dynamique augmente la surface des SANIs de $3,6\%$, cette augmentation est compensée par le gain de surface que nous réalisons au niveau des routeurs. En effet, la nouvelle topologie utilisée par le RNoC réduit la surface des routeurs de $2,5\%$, car elle modifie le nombre de ports de certains d'entre eux.

Interfaces réseau	Tables TDMA	Communications	Chemin
SANI P1	$\|C_0\|C_0\|C_0\|C_1\|C_1\|C_1\|$	$C_0(P_1, P_3)$	P_1, R_0, R_2, IP_3
		$C_1(P_1, P_5)$	P_1, R_0, R_4, IP_5
SANI P2	$TDMA_0$	$C_0(P_2, P_3)$	P_2, R_1, R_2, IP_3
	$\|C_2\|C_2\|C_1\|C_1\|C_0\|C_0\|$	$C_1(P_2, P_4)$	P_2, R_1, R_3, IP_4
		$C_2(P_2, P_5)$	P_2, R_1, R_4, IP_5
	$TDMA_1$	$C_0(P_2, P_3)$	P_2, R_1, R_2, IP_3
	$\|C_1\|C_1\|C_1\|C_0\|C_0\|C_0\|$	$C_1(P_2, P_4)$	P_2, R_1, R_3, IP_4
	$TDMA_2$	$C_0(P_2, P_3)$	P_2, R_1, R_2, IP_3
	$\|C_1\|C_1\|C_1\|C_0\|C_0\|C_0\|$	$C_1(P_2, P_5)$	P_2, R_1, R_4, IP_5
SANI P3	$TDMA_0$	$C_0(P_3, P_6)$	P_3, R_2, R_5, IP_6
	$\|C_2\|C_0\|C_0\|C_1\|C_1\|C_2\|$	$C_1(P_3, P_7)$	P_3, R_2, R_6, R_3, R_2, R_6, IP_7
		$C_2(P_3, P_9)$	P_3, R_2, R_0, R_8, IP_9
	$TDMA_1$	$C_0(P_3, P_6)$	P_3, R_2, R_5, IP_6
	$\|C_1\|C_0\|C_0\|C_0\|C_1\|C_1\|$	$C_1(P_3, P_7)$	P_3, R_2, R_6, R_3, R_6, IP_7
	$TDMA_2$	$C_0(P_3, P_6)$	P_3, R_2, R_5, IP_6
	$\|C_1\|C_0\|C_0\|C_0\|C_1\|C_1\|$	$C_1(P_3, P_9)$	P_3, R_2, R_0, R_8, IP_9
SANI P4	$\|C_1\|C_1\|C_1\|C_0\|C_0\|C_0\|$	$C_0(P_4, P_7)$	P_4, R_3, R_6, IP_7
		$C_1(P_4, P_8)$	P_4, R_3, R_7, IP_8
SANI P5	$TDMA_0$	$C_0(P_5, P_6)$	P_5, R_4, R_5, IP_6
	$\|C_0\|C_2\|C_2\|C_1\|C_1\|C_0\|$	$C_1(P_5, P_8)$	P_5, R_4, R_7, IP_8
		$C_2(P_5, P_9)$	P_5, R_4, R_0, R_8, IP_9
	$TDMA_1$	$C_0(P_5, P_6)$	P_5, R_4, R_5, IP_6
	$\|C_0\|C_1\|C_1\|C_1\|C_0\|C_0\|$	$C_1(P_5, P_8)$	P_5, R_4, R_7, R_1, R_7, IP_8
	$TDMA_2$	$C_0(P_5, P_6)$	P_5, R_4, R_5, IP_6
	$\|C_0\|C_1\|C_1\|C_1\|C_0\|C_0\|$	$C_1(P_5, P_9)$	P_5, R_4, R_0, R_8, IP_9
SANI P6	$\|X\|X\|X\|C_0\|C_0\|C_0\|$	$C_0(P_6, P_1)$	P_6, R_5, R_0, IP_1
SANI P7	$\|X\|X\|C_0\|C_0\|C_0\|X\|$	$C_0(P_7, P_2)$	P_7, R_6, R_1, IP_2
SANI P8	$\|C_0\|C_0\|X\|X\|X\|C_0\|$	$C_0(P_8, P_2)$	P_8, R_7, R_1, IP_2
SANI P9	$\|C_0\|C_0\|C_0\|X\|X\|X\|$	$C_0(P_9, P_1)$	P_9, R_8, R_0, IP_2

TABLE 7.6 – Paramètrage des interfaces réseau du RNoC pour l'application détection de mouvements et suivi d'objets

	NoC		RNoC	
Communications	Durée	Bande passante	Durée	Bande passante
$C(P_1, P_3)$	0,0046	15,87	0,0046	15,87
$C(P_1, P_5)$	0,0046	15,87	0,0046	15,87
$C(P_2, P_3)$	0,009	8,11	**0,0046**	**15,87**
$C(P_2, P_4)$	0,009	8,11	**0,0046**	**15,87**
$C(P_2, P_5)$	0,009	8,11	**0,0047**	**15,54**
$C(P_3, P_6)$	0,009	8,11	**0,0046**	**15,87**
$C(P_3, P_7)$	0,009	8,11	**0,0046**	**15,87**
$C(P_3, P_9)$	0,009	8,11	**0,0047**	**15,54**
$C(P_4, P_7)$	0,0046	15,87	0,0046	15,87
$C(P_4, P_8)$	0,0046	15,87	0,0046	15,87
$C(P_5, P_6)$	0,009	8,11	**0,0046**	**15,87**
$C(P_5, P_8)$	0,009	8,11	**0,0046**	**15,87**
$C(P_5, P_9)$	0,009	8,11	**0,0047**	**15,54**
$C(P_6, P_1)$	0,0046	15,87	0,0046	15,87
$C(P_7, P_2)$	0,0046	15,87	0,0046	15,87
$C(P_8, P_2)$	0,0046	15,87	0,0046	15,87
$C(P_9, P_1)$	0,0046	15,87	0,0046	15,87
Qualité de service (images/s)	80		97 (+21,25%)	

TABLE 7.7 – Comparaison des performances du NoC et du RNoC

7.2 Implantation du NoC μSpider II sur une plateforme FPGA

Durant cette thèse, nous avons étudié quatre versions du NoC μSpider II, une version classique, une version RNoC avec des FIFOs reconfigurables dynamiquement, une version RNoC utilisant le principe de reconfiguration dynamique des tables multi-TDMA compatibles, et une version avec des FIFOs et des tables TDMAs reconfigurables dynamiquement. Afin de comparer les performances de ces quatre versions du NoC μSpider II, nous les avons implantées matériellement sur une plateforme FPGA au sein de la même architecture multiprocesseur.

Ces implantations permettent d'un côté de confirmer et de valider les résultats obtenus en simulation pour l'application détection de mouvements et suivi d'objets. D'un autre coté, elles permettent de construire un démonstrateur qui accompagnera et complétera l'environnement de conception μSpider II. La description de la carte et de l'interface graphique de ce démonstrateur sont présentés dans l'annexe E.

7.2.1 Architecture du système multiprocesseur

L'architecture du système multiprocesseur de cette implantation représentée sur la figure 7.3 est constituée de quatre unités de traitement reliées par le réseau d'interconnexion μSpider II. Chaque unité de traitement est composée d'un processeur de type μBlaze, d'un bloc mémoire

Composants	NoC (Slices)	RNoC (Slices)
R_0	369	339
R_1	458	350
R_2	352	335
R_3	254	326
R_4	329	306
R_5	193	171
R_6	198	241
R_7	186	221
R_8	171	158
Total Routeurs	2510	2447 (-2,5%)
$SANI_{IP1}$	101	94
$SANI_{IP2}$	133	145
$SANI_{IP3}$	136	156
$SANI_{IP4}$	100	103
$SANI_{IP5}$	129	146
$SANI_{IP6}$	103	100
$SANI_{IP7}$	104	102
$SANI_{IP8}$	104	100
$SANI_{IP9}$	94	94
Total SANIs	1004	1040 (+3,6%)
Autre	246	243
Total	3760	3730 (-0,8%)

TABLE 7.8 – Comparaison de la surface occupée par le NoC et par le RNoC

BRAM [55] de 64 ko [56], d'un adaptateur réseau, et d'un bus PLB [57]. Le µBlaze est un cœur de processeur softcore à jeu d'instruction réduit RISC [58] et entièrement 32 bits. Dans cette cette architecture, nous utilisons des µBlazes $v7.20.d$, et nous leur connectons deux blocs mémoires de 64 ko, un bloc mémoire d'instructions et un bloc mémoire de données. L'adaptateur de protocole permet aux unités de traitement de communiquer en envoyant et en recevant des données via le NoC, l'architecture de l'adaptateur de protocole développée pour les plateformes FPGA du fabriquant Xilinx ainsi que les procédures d'échanges sont décrites précédemment dans la section **??** du chapitre **??**.

Le NoC (Fig.7.3) conçu pour relier les unités de traitement de l'architecture multiprocesseur est constitué de sept routeurs connectés avec une topologie irrégulière. Il utilise des liens avec une largeur de 34 bits, 2 bits pour l'étiquetage des paquets et 32 bits pour transporter des données. Il contient aussi quatre interfaces réseau auto-adaptatives, chacune d'elle est connectée à l'adaptateur de protocole de l'une des quatre unités de traitement du système multiprocesseur.

55. Block Random Access Memory
56. kilooctet
57. Processor Local Bus
58. Reduced Instruction Set Computer

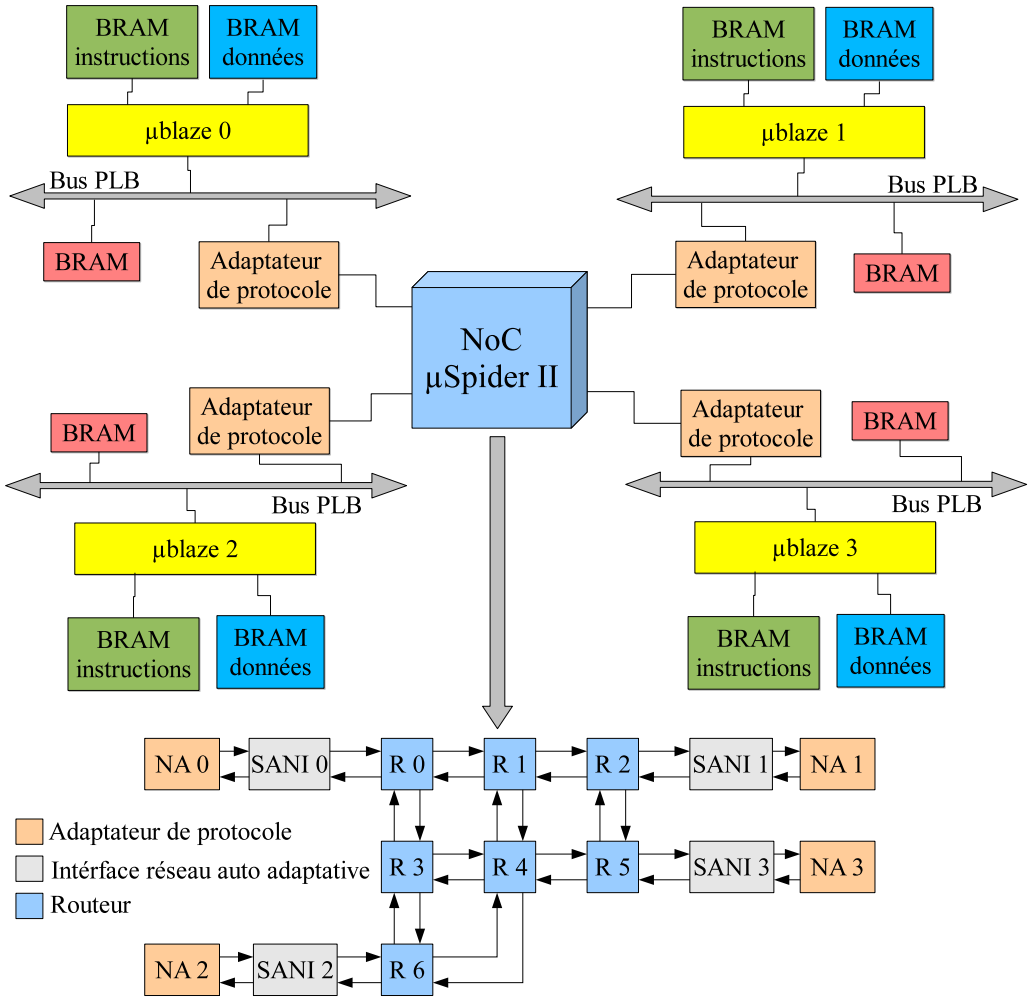


FIGURE 7.3 – Architectures du système multiprocesseur et du réseau µSpider II implémentées sur la plateforme FPGA

7.2.2 Tests et résultats

Le développement d'une application avec plusieurs tâches partitionnées et exécutées sur les différentes unités de traitement de l'architecture multiprocesseur pour créer des échanges de données à travers le NoC n'est pas obligatoire pour tester les performances d'un NoC. Il suffit, en effet, de créer un trafic aléatoire en réalisant plusieurs communications entre les différentes unités de traitement, pour calculer les bandes passantes réelles de ces communications. Donc, nous avons choisi de tester les différentes versions du NoC en utilisant un trafic simple sans exécuter une application réelle. Ce trafic est illustré sur la figure 7.4 avec les graphes de dépendance des communications, chaque graphe représente les communications d'une unité de traitement ainsi que l'ordre d'exécution de ces communications. Les graphes représentés avec les couleurs jaune, bleu, vert, et rouge décrivent respectivement les CDGs des unités de traitement $\mu Blaze_0$, $\mu Blaze_1$, $\mu Blaze_2$, et $\mu Blaze_3$.

Nous programmons chaque processeur $\mu Blaze$ de manière à réaliser les communications et respecter les dépendances décrites dans son graphe CDG. Nous utilisons les fonctions expliquées

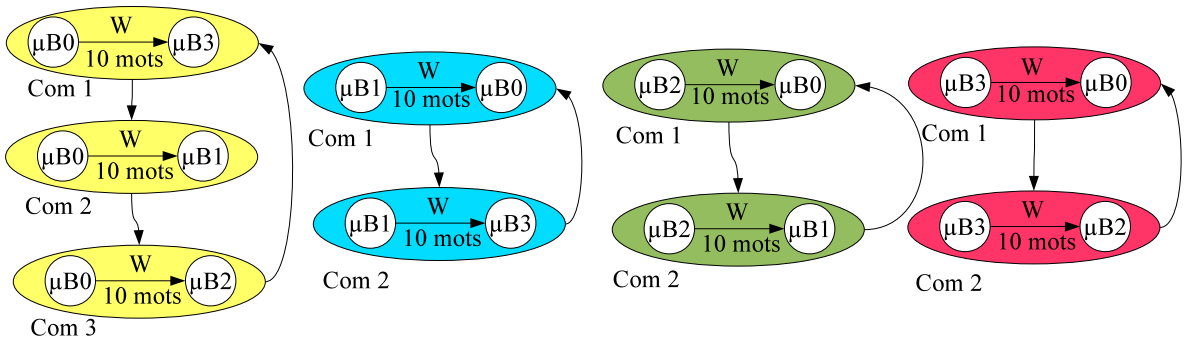


FIGURE 7.4 – Graphes de dépendance des communications du trafic réalisé dans le système multiprocesseur implémenté sur la plateforme FPGA

dans la section 6.3 du chapitre précédent, pour programmer les communications et calculer leur bande passante.

Pour comparer les différentes versions du NoC, nous avons choisi d'activer seulement l'auto-adaptation de l'interface réseau $SANI_0$ qui connecte l'unité de traitement $\mu Blaze_0$ au NoC. Ceci nous permet d'analyser et de comparer les débits des communications de l'unité de traitement $\mu Blaze_0$ selon les quatre versions suivantes :

- la version 1, **statique ou non reconfigurable** : nous considérons les débits des communications de cette version comme des références pour comparer les performances des versions reconfigurables ;
- la version 2, **FIFOs reconfigurables** utilise le principe de la reconfiguration dynamique des mémoires tampons de l'interface réseau $SANI_0$;
- la version 3, **TDMA reconfigurable** introduit l'approche de la reconfiguration dynamique des tables multi-TDMAs compatibles dans l'interface réseau $SANI_0$;
- et la version 4, **FIFOs et TDMA reconfigurables** associe les approches de reconfiguration des deux dernières versions.

Nous avons choisi d'observer la $SANI_0$, car le graphe CDG associé au $\mu Blaze_0$ est le seul à modéliser des communications vers tous les autres $\mu Blazes$, elle est donc représentative d'un cas où l'auto-adaptation peut tirer profit des partages de slots de temps et d'espace FIFO entre plusieurs destinataires. De plus, pour cette application, les communications des autres $SANI$ sont similaires à celles de la $SANI_0$ en terme de contraintes.

Par ailleurs, plus prosaïquement, le fait qu'une seule liaison série soit disponible sur la carte FPGA et que celle-ci soit contrôlée par un seul processeur complique énormément l'observation de plusieurs SANIS simultanément.

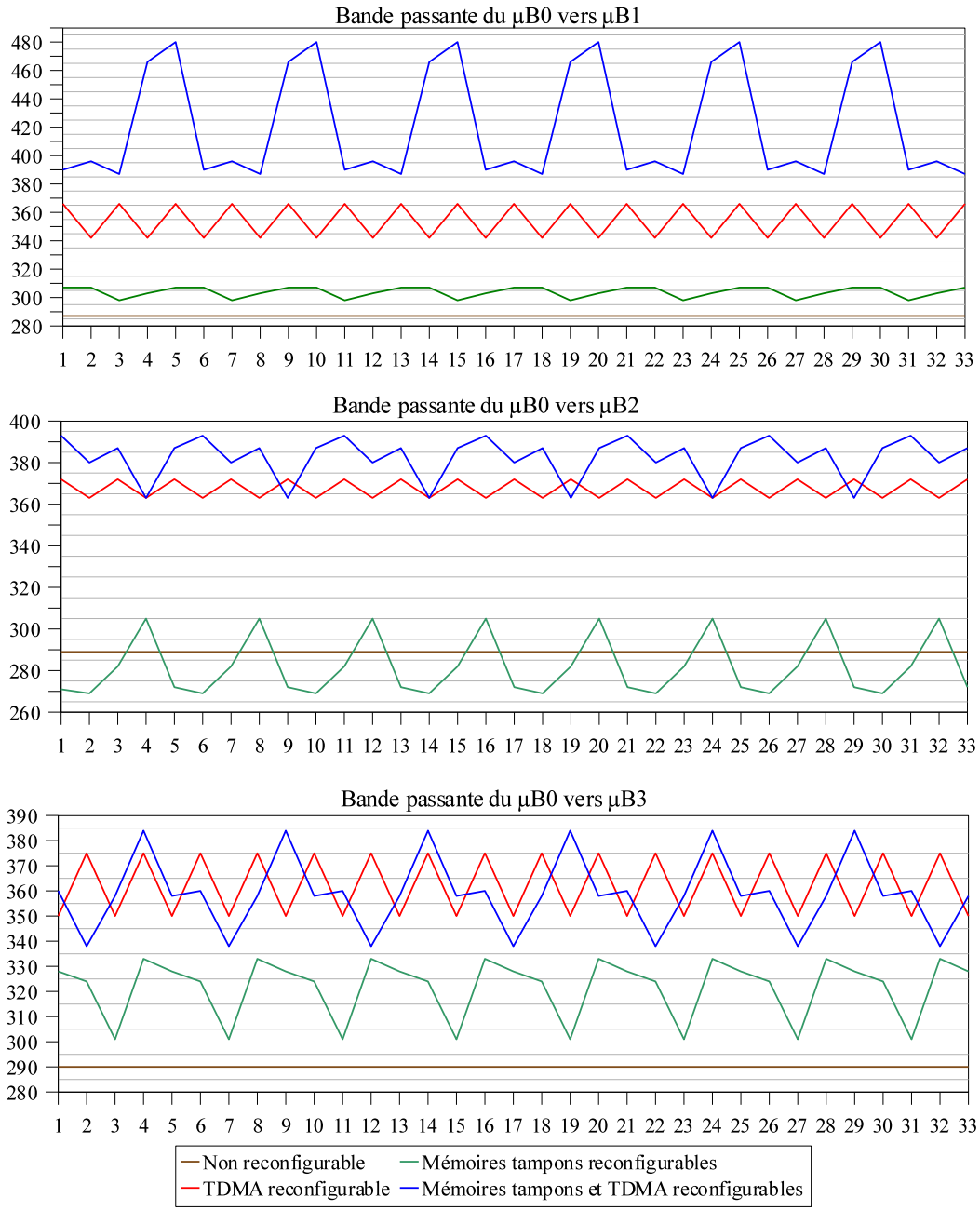


FIGURE 7.5 – Comparaison des performances des différentes versions du NoC implémentées sur une plateforme FPGA

Afin d'étudier avec précision le comportement du NoC et l'administration de la reconfiguration dynamique, nous avons répété l'exécution des graphes CDGs sur N itérations, et sans initialiser les paramètres du NoC à chaque itération. Ceci, nous permet de tester le NoC avec des solutions initiales différentes et qui changent selon les adaptations réalisées dans les itérations précédentes. Dans ce cas, le nombre d'itérations N est égale à 33, ce qui nous permet de représenter les bandes passantes des communications émises par la $SANI_0$ sur une large période

	Bande passante (Mbits/s)			
Communications	Version statique	Version FIFOs configurables	Version TDMA configurable	Version FIFOs et TDMA configurables
$C_0 : \mu Blaze_0 \rightarrow \mu Blaze_1$	287	$303,85$ $(5,87\%)$	$354,37$ $(23,47\%)$	$420,82$ $(46,63\%)$
$C_1 : \mu Blaze_0 \rightarrow \mu Blaze_2$	289	$281,67$ $(-2,54\%)$	$367,15$ $(27,04\%)$	$382,42$ $(32,33\%)$
$C_2 : \mu Blaze_0 \rightarrow \mu Blaze_2$	290	$321,7$ $(10,93\%)$	$362,12$ $(24,87\%)$	$358,90$ $(23,76\%)$

TABLE 7.9 – Bandes passantes moyennes des communications émises par la $SANI_0$

de temps.

La figure 7.5 regroupe les trois graphes qui représentent l'évolution des bandes passantes des communications émises par la $SANI_0$. Chaque graphe représente une seule communication, et il contient quatre courbes qui décrivent l'évolution de la bande passante de cette communication par rapport à la version du NoC utilisée. Ainsi, la courbe marron, la courbe verte, la courbe rouge, et la courbe bleu représentent respectivement les versions, NoC statique, RNoC avec FIFOs configurable, RNoC avec TDMA configurable, et RNoC avec FIFOs et TDMA configurables.

À la lecture de ces courbes, nous remarquons que le débit de toutes les communications évolue dans le temps pour les versions reconfigurables du NoC. Cette évolution est répétée tous les x itérations d'exécution des CDGs, le fréquence de répétition x change par rapport à la version RNoC utilisée. Cette répétition de l'évolution des débits est due aux décisions de configuration prises par le gestionnaire local (LM) de l'administration. Ces décisions évoluent également et de la même façon, car elles sont liées aux paramètres de la $SANI_0$ qui changent d'une configuration à une autre.

Pour mieux comparer les performances des différentes versions du NoC µSpider II implantées sur la plateforme FPGA. Nous avons calculé la moyenne des bandes passantes des trois communications émises par la $SANI_0$, par rapport au nombre de fois que les graphes CDGs ont été exécutés. Nous avons rassemblé ces résultats dans le tableau 7.9 en indiquant le pourcentage de gain ou de perte obtenu par rapport à la version statique du NoC.

Avant toute analyse, il est important de préciser que de manière générale, une baisse de bande passante peut être observée pour une communication en particulier, s'il s'avère que celle-ci a été surdimensionnée. C'est aussi l'intérêt de la méthode que de permettre une réallocation de bande passante utile. Dans le cas présent, nous avons choisi un cas de figure ou chaque processeur envoie autant de données que la bande passante disponible lui permet. Donc, la métrique de bande passante est ici révélatrice d'une optimisation dans tous les cas.

L'étude de ce tableau montre que la version statique du NoC offre presque le même débit au trois communications. Cette égalité est due au dimensionnement du NoC statique qui alloue le même nombre de slots de temps aux trois communications, ainsi que la même profondeur pour leurs FIFOs.

La version du RNoC utilisant des FIFOs avec des profondeurs configurables permet une faible augmentation du débit des communications C_0 et C_2, et elle diminue le débit de la communication C_1. La différence de débit constatée entre les trois communications est due principalement à leur ordre d'exécution présenté dans le graphe CDG (Fig.7.4). Cet ordre d'exécution influe les décisions prises par le gestionnaire local (LM) pour configurer la taille des FIFOs. Dans ce cas, les décisions du LM, permettent à la communication C_2 de profiter plus que les deux autres du principe d'extension, car elle est souvent réalisée quand sa FIFO adjacente est vide. Par contre, ces décisions diminuent la bande passante de la communications C_1, car sa FIFO emprunte des

Version du NoC	Statique	FIFOs configurables	TDMA configurable	FIFOs et TDMA configurables
Surface (slices)	2190	2212 (1%)	2082 (-4,94%)	2017 (-7,9%)
Surface (LUTs)	3942	4060 (3%)	3982 (1%)	4101 (4%)
Bande passante totale (Mbits/s)	866	907,22 (4,76%)	1083,64 (25,13%)	1162,14 (34,19%)
Efficacité (BP/LUTs)	0,219	0,223 (2,03%)	0,272 (24,26%)	0,283 (29,39%)

TABLE 7.10 – Comparaison de la surface occupée par les différentes versions du NoC implantées sur la plateforme FPGA

emplacements qu'elle n'arrive pas à récupérer lorsqu'elle en a besoin.

En revanche, le débit des trois communications augmente avec la version RNoC qui intègre à la $SANI_0$ le mécanisme de reconfiguration dynamique de la table TDMA. Nous remarquons que l'augmentation du débit est presque la même pour les trois communications. Cette similitude est due à la distribution équitable des slots de temps dans les différentes tables TDMAs utilisées par la $SANI_0$. Ainsi, le LM peut allouer aux trois communications le maximum et le même nombre de slots de temps quelque soit la situation, en passant d'une table TDMA à une autre.

Pour la quatrième version, nous avons voulu tester le comportement du gestionnaire local quand la $SANI_0$ intègre les deux approches de reconfiguration. Ce mélange s'avère très intéressant car le gain obtenu est plus élevé par rapport aux gains réalisés avec les autres versions. En revanche, il diminue le débit de la communication C_2 par rapport au débit obtenu avec la version TDMA configurables. Cette diminution est due principalement aux décisions différentes que le LM prend pour contrôler la taille des FIFOs dans les versions 2 et 4. En effet, dans la version 2, le LM augmente la profondeur de la FIFO allouée à la communication C_2, alors que dans la version 4, il la diminue.

Ce changement de comportement du LM pour la gestion de la taille des FIFOs est normal. Le LM adapte en effet sa stratégie de reconfiguration aux paramètres de la $SANI_0$ qui changent d'une version à une autre. Dans ce cas, c'est la distribution et le nombre de slots de temps alloué aux trois communications qui déclenchent cette adaptation.

Pour compléter la comparaison des différentes versions du RNoC, nous les avons évaluées selon la surface qu'elles occupent. Comme le montre le tableau 7.10, nous avons réalisé cette comparaison selon deux unités, l'unité "slice" et l'unité "LUT". Nous nous sommes rendu compte que la comparaison de la surface avec l'unité "slice" ne reflète pas la réalité, car l'algorithme de synthèse optimise l'architecture en réutilisant les slices employées initialement comme des simples registres. En revanche, la comparaison de la surface avec le nombre de tables de correspondance (LUT) utilisées permet d'évaluer plus précisément la logique combinatoire utilisée par le réseau.

Ce tableau montre que l'augmentation de la surface du réseau due à la reconfiguration dynamique est acceptable par rapport au gain obtenu en terme de débit. Il montre aussi que l'intégration des FIFOs avec des profondeurs configurables a un impact sur la surface occupée plus important que l'intégration de la table TDMA configurable.

Nous avons également comparé dans le tableau 7.10, le taux d'efficacité des différentes versions du réseau. Cette efficacité est le ratio de la bande passante totale des communications gérées par la $SANI_0$, et le nombre de LUTs occupé par le réseau. Nous observons bien que les réseaux reconfigurables dynamiquement sont plus efficaces que la version statique.

7.3 Conclusion

Les résultats obtenus avec les deux expériences de ce chapitre permettent d'affirmer que :

- la méthode de conception de topologie génère une solution optimisée, car d'une part elle occupe moins de surface que les topologies génériques. D'autre part, sa structure facilite et optimise la recherche des chemins spatio-temporels ;
- les mécanismes de reconfiguration dynamique permettent au réseau μSpider II d'être plus efficace, car la version RNoC augmente le débit du réseau dans la majorité des situations. De plus, le coût engendré par le RNoC sur la surface initialement occupée par le NoC est dérisoire par rapport au gain obtenu ;
- l'administration de la reconfiguration adapte son comportement selon les paramètres du réseau, les stratégies de reconfigurations intégrées, et les changement des contraintes de l'application.

Ces résultats sont généralisables à d'autres applications dans la mesure où celles-ci présentent des opportunités d'auto-adaptation à savoir des slots de temps inutilisés ou des espaces mémoire en FIFO non occupés. Ce type de situation est classique car en pratique un émetteur connecté au réseau réalise un seul transfert à la fois.

La portée de ces opportunités dépend du choix des emplacements des FIFO contigües en mémoire. Il s'agit là d'une perspective de ce travail qui résiderait dans une étape d'optimisation supplémentaire à intégrer au flot existant.

Conclusion générale

L'objectif principal affecté à cette thèse dans le projet AFANA était d'étudier et de concevoir des réseaux sur puce, en introduisant des mécanismes de reconfiguration dynamique, afin que celui-ci puisse s'adapter aux contraintes des communications de la nouvelle génération des systèmes sur puce. Les verrous à lever avec cette nouvelle génération des SoCs résident d'une part dans le nombre croissant des ressources hétérogènes que l'on peut intégrer dans la même puce et d'autre part dans la multiplicité des applications que l'on peut exécuter sur la même puce. Par conséquent, l'augmentation des ressources demande la conception d'un réseau performant, et la diversité ainsi que l'indéterminisme des applications imposent au NoC d'adopter un comportement qui s'adapte en temps réel aux changements des contraintes des communications. Sans cette adaptation du comportement, la conception d'un NoC qui répond à des contraintes évolutives devient très coûteuse car il doit être dimensionner au pire cas.

Nous avons commencé cette étude en réalisant un état de l'art sur les NoCs dans le chapitre 1. Pour y parvenir, nous avons identifié les paramètres des NoCs, en les classant conformément aux cinq couches du modèle OSI. Cette classification a permis de déterminer deux types de paramètres. Les paramètres architecturaux qui permettent de choisir la topologie, le placement des communicants, la largeur des liens et la taille des mémoires. Le deuxième type de paramètre détermine les protocoles utilisés par le NoC pour acheminer les données, tels que, la granularité de transmission, l'algorithme de routage, la politique de mémorisation, et l'arbitrage. Cette comparaison, que nous avons réalisée sur une sélection de NoCs par rapport à leur paramètres, a permis d'identifier trois classes. Des NoCs avec une qualité de service qui garantit le trafic (GT), en réservant certaines ressources du réseau à une communication pour garantir un débit ou/et une latence quelque soit la charge du réseau. Des NoCs où le trafic est acheminé au mieux (BE), cette classe ne réserve aucune ressource ce qui, par conséquent, n'offre aucune garantie de latence ou de débit. Et la troisième classe où le réseau intègre les deux types de trafic GT et BE.

La deuxième étape de notre démarche consistait à choisir le type de NoC, ses paramètres et son architecture. Après consultation avec les partenaires du projets AFANA, nous avons choisi de concevoir un NoC de type GT pour deux raisons. La première est due aux applications traitées dans le projet, où l'on se doit de satisfaire des contraintes imposées, en garantissant l'intégrité et le débit des communications. La deuxième raison vise à expliquer l'intérêt de la conception des NoCs reconfigurables dynamiquement car, l'étude approfondie des réseaux de type GT permet d'en montrer leurs limites en termes de flexibilité dans un contexte dynamique. Ainsi dans le deuxième chapitre, nous avons défini les paramètres du NoC μSpider II, et l'architecture des ses composants. Et dans le chapitre 3, nous avons détaillé le flot de conception et les algorithmes développés pour le dimensionner.

La conception du NoC μSpider II a débuté par spécification d'une topologie ad hoc, construite suivant le type de l'application, l'adressage mémoire,et les protocoles d'échanges appliqués. Pour concevoir la topologie, nous avons développé un algorithme de génération des topologies (voir section 3.3.3), composé de trois étapes. La première étape consiste à créer une topologie initiale selon la classe de l'architecture multiprocesseur à laquelle appartient le SoC, aussi nous avons développé deux méthodes de conception, une pour les classes SASM et SADM (voir section 3.3.1), et une deuxième pour la classe DADM (voir section 3.3.2). La deuxième étape de l'algorithme de génération des topologies optimise la topologie initiale, en remplaçant les routeurs qui ont un nombre de ports important, par plusieurs routeurs avec moins de ports dans le but de réduire la taille de la topologie. En effet, nous avons démontré dans la section 2.2.3, que ce n'est pas le nombre de routeurs qui influe sur la taille du réseau mais le nombre de ports d'entrée/sortie par

routeur. Enfin, la troisième étape de l'algorithme de génération des topologies optimise la solution de la deuxième étape, en supprimant les liens et les routeurs non utilisés ou sous-utilisés par les chemins des différentes communications. Pour tester cette approche, nous avons conçu pour l'application "détection de mouvements et suivi d'objets" un NoC utilisant la topologie générée par cet algorithme, et nous l'avons comparé à des NoCs utilisant des topologies génériques. La comparaison a montré que notre topologie permet au NoC d'occuper moins de surface, de plus elle permet de calculer des chemins spatio-temporels plus courts (voir section 7.1.2).

Pour que notre NoC μSpider II soit de type GT, nous l'avons paramétré en utilisant une commutation par paquets, un routage déterministe, une politique de mémorisation *wormhole*, et un contrôle du flux de bout en bout qui repose sur une nouvelle méthode expliquée dans la section 2.1. Ce paramétrage a dirigé nos choix pour la conception des différents composants du NoC. Ainsi, nous avons développé une interface réseau composée principalement de mémoires tampon pour stocker les données des différentes communications et d'une table TDMA pour ordonnancer les temps de départ des communications afin d'empêcher les conflits dans les routeurs et pour garantir une bande passante minimum (voir section 2.3). Nous avons créé aussi un routeur composé de décodeurs, d'une table de routage et d'un arbitre. Nous l'avons conçu d'une manière à transférer le paquet en un seul cycle, avec la possibilité de traiter plusieurs paquets en simultané. De plus, son architecture n'utilise que peu de mémoire, car elle n'intègre qu'un seul registre par port d'entrée (voir section2.2). Enfin nous avons développé un adaptateur de protocole avec une architecture indépendante du standard utilisé par le communicant, cette architecture a la particularité d'utiliser un DMA pour accéder directement à la mémoire locale du communicant et d'une mémoire cache pour diminuer le nombre d'accès (voir section2.4).

Après la définition de l'architecture des différents composant du NoC μSpiderII, nous les avons dimensionnés selon un flot de conception. L'une des particularités de ce flot de conception est l'utilisation des graphes de dépendance des communications, pour modéliser les contraintes de l'application en terme de bande passante, et pour représenter d'une manière claire les dépendances entre les processus de traitements et d'échanges. Cette représentation nous a permis de différencier les échanges exécutés séquentiellement et ceux exécutés en parallèles. Ce qui permet de calculer avec précision la valeur de la bande passante des différentes communications (voir section 3.2). Nous avons développé également un algorithme pour calculer la taille maximale des mémoires tampons en émission et en réception des interfaces réseau (voir section 3.5). Ainsi qu'un algorithme pour calculer nombre de slots de temps nécessaire pour garantir la bande passante de chaque communication (voir section 3.4). Enfin, nous avons créé une heuristique pour la recherche des chemins spatio-temporels, en utilisant une méthode de construction des chemins parallèle et gloutonne (voir section 3.6).

Afin de garantir le trafic, nous avons conçu un NoC qui alloue des intervalles de temps et qui réserve des ressources réseau pour chaque communication dans le but de garantir sa bande passante. Cette solution rend le NoC statique car, il ne peut pas adapter son comportement dans un système dynamique. Pour y remédier, nous avons introduit des mécanismes de reconfiguration dynamique sur le NoC μSpider II afin de le rendre adaptatif. Ainsi dans le chapitre 4, nous avons défini un espace de conception pour les RNoCs en créant un nouveau modèle qui lie le mécanisme de reconfiguration dynamique, les paramètres des composants réseau, et les protocoles des couches OSI. Ce modèle a révélé trois parties essentielles pour l'intégration de la reconfiguration dans les NoCs. La partie commande, la partie récolte d'informations, et la partie opérative. De plus, grâce à la corrélation définie dans ce modèle, nous avons mis en lumière trois axes de recherches fondamentaux dans le domaine de la conception des RNoCs : l'axe d'administration, et les axes de reconfiguration de l'architecture et des protocoles. Nous avons étudié chacun des

axes en définissant son usage et sa place dans le mécanisme de la reconfiguration dynamique dans les RNoC, en réalisant une analyse préalable des travaux qui traitent certaines de ces problématiques, et en révélant plusieurs points pertinents dont le potentiel reste à explorer. Nous avons défini également les principes de la gestion de l'administration de la reconfiguration. Cette administration se distingue par l'intégration d'une gestion partagée entre un gestionnaire global (GM) et plusieurs gestionnaires locaux (LMs). Le LM est intégré à chacune des interfaces réseau, dans le but de réaliser des modifications locales simples et très rapides à exécuter. En revanche le GM est incorporé à l'un des cœurs de traitement connecté au RNoC, son rôle consiste à effectuer des changements globaux en mettant à jour les configurations gérées par les LMs.

Cette étude des RNoCs nous a permis de mettre la lumière sur de nouveaux mécanismes que nous avons intégré à notre NoC μSpider II pour le rendre adaptatif, et améliorer ses performances. Ainsi dans le chapitre 5, nous avons présenté l'approche des mémoires tampons configurables dynamiquement qui permet d'adapter la profondeur des FIFOs dans les interfaces réseau en temps réel et selon les besoins des communications. Nous avons introduit également dans ce chapitre le mécanisme qui permet de configurer dynamiquement la table d'ordonnancement "TDMA", en adaptant le nombre d'intervalles de temps alloué aux communications selon les besoins.

L'intégration du principe des FIFOs configurables dynamiquement dans les interfaces réseau du NoC consiste à organiser les FIFOs dans la même mémoire physique, ainsi, chacune des FIFOs peut augmenter sa profondeur en s'étendant sur ces voisines. La mise en œuvre de ce mécanisme de reconfiguration dynamique a permis d'adapter la taille des FIFOs selon les besoins et d'augmenter les performances en diminuant le temps d'attente des communications pour accéder au réseau. Ce mécanisme a permis aussi de diminuer la complexité du contrôleur des FIFOs, car les implantations réalisées ont montré que la surface occupée par le contrôleur baisse en moyenne de 41% en utilisant la nouvelle organisation des FIFOs.

L'introduction de la reconfiguration dynamique sur un ordonnanceur de type TDMA consiste à redistribuer les intervalles de temps sur les différentes communications organisées dans la table TDMA. Cette redistribution permet d'augmenter la bande passante de certaines communications, en revanche elle diminue également la bande passante des autres car le nombre d'intervalles de temps de la table TDMA ne change pas. Pour réaliser cette technique de reconfiguration sur la table TDMA, nous avons utilisé plusieurs tables TDMA avec des distributions des slots différentes. Ces distributions sont compatibles entre elles, car chaque distribution organise les instants de départ des paquets avec des chemins précalculés et différents des autres distributions. L'utilisation de plusieurs tables TDMA dans les interfaces réseau a ajouté une nouvelle contrainte pour le calcul des chemins spatio-temporels. Ainsi, nous avons modifié notre heuristique afin de prendre en compte cette nouvelle contrainte (voir section 5.2.2).

Pour intégrer ces mécanismes de reconfiguration dans notre réseau, nous avons créé une nouvelle génération d'interface que nous avons nommée interface réseau auto-adaptative (SANI). Ainsi, nous avons modifié l'architecture des NIs pour qu'elles puissent gérer la reconfiguration, et nous leur avons ajouté également un module, nommé gestionnaire local (LM) qui permet d'administrer les changements réalisés sur la taille des différentes FIFOs et la distribution des slots de temps dans la table TDMA. Nous avons défini également le rôle du gestionnaire global (GM) dans notre RNoC μSpider II, qui permet de recalculer les chemins spatio-temporels du NoC dans le cas où certaines communications changent leurs caractéristiques. La solution heuristique offre des résultats satisfaisants avec des temps de calcul faibles, mais dans une démarche d'optimisation, nous avons développé également une métaheuristique qui repose sur un algorithme évolutionnaire et qui permet d'avoir des temps d'exécution plus faibles.

Dans le chapitre 7 nous avons vérifié la pertinence de l'intégration de la reconfiguration dy-

namique dans les NoCs, en comparant les performances du NoC statique et des RNoCs au sein de différents systèmes multiprocesseurs complexes. Ainsi, l'expérience d'implantation de l'application "détection d'objets et suivi de mouvements" dans un système multiprocesseur utilisant un réseau sur puce, a montré que l'introduction du mécanisme de configuration dynamique sur les tables multi-TDMAs compatibles diminue le temps de transmission de l'image entre les différents processeurs. Ainsi, l'utilisation du RNoC permet d'augmenter la qualité de service de l'application de $21,25\%$, en passant de 80 à 97 $images/s$. De plus le mécanisme de configuration dynamique n'augmente pas la surface occupée par le réseau (voir section 7.1.4). Nous avons réalisé aussi une deuxième expérience où nous avons testé les différentes versions du NoC dans un système multiprocesseur implanté sur une plate-forme FPGA. Cette expérience a montré également que l'utilisation du RNoCs permet d'augmenter les débit de $34,19\%$ avec une augmentation faible de la surface occupée qui est de l'ordre de 4%.

Au cours de ce travail, nous avons développé également différents environnements pour concevoir, simuler et implanter le réseau μSpider II. L'environnement de conception, constitué de plusieurs outils CAO, aide le concepteur du réseau à définir la structure du NoC, et les contraintes des communications, il permet également d'automatiser la génération de la description matérielle du NoC pour empêcher les erreurs dues à l'édition manuelle. Les environnements de simulation et d'implantation qui permettent de tester le fonctionnement et les performances du réseau. Nous avons créé aussi un démonstrateur sur une plateforme FPGA, qui permet d'accompagner l'environnement de conception et montrer la pertinence et la faisabilité de notre approche.

Perspectives

Le principe de configuration de la taille des FIFO dans les interfaces réseau a permis d'optimiser le taux de remplissage. Mais le succès de ce mécanisme dépend de l'emplacement des FIFOs les unes par rapport aux autres. Pour l'instant ce placement est déterminé pendant la phase du dimensionnement des FIFOs en utilisant une méthode de déduction. Cette méthodologie de placement est basique car elle se base seulement sur les dépendances des communications définies dans les graphes CDG. Ainsi, nous présentons l'optimisation de cette méthodologie de placement comme l'une des perspectives de ce travail. L'une des solutions pour optimiser ce placement est l'intégration d'un nouveau mécanisme de reconfiguration, qui permet de permuter en temps réel les emplacements des FIFOs selon les besoins réels d'extensions.

Une autre perspective consiste à tester le mode d'administration global, en implémentant le GM dans l'un des cœurs d'un système multiprocesseurs afin de tester son comportement. Cette expérience était prévue dans les objectifs du projet AFANA, mais malheureusement, elle n'a pas été réalisée, en raison de l'indisponibilité d'un partenaire.

Nous prévoyons aussi d'étendre l'intégration de la reconfiguration dynamique aux trafics de type BE, et de réaliser d'autres expériences avec des applications plus complexes en termes de bande passante et du nombre de communicants.

Annexes

A

Architecture d'une Slice Virtex-5

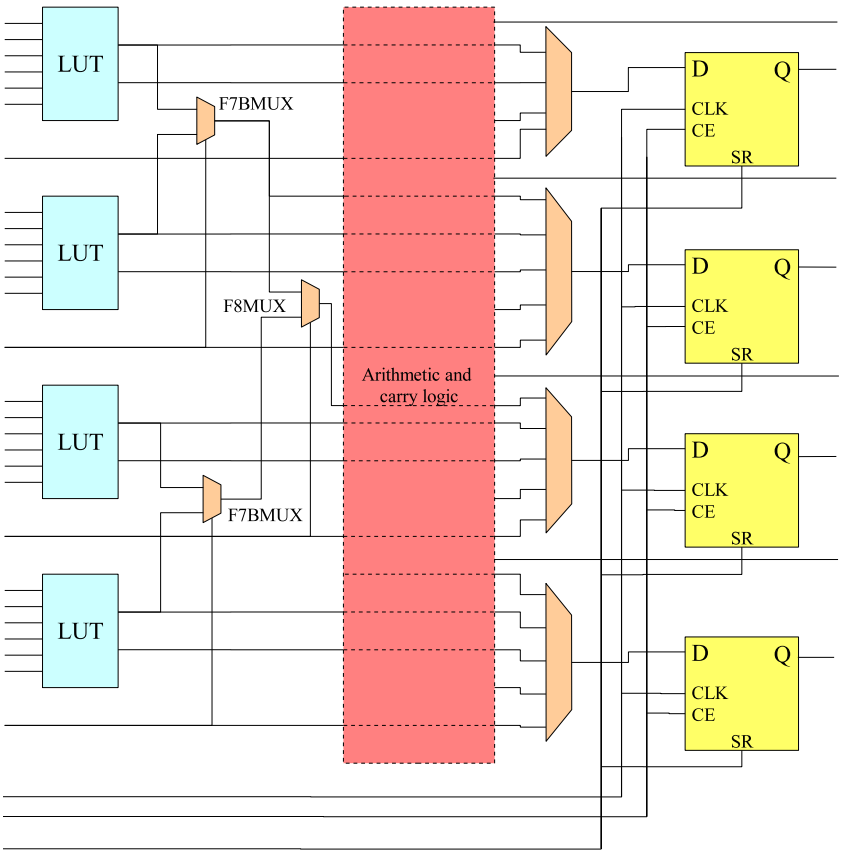


FIGURE A.1 – Description simplifiée de l'architecture de l'unité slice Virtex-5

La structure de la slice Virtex-5 [Tec] de Xilinx illustrée sur la figure A.1 est composée de :
- quatre tables de correspondance (LUTs [59]) avec un nombre d'entrées/sorties configurables, on peut configurer la LUT en 6 entrées et 1 bit de sortie, ou en 5 entrées et 2 bits de sortie ;
- trois multiplexeurs dédiés à la logique combinatoire (F7AMUX, F7BMUX, et F8MUX). les multiplexeurs (F7AMUX et F7BMUX) combinent les sorties de deux LUTs pour créer des circuits combinatoires avec 7 entrées. et le multiplexeur F8MUX est utilisé pour combiner les sorties des deux multiplexeurs (F7AMUX et F7BMUX) ;

59. Look-Up Tables

– un bloc dédié à la logique arithmétique, il est composé de deux additionneurs 1 bit avec propagation de retenue ;
– quatre registres 1 bit qui peuvent être configurés comme des bascules de type flip-flop ou latche. L'entrée de ces registres est sélectionnée avec les multiplexeurs AMUX-DMUX.

B

Jeux de test

Nous avons développé deux algorithmes pour résoudre le problème de recherche des chemins spatio-temporels, un algorithme heuristique et un algorithme mémétique. Pour tester et comparer les temps de résolution de ces deux algorithmes, nous avons créé trois jeux de test.

B.1 Jeu de test A

La première instance est un NoC simple avec une topologie irrégulière, il connecte quatre IPs et trois mémoires. La table TDMA attribuée à ce réseau est constituée de huit intervalles de temps. Comme le montre la figure B.1, chaque interface réseau (SANI) ordonnance ces communications dans la table TDMA en attribuant à chacune d'elle deux slots de temps.

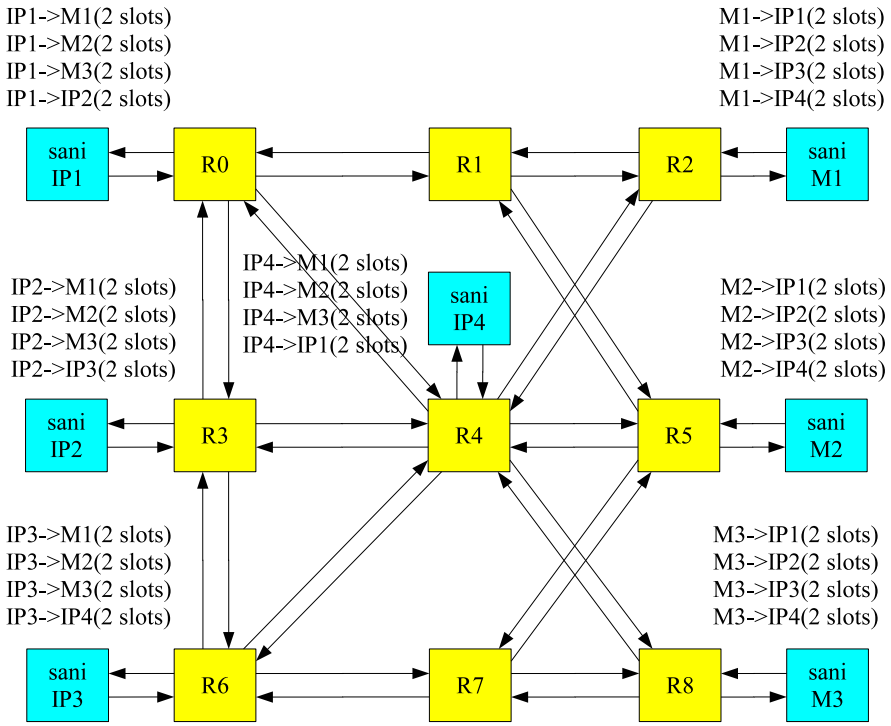


FIGURE B.1 – Architecture du NoC pour le jeu de test A

B.2 Jeu de test B

La figure B.2 illustre un deuxième jeu de test. Ce jeu de test est un réseau qui connecte dix IPs et il transmet 33 messages. Ce NoC utilise une table TDMA avec neuf intervalles et introduit le principe de multi-TDMA compatibles. Ce jeu test permet de vérifier nos algorithmes pour le cas des NoCs avec des tables TDMAs reconfigurables dynamiquement.

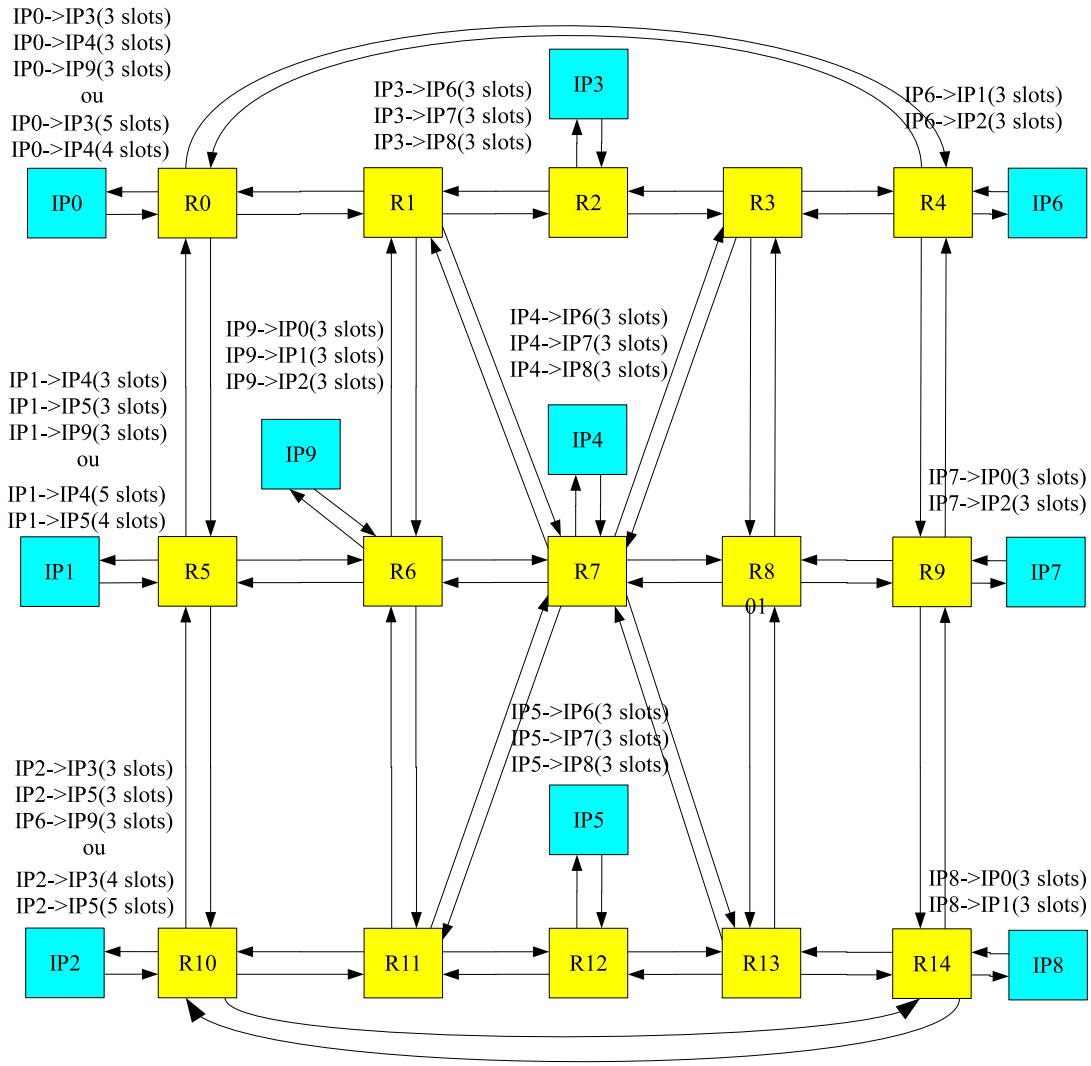


FIGURE B.2 – Architecture du NoC pour le jeu de test B

B.3 Jeu de test C

Enfin, nous testons un troisième réseau pour confirmer la robustesse des algorithmes et leurs performances. Ce NoC connecte les 28 IPs et les 7 mémoires de l'architecture du décodeur H264 développé au sein du centre de recherche *technicolor Rennes*. Pour satisfaire la bande passante des 209 communications de cette application, nous utilisons une table TDMA de 47 intervalles de temps.

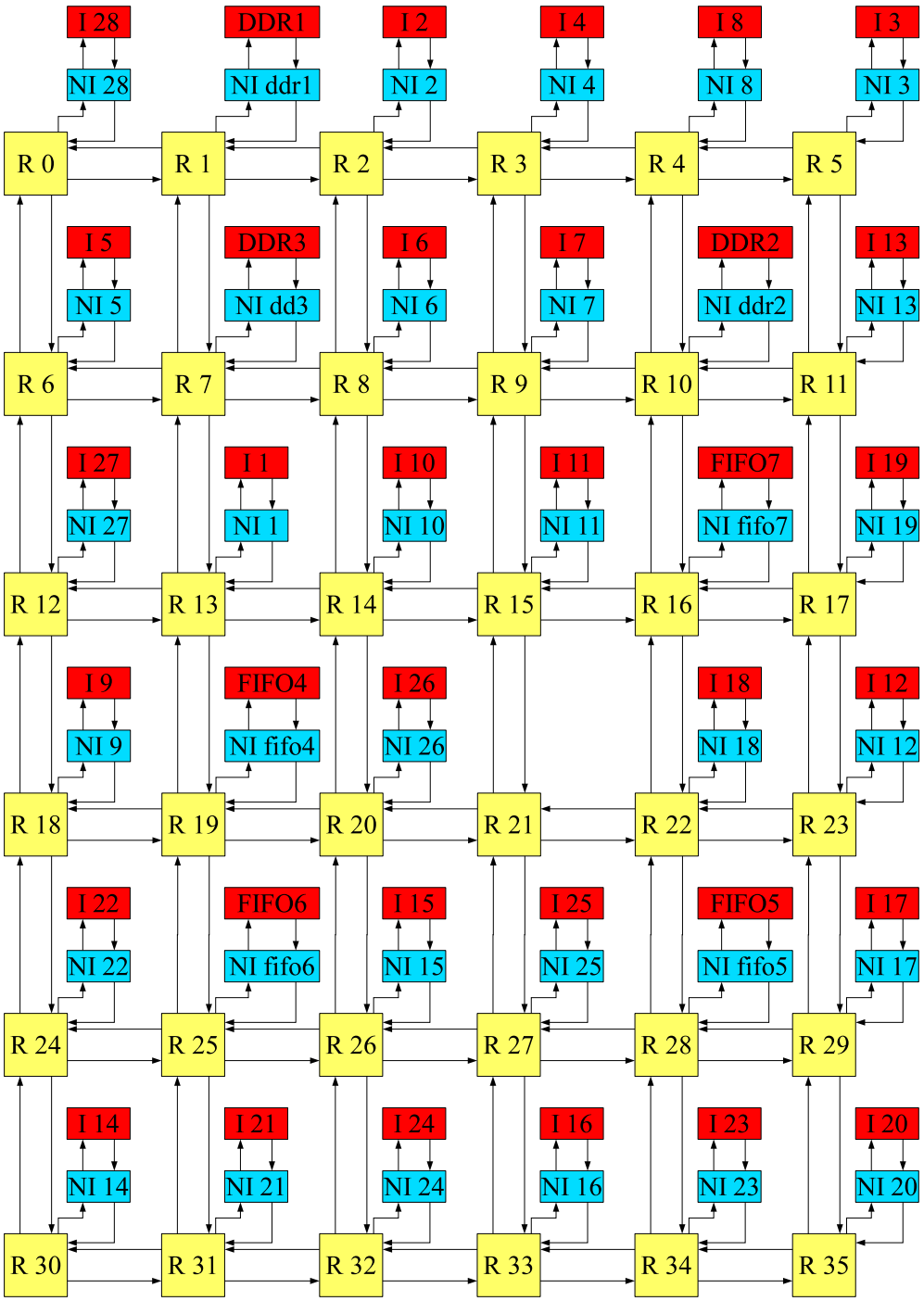


FIGURE B.3 – Architecture du NoC pour le SoC d'un décodeur H264

C

Environnement de conception μspider II

C.1 Outil d'exploration

C.1.1 Création et affichage des graphes CDGs

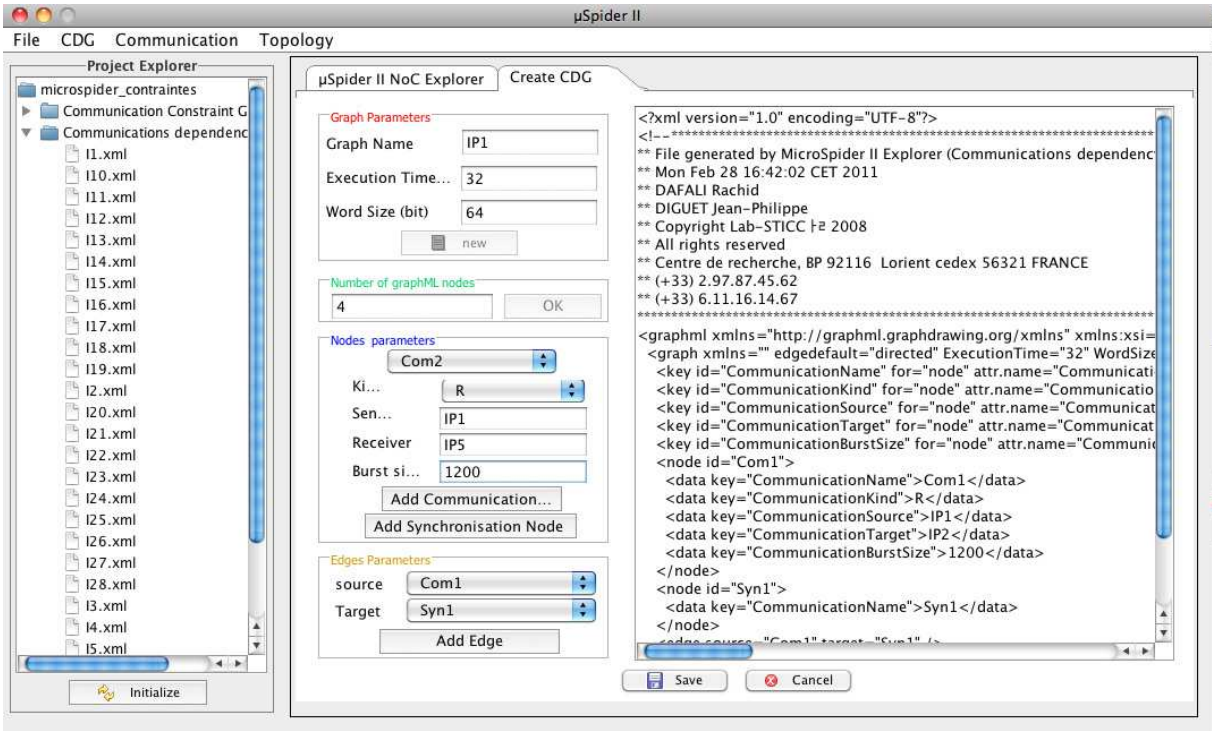


FIGURE C.1 – Création d'un graphe CDG

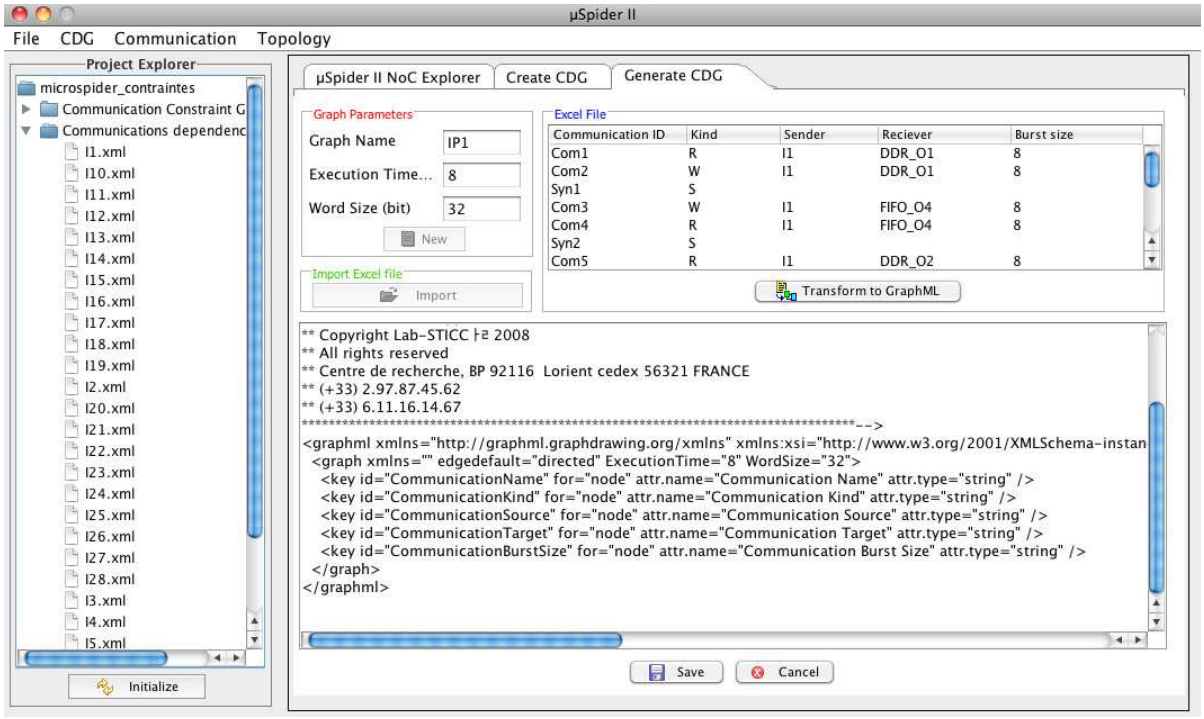


FIGURE C.2 – Génération d'un graphe CDG à partir d'un tableur

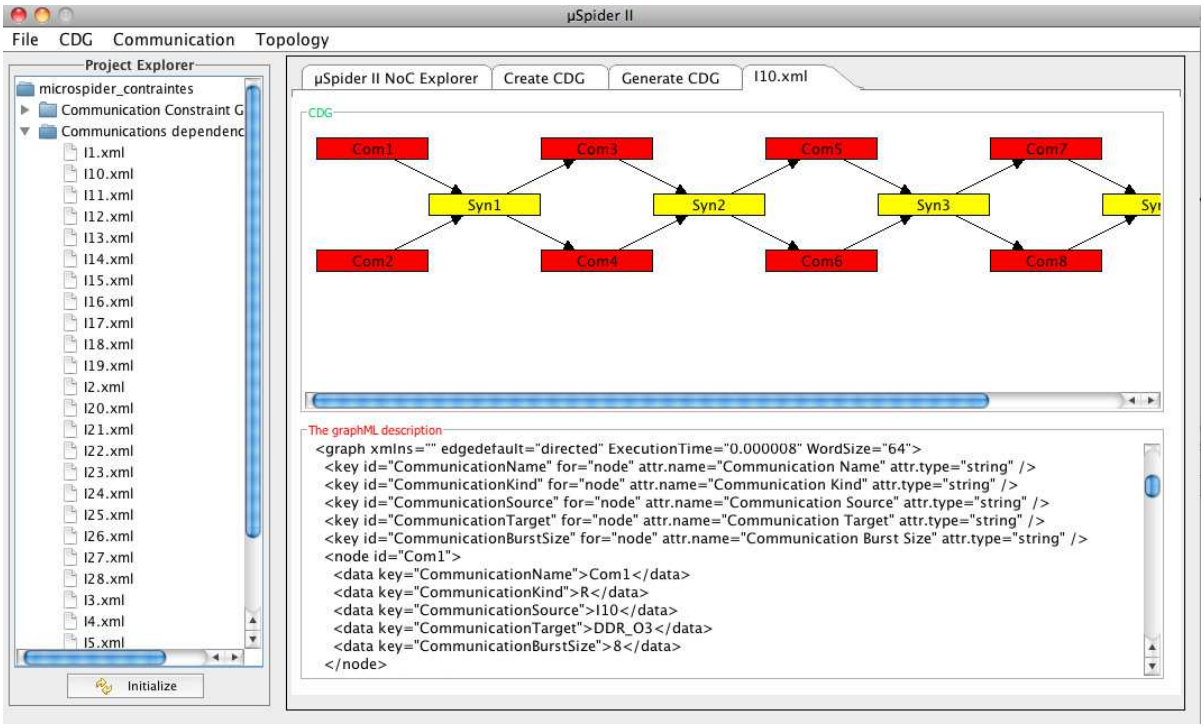


FIGURE C.3 – Affichage du graphe CDG

C.1.2 Calcul des contraintes et génération de la topologie

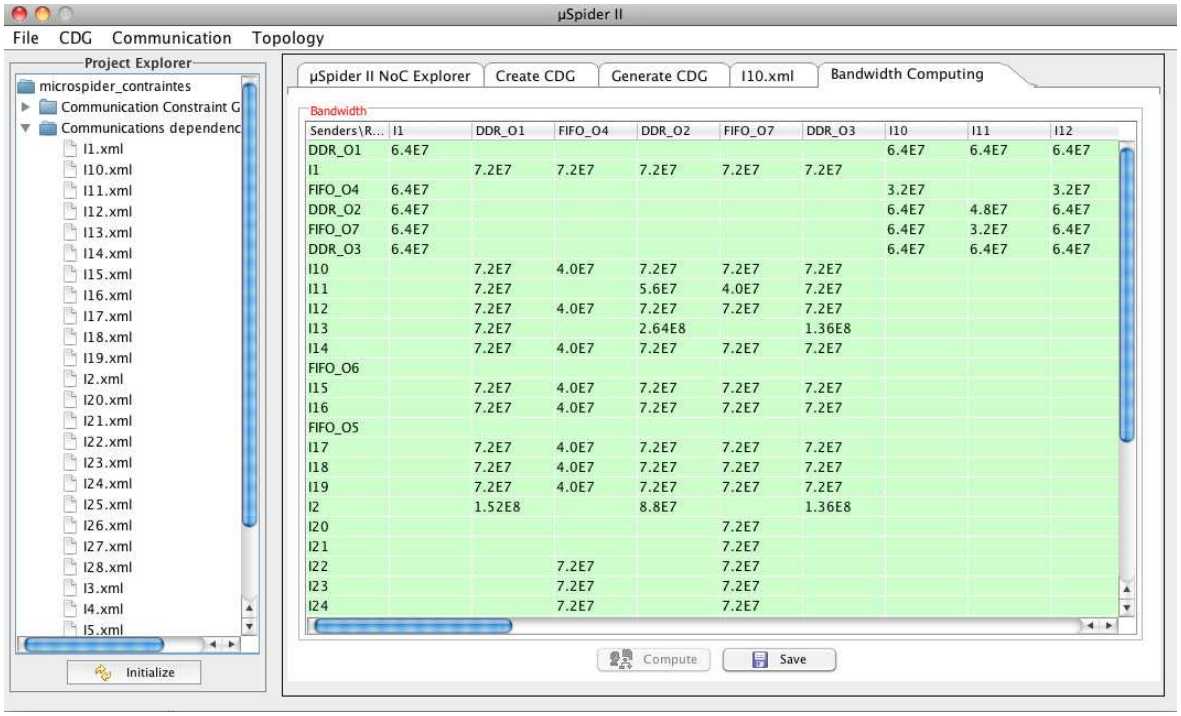


FIGURE C.4 – Calcul des contraintes réelles de l'application

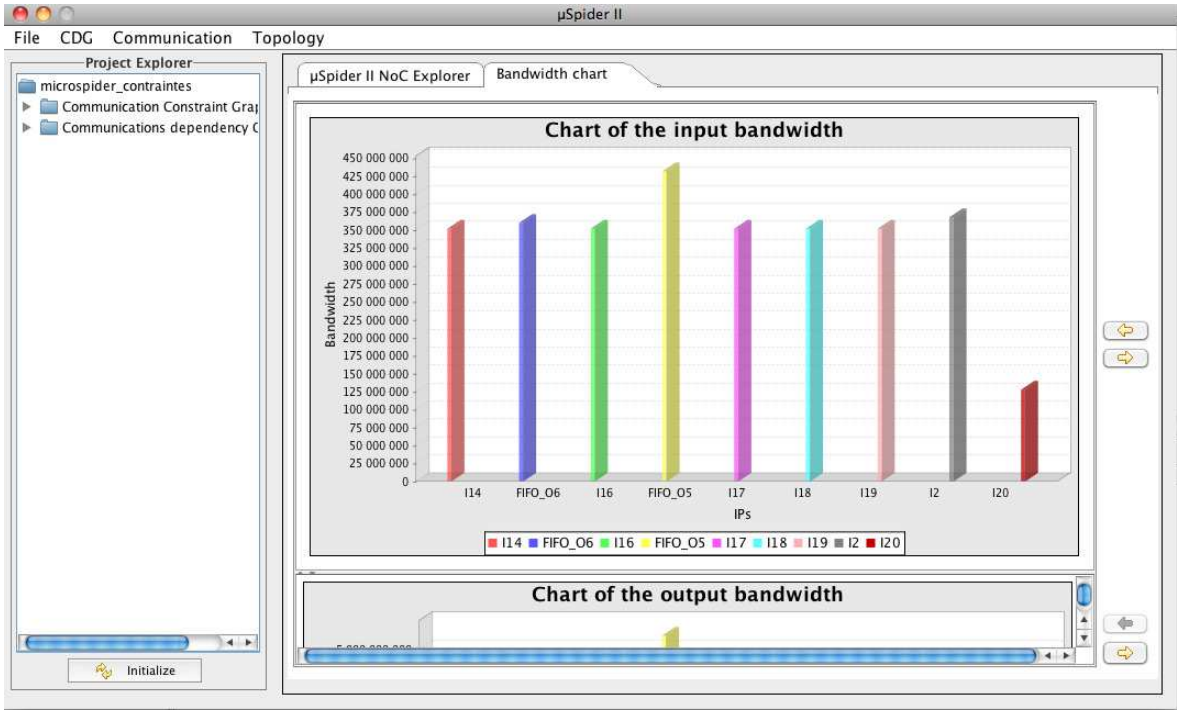


FIGURE C.5 – Affichage en diagramme des bandes passantes des différents communicants

C.2 Outil de génération

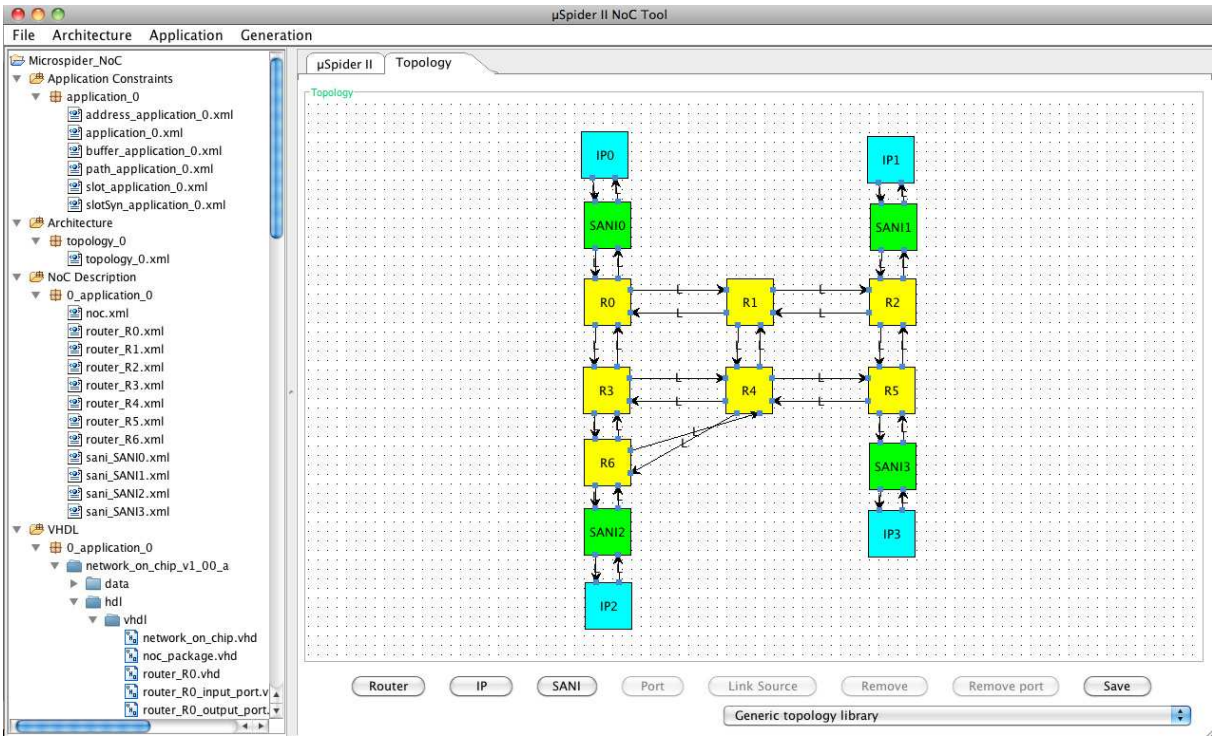


FIGURE C.6 – Édition de la topologie

D

Les CDGs de l'application détection et suivi d'objets

Les graphes CDGs de l'application détection et suivi de mouvement se composent de trois processus, un processus de traitement qui indique le numéro de la tâche exécutée et le numéro de l'image traitée. Un processus d'attente appelé "Réception", ce processus attend la fin de la réception de l'image avant d'autoriser son traitement. Et un processus d'échange indiquant l'identifiant de l'émetteur, l'identifiant du récepteur, la bande passante et le sens de la communication.

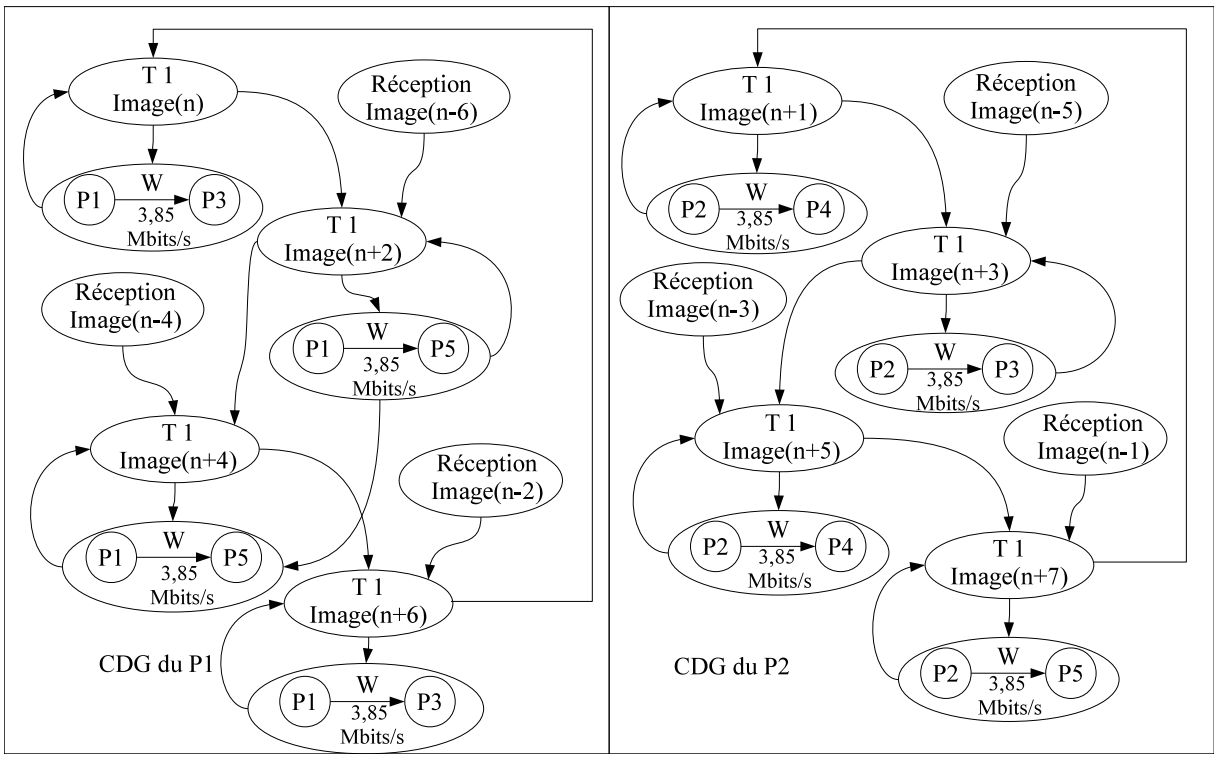


FIGURE D.1 – Graphes CDGs des processeurs qui exécutent la tâche 1

Les graphes CDGs illustrés dans la figure D.1 représentent l'organisation et les dépendances des différents processus qui composent le traitement réalisé par les processeurs 1 et 2. Le processus de réception dans ces deux graphes est différent de celui des autres graphes, car il n'attend pas la

réception de l'image qui sera traitée avec la tâche 1 mais la réception de l'image qui sera affichée. Ainsi, les processeurs 1 et 2 ordonnent le traitement des images de manière à organiser l'ordre d'affichage selon l'ordre de lecture.

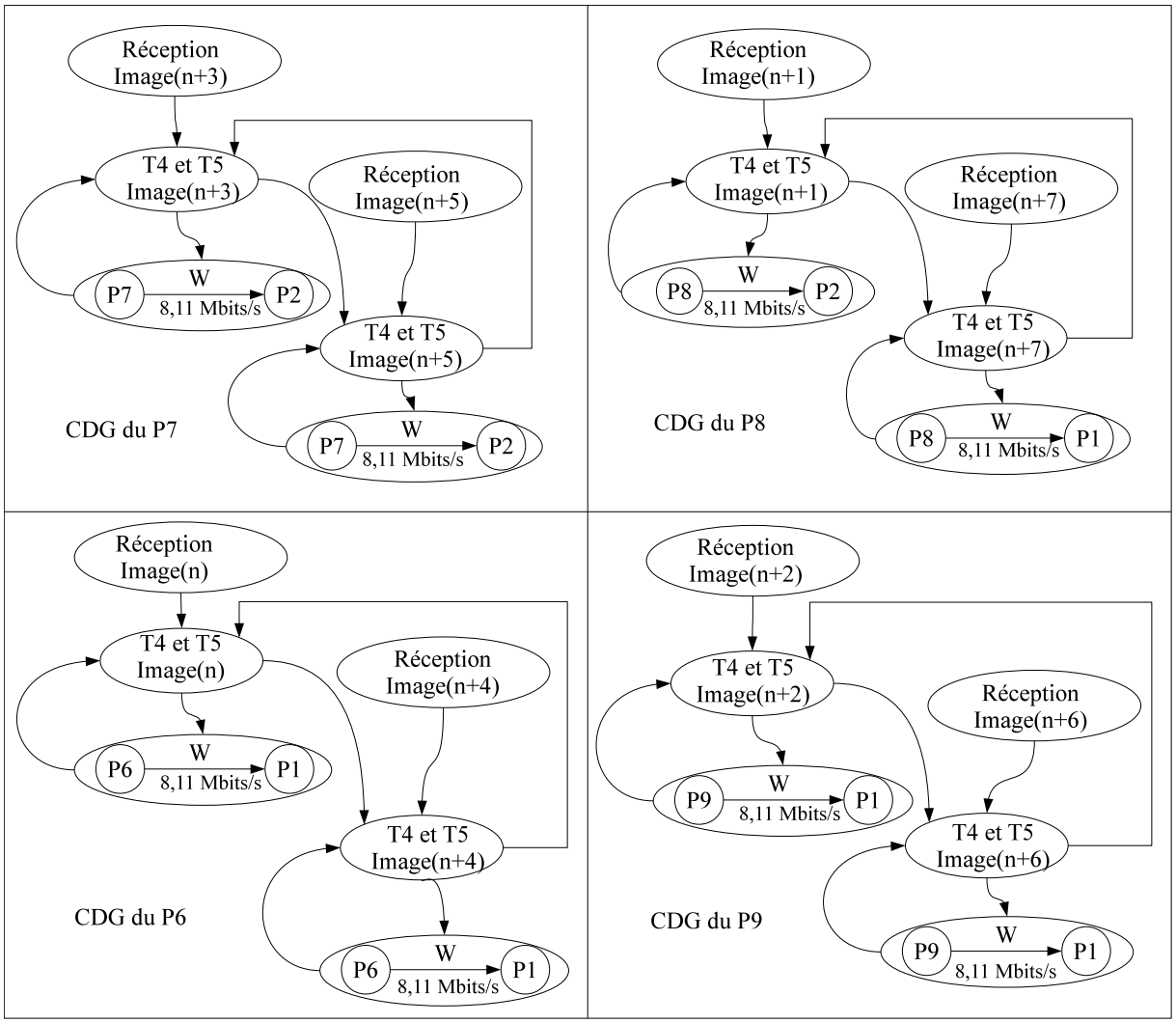


FIGURE D.3 – Graphes CDGs des processeurs qui exécutent les tâches 4 et 5

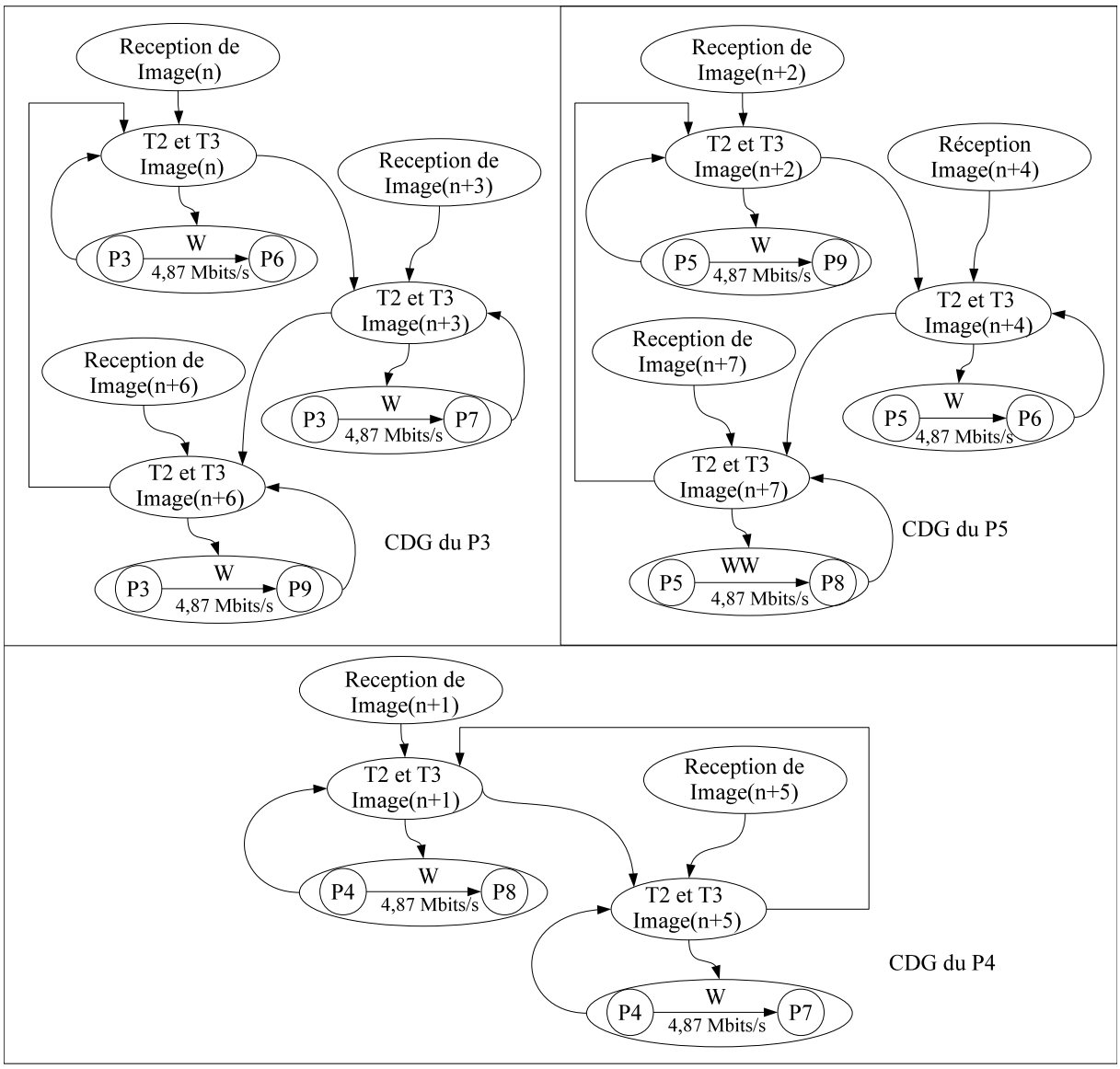


FIGURE D.2 – Graphes CDGs des processeurs qui exécutent les tâches 2 et 3

E

Démonstrateur

Le développement d'un démonstrateur est l'aboutissement des travaux de recherches que nous avons menés dans le domaine des réseaux sur puce. Ce démonstrateur a pour objectif d'évaluer et de comparer les performances des différentes versions du RNoC. En plus, il permet d'accompagner l'environnement de conception μSpider II, pour compléter le processus de conception qui commence avec une description de haut niveau et se termine avec une implantation matérielle.

Pour faciliter l'utilisation du démonstrateur, nous avons synthétisé quatre différents systèmes multiprocesseurs, chaque système permet d'évaluer une version du RNoC. Ces architectures multiprocesseurs réalisent le même test détaillé dans la section 7.2 du chapitre 7. Après la synthèse, nous stockons les résultats dans une carte mémoire externe sous un format de "bitstream". Ainsi, cette technique permet à l'utilisateur de programmer le FPGA avec l'un des quatre "bitstream", selon la version du RNoC qu'il souhaite évaluer.

Afin de réaliser ce démonstrateur, nous utilisons la plateforme FPGA XUPV5-LXT11OT illustrée sur la figure E.1, car elle permet de programmer le FPGA avec l'un des "bitstream" stocké sur la carte "flash Compact" en paramétrant les huit interrupteurs du "DIP swicth" selon le numéro du "bitstream" à utiliser. Après le choix du "bitstream", il faut appuyer successivement sur les boutons "programmation" et "reset" pour reprogrammer le FPGA. Dans notre cas, l'utilisateur choisit la version du RNoC à évaluer selon le paramétrage du "DIP switch" suivant :

- le DIP switch = "00010101" pour la version classique du NoC ;
- le DIP switch = "00110101" pour la version RNoC avec des FIFOs reconfigurables ;
- le DIP switch = "01010101" pour la version RNoC avec des tables TDMAs reconfigurables ;
- et le DIP switch = "10010101" pour la version RNoC avec des FIFOs et des tables TDMAs reconfigurables.

Le démonstrateur comporte en plus de la plateforme FPGA, une interface homme machine qui permet d'afficher les résultats des performances du réseau. Cette IHM [60] est conçue avec le logiciel de développement d'applications *LabVIEW*.

L'IHM illustrée sur la figure E.2 reçoit les données de la plateforme FPGA sur le port série "RS232". L'utilisateur choisit et paramètre le port série qui connecte la plateforme à l'ordinateur grâce à l'IHM qui permet de lister les ports séries et de configurer les différents paramètres du port choisi. L'IHM contient quatre interrupteurs qui permettent d'identifier le type des données reçues sur le port série, ainsi l'utilisateur met à 1 l'interrupteur qui correspond à la version du RNoC testée par la plateforme. Après le décodage des données reçues de la plateforme, l'IHM affiche les bandes passantes des différentes communications sous formes de graphes. Cette représentation graphique permet d'explorer et de comparer les performances des différentes versions du RNoC μspider II.

60. Interface Homme machine

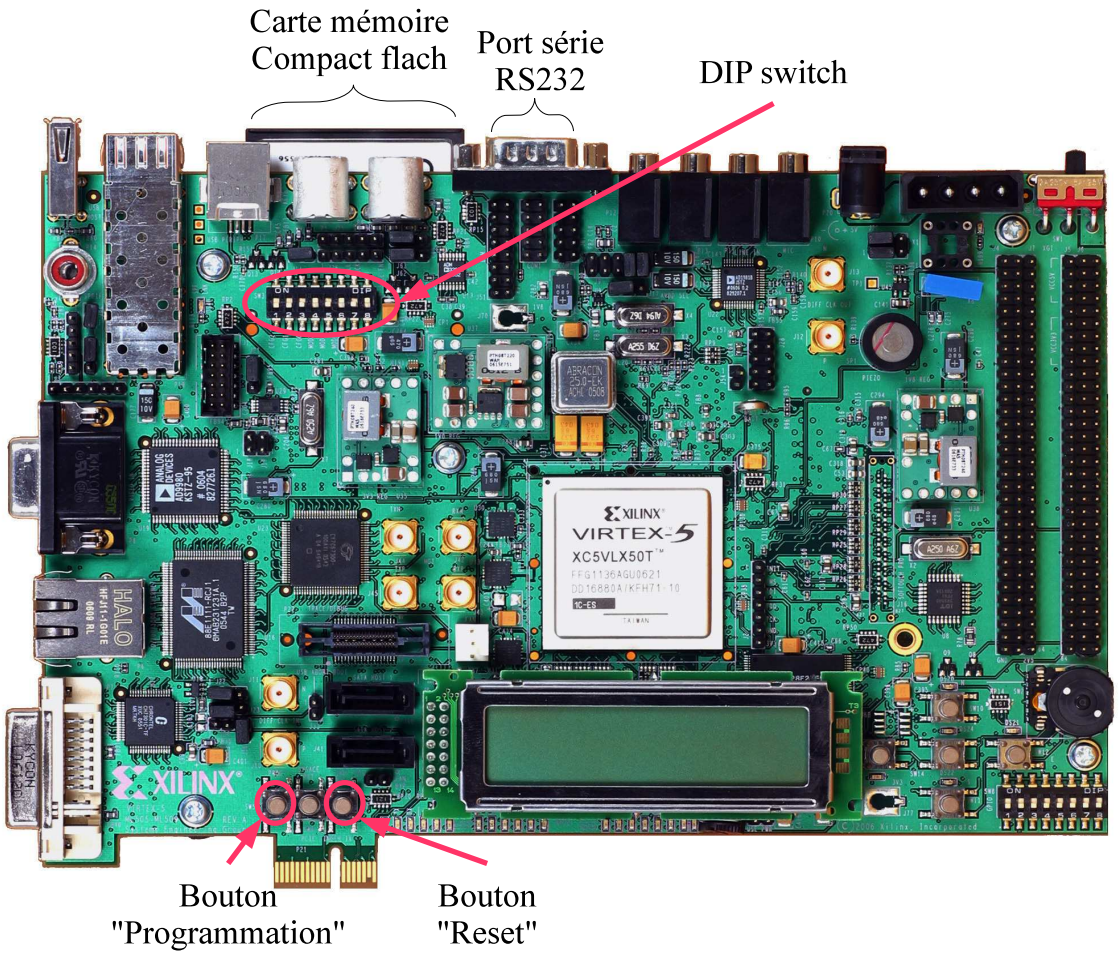


FIGURE E.1 – Description de la carte XUPV5-LXT100T

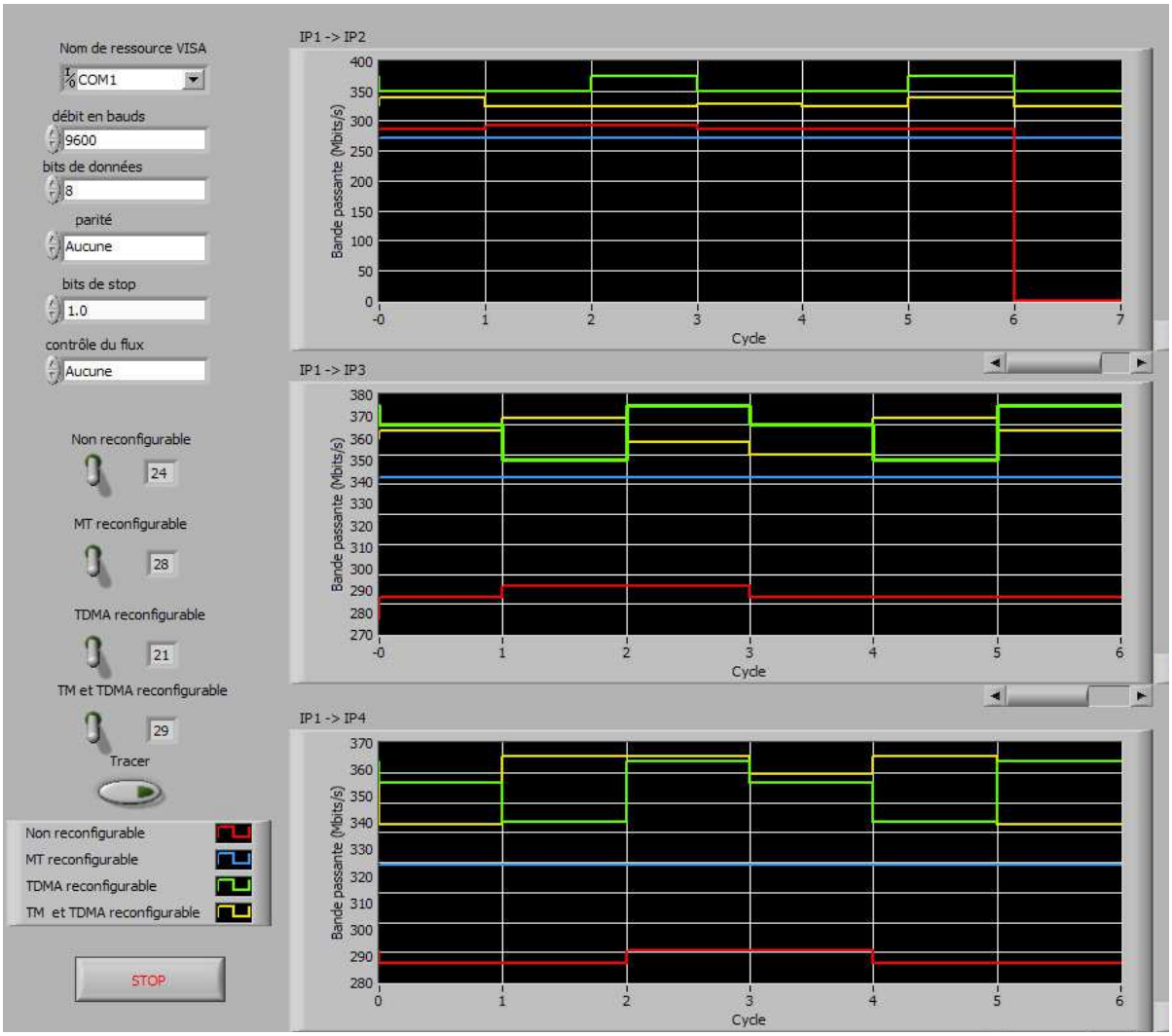


FIGURE E.2 – Interface homme machine du démonstrateur

Liste des acronymes

AFANA	Application-Field-Aware Adaptive Network on chip Architecture
API	Application Programming Interface
ASIC	Application Specific Integrated Circuits
AXI	Advance eXtensible Interface
BE	Best Effort $\Longleftrightarrow$ trafic au mieux
BP	Bandwidth $\Longleftrightarrow$ bande passante
BRAM	Block Random Access Memory
CAO	Conception Assistée par Ordinateur
CDG	Communication Dependency Graph $\Longleftrightarrow$ graphe de dépendance des communications
CTG	Communication Task Graph $\Longleftrightarrow$ graphe des communications des tâches
DADM	Distributed Address space Distributed Memory
DMA	Direct Memory Access $\Longleftrightarrow$ accès direct à la mémoire
EDA	Electronic Design Automation
F	Frequency $\Longleftrightarrow$ fréquence
FC	Flow Control $\Longleftrightarrow$ contrôle du flux
FIFO	First In First Out
flit	FLow control unIT $\Longleftrightarrow$ unité de contrôle de flux
FPGA	Field Programmable Gate Array $\Longleftrightarrow$ circuit logique programmable
GALS	Globally Asynchronous Locally Synchronous
GM	Global Manager $\Longleftrightarrow$ gestionnaire global
GT	Guaranteed Traffic $\Longleftrightarrow$ trafic garanti
IP	Input Port $\Longleftrightarrow$ port d'entrée
IP	Intellectual Property $\Longleftrightarrow$ propriété intellectuelle
ISO	International Organization for Standardization $\Longleftrightarrow$ organisation internationale de standardisation
ITU	International Telecommunication Union $\Longleftrightarrow$ Union Internationale des Télécommunications
LM	Local Manager $\Longleftrightarrow$ gestionnaire local
LW	Link Width $\Longleftrightarrow$ largeur d'un lien
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data

NI	Network Interface $\Longleftrightarrow$ interface réseau
NoC	Network on Chip $\Longleftrightarrow$ réseau sur puce
NORMA	No Remote Memory Access
NUMA	Non Uniform Memory Access
OCP	Open Core Protocol
OP	Output Port $\Longleftrightarrow$ port de sortie
OPA	Output Port Address $\Longleftrightarrow$ Adresse du port de sortie
OPL	Output Port Label$\Longleftrightarrow$ Étiquette du port de sortie
OS	Operating System $\Longleftrightarrow$ système d'exploitation
OSI	Open Systems Interconnection $\Longleftrightarrow$ interconnexion de systèmes ouverts
PA	Protocol Adapter $\Longleftrightarrow$ adaptateur de protocole
phit	PHysical unIT $\Longleftrightarrow$ unité physique
PLB	Processor Local Bus
QoS	Quality of Service $\Longleftrightarrow$ qualité de service
R	Router $\Longleftrightarrow$ routeur
RD	Reconfiguration Dynamique
RISC	Reduced Instruction Set Computer
RNoC	Reconfigurable Network-on-Chip $\Longleftrightarrow$ réseau sur puce reconfigurable
SADM	Single Address space Distributed Memory
SAF	Store and Forward
SANI	Self-Adaptive Network Interface $\Longleftrightarrow$ interface réseau auto-adaptative
SASM	Single Address space Shared Memory
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
TDMA	Time Division Multiple Access $\Longleftrightarrow$ accès multiple à répartition dans le temps
UMA	Uniform Memory Access
VCI	Virtual Component Interface
VCT	Virtual Cut Through
XML	eXtensible Markup Language

Bibliographie

[ACG+03] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C.A. Zeferino. SPIN : a scalable, packet switched, on-chip micro-network. In *DATE '03 Proceedings of the conference on Design, Automation and Test in Europe : Designers' Forum*, volume 2, pages 70 – 73 suppl., 2003.

[AIS09] A. Agarwal, C. Iskander, and R. Shankar. Survey of network on chip (NoC) architectures & contributions. *Journal of engineering computing and architecture*, 3(1), 2009.

[AN05] T. Ahonen and J. Nurmi. Integration of a NoC-based multimedia processing platform. In *International Conference on Field Programmable Logic and Applications*, pages 606 – 611, 2005.

[ASTBN04] T. Ahonen, A. Sigüenza-Tortosa, H. Bin, and J. Nurmi. Topology optimization for application-specific networks-on-chip. In *SLIP '04 : Proceedings of the 2004 international workshop on System level interconnect prediction*, pages 53–60, New York, NY, USA, 2004.

[BAM+05] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen. DyNoC : A dynamic infrastructure for communication in dynamically reconfugurable devices. In *International Conference on Field Programmable Logic and Applications*, pages 153 – 158, 2005.

[BB04] D. Bertozzi and L. Benini. Xpipes : a network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine, IEEE*, 4(2) :18 – 31, 2004.

[BCGK04] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC : QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50 :105–128, 2004.

[BHB+07] L. Braun, M. Hubner, J. Becker, T. Perschke, V. Schatz, and S. Bach. Circuit switched run-time adaptive network-on-chip for image processing applications. In *International Conference on Field Programmable Logic and Applications*, pages 688 – 691, 2007.

[BJM+05] D. Bertozzi, A. Jalabert, Srinivasan Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2) :113 – 129, 2005.

[BM06] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1) :1, 2006.

[BS04] T. Bjerregaard and J. Sparso. Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Norchip Conference*, pages 269 – 272, 2004.

[BS05] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Design, Automation and Test in Europe*, pages 1226 – 1231, 2005.

[CDRS10] J.-C. Créput, R. Dafali, A. Rossi, and M. Sevaux. Communications reconfigurables dans un NoC : optimisation par approche évolutionnaire. Rapport interne, Octobre 2010.

[CLM+04] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon : a novel on-chip communication network. In *International Symposium on System-on-Chip*, pages 15–20, 2004.

[DBG+03] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. Xpipes : a latency insensitive parameterized network-on-chip architecture for multiprocessor socs. In *21st International Conference on Computer Design*, pages 536 – 539, 2003.

[DZ83] J.D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12) :1334 – 1340, 1983.

[EDH06] S. Evain, J.-Ph. Diguet, and D. Houzet. NoC design flow for TDMA and QoS management in a GALS context. *EURASIP Journal on Embedded Systems*, 2006.

[EMEKG07] H. Elmiligi, A.A. Morgan, M.W. El-Kharashi, and F. Gebali. A topology-based design methodology for networks-on-chip applications. In *2nd International Design and Test Workshop*, pages 61– 65, 2007.

[Eus08] Yvan Eustache. *Reconfigurations Algorithmiques et Architectures Régulées : Contribution à l'Auto-Adaptation des Systèmes Embarqués.* PhD thesis, Université de Bretagne Sud, 2008.

[FF56] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3) :399–404, 1956.

[Fly72] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9) :948 – 960, 1972.

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip : concepts, architectures, and implementations. *Design Test of Computers*, 22(5) :414 – 421, 2005.

[GG00] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 250–256, 2000.

[HCG07] A. Hansson, M. Coenen, and K. Goossens. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. pages 1 – 6, apr. 2007.

[HKHT05] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann. Dynamic reconfiguration with hardwired networks-on-chip on future fpgas. pages 527 – 530, 2005.

[IBJ94] C. Izu, R. Beivide, and C. Jesshope. Mad-postman : A look-ahead message propagation method for static bidimensional meshes. In *Parallel and Distributed Processing, 1994. Proceedings. Second Euromicro Workshop on*, pages 117 – 124, 1994.

[JM05] H. Jingcao and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4) :551 – 562, 2005.

[KCZS01] M.E. Kreutz, L. Carro, C.A. Zeferino, and A.A. Susin. Communication architectures for system-on-chip. In *14th Symposium on Integrated Circuits and Systems Design*, pages 14 – 19, 2001.

[KM85] J. Kramer and J. Magee. Dynamic configuration for distributed systems. *IEEE Transactions on Software Engineering*, SE-11(4) :424 – 436, 1985.

[KM90] J. Kramer and J. Magee. The evolving philosophers problem : dynamic change management. *IEEE Transactions on Software Engineering*, 16(11) :1293 – 1306, 1990.

[Ler06] A. Leroy. *Optimizing the on-chip communication architecture of low power Systems-on-Chip in Deep Sub-Micron technology.* PhD thesis, 2006.

[LV05] G.M. Link and N. Vijaykrishnan. Hotspot prevention through runtime reconfiguration in network-on-chip. pages 648 – 649 Vol. 1, 2005.

[MBD+05] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal. Dynamic time-slot allocation for qos enabled networks on chip. In *3rd Workshop on Embedded Systems for Real-Time Multimedia*, pages 47 – 52, 2005.

[MC03] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. *Handbook of metaheuristics*, pages 105–144, 2003.

[MDM04a] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 896 – 901, 2004.

[MDM04b] S. Murali and G. De Micheli. SUNMAP : a tool for automatic topology selection and generation for nocs. In *Design Automation Conference*, pages 914 – 919, 2004.

[MHO05] R. Marculescu, Jingcao Hu, and U.Y. Ogras. Key research problems in noc design : a holistic perspective. In *Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 69 –74, 2005.

[NM93] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2) :62 – 76, 1993.

[NMA+05] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. pages 234 – 239 Vol. 1, 2005.

[NMV04] V. Nollet, T. Marescaux, and D. Verkest. Operating-system controlled network on chip. pages 256 – 259, 2004.

[OM05] U.Y. Ogras and R. Marculescu. Energy- and performance-driven noc communication architecture synthesis using a decomposition approach. In *Design, Automation and Test in Europe*, pages 352 – 357 Vol. 1, 2005.

[PAK06] T. Pionteck, C. Albrecht, and R. Koch. A dynamically reconfigurable packet-switched network-on-chip. In *Design, Automation and Test in Europe*, volume 1, page 8 pp., 2006.

[PKA06] T. Pionteck, R. Koch, and C. Albrecht. Applying partial reconfiguration to networks-on-chips. In *International Conference on Field Programmable Logic and Applications*, pages 1 – 6, 2006.

[Rai92] S. Raina. Virtual shared memory : A survey of techniques and systems. Technical report, Bristol, UK, 1992.

[RGR+03] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *Computers and Digital Techniques*, 150(5) :294–302, 2003.

[SKH08] E. Salminen, A. Kulmala, and T.D. Hamalainen. Survey of network-on-chip proposals. *white paper, OCP-IP*, pages 1–13, 2008.

[SS08] M.B. Stensgaard and J. Sparso. ReNoC : A network-on-chip architecture with reconfigurable topology. In *Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 55 – 64, 2008.

[STM] STMicroelectronics. Stmicroelectronics unveils innovative network-on-chip technology for new system-on-chip interconnect paradigm. http ://www.st.com/stonline/press/news/year2005/t1741t.htm.

[Tec] 1-CORE Technologies. Fpga logic cells comparison. http ://www.1-core.com/library/digital/fpga-logic-cells/fpga-logic-cells.pdf.

[WL03] D. Wiklund and Dake Liu. SoCBUS : switched network on chip for hard real time embedded systems. In *Parallel and Distributed Processing Symposium*, page 8 pp., 2003.

[Zim80] H. Zimmermann. OSI reference model–the ISO model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4) :425 – 432, 1980.

Liste des publications

Chapitre de livre

R.Dafali, J-Ph.Diguet, "**Keys for Administration of Reconfigurable NoC Self-Adaptive Network Interface Case Study**", in *Dynamic Reconfigurable Network-on-Chip Design : Innovations for Computational Processing and Communication*, IGI Global, Ed. Jih-Sheng Shen and Pao-Ann Hsiung, 2009.

Conférences internationales avec actes

R.Dafali and J-Ph.Diguet, "**MPSoC Architecture-Aware Automatic NoC Topology Design**", In *IFIP International Conference on Network and Parallel Computing (NPC'10)*, September 13-15, 2010, Zheng Zhou, China.

R.Dafali and J-Ph.Diguet, "**Self-Adaptive Network Interface (SANI) : local component of a NoC configuration manager**", *in Reconfig'09 International Conference on ReConFigurable Computing and FPGAs*, December 9-11, 2009, Cancun, Mexico.

R.Dafali, J-Ph.Diguet and M.Sevaux, "**Key Research Issues for Reconfigurable Network-on-Chip**", in *ReConFig'08, International Conference on ReConFigurable Computing and FPGAs*, December 3-5, 2008, Cancun, Mexico.

R.Dafali, S.Evain, J-Ph.Diguet and E.Juin, "**μSpider CAD tool : Case Study of NoC IP Generation for FPGA**", in *DASIP'07, Workshop on Design and Architectures for Signal and Image Processing*, November 27-29, 2007, Grenoble, France.