Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce (NoC)

R. Lemaire

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : « Micro et Nano Électronique »

préparée au **CEA-LETI**

dans le cadre de **l'Ecole Doctorale « Électronique, Électrotechnique, Automatique et Traitement du Signal »**

présentée et soutenue publiquement

par

Romain LEMAIRE

le 11 octobre 2006

Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce (NoC)

Directeur de Thèse

Ahmed Amine JERRAYA

JURY

M. Guy Mazaré	, Président
M. Stanislaw Piestrak	, Rapporteur
M. Lionel Torres	, Rapporteur
M. Ahmed Amine Jerraya	, Directeur de thèse
M. Didier Lattard	, Co-encadrant

Octobre 2006 – Version 1.0 – 40 exemplaires.

Remerciements

Pour débuter ce mémoire de thèse, je tiens à remercier toutes les personnes qui ont contribué à des degrés diverses au bon déroulement de ce travail de recherche.

Je remercie tout d'abord Monsieur Didier Lattard, Ingénieur-Chercheur au CEA-LETI, pour son encadrement, son expertise et son soutien pendant les trois années de préparation de cette thèse.

Je remercie vivement Monsieur Ahmed Jerraya, Directeur de recherche au CNRS, Chef du groupe SLS au laboratoire TIMA, pour ses conseils toujours pertinents en tant que directeur de thèse.

J'exprime également mes remerciements aux autres membres du jury : Monsieur Guy Mazaré, Vice-Président de l'INPG, chargé des systèmes d'information, pour m'avoir fait l'honneur de présider le jury, Messieurs Stanislaw Piestrak et Lionel Torres, Professeurs respectivement aux Universités de Metz et de Montpellier II pour avoir accepté d'être rapporteurs de ce travail et pour leurs remarques très constructives.

Je voudrais aussi remercier Monsieur François Bertrand de m'avoir accueilli dans le laboratoire IAN ainsi que tous les membres du laboratoire, en particulier Messieurs Fabien Clermidy et Yves Durand pour, entre autre, leurs aides précieuses pour la relecture de mes publications, mais également Christian, Claire, Didier V., Edith, Jean, Jérôme, Pascal, Sébastien, Yvain et les stagiaires successifs pour les nombreuses discussions techniques que nous avons pues avoir et qui ont enrichi ce mémoire.

Cette thèse s'est déroulée dans le cadre du service CME et je remercie Messieurs Hervé Fanet et Jean-René Lèquepeys de m'avoir permis d'y travailler. J'exprime également ma sympathie à toutes les personnes du service, notamment à Madame Armelle De Kerleau pour sa disponibilité au secrétariat.

Je tiens de plus à saluer tous mes collègues doctorants au LETI avec qui j'ai partagé cette expérience de recherche : Antoine, Arnaud, Benoît, Cédric, David, Diego, Fabrice, Matthieu, Philippe, Xuan Tu, ainsi que mes amis grenoblois, mes compagnons de randonnée pendant ces trois années et mes amis de salle-co qui se reconnaîtront.

Enfin je voudrais remercier toute ma famille : mes grands-parents, mes parents, ma soeur Tiphaine et mon frère Baptiste qui m'ont toujours donné confiance tout au long de cette thèse.

Bonne lecture !

Table des matières

Introduction : Les réseaux sur puce dans le contexte des systèmes de télécommunication sans-fil

En 1958, Jack Kilby, employé par Texas Instruments, réussit à connecter plusieurs transistors sur un substrat de germanium et devient ainsi l'inventeur officiel du circuit intégré. Il obtiendra le prix Nobel de physique en 2000 et plus anecdotiquement les bascules JK seront nommées ainsi en son honneur. Depuis les techniques d'intégration n'ont cessé de progresser, la microélectronique fait partie de notre quotidien et les puces de silicium contiennent des millions de transistors comme en témoigne l'évolution des microprocesseurs produits par la société Intel (Tableau 1).

Tab. 1 – Croissance du nombre de transistors dans les circuits intégrés : exemple des microprocesseurs Intel

Année de production	**Processeur**	**Nombre de transistors**
1971	4004	2.300
1978	8086	29.000
1982	80286	275.000
1989	80486	1.160.000
1993	Pentium	3.100.000
1995	Pentium Pro	5.500.000
1997	Pentium II	27.000.000
2001	Pentium 4	42.000.000
2004	Pentium Extreme Edition	169.000.000

Cette croissance du nombre de transistors intégrables a rendu possible la réalisation

de circuits numériques aux fonctionnalités toujours plus nombreuses. Aujourd'hui, les concepteurs peuvent implémenter un système électronique complet sur une unique puce : c'est le système sur puce, aussi appelé SoC (de l'anglais *System-on-Chip*). Un SoC intègre généralement des unités de traitement matérielles et logicielles, des interfaces d'entrée et de sortie, des unités mémoires, des unités reconfigurables. Un ou plusieurs système de communication permettent à ces différentes unités d'échanger des données entre elles.

Contexte de la thèse

Ces travaux de thèse s'inscrivent dans le cadre des problématiques actuelles du LETI[1] au sein du laboratoire d'Intégration des Architectures Numériques (IAN) qui dépend du service Conception pour les Microtechnologies Émergentes (CME). Ces problématiques sont doubles, elles se concentrent sur :

- l'intégration d'applications de télécommunication complexes dans un SoC, et
- le développement d'architectures de communication pour SoC.

Les applications de télécommunication visées par cette première problématique concernent les standards en cours d'élaboration qui sont désignés par le terme 4G (pour quatrième génération). Ces technologies réalisent la fusion entre le monde de la téléphonie mobile et celui des réseaux sans-fil. En terme de performances, le saut est significatif puisque les débits pourront atteindre 100 Mbps à comparer aux 2 Mbps d'un système UMTS[2] (3G). Au niveau des terminaux mobiles, ce gain en débit entraîne une augmentation de la complexité en particulier en ce qui concerne les traitements en bande de base. Différentes techniques de modulation sont combinées : les transmissions multiporteuses (comme l'OFDM[3]), les techniques d'étalement de spectre (par exemple le CDMA[4]) et l'exploitation de la diversité spatiale des antennes (système MIMO[5]). Pour les concepteurs, cela se traduit par des blocs de traitement plus élaborés mais également des chaînes de modulation et de démodulation moins linéaires et plus complexes à contrôler. Proposer un SoC implémentant l'ensemble de ces fonctions constitue alors un enjeu majeur au niveau de l'architecture du circuit et de la méthodologie de conception. Le LETI prend part à ces travaux de recherche dans le cadre de nombreux projets en

[1] Laboratoire d'Electronique et des Technologies de l'Information
[2] *Universal Mobile Telecommunications System*
[3] *Orthogonal Frequency Division Multiplexing*
[4] *Code Division Multiple Access*
[5] *Multiple-Input Multiple-Output*

particulier MATRICE [MATR06] et 4MORE [4MOR06] sur lesquels nous reviendrons par la suite.

La seconde problématique découle de la première mais peut également avoir une portée plus globale. L'évolution des technologies d'intégration sur silicium et la complexification des applications, notamment dans le domaine des télécommunications conduit les concepteurs à réaliser des SoC implémentant un nombre toujours croissant d'unités de traitement. La quantité de données échangées entre ces unités est également en constante augmentation. Les architectures de communication sur puce deviennent donc un élément qu'il est déterminant de maîtriser lors de la conception d'un SoC complexe. Les solutions d'interconnexion classiquement utilisées et majoritairement basées sur des systèmes de bus partagés montrent leurs limites en terme de performance, de consommation ou de surface. De paradigmes architecturaux innovants doivent alors être proposés. Un nouveau concept appelé *réseau sur puce* ou NoC (*Network-on-Chip*) semble particulièrement prometteur et fait l'objet de nombreux travaux de recherches dans les milieux académiques et industriels depuis 5-6 ans. Les NoC consistent en une adaptation des principes des réseaux informatiques pour réaliser des structures de communication flexibles et distribuées dans un circuit intégré. Ils offrent ainsi une plate-forme ouverte sur laquelle connecter un grand nombre de ressources de traitement. Les flots de données ne sont pas fixés et peuvent être reconfigurés en fonction des besoins des applications. Dans ce contexte, le LETI propose une architecture de réseau qui s'est concrétisée au cours du projet FAUST (F*lexible* A*rchitecture of* U*nified* S*ystems for* T*elecom*).

Motivations et objectifs

Les NoC apportent de réelles avantages au niveau des communications en tant que structure d'interconnexion et nous les détaillerons par la suite. Cependant les échanges de données sont moins déterministes et leurs gestions plus complexes. En effet, un réseau est par nature un système distribué sur lequel les transactions sont multiples et simultanées. Il faut alors gérer des problèmes de synchronisation, d'ordonnancement, de contrôle de flux pour l'ensemble des communications. Nous avons ainsi identifié un premier objectif qui est la prévision, l'évaluation et la validation des performances d'un NoC dans le cas d'un trafic applicatif concret. Autrement dit, le réseau offre aux ressources de traitement un service de communication fonctionnel mais comment vérifier avec suffisamment d'anticipation qu'il pourra supporter les contraintes de débits et de latence imposées par l'application. Le projet FAUST nous donne un cadre applicatif

très concret pour aborder ce type de problèmes.

Un des intérêts majeurs des réseaux sur puce est de fournir une structure d'interconnexion fiable et performante pour développer rapidement de nouveaux circuits à partir de blocs fonctionnels réutilisables, préalablement développés et validés. Pour cela, il est impératif d'avoir un protocole bien défini qui permet de découpler les traitements des communications. L'objectif que nous visons alors est la spécification des interfaces entre le réseau et les unités de traitement. Ces interfaces doivent pouvoir gérer l'ensemble des flux de communication en respectant le protocole. Cela concerne entre autre l'ordonnancement des transferts, la gestion des sources et destinations des données et l'adaptation des formats de données. De plus, plusieurs contraintes sont à prendre en compte, en particulier les interfaces réseau ne doivent pas être un facteur limitant pour les performances temporelles (latence, fréquence, débit) des unités de traitement et il faut trouver de bons compromis entre les fonctionnalités et la complexité de la logique à implémenter sur silicium. Pour finir, la conception de ces interfaces ne peut se faire complètement indépendamment des applications, cette thèse a donc pour objectif de présenter une structure d'interface à la fois bien adaptée aux besoins des systèmes de télécommunication précédemment cités mais également suffisamment générique pour anticiper les évolutions vers les standards futurs.

Les interfaces réseau permettent de connecter les unités de traitement au NoC. Elles implémentent les mécanismes pour la gestion locale des communications. L'étape suivante est de maîtriser le comportement global du système. C'est-à-dire de coordonner les différentes unités de traitement pour qu'elles réalisent la fonctionnalité pour laquelle le circuit est conçu. L'objectif est donc de mettre en place un système pour contrôler et synchroniser les différentes phases de traitement et de communication. Les difficultés viennent du fait que le système est distribué. Il n'y a pas de connexions directes entre les unités, les messages de contrôle circulent sur le réseau, il est donc complexe d'avoir une vision d'ensemble de l'état du système à un instant donné.

Contributions de la thèse

Cette thèse a pour cadre l'étude d'une architecture de réseau sur puce dans le contexte de l'implémentation d'un modem bande de base supportant des applications de télécommunication radiomobiles. Nous venons de présenter nos objectifs de recherche. Les contributions de ces travaux répondant à ces objectifs sont multiples et originales :

– Une étude bibliographique avancée nous a permis de synthétiser les concepts rela-

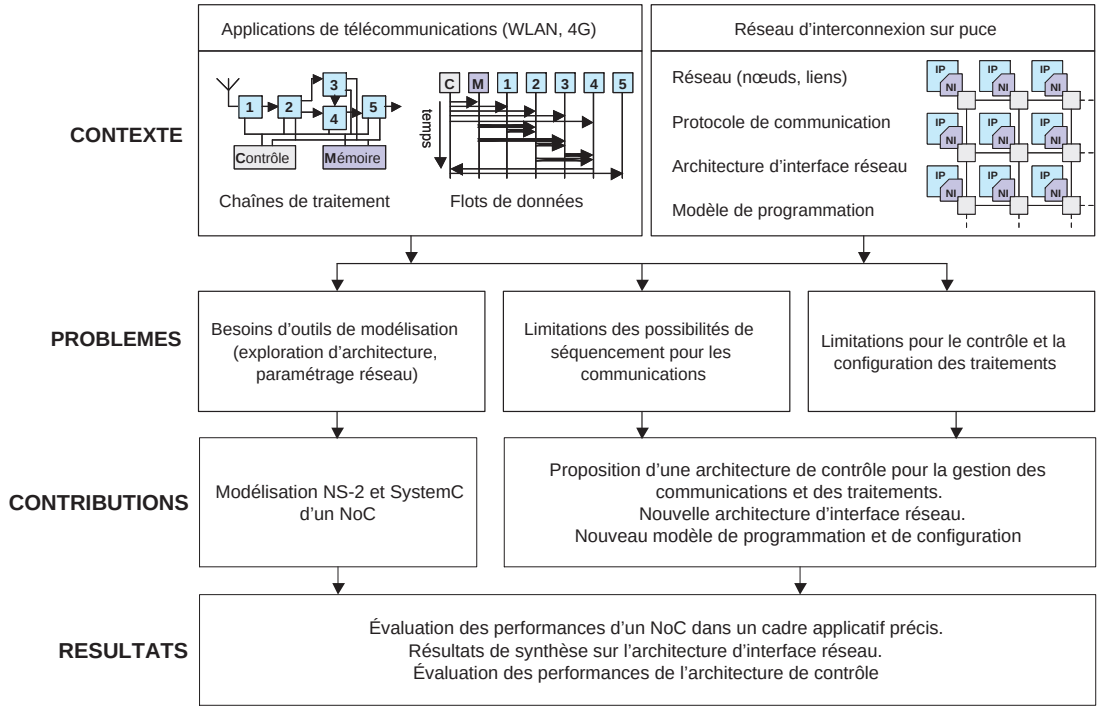


FIG. 1 – Flot de thèse

tifs aux réseaux sur puce et de dégager des problématiques nouvelles sur lesquelles nous avons travaillé : la modélisation des NoC, la gestion des communications et le contrôle des traitements applicatifs.

– Un environnement de modélisation haut-niveau d'applications sur NoC a été développé en utilisant le simulateur de réseau NS-2 [NS206]. Ce modèle permet d'évaluer et de valider les performances d'un NoC dans le cadre d'une application précise de façon anticipée dans le flot de conception.
– Une approche innovante pour le contrôle et la configuration d'une application distribuée sur un NoC est proposée [LLCB06]. Elle définit une architecture d'interface réseau intégrant un séquenceur d'instruction permettant à un système central de programmer des scénarios de chargement de configurations. Les configurations sont téléchargées de manière autonome par les interfaces réseaux grâce à un mécanisme d'envoi de requêtes vers un serveur de configuration. Ces fonctionnalités exploitent le réseau de communication et ne nécessitent pas de structure de connexion complémentaire.

La Figure 1 permet de situer ces contributions par rapport à la démarche que nous avons suivi au cours de cette thèse.

Plan du document

Outre cette introduction, ce document comprend quatre chapitres. Le premier chapitre présente un état de l'art sur la conception d'un système sur puce dans le contexte des architectures de réseaux sur puce. Il expose en détails les nouveaux concepts relatifs aux NoC et introduit les problématiques que nous avons traitées dans le cadre de cette thèse.

Le chapitre 2 aborde le domaine des télécommunications sans-fil. Il présente les applications sur lesquelles nous avons travaillées et expose les spécificités induites pour la conception d'un SoC. Nous détaillons ensuite l'architecture du réseau FAUST dans la perspective des problématiques introduites au chapitre précédent.

Les chapitres suivants constituent une présentation détaillée du travail effectué au cours de cette thèse et des résultats auxquels nous avons aboutis. Le chapitre 3 propose un environnement pour la modélisation et l'analyse d'une architecture de réseau sur puce. Notre approche se base sur l'outil de simulation NS-2 et sur un modèle SystemC du réseau FAUST. Ces environnements sont exploités dans le cadre concret d'une application MC-CDMA. L'approche NoC est confortée par des résultats de performance.

Le chapitre 4 se concentre sur les problématiques de gestion des flots de données et du contrôle des traitements répartis sur les ressources du réseau. Il présente notre solution architecturale basée sur une interface réseau modulaire associée à un protocole de configuration. Une description de l'architecture de l'interface en VHDL illustre une implémentation possible de cette approche. Des résultats de simulation valident le fonctionnement du protocole et permettent de dimensionner les paramètres d'instanciation.

Pour finir, nous présentons nos conclusions accompagnées de perspectives de travaux futurs.

Chapitre 1

Conception d'un système sur puce avec une architecture de réseau sur puce : état de l'art

Les systèmes intégrés sur puces (SoC[1]) sont de plus en plus complexes et intègrent un nombre croissant d'unités de traitement pour répondre aux besoins de nouvelles applications. Cette croissance entraîne une augmentation des besoins de communication dans les circuits pour les échanges de données et pour le contrôle des traitements. L'évolution des technologies silicium rend possible cette densité d'intégration. Cependant avec les échelles submicroniques, de nouvelles contraintes apparaissent. Le coût des interconnexions devient supérieure à celui de la logique. Les communications deviennent prépondérantes pour les performances du systèmes.

Dans ce contexte, les solutions d'interconnexion classiques généralement basées sur des bus partagés présentent des limites. Elles ne supportent pas les débits élevés, manquent de flexibilité et ne passeront pas à l'échelle pour les systèmes futures implémentant plusieurs centaines d'unités de traitement.

Ayant fait ce constat, différents groupes de recherches ont été amené à proposer un nouveau paradigme : le réseau sur puce (NoC[2]). Inspiré des réseaux informatiques commutés, ce type d'architecture présente de réels atouts au niveau conception, performance et implémentation.

[1] *System-on-Chip*
[2] *Network-on-Chip*

L'objectif de ce premier chapitre est de présenter les architectures NoC dans le contexte des travaux de thèse. Dans une première section, nous introduisons le réseau sur puce par rapport au contexte des SoC. La deuxième section détaille les différents concepts relatifs au NoC puis nous donnons des exemples concrets de réalisation issus de l'état de l'art dans le domaine. La dernière section de ce chapitre présente les nouveaux challenges liés à la conception d'un NoC que nous avons identifiés dans le cadre de la thèse.

A l'issue de ce chapitre, le lecteur doit avoir une large compréhension sur le fonctionnement d'un réseau sur puce avec une vision globale de ce qui a été fait dans le domaine. Il doit également comprendre quels sont les tenants et les aboutissants de la conception d'un SoC intégrant un NoC.

1.1 Problématiques des systèmes sur puce et comment l'introduction d'un réseau d'interconnexion peut y répondre

Dans cette première section, nous nous intéresserons aux évolutions actuelles dans le domaine de la conception de systèmes sur puce pour introduire le concept du réseau sur puce.

1.1.1 Les systèmes sur puce et les évolutions actuelles

L'évolution des technologies d'intégration sur silicium permet une constante réduction des dimensions des structures physiques qui peuvent être réalisées sur un circuit intégré (Tableau 1.1). Cette évolution respecte globalement la loi de Moore [Moor65] qui prévoit que les possibilités d'intégration de portes logiques (transistors) doublent tous les 18 mois. Cette augmentation du nombre de transistors composants les circuits permet d'intégrer de plus en plus de fonctionnalités sur une unique puce. Il est maintenant possible d'intégrer un système dans son ensemble et pas seulement une partie d'une application. On parle alors de système intégré sur puce et on utilise l'acronyme anglais de SoC pour *System-on-Chip*.

Parallèlement à l'évolution des technologies, les concepteurs doivent faire face à deux tendances. Tout d'abord l'augmentation de la complexité qui est liée à la convergence des applications. Aujourd'hui un circuit intègre des fonctionnalités hétérogènes pour répondre à des applications variées comme la communication ou le traitement des

TAB. 1.1 – Tendances futures pour les ASIC et les processeurs hautes-performances d'après [ITRS05]

Année de production	2005	2006	2007	2009	2012	2016	2020
DRAM 1/2 pas (nm)	80	70	65	50	36	22	14
Métal M1 1/2 pas (nm)	90	78	68	52	36	22	14
Nombre maximum de transistors (millions)	1928	2430	3061	4859	9718	24488	61707
Taille maximale du circuit (mm^2)	858	858	858	858	858	858	858
Fréquence interne maximale (MHz)	5204	6783	9285	12369	20065	39683	73122
Niveaux de métal	15	15	15	16	16	17	18
Puissance maximale (W)	167	180	189	198	198	NC	NC

images, de la vidéo et du son. Ensuite, les pressions du marché rendent les temps de développement de plus en plus courts. Si il y a dix ans, il était possible de développer en 18 mois un circuit de 250k portes, aujourd'hui, un circuit d'une dizaine de millions de portes doit être conçu en moins de 6 mois pour assurer sa rentabilité [CCHM99]. L'avenir de l'activité de conception repose sur la capacité à limiter les coûts pour un nouveau circuit.

Les investissements sont surtout orientés sur des aspects technologiques et moins sur les techniques de conception. C'est pourquoi des problèmes de productivité apparaissent lorsque les capacités à concevoir des circuits qui exploitent efficacement les nouvelles technologies n'arrivent pas à suivre les évolutions de ces technologies (*Design Productivity Gap* [ITRS05]) : il faudrait augmenter la taille des équipes et le temps de conception. Pour palier cette limite, les techniques de conception évoluent en continu. La tendance est de synthétiser un circuit à partir de blocs déjà validés qui implémentent une ou plusieurs fonctions et qui sont conçus pour être facilement réutilisés. On utilise le terme d'IP pour *Intellectual Property*. On parle alors de conception à partir d'IP ou d'*IP-based design*. Ainsi les circuits devraient devenir de plus en plus complexes en intégrant plusieurs centaines d'unités de traitement dans les années à venir (Tableau 1.2).

Se pose alors un nouveau problème qui est de savoir comment interconnecter ces nombreux blocs et comment les faire communiquer entre eux. Ceci est d'autant plus complexes que les IP peuvent être de nature très hétérogènes ce qui se traduit par des comportements et des modèles de communication variés. Certaines applications ont des

TAB. 1.2 – Evolution de la complexité des SoC

Année de production	2005	2006	2007	2009	2012	2016	2020
Nombre d'unités de traitement	16	23	32	63	133	348	878
Taille de la logique (normalisée à 2005)	1,00	1,27	1,62	2,81	5,50	13,77	34,15
Réutilisation de blocs (%)	32	33	35	38	42	49	55
Fonctions reconfigurables (%)	23	26	28	30	40	50	62

contraintes fortes en terme de débit, d'autres au niveau de la latence et le concepteur doit faire des compromis pour optimiser l'architecture.

Les réductions technologiques sont plus efficaces pour les transistors que pour les fils d'interconnexion. Les fils prennent une part de plus en plus importante dans les performances (consommation et surface). Au niveau système, il faut maintenant davantage se concentrer sur le transport de données et la communication que sur le calcul et le traitement [SSMK01]. Les communications deviennent un goulot d'étranglement dans les SoC. En effet, les unités de traitement ne pourront pas travailler à leur cadence maximale si les communications ne suivent pas. La conception se concentre alors sur les communications, on parle ainsi de *communication-based design* [KNRS00]. Au delà des blocs d'IP, c'est alors l'ensemble des ressources de communication qu'il faut rendre réutilisables.

1.1.2 Exigences des futures structures d'interconnexion

Pour faire face aux évolutions que nous venons de présenter, les architectures de communication devront répondre aux différentes exigences que nous allons détailler :

- Flexible et extensible

L'architecture de communication est conçue pour interconnecter de nombreux blocs d'IP de nature hétérogène, de plus en plus complexes intégrant des fonctions ayant des caractéristiques très diverses. Le protocole d'échange des données doit donc être suffisamment flexible pour s'adapter à ces modules qui peuvent être développés par différentes équipes, avec différents outils, dans différents contextes et qui sont destinés à être intégrés dans le même circuit. Cette flexibilité peut être obtenue en spécifiant de manière rigoureuse des interfaces qui permettent de découpler les fonctions de traitement et les fonctions de communication. On utilise le terme d'*interface-based design* [RoSV97]. Une architecture de communication flexible permet aussi de gagner en productivité. En effet,

elle pourra être réutilisée d'un circuit à l'autre et supporter les besoins de nombreuses applications. La notion d'extensibilité[3] est également un élément important. Il s'agit de la possibilité d'augmenter le nombre de blocs interconnectés tout en maintenant le même niveau de performance dans les communications.

- Implémentation distribuée

Avec l'évolution des technologies d'intégration, il semble de plus en plus difficile de maintenir une synchronisation globale dans l'ensemble d'un circuit. En effet les délais globaux des fils deviennent supérieurs aux délais des portes logiques [HoMH01]. Avec une technologie 35nm, moins de 0,4% du circuit pourrait être atteint en un cycle d'horloge [AHKB00]. Si l'on ajoute l'augmentation des fréquences, il est donc peu probable que les futurs circuits soit entièrement synchrones. Les structures de communication devront permettre de faire cohabiter plusieurs domaines d'horloge. On s'oriente vers des systèmes globalement asynchrones et localement synchrones (GALS) [MHKE99] [MVKF99]. Les modules synchrones devront ainsi organiser des transferts de façon autonome avec un système de communication distribué sans contrôle global.

- Robuste aux défauts technologiques

Les réductions technologiques font apparaître des phénomènes indésirables de plus en plus marqués. On parle d'effets submicroniques profonds[4] [BlGa01] pour les technologies en dessous de $0{,}18\mu m$. Il s'agit principalement de bruit sur les interconnexions qui entraîne des erreurs sur les transmissions. Les origines sont nombreuses : paramètres de fabrication, alimentation, diaphonie, couplage... Un concepteur qui doit intégrer plusieurs millions de transistors ne peut pas gérer tous ces effets indépendamment. Il utilise de ce fait des blocs préalablement caractérisés. Il faut alors veiller à ce que la méthode d'interconnexion n'introduise pas de nouveaux effets perturbateurs. Pour cela, il faut préférer des architecture de communication qui présentent des géométries régulières avec des propriétés électriques prévisibles.

1.1.3 Les solutions d'interconnexion actuelles et leurs limites

Les communications sur puce ont traditionnellement été assurées par des connexions directes (liaisons point-à-point), ou des bus partagés (Figure 1.1) [LaRD01].

Les connexions directes consistent à relier physiquement un émetteur à un récepteur. Ces canaux de communications dédiés pour relier les modules entre eux assurent les

[3] traduction du terme anglo-saxon de *scalability*
[4] *deep submicron effects*

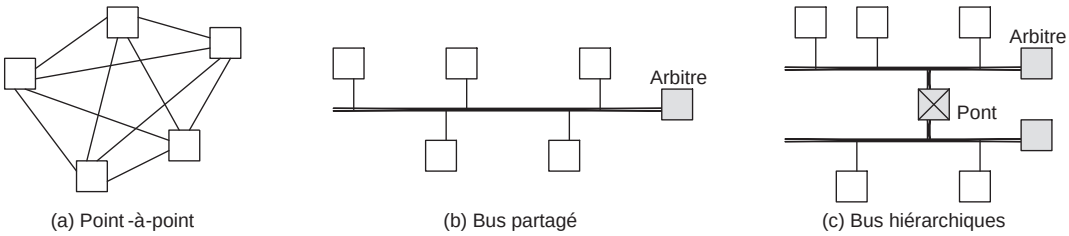


FIG. 1.1 – Architecture d'interconnexion sur puce

meilleures performances en terme de bande passante puisque celle-ci n'est pas partagée avec d'autre modules. De plus il n'y a pas d'arbitrage à effectuer pour accéder aux liens de communication ce qui réduit la complexité des ressources de contrôle. Cependant, ces interconnexions sont difficilement réutilisables et leur conception devient difficile lorsque les systèmes se complexifient [ZKCS02].

1.1.3.1 Bus partagés

Dans les systèmes sur puces complexes, le bus partagé est aujourd'hui le moyen de communication le plus répandu dans l'industrie. Il existe différents produits sur le marché (Tableau 1.3) qui sont en général proposés par des fournisseurs d'IP. Le bus offre alors une solution maîtrisée pour connecter entre eux un ensemble de composants compatibles avec le protocole de communication.

TAB. 1.3 – Architectures de bus pour les systèmes sur puce

Architecture de bus	**Entreprise / Propriétaire**	**Référence**
Advanced Microcontroller Bus Architecture (AMBA)	ARM	[Flyn97]
CoreConnect	IBM	[HoDr02]
CoreFrame	PalmChip	[Cord99]
STBus	STMicroelectronics	[STBU03]
SiliconBackplane	Sonics	[Wing01]
Peripheral Interconnect Bus (PI-Bus)	Open Microprocessor systems Initiative (OMI)	[PIBU95]
Wishbone Interconnect	Silicore Corporation	[WISH02]

Contrairement aux liaisons point-à-point, les bus sont flexibles et facilement réutilisables. Cependant ils ne permettent qu'une seule communication à la fois entre les

modules qui s'y connectent. La bande passante est donc partagée et les performances diminuent lorsque le nombre de modules augmente. Un mécanisme d'arbitrage est nécessaire pour gérer les accès simultanés. De plus les bus nécessitent des longueurs de fils importantes qui consomment beaucoup d'énergie puisque les données sont diffusées sur tout le bus. Lorsqu'on augmente le nombre de modules, la taille du bus augmente, ce qui a pour conséquence d'augmenter le délai de transmission des données au point de dépasser la période d'horloge [GPIS04]. Le bus est donc par nature limité dans sa capacité à interconnecter un grande nombre de blocs d'IP.

L'introduction de bus hiérarchiques permet de surmonter certains inconvénients. Le principe est de relier plusieurs bus entre eux par l'intermédiaire de ponts (*bridge*). Chaque bus possède son propre arbitre. Les unités de traitement sont groupées en fonction des communications. On réduit ainsi le nombre d'éléments connectés à un même bus, la taille des fils est réduite ce qui réduit la consommation. Ce type de solution reste cependant complexes notamment lorsque les informations doivent transiter par les ponts pour passer d'un bus à l'autre (problèmes de gestion de l'adressage).

1.1.3.2 Standards d'interface

Pour simplifier l'intégration d'IP autour d'une même structure d'interconnexion des standards d'interfaces ont été proposées par des consortia. Ces interfaces standardisées permettent de développer des composants réutilisables indépendamment du système de communication auquel il sera raccordé.

Le consortium VSIA [VSIA06] a proposé le standard VCI (*Virtual Component Interface*). Deux types d'IP sont définis : les initiateurs qui émettent des requêtes et les cibles qui y répondent. Les communications sont de type point-à-point et unidirectionnelle. Si une IP doit être à la fois initiateur et cible, les deux interfaces sont implémentées pour ce module.

La norme OCP (*Open Core Protocol*) soutenue par le consortium OCP-IP [OCPI06] définit également un protocole point-à-point aux fonctionnalités proches de celles de VCI. Elle incluent en plus des évolutions notamment dans la gestion de signaux de test.

Globalement l'utilisation de ces interfaces est encore assez peu répandue car elles sont trop génériques et des solutions propriétaires répondent mieux aux besoins spécifiques des applications. Une analyse plus détaillée des différents systèmes d'interface est présentées dans [CoBM02].

1.1.4 Introduction aux réseaux sur puce

Les bus ont été implémentés avec succès dans de nombreux types de systèmes sur puce. L'évolution des technologies d'intégration, la convergence des applications, la réduction des temps de mise sur le marché ont entraîné de profonds changements dans les architectures de bus. Ces évolutions ont eu des répercutions au niveau des blocs d'IP réutilisables. En effet, les protocoles de bus n'utilisent pas de pile protocolaire organisée en couche permettant de hiérarchiser les services. Ainsi l'ajout de nouvelles fonctionnalités induits des modifications majeurs jusqu'au niveau de l'interface entre le bus et l'IP [Arte06].

De plus comme nous l'avons vu précédemment, les bus présentent des limitations intrinsèques au niveau de la bande-passante et de la flexibilité qui ne sont pas compatibles avec les futures générations de circuits intégrant un très grand nombre de ressources de traitement. En conséquence, les bus ne permettent pas de suivre suffisamment rapidement l'évolution des technologies.

Dans le domaine des communications, les réseaux informatiques ont su faire face à des problèmes similaires en utilisant des structures d'interconnexion distribuées et bien hiérarchisées. Ils sont étudiés depuis des dizaines d'années aussi bien dans le contexte des réseaux locaux[5] [Tane96] que dans l'interconnexion de machines parallèles [CuSG99]. En découplant les couches de communications, il a été possible de faire évoluer les architectures au niveau physique (Ethernet, ADSL, fibre optique, sans-fil) et au niveau des transactions (internet, transport de la voix, de la vidéo) tout en gardant une compatibilité entre les systèmes.

C'est à partir de ces constations qu'est née l'idée d'intégrer un réseau de communication sur un circuit intégré dans le but de proposer une architecture réellement nouvelle et compétitive en terme de délais, consommation et surface.

La recherche dans ce domaine s'est réellement intensifiée autour des années 2000 quant il est devenu clair pour l'ITRS[6] que de nouvelles architectures de communication sur puce étaient nécessaires pour surmonter les problèmes de conceptions liés à l'augmentation de la densité d'intégration des transistors dans les circuits. Les premières approches ont abordés le problème à la fois d'une point de vue de la conception haut-niveau [HJKP00] [BeMi01] [SSMK01] et du point de vue physique [DaTo01] [HoMH01]. Rapidement d'autres contributions ont illustré les concepts par des propositions concrètes d'architecture [GuGr00] [RiGW01].

[5] LAN : *Local Area Network*
[6] *International Technology Roadmap for Semiconductors*

Concernant la terminologie, le terme le plus utilisé est celui de *réseau sur puce* traduction directe de *network-on-chip* (abrégé en *NoC*) qui dérive de *system-on-chip* (SoC). Les appellations du type *network on silicon* ou *on-chip network* sont plus rarement utilisées. La recherche sur les NoC réalise une synthèse multidisciplinaire entre les domaines du calcul distribué, des réseaux et des communications sur puce. Elle consiste en une adaptation des solutions éprouvées dans le cas des réseaux non-intégrés au cas spécifiques de la conception de SoC.

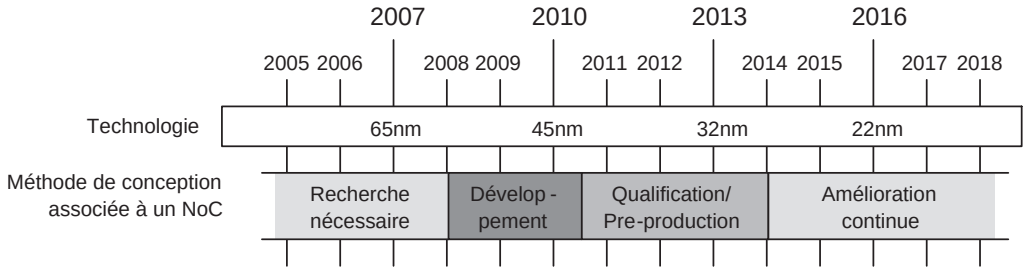


FIG. 1.2 – NoC : De la recherche à l'utilisation industrielle (d'après [ITRS05])

Aujourd'hui, tous les aspects du NoC n'ont pas encore été abordés et il n'existe pas de solution industrielle avec un niveau de maturité comparable à celui des bus. Les NoC sont encore au stade de recherche (Figure 1.2)

1.2 Concepts relatifs aux réseaux sur puce

Après avoir introduit les réseaux sur puce dans le contexte de la conception des systèmes sur puce, nous entamons cette seconde section dont l'objectif est de présenter un certain nombre de concepts relatifs aux NoC. Ces différentes notions sont issues des publications qui ont été faite dans le domaine. Cette section se décompose en deux parties. Nous nous intéresserons d'abord au NoC du point de vue des mécanismes de communication. Nous nous placerons ensuite d'un point de l'application et des ressources de traitement. En complément, nous présentons en Annexe A une comparaison entre les réseaux sur puce et les solutions d'interconnexion classiques.

1.2.1 Structures et mécanismes de communication dans les NoC

L'objectif principal d'un réseau sur puce est de fournir un système permettant d'échanger des données entre différentes ressources de traitement (ou coeur d'IP). Le

réseau est composé de *noeuds* (aussi appelés *switchs* ou *routeurs*). Les données transitent entre les noeuds grâce à des liens (ou canaux) de communication point à point. L'ensemble formé par les noeuds et les liens constitue le *système de communication*. Chaque noeud comprend un ensemble de ports de communications. Ces ports connectés aux liens permettent de relier les noeuds entre eux. Suivant les architectures, les ressources sont connectées directement aux noeuds ou par l'intermédiaire de liens. Les noeuds prennent les décisions de routage (où transmettre une donnée) ainsi que celles d'arbitrage (quelle donnée transmettre).

1.2.1.1 Topologies des réseaux sur puce

La topologie spécifie l'organisation physique du réseau. Elle définit donc comment les noeuds et les liens sont connectés entre eux. Deux nombreuses topologies sont envisageables, nous avons choisi de présenter les plus couramment utilisées sur la Figure 1.3. Ces topologies sont dîtes régulières compte tenu de leur loi de construction géométrique. Pour comparer les topologies entre elles, différents critères physiques sont utilisés. Les plus courant sont :

- Le *diamètre*. C'est le nombre maximal de liens qui séparent deux ressources quelconques du réseau (en considérant les plus courts chemins).
- La *distance moyenne*. C'est le nombre moyen de liens entre deux ressources.
- La *connectivité*. C'est le nombre de voisins directs des noeuds dans le réseau.
- La *largeur de bissection*. C'est le nombre minimal de liens qu'il faut couper pour séparer le réseau en deux parties égales (plus ou moins un noeud). Cela permet d'évaluer le coût de transférer les données d'une moitié du réseau à l'autre.

La plupart des propositions d'architecture NoC sont basées sur une topologie de maillage simple à deux dimensions[7] [STRR05]. Les trois raisons principales qui expliquent ce choix sont [MNTK04] :

- La facilité d'implémentation. En effet, avec les technologies silicium, il est plus simple de construire des structures planes. Les topologies à dimensions supérieures (tels que les cubes ou hypercubes) qui sont utilisées pour des réseaux non-intégrées ne sont donc pas reprises dans le cas des NoC. De plus les noeuds ont la même architecture et les liens sont de même longueur ce qui facilite les étapes de placement et de routage.
- Une connectivité suffisante. Les architectures de NoC sont étudiées dans le but de réaliser des circuits intégrés complexes qui répondent le plus souvent à des be-

[7] la dénomination anglo-saxonne couramment employée est *2D mesh*

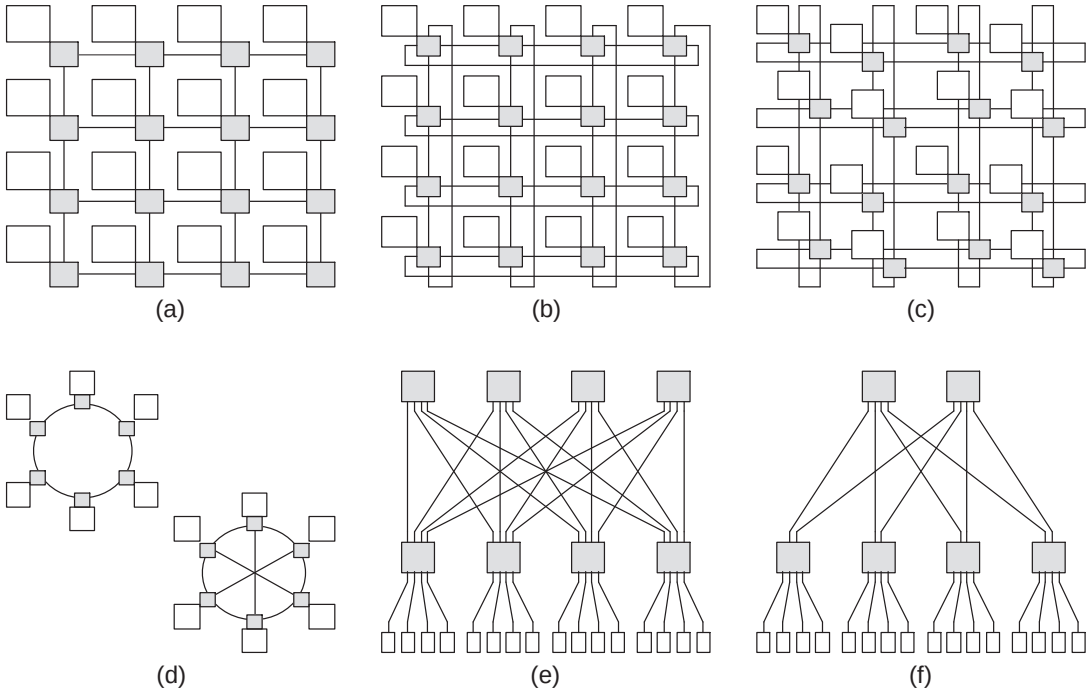


FIG. 1.3 – Topologies usuelles de réseaux sur puce : (a) réseau à maillage simple à deux dimensions (b) réseau maillé en tore (c) réseau maillé en tore enfoui (d) réseau en anneau avec ou sans corde (e) réseau en arbre élargi (f) réseau en arbre élargi en papillon (*butterfly fat-tree*)

soins dans le domaines des télécommunications, du multimédia ou du traitement du signal au sens large. Dans la plupart des ces applications, les schémas de communication sont locaux. Une ressource du réseau n'a pas besoin de communiquer avec toutes les autres. La topologie peut donc rester simple. Ceci est à mettre en contraste avec les architectures de calculs massivement parallèles pour lesquelles la topologie est optimisée pour des communications arbitraires avec l'ensemble des ressources.

– Des propriétés intrinsèques intéressantes. La complexité des noeuds est limitée à au plus cinq ports d'entrée/sortie ce qui simplifie l'algorithme de routage des données (cf. paragraphe 1.2.1.4). Il est facile de construire différents chemin entre une source et sa destination ce qui apporte de la robustesse dans cas de perturbation sur le réseau.

Parmis les topologies maillées à deux dimensions, le choix entre le type maillage simple[8], tore[9] ou tore enfoui[10] est souvent discuté [DaTo01] [ZeSu03]. En effet, les interconnexions en tore offrent une meilleure utilisation des ressources réseau, puisque pour un même nombre de noeud, il y a plus de liens ce qui a pour effet de diminuer le diamètre et d'augmenter la bande-passante (largeur de bissection double). En contre partie, la structure est moins régulière que pour un réseau maillé et les fils utilisés pour boucler le réseau sont plus longs et pénalise donc les performances des liens. Le tore enfoui est un compromis au niveau de la longueur des fils cependant il introduit un complexité dans la détermination des chemins de routage entre sources et destinations.

Les réseaux de types anneau à corde[11] ou en arbre élargi[12] ont un diamètre plus petit que les réseaux maillés. Ils permettent donc de réduire la latence mais leurs structures sont moins régulières. Une étude comparative a été effectuée [PGJI05] et on observe qu'avec un trafic uniformément réparti, le débit atteignable avant saturation est plus élevé avec ces topologies par rapport aux topologies maillées. Cependant, l'étude montre que les topologies maillés exploitent avantageusement le fait que dans les SoC le trafic sera majoritairement local.

Des réseaux sur puces peuvent aussi être construits à partir de topologies non régulières. Le principes est de déterminer la topologie la mieux adaptée aux flots de donnée et aux contraintes de performances imposées. L'architecture du NoC est construite à partir de blocs d'IP paramétrables que les concepteurs peuvent assembler en fonction de l'application ciblée. Dans [BeBe04], des travaux, basés sur une architecture de NoC pour du décodage vidéo, comparent une topologie maillée générique avec deux topologies spécifiques et montrent des gains en surface et en consommation tout en obtenant de meilleurs performances au niveau latence. Des topologies hybrides sont également possibles comme proposées par [WiGo02]. Les ressources sont connectées localement par une architecture classique de bus. Les différents bus du circuit sont reliés entre eux par un réseau. Cela permet de combiner les performances de trafic local d'un bus de petite taille avec la flexibilité des interconnexions du réseau.

Au final, le choix de la topologie est complexe. Il est fortement lié aux contraintes de l'application et au trafic sur le réseau. Il résulte d'un compromis entre les performances intrinsèques de la topologie et le coût que représente son implémentation.

[8] *mesh*
[9] *torus*
[10] *folded torus*
[11] traduction de l'anglais *chordal ring network*
[12] traduction de l'anglais *fat-tree network*

1.2.1.2 Mécanismes de communication et modes de commutation

Après avoir sélectioné une topologie, il faut spécifier comment les informations vont transiter sur le réseau : c'est le *mécanisme de commutation*. Deux méthodes sont principalement utilisées [MMMO03] :

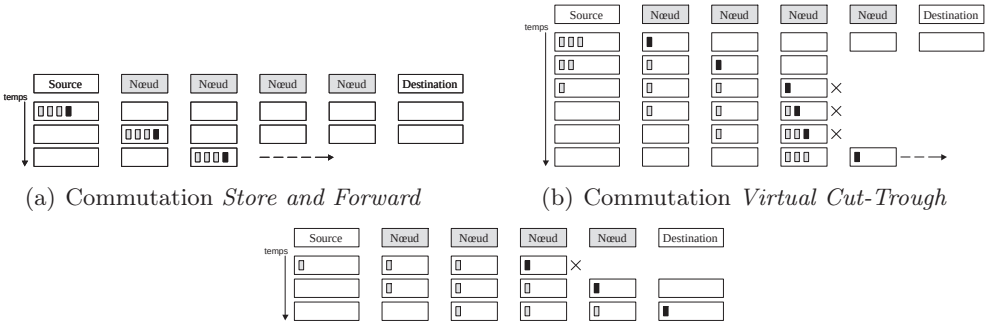
- La commutation de circuit[13]. Dans ce cas une connexion est établie entre la source de données et sa destination. Cela signifie que des éléments du réseau (liens, noeuds) sont alloués pour un transfert de données et ne peuvent pas être utilisés par d'autres ressources durant le temps de la connexion. Il existe un lien physique (souvent pipeliné) entre les deux ressources. Les informations de contrôle (pour le début et la fin de connexion par exemple) sont séparées des données. Ce mécanisme est bien adapté lorsque l'on veut effectuer des transferts de données importants puisqu'il faut rentabiliser le temps passé à négocier la connexion. Physiquement les noeuds sont simples puisqu'ils établissent seulement un lien entre un port d'entrée et un port de sortie. Il n'y a pas de blocage : la connexion est ou n'est pas établie. La commutation de circuit est souvent utilisée lorsque l'on veut des garanties de services (cf. paragraphe 1.2.2.1).

- La commutation de paquet[14]. Avec ce mécanisme, la structure d'échange au niveau du réseau est le paquet. Un paquet peut contenir une fraction, un ou plusieurs messages. Les paquets sont transmis sans qu'il y ait de connexion établie préalablement. L'avantage est de pouvoir accéder au réseau plus rapidement et de partager les ressources. Le réseau est localement commuté entre deux noeuds mais pas entre deux ressources comme dans la commutation de circuit. Les paquets doivent contenir des informations sur leurs destinations : les informations de contrôle sont envoyées en même temps que les données. Les noeuds sont souvent plus complexes, ils ont parfois à modifier le contenu des paquets pour mettre à jour des informations de routage par exemple, la latence est plus grande. Il faut introduire des systèmes de gestion des blocages et des priorités. Les paquets peuvent arriver de façon désordonnée et il y a moins de garantie temporelle sur le transfert.

Dans le cas de la commutation par paquet, il faut définir le mode de commutation, c'est à dire la façon dont les paquets vont transiter d'un noeud du réseau au suivant. Les trois principaux modes sont [RGRD03] :

- Store-and-forward. Ce mode de commutation consiste à faire transiter le paquet dans son ensemble d'un noeud à l'autre (Figure 1.4(a)). Cela implique que chaque noeud doit être en mesure de stocker la totalité d'un paquet. Il existe donc une taille de paquet

[13] traduction de *circuit-switching*
[14] traduction de *packet-switching*



(a) Commutation *Store and Forward*

(b) Commutation *Virtual Cut-Trough*

(c) Commutation *Wormhole*

FIG. 1.4 – Différents modes de commutation

maximale. De plus la latence des échanges est importante puisqu'il faut multiplier le temps de transfert d'un paquet entre deux noeuds par le nombre de noeuds du réseau traversés par le paquet de sa source à sa destination.

- Virtual cut-through. Avec ce mode un noeud peut commencer à envoyer un paquet si le noeud suivant lui garantit qu'il peut stocker le paquet dans sa totalité, dans le cas contraire le noeud doit pouvoir garder le paquet (Figure 1.4(b)). La capacité de mémorisation du noeud est donc la même que pour le mode *store-and-forward* mais la latence est diminuée puisqu'il n'est plus nécessaire d'attendre la réception complète du paquet pour qu'il passe d'un noeud à l'autre.

- Wormhole. Ce mode a pour but de réduire la taille des mémoires dans les noeuds sans limiter la taille des paquets. Les paquets sont découpés en éléments de taille fixe appelés flits (*flow control unit*). Les flits passent de noeud en noeud dès qu'il y a de la place pour un flit et pas nécessairement pour un paquet complet (Figure 1.4(c)). Le flit de tête[15] contient des informations de contrôle, en particulier sur la destination du paquet. Tous les flits qui suivent doivent emprunter le même chemin que le flit d'en-tête. Un même paquet peut donc être réparti sur plusieurs noeuds du réseau. La commutation *wormhole* permet donc de réduire la latence. En contre partie, les risques de blocage sont plus importants comme nous verrons par la suite.

Le Tableau 1.4 résume différents aspects à prendre en compte lors du choix du mode de commutation. En pratique, le mode wormhole est le plus utilisé car le critère de mémorisation est une contrainte forte pour l'implémentation silicium.

[15] appelé en-tête ou *header*

TAB. 1.4 – Comparatif de différents modes de commutation (D'après [RGRD03])

Mode	Store-and-forward	Virtual cut-through	Wormhole
Taille mémoire du noeud	Paquet complet	Paquet complet	Fraction d'un paquet
Latence	Grande	Faible	Faible
Interblocage	Non	Non	Possible

1.2.1.3 Stratégies de stockage et canaux virtuels

Nous venons de voir que les noeuds du réseau ont besoin d'éléments mémoire quelque soit le mode de commutation retenu. Le positionnement des mémoires par rapport aux ports d'entrée et de sortie du noeud peut se faire selon différentes stratégies [RGRD03] :

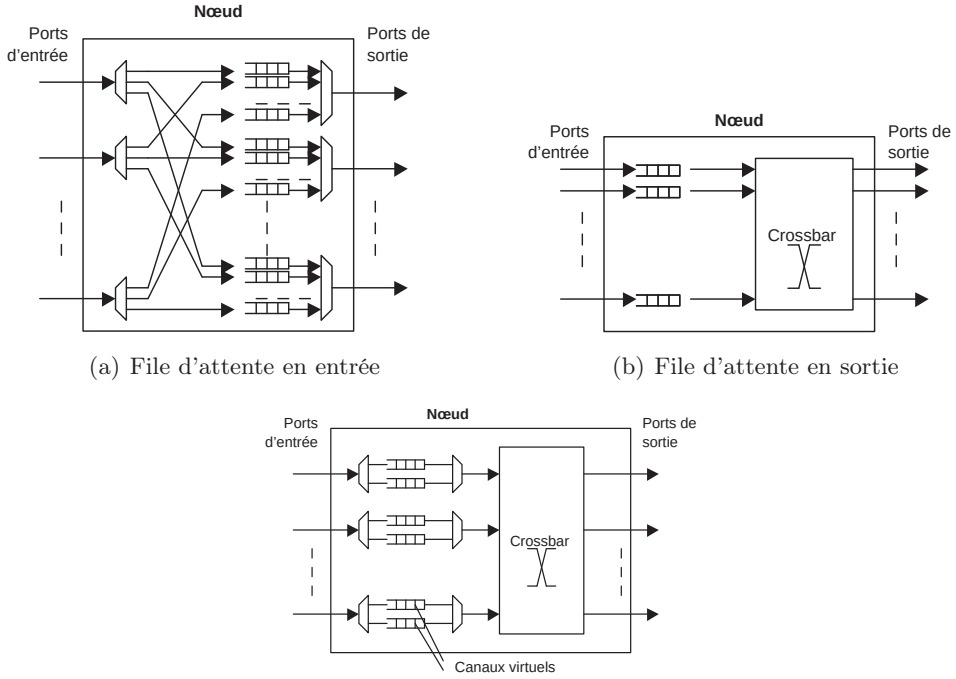
- File d'attente en sortie[16]. Chaque port de sortie a autant de files d'attente qu'il existe de ports d'entrée. Le noeud peut ainsi recevoir des paquets sur toutes ses entrées simultanément avant d'effectuer un arbitrage pour chaque sortie. C'est la stratégie qui offre les meilleures performances mais elle nécessite beaucoup de fils de connexion et un espace mémoire important : pour un noeud à N entrées et N sorties, il faut N^2 files d'attente.

- File d'attente en entrée[17]. On utilise une seule file d'attente pour chaque entrée du noeud. Pour un noeud à N entrées, il y a donc N files. Si cette technique présente l'avantage de réduire l'espace mémoire, elle entraîne un saturation rapide [KaHM87] à moins de 60% d'utilisation des capacités théoriques du noeud à cause du blocage en tête de ligne[18]. Ce phénomène se produit lorsqu'une donnée en tête de file n'a pas accès à un port de sortie et bloque les données suivantes dans la file.

- File d'attente en sortie virtuelle[19]. Cette stratégie a pour but de combiner les avantages des deux précédemment citées. Le principe est d'utiliser plusieurs files d'attente sur les ports d'entrée et d'y répartir les paquets suivant leurs destinations ou un niveau de priorité. On crée ainsi des canaux virtuels [Dall90]. Pour un canal physique (lien entre deux noeuds), il y a ainsi plusieurs files d'attente qui correspondent à des canaux virtuels et qui se partagent la bande passante du lien. De cette façon un paquet

[16] Traduction de *Output queuing*
[17] Traduction de *Input queuing*
[18] Traduction de *head-of-line blocking*
[19] Traduction de *Virtual output queuing*

provisoirement bloqué et stocké sur plusieurs noeuds du réseau peut être doublé par un autre paquet utilisant le même canal physique mais stocké sur un autre canal virtuel. On peut ainsi augmenter l'utilisation des capacités des liens en réduisant le phénomène de blocage en tête de ligne. En contre partie, la latence des paquets augmente avec le nombre de canaux virtuels. Des études [PGJI05] montrent que quatre canaux virtuels constitue un bon compromis entre débit élevé et latence raisonnable.



(a) File d'attente en entrée

(b) File d'attente en sortie

(c) *File d'attente en sortie virtuelle*

FIG. 1.5 – Différentes stratégies de stockage dans le noeud

1.2.1.4 Algorithmes de routage et interblocages

- L'algorithme de routage détermine quelle série de noeuds les données vont emprunter entre la ressource émettrice à la ressource réceptrice. C'est un point déterminant dans la conception d'un NoC. Il faut réussir le meilleur compromis entre une utilisation optimale des canaux de communication du réseau et un algorithme suffisamment simple à implémenter ne nécessitant pas des ressources matérielles trop importantes.

Il existe de nombreux algorithmes de routage avec différentes propriétés. Le routage est de type *source* lorsque c'est l'émetteur qui définit le chemin de routage. Au contraire,

si chaque noeud choisit la destination du paquet en fonction de son adresse cible (et d'autres critères éventuels), il s'agit d'un routage *distribué*.

Lorsque le routage dépend seulement de l'expéditeur et du destinataire il est dit *déterministe*. Au contraire, si le chemin peut varier en fonction du trafic, en tenant compte des zones de congestions par exemple, l'algorithme de routage est dit *adaptatif*. Le routage adaptatif est utilisé dans le cas d'application avec un trafic irrégulier mais si les échanges de données sont fortement prévisibles, un routage déterministe est préférable et plus simple à implémenter [BeMi02]. Dans le cas d'un routage adaptatif, le choix peut être *partiel*, c'est à dire effectué parmi un nombre restreint de chemins. Il peut également être *global*, le paquet peut être routé dans l'ensemble du réseau avant d'arriver à destination.

Les NoC sont des structures de communication distribuées. Les paquets circulent de noeud en noeud sans contrôle global de leurs cheminement. Ce type de fonctionnement peut potentiellement créer des situations de blocage. Ils faut alors mettre en place des mécanismes pour les empêcher. L'existence de ces situations de blocage est fortement liée au l'algorithme de routage comme nous allons le voir.

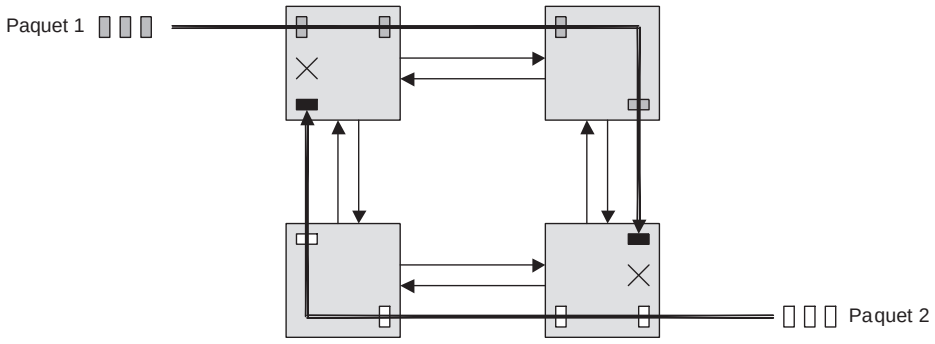


FIG. 1.6 – Exemple d'une situation d'interblocage statique

- Les interblocages statiques[20]. Ces interblocages se produisent lorsque plusieurs paquets se bloquent mutuellement à cause d'une dépendance cyclique. Chaque noeud est alors en attente d'envoyer un paquet vers un noeud qui est déjà en attente. La commutation *wormhole* est fortement sujette à ce type d'interblocage comme l'illustre la Figure 1.6. Le choix de l'algorithme de routage peut permettre d'éviter ces blocages. Avec une topologie maillée, le fait d'interdire certains virages supprime les dépendances circulaires par exemple [GlNi94]. On peut également utiliser des canaux virtuels, de cette façon un paquet provisoirement bloqué et stocké sur plusieurs noeuds du réseau

[20]Traduction de *Deadlock*

peut être doublé par un autre paquet utilisant le même canal physique mais stocké sur un autre canal virtuel.

- Les interblocages dynamiques [21]. Ce phénomène se produit lorsqu'un paquet n'arrive jamais à destination tout en continuant à circuler sur le réseau. Par exemple, avec un algorithme de routage adaptatif, un paquet est toujours dérouté par rapport à sa destination à cause de congestion sur le réseau. Un paquet peut également être bloqué de manière dynamique à cause d'une mauvaise gestion des priorités au niveau des noeuds si par exemple le liens de sortie est toujours occupé par des paquets considérés comme plus prioritaires. Ceci crée un phénomène dit de *famine*[22].

De nombreuses propositions sont basées sur le routage XY [MTCM05] [BCGK04] [ZeSu03]. Ce routage est lié aux topologies maillées à deux dimensions. Il consiste à faire circuler les paquets d'abord dans la direction X (Est-Ouest) puis dans la direction Y (Nord-Sud). C'est un routage déterministe, peu coûteux à implémenter. D'autres algorithmes déterministes sont utilisés mais les détails d'implémentation ne sont pas donnés [BeBe04] [STNu02] [WiGo02]. En général, ils sont de type source ce qui offre plus de souplesse dans le choix du chemins de routage. Le choix d'un routage adaptatif est plus rare car leurs mécanismes sont plus complexes. L'objectif est d'éviter la formation de points de congestion[23]. Il est ainsi parfois intéressant de détourner un paquet en rallongeant son itinéraire dans le réseau pour réduire les phases d'attente dans les noeuds afin de gagner en latence. Un mécanisme basé sur ce principe est proposé dans [YeBM03b]. Le gain par rapport à un routage déterministe est cependant très faible dans les résultats de simulation présentés.

1.2.2 Services, protocoles et communications dans les NoC

Nous venons de présenter différents concepts sur les mécanismes de communication dans les NoC. A présent, nous allons nous intéresser aux aspects liés à la mise en oeuvre d'un NoC : les services, le protocole et l'intégration des unités de traitement par le biais d'interfaces réseau.

1.2.2.1 Notions de services et de qualité de service

Comme pour tout système de communication sur puce, les échanges de données sur un NoC sont associées à différents niveaux de service. Les Laboratoires de Recherche Phi-

[21] Traduction de *Livelock*
[22] Traduction de *Starvation*
[23] *hot-spot*

lips ont présentés ces notions dans diverses publications [GDMP03] [RGRD03] [GPDG05]. Les services considérés comme essentiels sont :

- *Intégrité des données.* Ceci signifie que les données sont transmises sans altération.
- *Absence de perte de données.* Lors des transmission, aucun paquet ou flit n'est perdu dans le réseau. Une mauvaise gestion des flux ou espaces mémoire dans les noeuds peut être la cause de ces pertes.
- *Ordonnancement des données.* Les données sont reçues dans l'ordre où elles ont été émises. Avec un routage déterministe, il suffit d'éviter aux paquets de se doubler. Par contre lorsque le routage est adaptatif il faut mettre en place à la réception des mécanismes de réordonnancement basé sur une numérotation des paquets par exemple.

D'autres services peuvent définir des contraintes associées aux performances souhaitées pour les transferts sur le réseau :

- *Débit.* Il définit la quantité de données qui transitent d'une ressource à l'autre par unité de temps.
- *Latence.* C'est le temps entre l'émission d'un donnée sur le réseau et sa réception par la ressource destinatrice.

Pour fournir ces services, il existe deux niveaux d'implémentation :

- Les réseaux à *services garantis*[24]. Dans ce type de réseau, des mécanismes permettent de réserver des ressources de communication pour garantir un certain débit et ou une certaine latence quelques soit la charge du réseau. Ce type de garantie a pour effet de surdimensionner les ressources. De telles garanties peuvent être nécessaires pour des messages très prioritaires comme des interruptions par exemple.
- Les réseaux à *services meilleur-effort*[25]. Le réseau est dimensionné pour avoir de bonnes performances moyennes mais il n'offre aucune garantie de latence ou de débit dans les situations les moins favorables. En effet les ressources ne peuvent pas être réservées et se partagent entre toutes les ressources du réseau.

Assurer une Qualité de Service [26] revient à trouver un bon compromis entre les services imposés par l'application visée et le coût des ressources matérielles.

[24] *guaranteed services*
[25] *best-effort services*
[26] QoS - Quality of Service

1.2.2.2 Protocole de communication

Un ensemble de règles et de méthodes sont nécessaires pour transférer une information d'une ressource à une autre sur le réseau. Ces règles constituent le *protocole de communication*. Pour simplifier l'implémentation du protocole, celui est divisé en différentes couches avec des fonctionnalités spécifiques et des interfaces. L'ensemble de ces couches constitue la pile protocolaire[27].

Différentes publications [SSMK01] [MNTK04] [BeMi02] ont proposé de formaliser les protocoles de communication implémentés dans les NoC en adaptant la structure hiérarchique du modèle de référence OSI[Zimm80] dont les sept couches sont présentées Figure 1.7. Dans la plupart des cas seules les quatre couches basses du modèle OSI sont implémentées dans le NoC. Les couches supérieures sont en général plus spécifiques à l'application et prises en charge par les ressources (IP, CPU) connectées au réseau

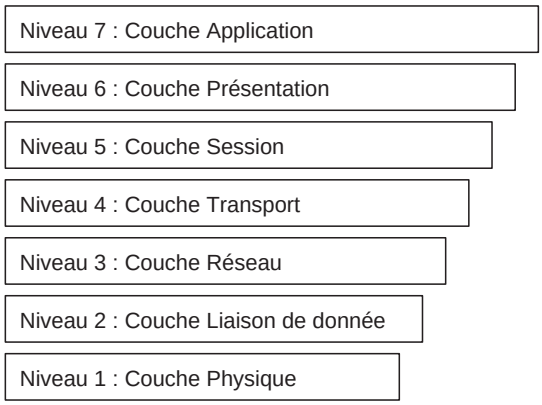


FIG. 1.7 – Sept niveaux du modèle OSI

- La couche physique : elle correspond au niveau le plus bas dans la description de la transmission des données. Elle définit les détails d'implémentation : la tension et les temps caractéristiques des signaux, le nombre et la longueur des fils d'interconnexion entre les noeuds. Cette couche prend aussi en charge la synchronisation des signaux dans le cas où il existe plusieurs domaines d'horloge sur le réseau. Elle est le support de transmission de l'information au niveau bit ou mot entre deux noeuds.

- La couche liaison de données : elle définit la façon de transmettre les informations entre deux noeuds. Elle établit une connexion logique entre les noeuds et assure la fiabilité de la transmission. Elle peut inclure des mécanismes de synchronisation, de contrôle de flux et de correction d'erreur.

[27] *protocol stack*

Les deux couches précédentes sont dépendantes de la technologie utilisée (par exemple un message doit être envoyer en pipeline s'il n'y a pas assez de fils pour l'envoyer en une seule fois).

- La couche réseau : elle définit comment envoyer un paquet à travers le réseau d'un expéditeur à un destinataire. Cette couche dépend également de la technologie (nombre de fils disponible pour la taille d'un paquet et des adresses de destination). Elle prend en charge les mécanismes de commutation et les algorithmes de routage. Dans le cas d'une commutation wormhole, elle assure le découpage des paquets en flits.

- La couche transport : Elle permet aux ressources de faire abstraction des mécanismes de la couche réseau. Elle assemble et désassemble les paquets. Elle prend également en charge le contrôle de flux. Le contrôle de flux est un ensemble de mécanisme pour éviter la surcharge du réseau et réguler le trafic. Il existe trois techniques principalement utilisées :

- Technique de fenêtrage : une fenêtre glissante détermine combien de paquets peuvent circuler dans le réseau pendant une période prédéfinie.
- Technique de contrôle de débit : le débit d'envoi de chaque émetteur est contrôlé et limité.
- Technique des crédits : avant d'envoyer un paquet, l'émetteur doit avoir reçu des crédits en provenance du destinataire. Régulièrement, le destinataire envoie des crédits à l'émetteur en fonction de ses capacités de stockage de nouveaux paquets. L'émetteur décrémente son compteur de crédit après avoir envoyer un paquet.

Ces quatre couches constituent un cadre pour la spécification d'un protocole pour un NoC. Il n'existe pas aujourd'hui des protocole de NoC standardisé. La tendance est à la création de réseaux spécifiques en cherchant à optimiser les performances en fonction de classes d'applications bien précises.

1.2.2.3 Concepts, fonctionnalités et implémentation des interfaces réseau

L'intégration d'une unité de traitement dans le système de communication est un aspect important à prendre en compte lors de la conception d'un NoC. La tendance avec les systèmes sur puces complexes actuels est de faciliter l'intégration de blocs réutilisables (IP). Pour cela, il paraît intéressant de pouvoir concevoir une unité de traitement indépendamment du réseau d'interconnexion sur lequel elle sera connectée.

Dans ce but, les différentes contributions dans le domaine des NoC introduisent généralement le concept d'interface réseau[28] (NI). Cette interface fournit un ensemble de

[28] *Network Interface*

services à l'unité de traitement tout en masquant les mécanismes internes de fonctionnement du réseau [MNTK04]. Une ressource est donc constituée de l'association d'une interface réseau avec une unité de traitement (Figure 1.8).

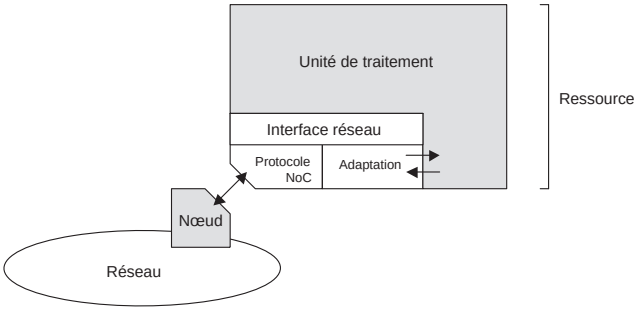


FIG. 1.8 – Positionnement de l'interface réseau dans le contexte d'un NoC

L'interface est connectée au réseau. On peut distinguer deux parties dans l'interface. La partie connectée au réseau est indépendante de l'unité de traitement. Elle implémente les trois couches les plus basses du protocole de communication permettant la gestion des communications sur le réseau. Une seconde partie connectée à l'unité de traitement implémente la couche transport. Elle réalise la conversion entre le protocole de communication au niveau réseau et les échanges de données au niveau de l'unité de traitement.

L'interface réseau est donc le point d'accès au réseau. Elle masque les mécanismes dépendant du réseau au niveau de la couche transport et permet ainsi de réutiliser une unité de traitement quelque soit la façon dont l'architecture des couches basses du réseau évolue. C'est un décisif à prendre en compte dans le processus de conception de systèmes sur puce.

L'intégration des interfaces dans le système peut se faire de différentes manières. Dans certains cas, une NI est connectée à chaque unité de traitement [MNTJ04]. Cette approche permet plus de souplesse dans l'organisation de l'architecture mais entraîne un coût important car il faut multiplier le nombre de NI. Une seconde approche, consiste à grouper plusieurs unités de traitement sur une même NI. Cela réduit le nombre de NI mais en contre partie les NI sont plus complexes. De plus, il faut pouvoir gérer les communications entre unités de traitement connectées sur la même NI. Dans l'exemple de la Figure 1.9, l'interface dispose de quatre ports permettant de connecter des IP compatibles avec les protocoles DTL et AXI [DRGR03].

Les services fournis par l'interface réseau peuvent s'organiser selon trois niveaux [ScRF04] :

- Services d'adaptation

La fonction principale de l'interface réseau est de gérer la communication avec le réseau pour permettre à l'unité de traitement d'échanger des données. Il s'agit de l'adaptation des débits et latences entre les flux de l'unité de traitement et les flux réseau. Il peut aussi y avoir des adaptations de fréquences d'horloge. Il faut également mettre en forme les blocs de données dans un format compatible avec les modes de transfert du réseau, généralement cela se traduit par une mise en paquets des données (et inversement en réception). Dans [BhMa03], une étude présente différents types d'implémentation pour la mise en paquet en s'intéressant aux solutions matérielles ou logicielles.

- Services réseau

Au niveau de la couche transport, l'interface réseau prend aussi en compte l'ordonnancement des envois, la gestion des priorités, le contrôle de flux, la gestion des adresses de destination, la création des connexions (circuits commutés).

- Services fonctionnels

L'interface réseau peut également intégrer des fonctionnalités supplémentaires au niveau de la gestion des unités de traitement. Il s'agit de fonctions de contrôle et de configuration. Le NoC est un système distribué. L'interface réseau doit permettre de pouvoir gérer une unité de traitement à distance, par l'intermédiaire du réseau. Dans [MMBM03], par exemple, les unités de traitement sont reconfigurables et l'interface réseau permet de charger dynamiquement les informations de reconfiguration.

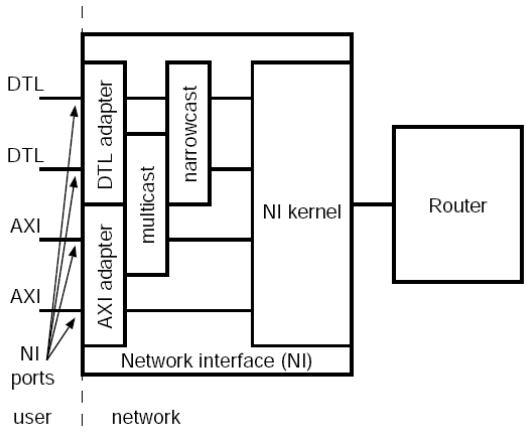


FIG. 1.9 – Exemple de l'interface réseau dans le réseau ÆTHEREAL (d'après [DRGR03])

1.3 État de l'art des architectures de réseaux sur puce

Pour illustrer les concepts présentés dans la section précédente, nous proposons maintenant quelques exemples concrets d'architectures de NoC représentatifs de l'état de l'art dans le domaine.

1.3.1 Réseaux meilleur-effort

Pour commencer cet état de l'art sur les architectures de réseau sur puce, nous présentons deux architectures de type meilleur-effort.

- SPIN

Le réseau SPIN[29] est développé par le laboratoire LIP6 de l'Université Pierre et Marie Curie de Paris [GuGr00]. C'est historiquement une des premières propositions concrètes de NoC par commutation de paquet. Ce réseau s'appuie sur une topologie en arbre élargi qui offre une latence limité grâce à un faible diamètre. Cette topologie est par contre moins flexible à intégrer et peu extensible. La commutation est de type *wormhole* avec un algorithme de routage adaptatif et distribué ce qui impose un réordonancement des paquets à la réception. Le contrôle de flux est assuré par un mécanisme de crédits sur des connexions point à point bidirectionnelles (36 bits de données et 2 bits de contrôle). Le noeud (RSPIN) utilise des files d'attente en entrée ainsi que deux files centrales (une pour les paquets montant et l'autre pour les paquets descendants) pour éviter le phénomène du blocage en tête de ligne.

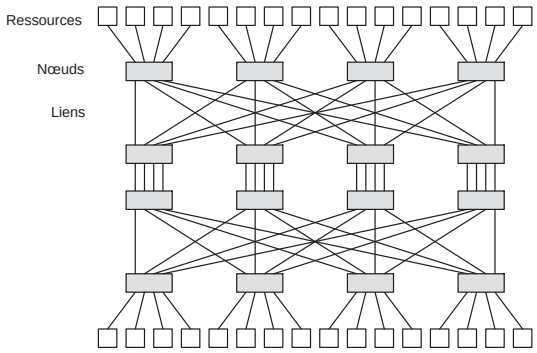


FIG. 1.10 – Topologie du réseau SPIN à 32 ports

Dans [ACGM03], une étude comparative avec une architecture de bus est présenté et démontre la supériorité du réseau SPIN en terme de cycles nécessaires pour assurer

[29] *Scalable, Programmable, Integrated Network*

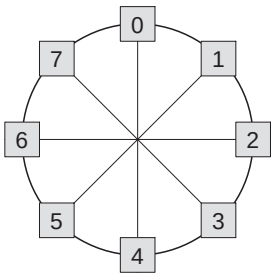
la transmission d'une quantité de données prédéfinies dès que le nombre de ressources dépasse la douzaine. Des synthèses et placements-routages ont été réalisés [AnGr03] pour une technologie CMOS 0,13μm. Le noeud RSPIN occupe une surface de 0,24mm^2 et une macro cellule SPIN32 correspondant à un réseau 16 noeuds avec 32 ports de connexion pour des ressources (Figure 1.10) nécessite 3,9mm^2 de silicium.

- OCTAGON

Cette architecture a été proposée en 2001 par STMicroelectronics et l'Université de Californie à San Diego [KNDR01]. Elle est destinée à répondre à des besoins en bande-passante importants rencontrés lors de la conception de routeurs Internet qui s'appuient sur des circuits de traitement de paquets IP. La topologie du réseau est un anneau octogonal avec des liens supplémentaires entre noeuds diamétralement opposés (Figure 1.11(a)). Les avantages mis en avant sont :

– un diamètre faible (maximum de deux liens entre deux noeuds)
– un algorithme de routage simple
– un débit effectif supérieur à celui d'un *crossbar* avec moins de fils de connexion

La mode de commutation peut se faire par circuit ou par paquet suivant les choix de l'arbitre de connexion. L'algorithme de routage se base sur l'adresse destination du paquet (codé sur 3 bits) et l'adresse du noeud courant par un mécanisme d'adresse relative (Figure 1.11(b)).



(a)

Adr_rel = Adr_paquet - Adr_noeud (modulo 8)
Si Adr_rel = 0, le paquet est arrivé
Si Adr_rel = 1 ou 2 le paquet est routé dans le sens horaire
Si Adr_rel = 6 ou 7 le paquet est routé dans le sens anti-horaire
Sinon le paquet est envoyé vers le noeud opposé dans l'octogone

(b)

FIG. 1.11 – Topologie Octagon et algorithme de routage

Il est possible d'augmenter la taille du système de façon linéaire en juxtaposant des structures OCTAGON par l'intermédiaire de noeuds remplissant la fonction de pont [KaND02]. Il faut alors utiliser un système d'adressage d'ordre supérieur (3 bits de poids fort pour désigner la structure de destination et 3 bits de poids faible pour désigner un noeud dans cette structure).

1.3.2 Réseaux avec qualité de service

Les services de bases (intégrité des données, absence de pertes...) sont assurés par les mécanismes du réseau. La maîtrise du débit et de la latence est plus problématique. La plupart des réseaux se basent sur l'hypothèse qu'ils sont correctement dimensionnés de façon à fonctionner loin de leur régime de saturation et ainsi d'éviter les cas les plus défavorables. Par exemple, pour le réseau SPIN [GuGr00], les simulations montrent que les probabilités de latence sont exponentiellement décroissantes. Ainsi avec une charge du réseau de 30%, la probabilité d'avoir une latence de 25 cycles et de l'ordre de 1% et elle tombe en dessous de 0,01% pour des latences supérieures à 150 cycles. Avec les réseaux meilleur-effort (cf. paragraphe 1.3.1), il faut donc limiter l'utilisation des resssources de communication pour s'assurer des performances compatibles avec les applications visées.

C'est pourquoi certaines architectures proposent des mécanismes spécifiques pour assurer des services garantis. Le réseau QNOC introduit des niveaux de priorités différents suivant la nature des messages. Le réseau NOSTRUM introduit le concept de circuits virtuels qui permet de garantir des degrés de débit avec une latence fixé. Le réseau ÆTHEREAL permet de créer des connexions entre deux ressources qui communiquent par l'intermédiaire de circuits commutés.

- QNOC

L'architecture QNOC[30] est proposée par l'Institut Technologique du Technion (Israël) [BCGK04]. La topologie est maillée à deux dimensions. Les ressources de traitement sont hétérogènes. L'algorithme de routage est de type XY ou YX en fonction des positions de l'émetteur et du récepteur de manière à ce que le chemin soit le même lors d'échanges bidirectionnels. Un mécanisme de crédit permet de contrôle le flux de messages.

Quatre niveaux de priorités ont été définis pour proposer des niveaux de services en fonctions des types de messages échangés et des contraintes de latence et de débit souhaitées. Le niveau *Signaling* est utilisé pour de courts messages très prioritaires, qui nécessitent une faible latence. Il s'agit typiquement de messages d'interruption ou de contrôle. Le niveau *Real-Time* permet de transférer des données en temps réel[31]. Le niveau de service *Read/Write* est utilisé pour gérer des échanges de données tels qu'ils se font habituellement sur un bus. Le dernier niveau *Block-Transfer* est utilisé pour transférer d'importantes quantités de données. Les paquets transmis sont identifiés en

[30] QoS NoC
[31] Flux audio ou vidéo par exemple

fonction du service désiré. Au niveau du noeud, les files d'attente d'entrée sont distinctes pour chaque niveau de service. Cela permet par exemple à un message prioritaire de type *Signaling* de s'insérer dans un flot de donnée *Real-Time*.

L'architecture du réseau a été simulée pour différents scénarios de trafic. Les simulations ont validé que les contraintes temporelles attribuées à chaque niveau de service sont bien respectées et ceci pour différents niveaux de charge du réseau.

- NOSTRUM

L'Institut Royale de Technologie de Suède (KTH) a publié un grand nombre de propositions en matière d'architectures et de méthodologies de conception de réseau sur puces. Après avoir présentée une architecture en nid d'abeille[32] [HJKP00], le choix s'est porté sur un réseau maillé à deux dimensions dans le cadre du projet NOCARC[33] [KJSF02]. Les derniers travaux présentent le concept d'une plate-forme de communication par commutation par paquets appelée NOSTRUM avec une pile protocolaire définie

L'architecture du réseau s'organise autour d'une topologie maillée à deux dimensions. L'algorithme de routage, dit *hot-potato* [Nils02], est assez original. Les paquets ont une taille fixé à 128 bits. Le principe est d'envoyer l'ensemble des paquets reçus par un noeud directement au cycle suivant sans avoir à les stocker. Le routage qui en résulte est non déterministe. En effet, si un noeud reçoit un paquet sur ses quatre ports d'entrée (Nord, Sud, Est et Ouest), il doit au cycle suivant renvoyer les quatre paquets en les orientant au mieux vers leur destination finale sur les quatre ports de sortie. Il se peut donc qu'un paquet ne prenne pas le chemin le plus court à cause d'un trafic concurrent sur un port.

Pour gérer les priorités et éviter qu'un paquet se déplace sans fin dans le réseau, chaque paquet possède un compteur[34], mis à jour à chaque noeud, qui indique le nombre de liens empruntés. Un mécanisme supplémentaire a été ajouté pour éviter les phénomènes de congestion. Le mécanisme de *Proximity Congestion Awareness* (PCA) [NMOJ03] calcule au niveau de chaque noeud une valeur représentative de sa charge. Cette valeur est distribuée aux noeuds voisins qui pourront ainsi éviter de transmettre des paquets vers un noeud saturé. Le routage est donc adaptatif et tient compte de manière locale du trafic global. Ce mode de routage a cependant pour inconvénient d'augmenter la latence et les ressources peuvent avoir des difficultés à injecter des don-

[32] Topologie *honeycomb*
[33] *NoC Architectures*
[34] *hop-counter*

nées dans le réseau.

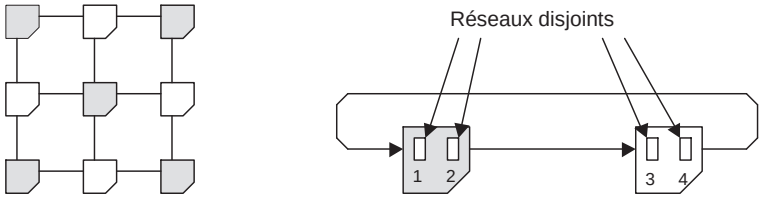


FIG. 1.12 – Concept de réseaux disjoints en raison de la topologie et des étages de mémorisation

Pour permettre des transferts avec des garanties de débits et de latences, le réseau NOSTRUM s'appuie sur les concepts de *Temporally Disjoint Networks* (TDN) et de *Looped Container Virtual Circuits* (VC). En effet, le routage *hot-potato* introduit dans le réseau un multiplexage temporel implicite qui crée des réseaux disjoints (TDN). Autrement dit, il est possible de déterminer avec certitudes que des paquets introduits à un instant et dans des noeuds particuliers ne vont jamais entrer en conflit sur une décision de routage [MNTJ04]. Ainsi en tenant compte du facteur de topologie et du nombre de registres de stockage, il existe quatre réseaux disjoints dans l'exemple Figure 1.12. Les *Looped Containers* sont des paquets alloués à l'avance à des ressources qui transitent en permanence sur des circuits (VC) pré-routés. Cela permet d'assurer à une ressource qu'elle pourra accéder au réseau à des instants précis sans attente et que le paquet sera transmis avec une latence fixe. Le nombre de containers sur un VC permet d'allouer la bande-passante par paliers. Dans l'implémentation actuelle, les VC ne peuvent pas être configurés dynamiquement et la bande-passante est en partie perdue lorsque les containers sont transmis alors qu'ils ne contiennent pas de donnée.

- ÆTHEREAL

ÆTHEREAL est une architecture de NoC proposée par Philips[35]. De nombreuses publications sont issues de ces travaux. Le but du réseau ÆTHEREAL est d'assurer à la fois un service meilleur-effort et de proposer des services garanties (de débit en particulier) [GDMP03].

La topologie est à deux dimensions mais ne se limite pas aux réseaux maillés (Figure 1.13(a)). Le routage est de type source avec un flot de donnée *virtual-cut through* ou *wormhole*. Le service meilleur-effort est géré de façon classique par commutation de paquet

[35] *Philips Research Laboratories* à Eindhoven aux Pays-Bas

Un mécanisme de commutation de circuit couplé à du multiplexage temporel [36] permet de créer des connexions avec un débit et une latence garantis. Le principe est le suivant : tous les noeuds ont une notion de temps commune qui est basée sur compteur de *time slot* et ils gèrent les correspondances entre les ports d'entrée et de sortie avec une table de *time slots*. A l'initialisation, cette table est vide, les ressources connectées envoient des paquets spéciaux pour créer des nouvelles connexions qui vont réserver des *time slot* dans chacun des noeuds nécessaires au routage. Une fois la connexion établie, les échanges de données auront donc un débit garanti (fonction du nombre de *time slot* réservés) et une latence fixe (fonction du nombre de noeuds traversés). L'utilisation de tables de *time slot* évite tout risque de conflits entre deux connexions.

Dans le noeud (Figure 1.13(b)), les services meilleur-effort sont réalisé en utilisant les *time slots* non réservés pour les connexions. Une version prototype du noeud a été synthétisée en technologie CMOS 0.12μm. Les paramètres retenus sont 5 ports, un flot wormhole, une file d'attente de 8 flits pour chaque entrée, des flits de 3 mots de 32 bits et une table de 256 *time slots*. Le noeud occupe une surface de 0,26mm^2 avec une fréquence de fonctionnement de 500MHz.

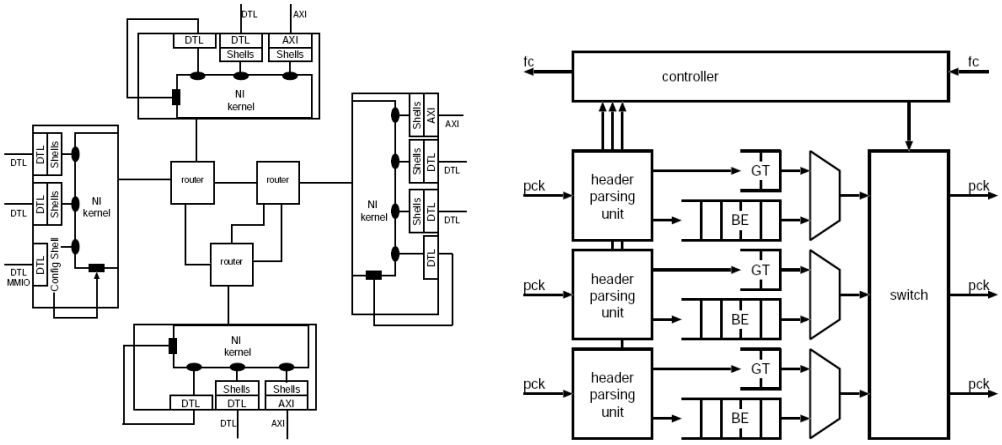


FIG. 1.13 – Exemple de réseau ÆTHEREAL et architecture d'un noeud (d'après [DRGR03])

[36] *time-division mutliplexed circuit-switching*

1.3.3 Réseaux asynchrones

Nous avons expliqué précédemment (cf. paragraphe 1.1.2) que le problème de la propagation du signal d'horloge allait être de plus en plus présent dans des circuits intégrés à venir. C'est pourquoi des solutions de communication sur puce basées sur des techniques asynchrones commencent à voir le jour.

- CHAIN

Le réseau CHAIN (*CHip Area INterconnect*) développé à l'Université de Manchester [BaFu02] est basé sur une implémentation entièrement asynchrone. Il utilise un protocole de synchronisation par poignée de main[37] insensible aux délai. Les données sont échangées par commutation de paquets et le routage est de type source. Le réseau est formé d'un ensemble de blocs de base comprenant des arbitres associés à des aiguillages de routage. Il cible des applications basse-consommation et a permis de concevoir un système pour carte à puce en technologie CMOS 0,18 μm [BaPF04].

- MANGO

Le réseau MANGO (*Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interface*) est proposé par l'Université Technique du Danemark. Il s'organise autour d'une topologie maillée à deux dimensions et s'appuie sur un noeud asynchrone supportant du trafic *best-effort* et également un mode connecté pour des services garantis. Une implémentation du noeud est présenté en technologie 0,12 μm.

1.3.4 Synthèse de l'état de l'art sur les NoC

Pour clore cette section, nous présentons un tableau synthétique regroupant un ensemble d'architectures de réseaux sur puce permettant un aperçu global de l'état de l'art actuel (Tableau 1.5).

1.4 Les enjeux nouveaux liés à la conception d'un NoC

Les réseaux sur puce permettent de réaliser des structures de communication flexibles et reconfigurables. Il s'agit également de systèmes distribués qui peuvent faire apparaître des comportement moins déterministes qu'avec structures d'interconnexion plus

[37] *handshake*

TAB. 1.5 – Tableau récapitulatif des architectures NoC

Référence	**NoC**	**Topologie**	**Commutation**	**Routage**	**Liens**	**Technologie - Noeud**	**Applications visées**
NoC meilleur effort (best effort)							
[ACGM03]	SPIN	Arbre élargi	*Wormhole*	Distribuée et adaptatif	32 bits données + 4 bits contrôle	CMOS 0,13μm / 0,24mm^2	
[STNu02]	PROTEO	Anneau variable	*Virtual cut-through*	Source		CMOS 0,18μm	Vidéo
[BeBe04]	×PIPES	Variable	*Wormhole*	Source	Lien série		
[KaND02]	OCTAGON	Anneau avec cordes	Circuit ou paquet	Source	variable + 3 bits adresse		Processeur de réseau
[LiST00]	ASOC	2D maillée	Circuit	Tabulé dans les noeuds	32 bits	50,000 transistors	communication, multimédia
[WiLi03]	SOCBUS	2D maillée	Circuit	Distribuée et adaptatif			Téléphonie
[DaTo01]	NoC de l'Univ. de Stanford	2D tore enfoui	Paquet	XY	256 bits données + 38 bits contrôle	CMOS 0,10μm / 0,59 mm^2	
[PGIS03]	NoC de l'Univ. de Colombie Britannique	Arbre élargi en papillon	*Wormhole*	Distribuée et adaptatif	42 bits	20000 portes	
NoC avec qualité de service (QoS)							
[WiGo02]	ÆTHEREAL	2D variable	Circuit et paquet	Source	32 bits	CMOS 0,12μm / 0,26mm^2	
[MNTK04]	NOSTRUM	2D maillée	Paquet	*Hot-potato*	128 bits donnée		
[BCGK04]	QNOC	2D maillée variable	*Wormhole*	XY	16 bits données + 10 bits contrôle		
[DuBL05]	FAUST	2D maillée	*Wormhole*	Source	32 bits données + 4 bits contrôle	CMOS 0,13μm / 35k portes	Réseaux sans-fil
NoC sur FPGA							
[MTCM05]	HERMES	2D maillée	*Wormhole*	XY partiellement adaptatif	8 bits donnée + 2 bits contrôle	FPGA Virtex II / 316 slices	
[MBVV02]	NoC de l'IMEC	2D tore	*Wormhole*	XY	16 bits donnée + 3 bits contrôle	Virtex FPGA XCV800 / 446 slices	Application multimédia
[ZeSu03]	SOCIN	2D maillé ou tore	*Wormhole*	XY	32 bits donnée + 6 bits contrôle	FPGA EPF10K200 / 420-1754LCs	
NoC Asynchrone							
[BaFu02]	CHAIN	Variable	Paquet	Source		CMOS 0,18μm	smart card
[BjSp05]	MANGO	2D maillée	Circuit et paquet	Source	32 bits donnée + 2 bits contrôle	CMOS 0,12μm / 0,188mm^2	

classiques. Dans le cadre de cette thèse, nous avons identifié trois problématiques nouvelles liés à la conception d'un SoC basé sur un NoC que nous allons présenter dans cette nouvelle section. Tout d'abord nous nous intéresserons à l'évaluation des performances d'un système fonctionnant avec un NoC, ensuite nous introduirons les enjeux liés à la gestion des communications et au contrôle des traitements applicatifs.

1.4.1 Dimensionnement du réseau, prévision et évaluation des performances

Lorsque l'on souhaite évaluer les caractéristiques d'une architecture de réseau sur puce, de nombreux critères peuvent être pris en compte. Dans l'idéal, on désire une architecture proposant des débits de transferts élevés, avec une faible latence, en minimisant la consommation d'énergie, le tout occupant une faible surface de silicium. Dans la suite, nous allons détailler ces différents critères. En pratique, dans le flot de conception d'un SoC intégrant un réseau, nous pouvons distinguer deux phases :

Dans un premier temps, il s'agit de définir l'architecture des briques de base du réseau, c'est-à-dire les noeuds de routage. C'est ce qui va déterminer le mécanisme et le mode de communication, le type de topologie ou le mode de routage (cf. section 1.2). Un certain nombre de paramètres comme la taille des liens ou la taille des mémoires dans les noeuds peuvent également être dimensionnés lors de cette phase. Dans le cadre de cette thèse, nous considérerons cette phase comme achevée. Au chapitre 2, nous présenterons une structure de réseau que nous prendrons comme hypothèse de départ pour la suite de l'étude.

Une fois les éléments constitutifs du réseau définis, on peut construire un système complet à savoir l'association d'un réseau avec des ressources de traitement. A ce niveau là, de nombreux choix sont encore à faire. Il faut en particulier disposer de façon judicieuse les ressources par rapport au réseau de manière à limiter les temps de transferts et les points de congestion. La topologie finale du réseau peut être modifiée en ajoutant des liens et des noeuds pour augmenter la bande-passante par exemple. Il faut aussi déterminer les chemins de routage, la taille des paquets et leurs priorités lorsque ces choix sont possibles. Une fois le système mis en place, il faut valider qu'il est fonctionnel et qu'il répond bien aux contraintes temporelles que les spécifications imposent.

1.4.1.1 Débit de données

Le *débit de données* doit permettre de quantifier le volume de données qui circulent sur le réseau de communication. Chaque élément du réseau (liens, noeuds, etc) peut être caractérisé par sa *bande-passante*. Cette valeur, couramment exprimée en bit par seconde (bps) n'est pas représentative du fonctionnement réel du réseau. A cause de congestions et de conflits de routage, les éléments du réseau ne peuvent jamais être utilisé au maximum de leurs capacités. C'est pourquoi, il est plus pertinent de s'intéresser au *débit de données effectif*. Ce débit correspond au débit de données en réception cumulé sur l'ensemble des ressources. Dans un réseau où il n'y a pas de perte de paquet, il est égal au débit en émission. Cette définition est préférable à la somme du débit sur chacun des liens qui est moins précise puisque les données transitant sur un plus grand nombre de liens vont avoir un poids artificiellement plus élevé dans le résultat.

1.4.1.2 Latence de transfert

Un critère de performance important dans le cadre des réseaux sur puce est l'estimation de la *latence* des transferts. Elle peut se définir à plusieurs niveaux : flit, paquet (ensemble de flits) ou message (ensemble de paquets). Pour un flit, il s'agit du temps écoulé entre l'introduction du flit dans le réseau et sa réception par la ressource destinatrice. Pour un paquet, la latence correspond au temps entre l'envoi du premier flit et la réception du dernier. Dans le cas d'un message, c'est le temps entre l'envoi du premier paquet et la réception du dernier. La latence dépend d'abord de critères statiques. Il s'agit principalement de la position relative de l'émetteur par rapport au récepteur sur le réseau. Plus le nombre de liens et de noeuds à parcourir sera important, plus la latence va augmenter, celle-ci résultant principalement du temps de routage dans les noeuds. La latence peut également varier de manière dynamique. Cela résulte du partage du réseau entre plusieurs flots de paquets simultanément. Les choix d'arbitrage au niveau des noeuds vont augmenter la latence des paquets qui devront attendre leur routage. Dans le cas de réseau à commutation de paquets, la latence est un critère difficile à maîtriser. En effet, même si elle peut être limitée en moyenne, il existe potentiellement des situations de pire-cas [ACGM03].

1.4.2 Configuration et gestion des flots de données

Lorsque l'on évoque une architecture de réseau sur puce, on visualise un ensemble d'unités de traitement qui communiquent entre elles par l'intermédiaire d'une structure

d'interconnexion. Le problème est de définir quel système va fonctionnellement prendre en charge la gestion des communications. En effet, le réseau offre seulement des services d'échange de données mais ce n'est pas lui qui prend les initiatives d'effectuer les transferts de données. C'est un outil de communication et non un acteur. Les unités de traitement réalisent un calcul sur les données elles n'ont donc pas a priori une connaissance du réseau, des sources ou des destinations des données. Le principe mis en avant est le découplage entre les communications et les calculs. C'est pourquoi les concepteurs on introduit le concept d'interface réseau (cf. paragraphe 1.2.2.3).

Le problème est alors de définir l'architecture de ces interfaces pour qu'elles répondent aux besoins liés aux applications visées en matière de flots de données.

L'organisation des flots de données s'établie selon trois niveaux :

- **La gestion des communications**. Le but est de rendre possible les différents échanges de données nécessaires au fonctionnement des unités de traitement du système. Cela concerne donc la gestion des adresses de sources et de destination ainsi que la quantité de données à transférer.

  Exemple :

(source) vers (destination)	quantité
A => C	n bits
B => C	m bits

- **La gestion du séquencement**. La contrainte sur l'ordre des transferts est ajouté. Pour une unité de traitement, les données doivent être reçues dans un ordre précis. De même, en sortie d'une unités, suivant les étapes de traitement, les données n'auront pas les mêmes destinations. Il est donc important de séquencer les transferts pour les synchroniser avec les phases de traitement.

  Exemple :

temps	(source) vers (destination)	quantité
t_0	A => C	n/2 bits
t_1	B => C	m/2 bits
t_2	A => C	n/2 bits
t_3	B => C	m/2 bits

- **La gestion du temps**. Il s'agit de respecter les contraintes temporelles d'un système applicatif. Garantir les transferts et leurs séquencements ne suffit pas à assurer un fonctionnement correct. Il faut également respecter les délais imposés par l'application. Ces délais doivent être compatibles avec la latence des transferts, la cadence des traitements ou les temps de réactivité du système sur certains évènements.

  Exemple : t_1-t_0<d_1
  t_2-t_1<d_2
  t_3-t_2<d_3

1.4.3 Contrôle d'une application distribuée sur un réseau

Les réseaux sur puce créent une infrastructure de communication distribuée. Il n'y a pas de lien direct entre les ressources de traitement. Il est donc difficile d'avoir une vision globale de l'état de toutes les ressources à un instant donné. La problématique est alors de contrôler l'exécution d'une application répartie sur les différentes ressources. Il faut pour cela mettre en place des mécanismes pour configurer les ressources et assurer un séquencement et une synchronisation entre les différentes tâches et les données à traiter.

Si l'on se réfère à l'état de l'art, on constate que peu de groupes de recherche présentent des architectures de réseau sur puce dans le cadre d'applications concrètes. On peut cependant citer quelques exemples. [CTCH04] présente un réseau sur puce nommé RCA[38] qui implémente la couche physique de différents standards de télécommunication sans-fil avec en particulier IEEE 802.11a. [TLVI05] implémente un décodeur LDPC sur un réseau maillé à deux dimensions. [XWHC04] propose un système de traitement vidéo embarqué sur un NoC. Pourtant, même lorsqu'une application est clairement associée à une architecture, la problématique du contrôle est rarement abordée.

Les solutions les plus souvent envisagées sont basées sur des approches centralisées. Dans ce cas une ressource spécifique prend en charge le contrôle de l'ensemble des ressources connectées au réseau. Nous allons illustrer ce principe de fonctionnement par deux exemples de travaux sur le domaine.

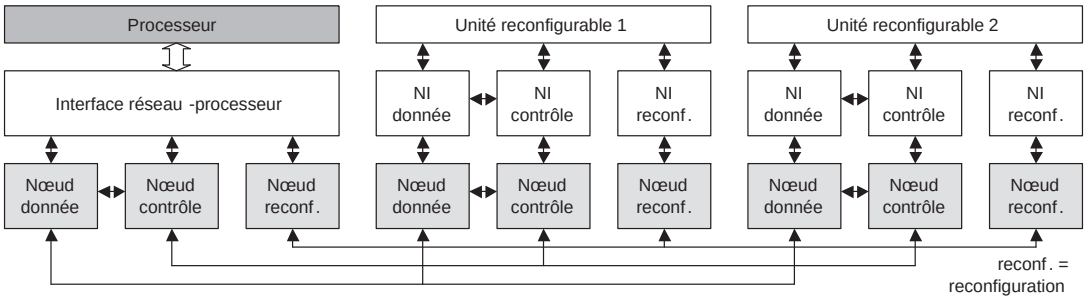


FIG. 1.14 – Système d'interconnexion basé sur trois réseaux (donnée, contrôle et reconfiguration) d'après [MMBM03]

Une proposition détaillée est présentée par les laboratoires de l'IMEC [MMBM03] [NoMV04] [NMAV05]. Il s'agit d'une architecture de NoC maillé à deux dimensions destinée à être implémentée dans un FPGA. La gestion de l'application est réalisée

[38] *Reconfigurable Communication Architecture*

par un système d'exploitation (OS[39]) embarqué dans un processeur centralisé. Chaque unité de traitement est connectée à une interface réseau (NI). L'interface réseau est composée de trois parties aux fonctionnalités distinctes : gestion des données, contrôle et reconfiguration (Figure 1.14). Cette interface assure un support matériel aux fonctions de l'OS.

Une seconde proposition est faite dans le cadre du réseau Æthereal (présenté au paragraphe 1.3.2). Là encore, il s'agit d'un schéma de programmation centralisé avec l'utilisation d'interfaces réseau reconfigurables. Un unique module de configuration a la charge d'établir les connexions entre les différentes ressources qui communiquent entre elles

Le défaut principal de ces deux solutions est la charge importante qu'elles imposent au contrôleur central qui doit gérer toutes les ressources. Elles sont donc limitées à des réseaux de petites tailles (environ 10 noeuds) et ne passent pas à l'échelle avec des systèmes plus importants [RDPG05]. De plus, le trafic réseau pour les messages de contrôle est élevé. Les latences de transfert peuvent alors devenir critiques ce qui impose l'usage de réseaux séparés pour le contrôle et les données comme c'est le cas dans la proposition de l'IMEC.

L'autre alternative est de mettre en place un système distribué. Dans ce cas, il y a plusieurs systèmes de contrôles répartis sur le réseau. Ces solutions sont plus souvent envisagées dans le cadre de systèmes sur puce multiprocesseurs (MPSoC[40]) [JeTW05] telle que la plate-forme StepNP proposée par STMicroelectronics [PPBL04]. Cependant, dans le cadre de cette thèse nous nous intéressons à des architecture de NoC hétérogènes avec des unités matérielles de traitement dédiées. Ceci diffèrent des approches basées sur plusieurs processeurs fonctionnant en parallèles entraînant d'autres problématiques de contrôle.

1.5 Conclusion

Ce premier chapitre nous a permis de présenter un large panorama sur les réseaux sur puce. Après avoir identifié les enjeux actuels liés à la maîtrise des communications sur puce, nous avons détaillé les caractéristiques des NoC en nous appuyant sur des exemples concrets issus des travaux de recherches actuels dans le domaine. Les NoC permettent de construire des architectures flexibles et extensibles. Grâce à un protocole et des interfaces

[39] *Operating System*
[40] *Multi-Processor System-on-Chip*

bien-définis, les communications et les traitements sont découplés. L'intégration d'IP est ainsi facilitée. Au delà de la structure de communication, il existe cependant certaines problématiques nouvelles lors de la conception d'un système applicatif. Il s'agit d'abord d'évaluer si le réseau est compatible avec le trafic de l'application. Ensuite, il faut mettre en place des mécanismes pour gérer les flots de communication de l'application et assurer le contrôle des traitement sur un système distribué. Dans le chapitre suivant, nous allons affiner nos axes d'études dans le contexte de systèmes de télécommunications et du réseau FAUST.

Chapitre 2

Contexte des travaux : applications de télécommunication et réseau FAUST

Dans le premier chapitre, nous avons présenté les architectures de réseau sur puce comme une solution efficace pour supporter les besoins en communication d'un système sur puce complexe intégrant un grand nombre d'unités de traitement. Ce nouveau chapitre doit nous permettre de nous focaliser d'une part sur les applications de télécommunication étudiées au LETI et d'autre part sur l'architecture de NoC intitulée FAUST qui est développée dans ce contexte.

La première section du chapitre est consacré au contexte des réseaux de télécommunication sans-fil haut-débit et à leurs évolutions actuelles. Nous détaillons ensuite deux projets (MATRICE et 4MORE) qui ont pour objectif de proposer une implémentation fonctionnelle de nouvelles techniques de communication basées sur des modulations multiporteuses, des accès multiutilisateurs et des architecture multiantennes. La troisième section sera consacrée à l'exposé de l'architecture du réseau FAUST qui sert de plateforme d'intégration dans le cadre de ces projets. Nous finirons ce chapitre en rediscutant les problématiques liées à la conception d'un NoC introduites au chapitre 1 dans le cadre spécifique du réseau FAUST.

2.1 Applications de télécommunications sans-fil et leurs évolutions

Les systèmes de télécommunication sans-fil représentent un secteur majeur de l'industrie de la microélectronique. A titre d'illustration, en 2005, un total de 816,6 millions de téléphones portables ont été vendus [GART06] soit un progression de 21% par rapport à 2004. Les industriels doivent constamment proposer de nouvelles générations de systèmes pour faire face à l'augmentation des besoins en communication sans-fil. Les progrès techniques sont principalement motivés par l'augmentation du débit de données dans le cas d'un usage statique et l'amélioration de la robustesse des échanges dans le cas d'une communication mobile. D'autres besoins comme la réduction de la consommation, l'augmentation de la qualité de services ou la compatibilité multistandard doivent également être pris en compte.

Dans cette section nous allons présenter les différents standards actuels de télécommunications sans-fil puis nous nous intéresserons aux évolutions à venir dans le but d'introduire les systèmes qui seront présentés dans section suivante.

2.1.1 Panorama des solutions actuelles pour les télécommunications sans-fil

Les réseaux de communications sont généralement classés en fonction de la distance sur laquelle ils s'étendent. Dans le cas des systèmes sans-fil nous pouvons distinguer quatre types de réseaux [Vulm04] :

- **PAN** (*Personal Area Network*). Les réseaux personnels ont une portée limitée à quelques mètres. Ils sont surtout destinés à établir des connexions entre terminaux plus ou moins évolués (souris, oreillette, etc). La norme la plus répandue est Bluetooth [BLUE06]. Cette norme est gérée par le groupe de travail IEEE 802.15.i. Il existe 4 normes dont 802.15.4 (Zigbee) dont le but est de proposer une puissance d'émission extrêmement faible. La technique UWB[1] [UWB06] devra bientôt permettre d'augmenter le débit des PAN.
- **LAN** (*Local Area Network*). Les réseaux locaux concernent surtout les réseaux intra-entreprise. Les WLAN (Wireless LAN) regroupent plusieurs technologies comme HomeRF, Hiperlan2 et en particulier les technologies issues de la norme IEEE 802.11 qui sont aujourd'hui les plus répandu sous la dénomination Wifi.

[1] *Ultra Wide Band*

- **MAN** (*Metropolitan Area Network*). Ils assurent l'interconnexion des entreprises, voire même des particuliers, à l'échelle d'une métropole sur un réseau haut débit. La norme en cours de déploiement est WiMAX[2] [WIM06] correspondant au groupe de travail IEEE 802.16.
- **WAN** (*Wide Area Network*). Ce sont principalement les réseaux de type téléphonie mobile répondant aux normes GSM, GPRS, EDGE, UMTS. Les débits vont de quelques dizaines de kbits/s au Mbit/s pour l'UMTS. Ces débits sont insuffisants pour des applications informatiques de plus en plus exigeantes en débit, mais correspondent à des applications professionnelles et grand public parfaitement ciblées.

Le Tableau 2.1 permet de comparer les standards de télécommunication sans-fil précédemment cités.

TAB. 2.1 – Comparaison entre différents standards de télécommunication

	Filaire (pour comparaison)			**Sans fils**					
	Ethernet	**ADSL**	**USB2**	**Wifi**	**Wimax**	**Bluetooth**	**Zigbee**	**UMTS**	**UWB**
Débit	100 Mbps	512kbps à 2Mbps	480Mbps	54Mbps	70Mbps	57kbps à 1Mbps	20 à 250 kbps	50kbps à 2Mbps	54 à 480 Mbps
Distance	100m	1km	5m	45m	10km	3m	10m	1km	1 à 10 m

2.1.2 Évolutions des standards de télécommunication sans-fil

Le passage de la deuxième (2G) à la troisième génération (3G) de système de téléphonie mobile a été motivé par l'introduction de services multimédia mobile. Tout porte à croire que les besoins en débit et en mobilité vont continuer à augmenter dans l'avenir. Les systèmes actuels ne permettent pas de cumuler le haut-débit avec la mobilité. Aujourd'hui les travaux de recherche et développement se focalisent donc sur l'après 3G (B3G[3]) ou la quatrième génération (4G). La Figure 2.1 permet de situer ces différentes générations de standards. La 4G devra permettre la convergence entre les systèmes 2G, 3G et WLAN.

Pour répondre aux fortes contraintes de débit et de mobilité, les futures systèmes 4G devront utiliser au mieux la ressource radio en offrant davantage de robustesse dans

[2] *Worldwide Interoperability for Microwave Access*
[3] *Beyond 3G*

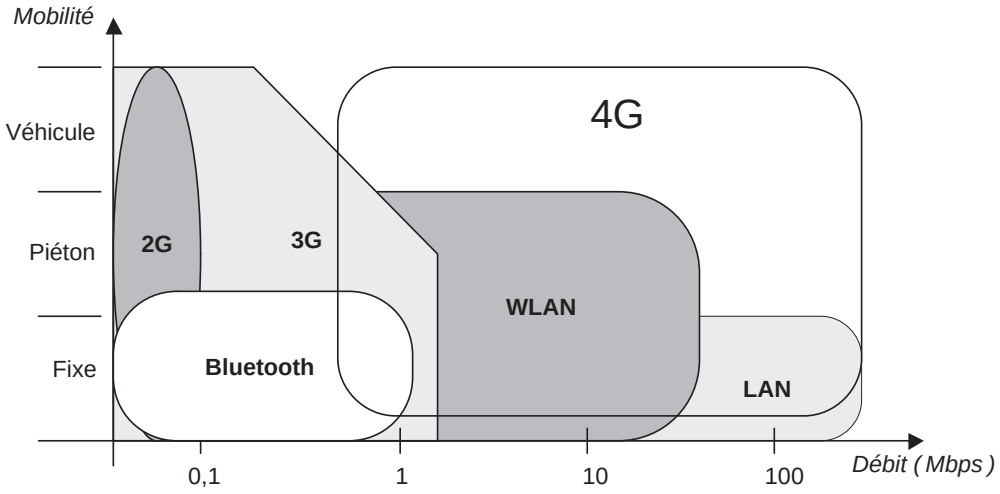


FIG. 2.1 – Mobilité en fonction du débit pour différents types de standards de télécommunication

les transmissions avec une meilleure efficacité spectrale. En effet, avec l'augmentation de nombre de standards de télécommunication qui coexistent sur les ondes, le spectre radio est de plus en plus saturé. Cela impose donc de fortes contraintes sur la bande-passante disponible pour une application. Par exemple entre les standards IEEE 802.11b (11Mbps) et 802.11g (54Mbps), l'efficacité spectrale a quasiment été multipliée par cinq pour atteindre 2,7bps/Hz [Holt05]. Cette tendance se confirme avec le futur standard IEEE 802.11n qui devrait permettre des débits d'au moins 100Mbps avec une efficacité spectrale d'au moins 5bps/Hz.

L'introduction de nouvelles techniques permet de répondre à ces évolutions. La technique multiporteuse d'accès multiple par répartition de code (MC-CDMA[4]) permet des transmissions multiutilisateur à haut-débit [YeLF93] [FaKa03]. Elle combine les avantages des techniques multiporteuses OFDM qui optimisent l'efficacité spectrale et les techniques d'étalement de spectre CDMA qui permettent les accès multiple au canal de transmission par l'attribution de codes spécifiques à chaque utilisateur [JVLZ05].

Les techniques à antennes multiples permettent d'exploiter la dimension spatiale du canal radio. Parmi elles, la technique MIMO (*Multiple Input Multipe Output*[5]) tire partie de la décorrélation qu'il existe entre les différents canaux radio formés par les antennes émettrices et réceptrices [Hong05]. Ceci permet d'augmenter, au choix, le débit ou la robustesse des transmissions en fonction de l'environnement. Les futurs standards IEEE

[4] *MultiCarrier - Coded Division Multiple Access*
[5] En opposition au principe SISO (*Single Input Single Output*)

802.11n et 802.16e (*Mobile Wimax*) utiliseront cette technique.

2.2 Les systèmes de télécommunication dans le contexte de la thèse

Le déploiement de la troisième génération de systèmes mobiles terrestres est en cours et il y a déjà une activité de recherche importante pour les systèmes 4G. Dans ce contexte, la communauté européenne finance différents projets sous le label IST[6] qui seront forces de proposition dans l'établissement de futurs standards. L'objectif est la réalisation de systèmes intégrés multi-standards supportant des débits de 20 Mbps en environnement mobile et jusqu'à plus de 100Mbps en environnement fixe et piéton.

Dans cette section nous allons présenter deux projets pour lesquels le LETI est partenaire et qui constitue un contexte applicatif dans le cadre de cette thèse.

Le projet MATRICE (M*ulticarrier CDM*A TR*ansmission Techniques for* I*ntegrated Broadband* CE*llular Systems* [MATR06]) a pour objectif de définir et de valider les concepts d'accès et de transmission basés sur la techniques MC-CDMA tout en proposant une architecture d'implémentation matérielle. Le projet 4MORE (4*G MC-CDMA* M*ultiple Antenna SOC for* R*adio* E*nhancements* [4MOR06]) s'appuie sur les résultats de MATRICE et doit permettre de progresser vers l'implémentation d'un système sur puce (SoC) complet pour un terminal 4G en combinant les techniques MC-CDMA et MIMO.

2.2.1 Présentation des systèmes de télécommunication MATRICE et 4MORE

Les réseaux MATRICE et 4MORE ont une organisation de type *infrastructure*. Des points d'accès (ou station de base[7]) sont répartis sur la zone à couvrir et reliés entre eux par un réseau filaire. Les utilisateurs mobiles (appelés terminaux mobiles[8] se connectent au point d'accès qui leur fournit la couverture radio pour accéder au réseau. La communication entre deux terminaux est réalisée par l'intermédiaire de ce point d'accès. Ce mode s'oppose au mode *ad-hoc* dans lequel chaque terminal mobile est un noeud actif du réseau qui est capable de dialoguer directement avec un autre noeud du réseau sans passer par un tiers de communication.

[6] *Information Technologies Society*
[7] en anglais *Base Station* ou BS
[8] en anglais *Mobile Terminal* ou MT

On définit la voix montante (UL[9]) qui correspond à l'envoi de données du terminal mobile vers la station de base. A l'inverse, on parle de voix descendante (DL[10]) lorsque les données sont transmises de la station de base vers le terminal mobile.

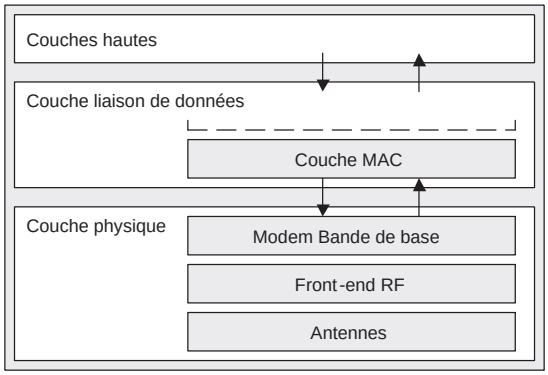


FIG. 2.2 – Couches protocolaires et structure d'implémentation d'un système de télécommunication

Les systèmes de télécommunication définis dans le cadre des projets MATRICE et 4MORE sont divisés en modules qui opèrent à différents niveaux selon les couches protocolaires du modèle OSI [Zimm80]. Nous nous intéressons en particulier au couches basses (Figure 2.2). La sous-couche *Media Access Control* ou MAC est la partie inférieure de couche de liaison de données. Elle sert d'interface entre la partie logicielle et la couche physique (matérielle). Elle détermine et contrôle l'accès au médium de communication. Le modem bande de base effectue la modulation (en émission) et la démodulation (en réception) des données numériques. Le front-end RF effectue la conversion entre le domaine numérique et le domaine analogique. Il réalise également les transpositions de fréquences et les corrections du signal (gestion du gain). Les antennes transmettent et reçoivent le signal radiofréquence.

2.2.1.1 Structures des trames MATRICE et 4MORE

MATRICE et 4MORE sont conçus pour permettre des communications bidirectionnelles TDD (*Time Division Duplex*). Les données sont échangées dans le cadre d'une structure temporelle appelée *supertrame*. Cette supertrame est découpée en trames qui peuvent être de type UL ou DL. Les trames on une durée de 666,66μs. La supertrame contient 15 trames, sa durée est ainsi fixée à 10ms. Le protocole détermine pour chaque

[9] *Up Link*
[10] *Down Link*

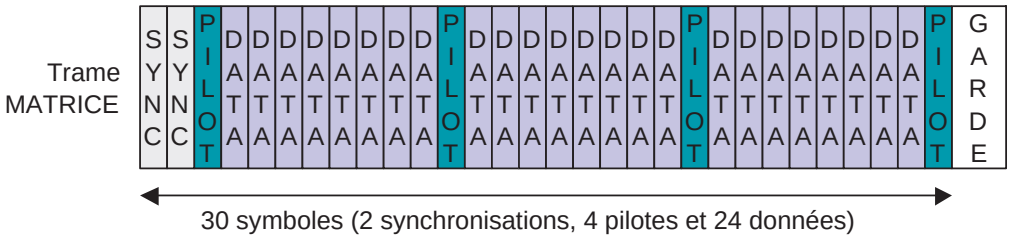


FIG. 2.3 – Structure d'une trame MATRICE

supertrame le nombre et la position des trames UL et DL. Pour cela des trames de contrôle sont envoyées en début de supertrame.

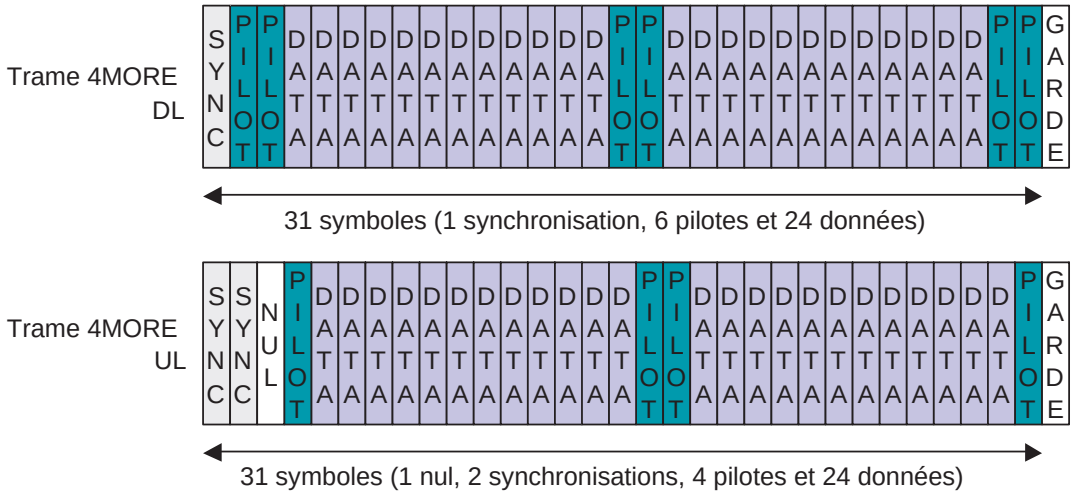


FIG. 2.4 – Structures des trames 4MORE en DL et UL

La Figure 2.3 présente la structure d'une trame MATRICE. Cette structure est identique pour toutes les trames (UL et DL). Elle est composée de 30 symboles OFDM de 1240 échantillons excepté les deux premiers (80 échantillons chacun) qui sont utilisés pour la synchronisation. 4 symboles pilotes répartis sur la trame servent à l'estimation du canal de transmission. 24 symboles contiennent les données. La trame se termine par un intervalle de garde qui correspond à un instant entre deux trames où aucune donnée n'est transmise. Une trame correspond à 38400 échantillons à une fréquence de 57,6MHz. Ainsi, avec des échantillons codés sur 32 bits, cela représente un débit moyen de 1,8Gbps sur la durée d'une trame.

Dans 4MORE, les trames sont découpées en 32 intervalles de temps identiques. Elles contiennent 40960 échantillons (Fréquence de 61,44MHz). Les trames UL et DL

TAB. 2.2 – Paramètres OFDM des systèmes MATRICE et 4MORE

	4MORE		MATRICE
	DL	UL	
Taille de la FFT	1024		1024
Nombre de porteuses modulées	694	672	736
Nombre de porteuses nulles	330	352	288
Nombre de porteuses pilotes	22	96	0
Nombre de porteuses de données	672	576	736
Durée d'un symbole (µs)	20,83		21,5
Préfixe cyclique	256		216
Nombre d'échantillons par symbole de données	1280		1240

sont différentes (Figure 2.4). La structure de trame DL est composée de 24 symboles de données, 6 symboles pilotes, un symbole de synchronisation et un intervalle de garde. La trame UL contient 24 symboles de données, 4 symboles pilotes et un intervalle de garde. 3 symboles (2 motifs de synchronisation et un nul) sont utilisé pour la synchronisation.

2.2.1.2 Paramètres de modulation et classes de fonctionnement

La modulation OFDM utilise 1024 fréquences porteuses. Les données sont positionnés sur les porteuses dans le domaine fréquentielle, le signal temporel transmis est calculée par une opération de FFT inverse. Le Tableau 2.2 présente les paramètres OFDM des systèmes MATRICE et 4MORE. Les symboles OFDM ont des caractéristiques assez semblables. Ils sont complétés par un préfixe cyclique pour se prémunir des interférences inter-symboles.

Les systèmes MATRICE et 4MORE proposent de nombreuses classes de fonctionnement qui en fonction l'environnement d'utilisation et du type de service souhaité permettent d'obtenir un compromis optimal entre le débit de donnée et la fiabilité des transmissions.

Dans MATRICE, il est possible d'utiliser deux types de modulation (QPSK ou 16QAM). Le codage canal est réalisé par un codeur convolutif associé à une unité de poinçonnage qui permet de choisir entre deux rapports de codage (1/2 ou 3/4). Enfin l'accès multiple par répartition de code (CDMA) alloue entre 1 à 32 codes par utilisateur

TAB. 2.3 – Classes de fonctionnement du système MATRICE

	Classe 1		**Classe 2**		**Classe 3**		**Classe 4**	
Nombre de codes	1	32	1	32	1	32	1	32
Modulation	QPSK	QPSK	QPSK	QPSK	16-QAM	16-QAM	16-QAM	16-QAM
Rapport de codage	1/2	1/2	3/4	3/4	1/2	1/2	3/4	3/4
Bits de données par trame	552	17664	828	26496	1104	35328	1656	52992
Débit donnée (Mbps)	0,83	26,50	1,24	39,74	1,66	52,99	2,48	79,49

ce qui constitue une variable supplémentaire pour caractériser une trame. Le terme utilisateur est à prendre au sens applicatif, il correspond à un besoin en bande-passante pour un service (donnée, voix, vidéo). Plusieurs utilisateurs peuvent être associés à un même terminal mobile. Le Tableau 2.3 présente les différentes classes de fonctionnements possibles et les débits de données associés. Le débit étant proportionnels au nombre de codes, il est possible de déduire les débits intermédiaires entre 1 et 32 codes.

TAB. 2.4 – Classes de fonctionnement du système 4MORE en DL

Codage canal	Convolutif												LDPC		
Modulation	QPSK	8PSK	16QAM	64QAM	QPSK	8PSK	16QAM	64QAM	QPSK	8PSK	16QAM	64QAM	QPSK	16QAM	64QAM
Rapport de codage	1/2	1/2	1/2	1/2	2/3	2/3	2/3	2/3	3/4	3/4	3/4	3/4	0,794	0,794	0,794
Bits de données par trame	504	756	1008	1512	672	1008	1344	2016	756	1134	1512	2268	800	1600	2400
Débit donnée par code (Mbps)	0,8	1,1	1,5	2,3	1,0	1,5	2,0	3,0	1,1	1,7	2,3	3,4	1,2	2,4	3,6
Débit total (Mbps)	24,2	36,3	48,4	72,6	32,3	48,4	64,5	96,8	36,3	54,4	72,6	108,9	38,4	76,8	115,2

Comme nous l'avons déjà évoqué précédemment, avec 4MORE, la modulation n'est pas équivalente en UL et DL. Pour les trames DL, le codage canal est réalisé avec un codeur convolutif (rapport de codage 1/2, 2/3 ou 3/4) ou un codeur LDPC [11]. 4 types de modulations sont possibles (QPSK, 8PSK, 16QAM et 64QAM). Enfin un utilisateur peut se voir attribuer de 1 à 32 codes d'étalement de spectre. Le Tableau 2.5 récapitule les différentes classes ainsi disponibles. Le débit total est atteint lorsque les 32 codes sont utilisés par le même utilisateur.

Pour les trames UL, le paramétrage est différent puisqu'il doit permettre de distinguer les données issues de chaque terminal mobile (MT). Les 576 porteuses de données d'un symbole OFDM sont réparties en 24 sous-bandes. On utilise le terme de SS-MC-MA (*Spread Spectrum - MultiCarrier - Multiple Access*) pour définir ce mode d'accès

[11] *Low Density Parity Check*

TAB. 2.5 – Classes de fonctionnement du système 4MORE en UL en pleine charge

Modulation	QPSK	8PSK	16QAM	64QAM	QPSK	8PSK	16QAM	64QAM	QPSK	8PSK	16QAM	64QAM
Rapport de codage	1/2	1/2	1/2	1/2	2/3	2/3	2/3	2/3	3/4	3/4	3/4	3/4
Bits de données par trame	576	864	1152	1728	768	1152	1536	2304	864	1296	1728	2592
Débit donnée par sous-bande (Mbps)	0,86	1,30	1,73	2,59	1,15	1,73	2,30	3,46	1,30	1,94	2,59	3,89
Débit total (Mbps)	20,7	31,1	41,5	62,2	27,6	41,5	55,3	82,9	31,1	46,7	62,2	93,3

à la radio. Chaque MT se voit affecter un certain nombre de sous-bande. L'étalement de spectre utilise 8 codes qui sont répartis entre les utilisateurs d'un même MT. On distingue trois modes : pleine-charge, demi-charge et quart-charge correspondant respectivement à l'utilisation de 8, 4 et 2 codes. Le Tableau 2.5 résume les différentes classes des fonctionnement dans le cas pleine-charge. Les débits doivent être divisés par 2 ou 4 pour les cas demi et quart-charge. Le débit total correspond à l'utilisation des 24 sous-bandes par le même couple MT-utilisateur.

2.2.2 Présentation des chaînes de traitement MATRICE et 4MORE

Nous allons dans ce paragraphe présenter les chaînes de traitement pour la modulation et la démodulation des systèmes MATRICE et 4MORE. Cela représente quatre chaînes indépendantes. Il ne s'agit pas de faire une description fine des algorithmes de traitement du signal mais plutôt d'identifier les différentes unités de traitement et les flots de données qui existent entre elles.

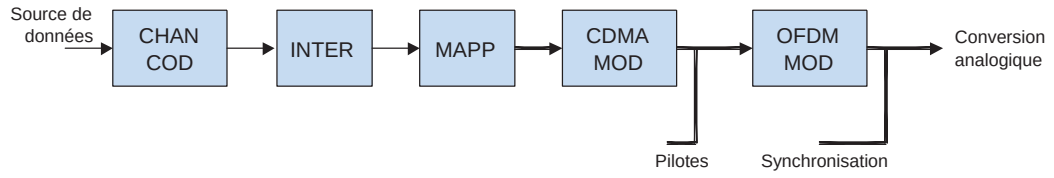


FIG. 2.5 – Modules de la chaîne d'émission MATRICE

La chaîne de modulation MATRICE est présentée Figure 2.5. Les symboles de synchronisation sont précalculés est transmis directement. Le flot de données binaires à envoyer est encodé dans l'unité de codage canal (CHAN_COD). Un système d'entrelacement (INTER) permet de réorganiser les données ce qui permet en réception de répartir les erreurs ponctuelles après désentrelacement et ainsi de rendre possible leurs

corrections. L'unité mappeur (MAPP) positionne les bits sur une constellation en fonction de la modulation (QPSK ou 16QAM). Les données sont alors représentées par des nombreux complexes (voix I et Q). Les codes d'étalement sont introduits dans l'unité de modulation CDMA (CDMA_MOD). Il s'agit principalement de réaliser une FHT[12]. Enfin les données sont répartis sur les différentes sous-porteuses et converties en symboles OFDM par une FFT[13] dans l'unité OFDM_MOD. Les échantillons temporelles ainsi générés peuvent être transmis sur le canal radio. Les symboles pilotes sont générés à partir de porteuses précalculés par l'unité OFDM_MOD.

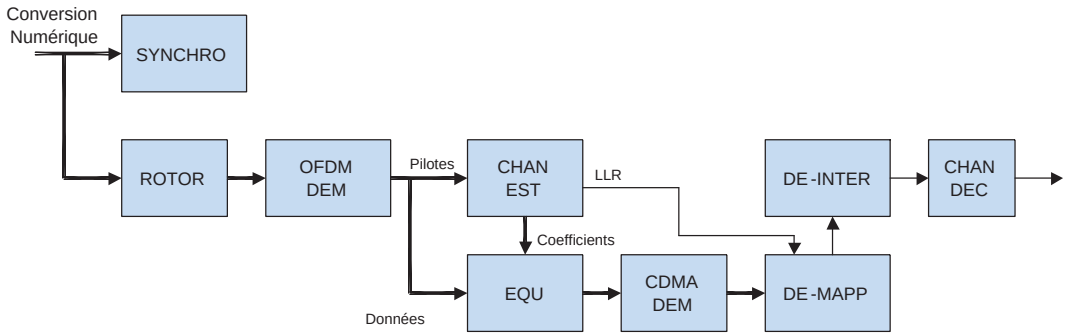


FIG. 2.6 – Modules de la chaîne de réception MATRICE

La chaîne de réception MATRICE (Figure 2.6) va permettre de réaliser les opérations inverses pour récupérer les données binaires après échantillonnage et numérisation du signal radio. Dans un premier temps, une unité SYNCHRO effectue une corrélation entre le signal reçu et les valeurs de synchronisation pour identifier le début d'une trame et ainsi caler le récepteur par rapport aux symboles OFDM. L'unité ROTOR effectue des corrections fréquentielles (écart entre les oscillateurs du côté de l'émetteur et du récepteur) et temporelles (écart au niveau des instants d'échantillonnage). L'unité OFDM_DEM permet d'extraire les valeurs des porteuses des symboles temporels. Les porteuses pilotes sont envoyés à un estimateur de canal (CHAN_EST) qui calcule les coefficients d'égalisation. Les données sont corrigées dans l'égaliseur (EQU). On retrouve ensuite le dual de la chaînes d'émission avec la démodulation CDMA, le démappeur, le désentrelaceur et de décodage canal. Il faut noter que le démappeur convertit les échantillons complexes en *soft bits* en fonction des coefficients de vraisemblance LLR (*Log Likelihood Ratio*) fournis par l'estimateur de canal. Ces *soft bits* servent de métriques au décodeur de Viterbi qui est utilisé par l'unité CHAN_DEC.

[12] *Fast Hadamard Transform*
[13] *Fast Fourier Transform*

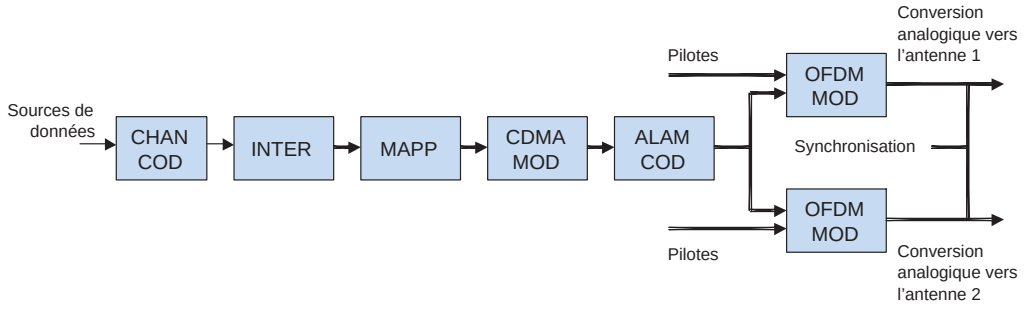


FIG. 2.7 – Modules de la chaîne d'émission 4MORE

La chaîne d'émission 4MORE (Figure 2.7) est une évolution de la chaîne MATRICE avec l'utilisation d'un système MIMO à deux antennes. Les données à émettre sont ainsi réparties selon un codage spatio-temporel sur deux antennes dans l'unité de codage d'Alamouti [Alam98] (ALAM_COD).

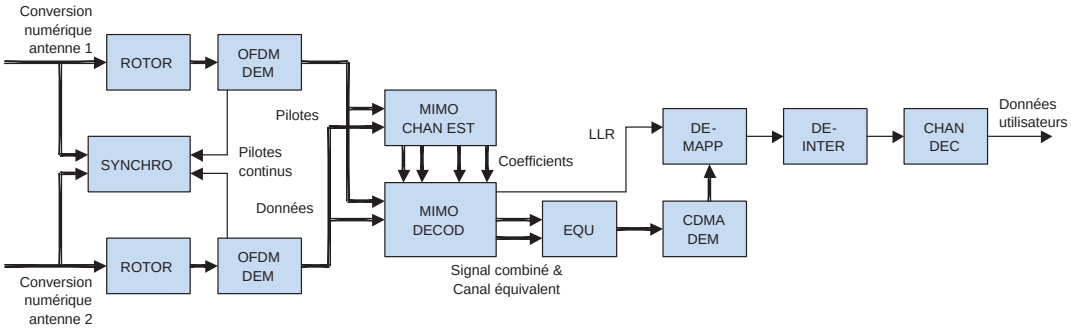


FIG. 2.8 – Modules de la chaîne de réception 4MORE

En réception la chaîne 4MORE doit effectuer la combinaison entre les signaux reçus sur chacune des deux antennes. Les opérations de démodulation OFDM sont effectuées de manière indépendante pour chaque antenne. Les porteuses correspondant aux pilotes sont utilisées pour effectuer des corrections sur les données. Les symboles pilotes continus sont envoyés à l'unité d'estimation de canal (MIMO_CHAN_EST). Cette unité va calculer quatre séries de coefficients en relation avec les quatre canaux formés par les couples d'antennes d'émission et de réception. Les données issues des deux antennes sont regroupées dans l'unité de décodage MIMO. Cette unité fournit ainsi un signal combiné et des coefficients équivalents à un canal radio unique pour l'unité d'égalisation. Une fois les signaux combinés, la suite de la chaîne de traitement est comparable à celle de MATRICE.

2.2.3 Spécificité d'implémentation des systèmes de télécommunication

Nous venons d'exposer le fonctionnement de systèmes de télécommunication haut-débit sans-fil. Les traitements en bande de base associés à ces applications présentent un certain nombre de caractéristiques spécifiques. La prise en compte de ces caractéristiques est déterminante pour les choix d'architecture et les compromis d'implémentation dans l'optique de réaliser un système sur puce supportant ces applications.

- Système hétérogène

Les chaînes d'émission et de réception sont divisées en différentes unités de traitement. Il existe une grande hétérogénéité entre ces unités. Le format des données varie beaucoup d'une unité à l'autre. Pour le codage canal, les opérateurs travaillent le plus souvent au niveau bit, un calcul de FFT[14] peut faire intervenir des nombres complexes codés sur 32 bits, un démappeur manipule des formats intermédiaires avec 5 ou 6 bits par exemple. Ces unités ont également des temps de traitement différents et ne travaillent pas sur les mêmes quantités de données. Un calcul de FHT[15] dure plusieurs cycles sur un bloc de données bien défini, au contraire un système d'égalisation travaille à flux tendu sur un échantillon temporel. Cela crée des variations dans les débits de données tout au long des chaînes de traitement. Ces éléments ont pour conséquence de rendre complexe la mise en place d'un système de communication et de contrôle bien adapté à cette grande variété de comportement au sein des unités de traitement

- Système flots de données

Pour les traitements de modulation et de démodulation numériques du signal, une grande quantité de données doit être manipulée et échangée entre les unités de traitement avec des débits élevés. Les traitements sont assez indépendants les uns des autres. Au niveau arithmétique les calculs ne sont pas trop complexes : il y a peu de calculs itératifs et les dynamiques sont limités. La difficultés provient donc de l'important brassage des données qu'il faut réaliser : algorithme d'entrelacement, répartition en fonction de codes, de sous-bandes, de sous-porteuses. L'architecture du système doit donc être organisé de manière à supporter de multiples flots de données avec des débits souvent supérieurs à 1Gbps (cf. paragraphe 2.2.1.1).

- Système avec des contraintes temps réelles fortes

Un système de télécommunication doit répondre à un certain nombres de contraintes

[14] *Fast Fourier Transform*
[15] *Fast Hadamard Transform*

temporelles imposées par le protocole de communication de la norme ciblée. Cela concerne en particulier la durée des symboles de données, la durée des trames ou les intervalles entre deux trames. Le système doit être capable de traiter les données en temps réel. Certaines latences doivent également être respectées. Un système de communication introduit des rebouclages : il faut par exemple un temps minimum entre la réception d'un message de contrôle et l'émission d'un message d'acquittement. L'optimisation temporelle est donc un aspect primordial à prendre en compte lors de la conception de ces systèmes.

- Contrôle et reconfiguration

Le contrôle et la synchronisation d'unités de traitement de natures différentes n'est pas aisé d'autant plus si les différentes unités sont distribuées sur un système d'interconnexion introduisant des latences de communication. Pour un système de télécommunication, il existe aussi de nombreux modes de fonctionnement. Il peut s'agir de différentes phases (synchronisation, réception...) dans l'exécution d'une application. Cela peut également être différents paramètres de configuration (débit, constellations, rapport de codage...). Enfin, on peut être amené à changer d'application (système multistandard). Si l'on souhaite réaliser un système (en l'occurrence un système sur puce) supportant ces différents modes de fonctionnement, sans pour autant multiplier les sous-systèmes, il faut travailler sur des architectures reconfigurables, et même dynamiquement reconfigurables. Ceci constitue donc une contrainte d'intégration supplémentaire.

2.3 Le réseau FAUST

Les deux premières sections du chapitre nous ont permis de présenter le contexte précis des systèmes de télécommunications pour la 4G en particulier dans le cadre de projets qui visent à l'intégration de ces fonctionnalités dans une solution à base de système sur puce (SoC). Au niveau de l'architecture des terminaux de communication, ces applications induisent trois tendances :

- Une augmentation de la complexité. Les nouveaux algorithmes imposent des puissances de calculs plus élevées avec des débits de données plus importants.
- Des mécanismes de reconfiguration. La convergence des standards n'est possible qu'avec des systèmes multiapplications. Les ressources de traitement sont mutualisées et reconfigurées en fonction du type de fonction réalisée.
- Des plateformes d'intégration ouvertes et flexibles. Les standards changent rapidement, et il faut donc être capable de faire évoluer les circuits en conséquence.

Ces considérations ont amené le LETI à proposer une solution de SoC pour les télécommunications organisée autour d'une architecture de réseau sur puce que nous allons présenter dans cette section.

2.3.1 Architecture du réseau

L'architecture de réseau sur puce que nous allons présenter est développée au LETI dans le cadre du projet FAUST (F*lexible* A*rchitecture of* U*nified* S*ystem for* T*elecom*). L'objectif recherché est de proposer une structure de communication flexible, performante (bonne qualité de service en termes de latence et de débit) tout en limitant la complexité des mécanismes de communication.

2.3.1.1 Topologie du réseau

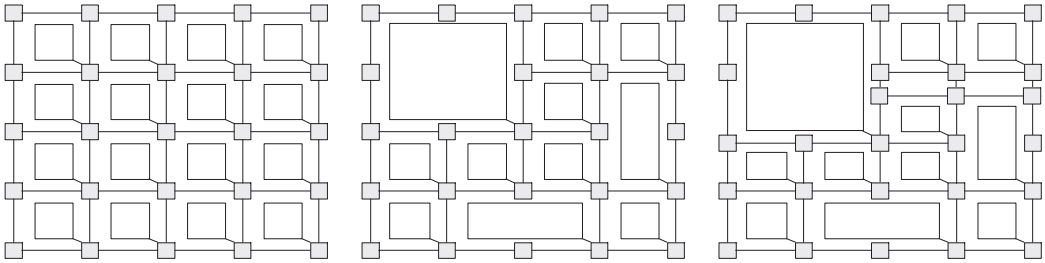
Le réseau FAUST est une architecture d'interconnexion pour des applications de types flots de données. Les ressources de traitement connectées au réseau sont de nature hétérogène et donc a priori de tailles variables. Au niveau du réseau, il est préférable d'avoir une structure régulière pour limiter la complexité et des liens courts pour de meilleurs performances temporelles.

Ces contraintes ont mené au choix d'une topologie maillée à deux dimensions (Figure 2.9(a)). Chaque noeud est relié à ses quatre voisins les plus proches ainsi qu'à une ressource de traitement. Cette structure sert de base de départ pour développer un noeud générique et un mode de routage. En fonction de l'application, la topologie peut être adaptée à la taille des ressources. On obtient alors un réseau (Figure 2.9(b)) avec des tailles de ressources multiples d'un bloc minimal correspondant à l'espace entre quatre noeuds.

Les ressources de tailles plus importantes peuvent entraîner la suppression de certains liens. Ceci affecte le débit global du réseau est peut créer localement des phénomènes de congestion. Pour y remédier, il est possible de d'augmenter la bande-passante en dédoublant des noeuds et en adaptant le routage (Figure 2.9(c))

2.3.1.2 Mécanismes de communication

FAUST est un réseau commuté par paquet. Le mode de commutation est de type *wormhole* (cf. paragraphe 1.2.1.2). Les flits sont composés de 34 bits. Les données (signal



(a) Topologie régulière (b) Topologie adaptée au ressource (c) Topologie adaptée au débit

FIG. 2.9 – Approche topologique pour FAUST

data) sont codées sur 32 bits. Un paquet est toujours composé d'un flit de tête[16] et éventuellement d'un ou plusieurs flits[17]. L'algorithme de routage est déterministe de type source (cf. paragraphe 1.2.1.4). Les chemins de routage[18] sont pré-calculés par un algorithme de routage qui garantit l'absence d'interblocage [GlNi94]. Ils sont ensuite chargés dans les ressources en fonction de leurs besoins.

Chaque port d'entrée des noeuds implémente une stratégie de file d'attente avec deux canaux virtuels (cf. paragraphe 1.2.1.3). Ces canaux virtuels sont utilisés pour définir deux niveaux de priorités différents. L'arbitre du noeud fait passer prioritairement les paquets envoyés sur le canal '0' par rapport au canal '1'.

Il existe deux implémentations possibles pour ces mécanismes de communication. L'une est faite en logique synchrone. Dans ce cas tout le réseau est synchronisé sur un même arbre d'horloge. L'autre est réalisée en logique asynchrone (quasiment insensible aux délais [BCVC05]). Il n'y a alors plus de signal d'horloge, les noeuds se synchronisent mutuellement par un mécanisme de poignée de main[19]. A haut-niveau, le réseau FAUST fonctionne alors comme un système GALS (cf. paragraphe 1.1.2). Cette approche renforce l'idée d'une architecture distribuée au niveau des communications et du contrôle. Il n'existe pas de signaux globaux et l'ensemble des informations transitent par le réseau entre les différents îlots synchrones formés par les ressources.

2.3.2 Le protocole de communication

Le protocole de communication de FAUST est hiérarchisé en 4 niveaux [Bern04] :

[16] *header*
[17] flits de données
[18] appelés *path to target*
[19] *handshake protocol*

- Le niveau flit. Il correspond au niveau d'échange local d'un flit entre deux noeuds ou entre une ressource et un noeud. Cet échange est géré par un mécanisme de synchronisation basé sur deux signaux : *Send* et *Accept* (un couple pour chaque canal virtuel). *Send* indique qu'un flit est envoyé. *Accept* reste valide tant que les flits peuvent être reçus (exemple Figure 2.10). Ce protocole peut être implémenté en logique synchrone ou asynchrone.

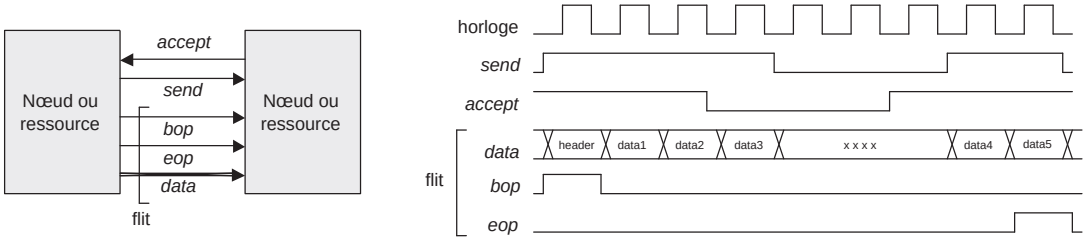
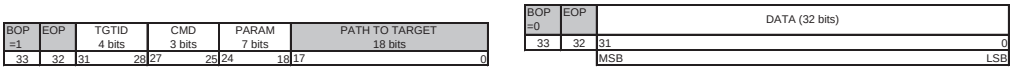


FIG. 2.10 – Mécanisme de synchronisation *send*/*accept* pour la couche flit

- Le niveau paquet. C'est le niveau d'échange des paquets à travers le réseau. Le *header* (Figure 2.11) contient les informations nécessaires au routage du paquet à travers les noeuds. Il est identifié par le bit BOP[20] à l'état 1. Le dernier flit (qui peut être le *header*) est repéré par le bit EOP[21]. Ceci permet de garder la cohésion des flits d'un même paquet au niveau de l'arbitrage des noeuds. Pour le routage, la topologie maillée à deux dimensions définit cinq directions : Nord, Sud, Est, Ouest et Ressource. La succession des directions à prendre est codée dans le champ PATH_TO_TARGET du *header*. Chaque direction est codée sur deux bits[22]. La ressource est adressée en retournant le paquet sur la direction dont il provient. Un paquet ne peut pas faire demi-tour. La longueur actuelle du PATH_TO_TARGET est de 9 changement de directions.



(a) Format du *header*

(b) Format d'un flit de donnée

FIG. 2.11 – Format des flits FAUST

- Le niveau communication est le niveau d'échange entre deux ressources. Il découpe les blocs de données du niveau application en paquets de taille raisonnable pour le réseau. Ce niveau permet aussi de garantir que le réseau ne se bloquera pas à

[20] *begining of packet*
[21] *end of packet*
[22] Nord = 00, Est = 01, Sud = 10, Ouest = 11

cause d'un engorgement. Une communication est définie entre une ressource émettrice et une ressource réceptrice. Les deux ressources s'échangent des paquets de service pour réguler la communication : FAUST utilise pour cela un mécanisme de gestion de flux par crédit (cf. paragraphe 1.2.2.2). L'en-tête du paquet contient des paramètres intervenant au niveau communication. Tout d'abord le champ TGTID[23] indique la destination du paquet dans la ressource (Tableau 2.6(a)). Lorsque les données sont à destination des registres de configuration, le second flit donne l'adresse du registre. Le champ CMD (Tableau 2.6(b)) associe une commande au paquet ou précise s'il s'agit d'un paquet de crédit. Dans ce cas le paquet ne contient qu'un flit et le nombre de crédit est indiqué dans le champ PARAM. Nous détaillerons la commande *initial write* (INIT_WRITE) dans le paragraphe 2.4.3

TAB. 2.6 – Paramètres du *header* au niveau communication

(a) TGTID

Valeur	**Destination**
0000	Registres de configuration (RWD)
0001 à 1111	Mémoire FIFO (jusqu'à 15)

(b) CMD

Code	**Commande**
000	read
001	write
010	initial write
011	send credit

- Le niveau application est le niveau de définition des données utilisateur. A ce niveau, les ressources s'échangent des messages (ou blocs de données) de différents types, permettant le transfert de données, la configuration et le contrôle des ressources, les interruptions du processeur.

Dans FAUST, les trois couches basses sont matérielles. Le processeur configure la couche communication.

2.3.3 Intégration des unités de traitement

Le réseau FAUST est conçu comme une plate-forme de communication ouverte permettant d'intégrer différentes unités de traitement de nature hétérogène. L'intégration de ces unités est facilitée grâce à une architecture modulaire d'interface réseau. Le concept d'interface réseau (ou NI[24]) a été présenté aux paragraphes 1.2.2.3 et 1.4.2.

[23] identification de la cible - *target identity*
[24] *Network Interface*

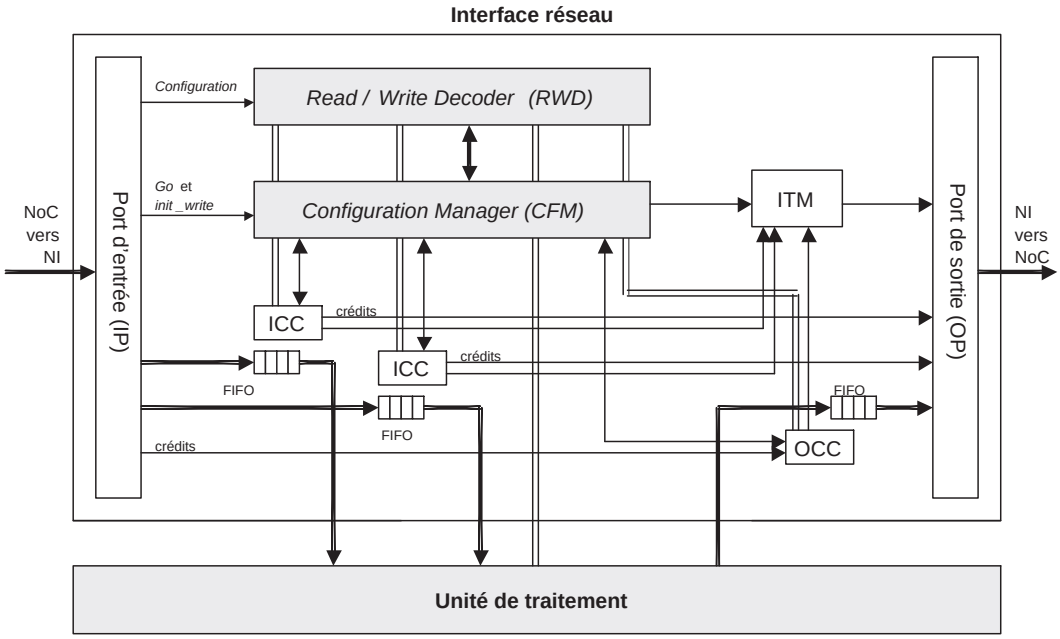


FIG. 2.12 – Architecture de l'interface réseau FAUST

2.3.3.1 Architecture de l'interface réseau

Dans FAUST, l'interface réseau a une double fonctionnalité. Elle gère à la fois les communications entre l'unité de traitement et le réseau mais également la configuration des traitements et leurs enchaînements.

L'architecture de l'interface est paramétrable. Elle est constituée de différents éléments qui peuvent être assemblés et configurés en fonction des besoins de l'unité de traitement. La Figure 2.12 présente un exemple d'architecture de l'interface FAUST. Nous allons détailler les différents éléments constitutifs de l'interface. On se focalise ici sur l'architecture. Les aspects fonctionnels seront repris aux paragraphes 2.4.2 et 2.4.3.

- Ports d'entrée et de sortie[25]

Les portes d'entrée et de sortie [IP03] [OP03] ont pour fonctions de gérer les communications aux niveaux flit et paquet : gestion des signaux *Send* et *Accept* pour les deux canaux virtuels, gestion des bits BOP et EOP. En réception, le port d'entrée reçoit les paquets. Il décode l'en-tête et dirige les données dans l'interface en fonction de la commande (CMD) et de la destination (TGTID). En émission, le port de sortie effectue l'arbitrage entre les différents modules qui veulent envoyer un paquet.

[25]IP pour *Input Port et OP pour* Output Port

- Contrôleurs de communication d'entrée et de sortie[26]

L'émission et la réception des données entre l'unité de traitement et le reste du circuit s'effectue par l'intermédiaire de mémoires FIFO. Le traitement et les communications sont ainsi totalement découplés. A chaque mémoire FIFO est associée un contrôleur de communication [ICC04] [OCC04]. Le nombre de mémoires FIFO et donc le nombre de contrôleurs de communication dépend ainsi directement du nombre d'entrées et de sorties de l'unité de traitement. En réception les contrôleurs (ICC) assure l'envoi des crédits en fonction du contenu des mémoires FIFO pour le mécanismes de contrôle de flux. En émission les contrôleurs (OCC) envoie les paquets de données lorsque suffisamment de crédits en provenance de la ressource de destination ont été reçus.

- Gestionnaire de configuration [27]

Le fonctionnement des contrôleurs de communication et de l'unité de traitement dépend d'un certains nombre de paramètres de configuration. Le chargement de ces paramètres est assuré par le gestionnaire de configuration. Nous détaillerons les mécanismes de ce gestionnaire dans les paragraphes décrivant la gestion des flots de données (2.4.2) et le contrôle des traitement (2.4.3).

- *Read/Write Decoder*[28]

Le Read/Write Decoder (RWD) [RWD04] contient les registres de configurations des contrôleurs de communications et de l'unité de traitement. Le RWD est donc spécifique à chaque ressource. Il décode et exécute les ordres d'écriture des registres de configuration, en provenance du réseau. Ces registres sont ensuite accessibles à leurs destinataires respectifs.

- Gestionnaire d'interruption[29]

Le gestionnaire d'interruption [ITM04] génère des paquets permettant de signaler au processeur de contrôle (cf. paragraphe 2.4.3) certain événements au niveau de l'interface. Cela peut être pour indiquer la fin de l'exécution d'une configuration ou en cas d'erreur dans le chargement d'une configuration.

2.3.3.2 Configurations des contrôleurs de communication

Les contrôleurs de communication sont configurés par une série de paramètres qui sont stockés dans le RWD. Le CFM commande l'exécution de ces configurations et les

[26] ICC pour *Input Communication Controler* et OCC pour *Output Communication Controler*
[27] CFM pour *Configuration Manager*
[28] RWD
[29] ITM pour *Interrupt Manager*

contrôleurs signalent lorsqu'ils sont prêts à charger une nouvelle configuration. Dans la suite du paragraphe, nous allons détailler les paramètres de configuration de ces contrôleurs qui seront utilisés dans le système présenté au chapitre 4.

L'organisation des registres de configuration de l'ICC sont présentés Figure 2.13. l'ICC est configuré pour envoyer un certain nombre de crédit (*Total Crédit*). Les crédits sont envoyés si la place disponible dans la mémoire FIFO est supérieur au *Seuil Crédit*. La destination des crédits est fixée par le *Chemin de routage* et le canal virtuel utilisé par le champ *Prio*. Le champ *TGTID* correspond au registre de stockage des crédits dans la ressource réceptrice (cf. *Séléction crédit* dans la configuration de l'OCC). Une interruption peut être envoyée en fin de configuration en fonction du *Masque IT*.

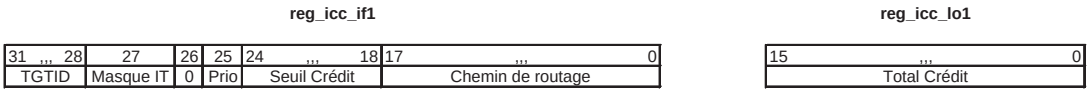


FIG. 2.13 – Registres de configuration du contrôleur de communication d'entrée (ICC)

Les registres de configuration des OCC sont présentés à la Figure 2.14. L'OCC est configuré pour envoyer un nombre de flit correspondant à *Total donnée* avec une *Taille paquet* fixée. La configuration peut être répétée en fonction du *Nombre de boucle* spécifié. La destination des données est paramétrée par le *Chemin de routage*, le canal *Prio* et le numéro de la mémoire FIFO de destination *TGTID*. L'envoi des données est conditionné par la présence de crédits dans le compteur *Sélection crédit* sauf si l'option *Crédit infini* est utilisée. L'utilité de plusieurs compteurs de crédits distincts est de gérer simultanément plusieurs destinataires sans mélanger leurs crédits. Les paquets sont envoyés avec une commande *CMD* et un *Numéro de configuration*. Nous expliquerons ces champs au paragraphe 2.4.3.

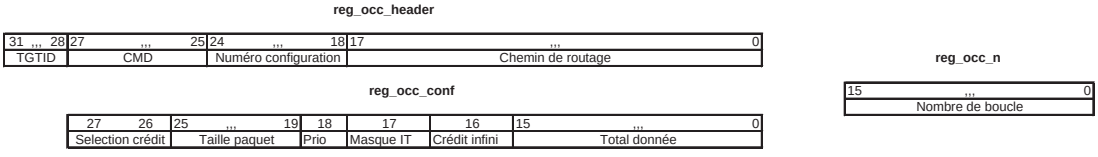


FIG. 2.14 – Registres de configuration du contrôleur de communication de sortie (OCC)

2.3.4 Mise en oeuvre du réseau FAUST : le circuit FAUST

Les concepts sur le réseau FAUST que nous venons de présenter ont été implémentés et validés dans un circuit conçu au LETI. Ce circuit a été réalisé en technologie 0,13μm

de STMicroelectronics et les premiers exemplaires ont été testés en janvier 2006. La conception du circuit par le laboratoire IAN s'est donc faite en parallèle de l'avancement de cette thèse.

Le circuit est représenté de manière simplifiée sur la Figure 2.15. Il comporte 20 noeuds asynchrones et une vingtaine de ressources de traitement. Les ressources ont été spécifiées et sélectionnées dans le but de réaliser les traitements correspondant à la couche physique des systèmes de télécommunication associés aux projets MATRICE et 4MORE (présentés dans la section 2.2). Certaines ressources correspondent à la chaîne d'émission (TX[30]), d'autres à celle de réception (RX[31]). Le contrôle du circuit est réalisé par une ressource spécifique (*CPU*) qui implémente un coeur de processeur ARM946E-S [ARM06] (4ko de cache de donnée et 4ko de cache d'instruction). Le processeur dispose d'un accès direct à deux mémoires embarquées (128ko chacune) et une mémoire externe via un bus AHB[32] [AMBA06]. Un pont permet de connecter le bus au réseau. L'utilisation de ce processeur et des mémoires embarquées seront détaillées au paragraphe 2.4.3.

L'implémentation des noeuds du réseau est faite en technique asynchrone. Au niveau performance, le noeud asynchrone permet des transferts à une cadence de l'ordre de 160 Mflits/s sur chacun des ports de sortie. La complexité du noeud est de 20k portes. Entre chaque noeud et ressource, une interface synchrone/asynchrone effectue l'adaptation des signaux. Cette interface représente 15k portes. FAUST est donc une architecture GALS (cf. paragraphe 1.1.2). La ressource *Horloge et Test* génère les 24 signaux d'horloge qui cadencent les ressources de traitement. Chaque horloge peut être réglée individuellement pour optimiser la consommation du circuit en fonction des contraintes de calcul (Principe du DFS : *Digital Frequency Scaling* [MuRB06]). La ressource *NoC Performance* permet de mesurer les caractéristiques du réseau (débit, latence).

Le circuit communique avec l'extérieur grâce à un port Ethernet. La ressource *Ethernet* effectue la conversion de protocole. Il existe également deux liens bidirectionnels du réseau qui sont connectés à des plots d'entrée et de sortie. Ceci offre la possibilité d'étendre le réseau à l'extérieur du circuit. Le bloc *I/O* permet de choisir entre une extension synchrone ou asynchrone. Ainsi, dans le cadre du projet 4MORE, une carte prototype a été réalisée incluant deux circuits FAUST et deux composants programmables (FPGA Xilinx Virtex-4 [VIRT06]). Les FPGA contiennent des noeuds synchrones du réseau. Tous ces composants communiquent de manière complètement transparente grâce

[30] *Transmitter*
[31] *Receiver*
[32] *Advanced High-speed Bus*

au protocole FAUST.

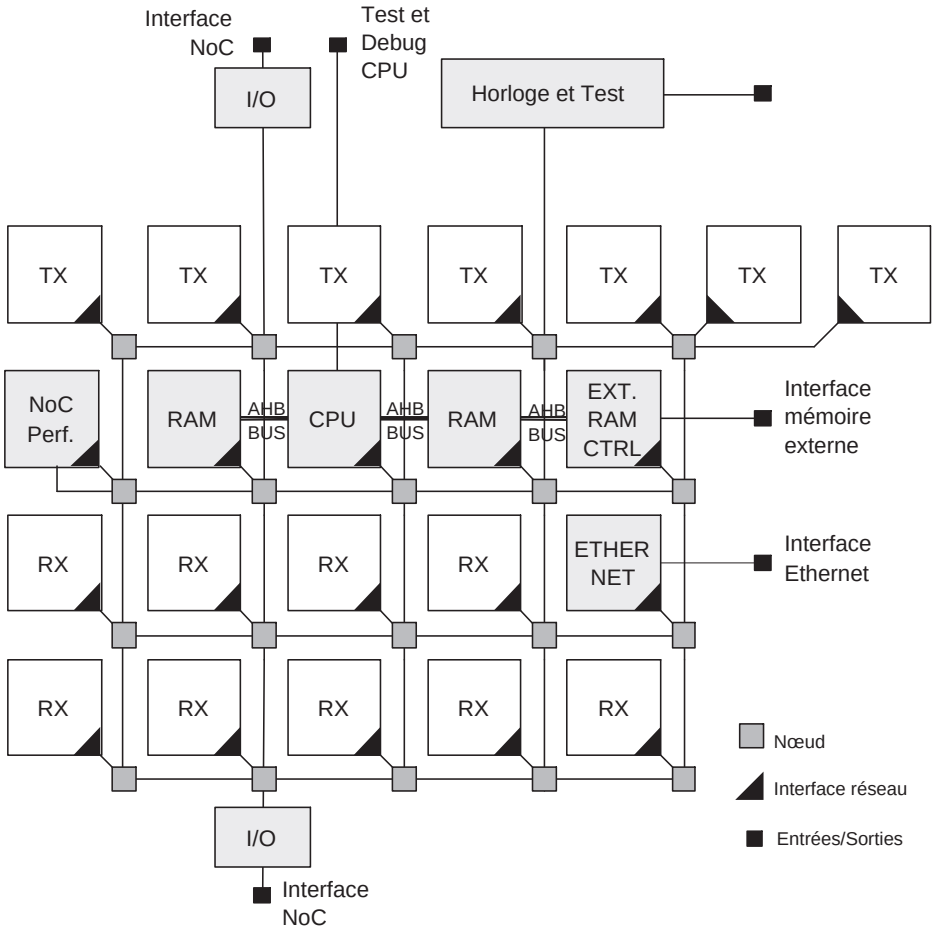


FIG. 2.15 – Architecture du circuit FAUST

2.4 Conception d'un système sur puce avec le réseau FAUST

Dans le chapitre 1, nous avons présenté des enjeux nouveaux liés à la conception d'un NoC dans un cadre général (cf. section 1.4). Cette nouvelle section a pour objectif de focaliser ces problématiques dans le contexte du projet FAUST. Nous verrons ainsi comment le réseau FAUST apporte des réponses en ce qui concerne l'évaluation des performances, la gestion des communications et le contrôle des traitements.

2.4.1 Modélisation TLM/SystemC du réseau FAUST

Dans le cadre du projet FAUST, le laboratoire IAN a mis en place un modèle de simulation pour évaluer les performances du réseau. Ce modèle permet de simuler les mécanismes de communication au niveau de noeuds et des interfaces. Le langage SystemC et la bibliothèque TLM développée par STMicroelectronics ont été utilisés. Une introduction au langage SystemC et à la modélisation TLM est présentée en Annexe B.1.

Ce modèle propose différentes briques de bases permettant au concepteur de construire un NoC spécifique basé sur le protocole FAUST. Dans les paragraphes suivants nous présenterons ce modèle d'un point de vue fonctionnel. Pour des détails d'implémentation logiciel on peut se référer à [BCVC05]. Dans le chapitre 3, nous utiliserons ce modèle pour simuler le trafic réseau correspondant à une application MC-CDMA.

2.4.1.1 Modélisation du noeud et gestion des transactions

Dans FAUST, les mécanismes de communication sont gérés par les noeuds. Le module *anoc_ node* (dérivé de *sc_ module*) modélise un noeud du réseau. Il contient des ports d'entrée `node_in[nb_ports]` de type `anoc_in` et des ports de sortie vers le réseau `node_out[nb_ports]` de type `anoc_out`. Pour la prise en compte du temps, une unique valeur `node_wait_state` sert à modéliser le temps minimum entre deux transferts successifs sur une même sortie. Un événement `sc_event node_event` permet de synchroniser le noeud avec son entourage.

La fonctionnement du réseau nécessite un transfert bi-directionnel entre les noeuds. Ce transfert obéit à un protocole de synchronisation local (cf paragraphe 2.3.2). Les signaux *data* et *accept* sont modélisés par l'exécution d'une fonction de `transport` unique définie dans l'API [33] TLM/SystemC. Il s'agit de la fonction :

```
tlm_status transport(tlm_transaction & transaction)
```

Une transaction peut être de deux types : DATA ou ACCEPT. Une transaction de type DATA contient les informations correspondant à un flit. Une transaction ACCEPT correspond au signal de validation du mécanisme de synchronisation.

Dans le noeud, la fonction `transport()` permet d'écrire la valeur d'une transaction entrante ce qui génère un événement `node_event` qui va déclenché le processus

[33] Application Programming Interface

(`sc_thread`) principal du noeud. Ce processus va arbitrer toutes les requêtes et générer les transactions DATA et ACCEPT sortantes.

La connexion entre les noeuds se fait grâce au mécanisme de *binding* SystemC comme par exemple : `nodeA->node_out[NORTH]->bind(nodeB->node_in[SOUTH]) ;`

2.4.1.2 Modélisation de l'interface réseau

Le module *anoc_ni* contient une description SystemC de l'interface réseau du réseau FAUST (cf. paragraphe 2.3.3). Un fichier de paramètres permet de définir la structure de chaque interface : nombre et tailles des mémoires FIFO, nombre de configurations. L'interface implémente la fonction `transport()` pour recevoir les transactions. En fonction du type de transaction reçue, l'interface met à jour différentes variables (compteur de crédits, tables d'ACCEPT, registre de configurations...) ou écrit des données dans les mémoires FIFO. Ces différentes opérations génèrent des événements (`sc_event`) qui déclenche la fonction `write_packet_main()` (sc_thread). Cette fonction réalise l'arbitrage des différentes requêtes et le cas échéant envoie des transactions sur le noeud connecté à l'interface.

2.4.2 Gestion des flots de données dans le circuit FAUST

Dans le réseau FAUST, la gestion des flots de données est réalisée au niveau des interfaces réseau (cf. paragraphe 2.3.3.1).

En réception, chaque flot de donnée est contrôlé par un contrôleur de communication d'entrée (ICC). Le gestionnaire de configuration (CFM) permet de gérer deux configurations par ICC. Une commande de démarrage précise si le gestionnaire doit charger une configuration ou l'autre ou bien l'une puis l'autre. Un paramètre de boucle indique le nombre de répétition à effectuer. Ainsi en nommant les configurations 1 et 2 et en programmant un chargement en alternance avec un paramètre de boucle de 5, on obtient de séquencement suivant : 1 2 1 2 1. Ce mécanisme de séquencement est utile pour aller chercher des données alternativement dans deux endroits différents et pour les réceptionner dans la même mémoire FIFO. En effet, chaque configuration permet de sélectionner l'origine des données grâce au mécanisme de crédit.

En émission, les flots sont gérés par les contrôleurs de communication de sortie (OCC). Il peut y avoir jusqu'à quatre configurations possibles par OCC. Les configurations sont chargées dans le RWD puis l'écriture d'un registre permet de les valider.

Le chargement des configurations par le CFM est conditionné par la réception d'un paquet contenant la commande *initial write* (INIT_WRITE, cf. Tableau 2.6(b)) dans son en-tête associée à un numéro de configuration (CONFIGID). Ainsi, contrairement aux ICC, le séquencement des configurations dans les OCC n'est pas géré par le CFM mais dépend des données reçues. Une fois chargée, une configuration est exécutée jusqu'à l'envoi de toutes les données (éventuellement plusieurs fois de suite suivant le nombre de boucle). Si une nouvelle commande INIT_ WRITE est reçue pendant ce temps, elle est stockée dans une file d'attente et dépilée par le CFM ensuite.

Ces mécanismes sont compatibles avec les applications visées par le circuit FAUST. Ils présentent cependant un certains nombre de limitations. Au niveau des ICC, le séquencement est extrêmement simple. Il ne permet pas de gérer plus de deux configurations et les scénarios se limitent à un va-et-vient entre ces configurations. De plus les ICC ne sont pas suffisamment indépendants puisqu'ils sont démarrés au même instant par le CFM.

Pour les OCC, le séquencement des configurations n'est pas programmable. Il dépend du découpage des données en blocs associés à la commande INIT_WRITE. Il est possible de gérer plusieurs flux en parallèle en implémentant plusieurs OCC mais les configurations doivent être chargées au même instant sur chaque OCC.

Les contrôleurs de communication ont des registres de configurations dédiés dans le RWD ce qui limite la flexibilité dans la gestion de l'espace de stockage.

Dans le but de répondre à des besoins plus complexes au niveau des trafics à gérer dans des applications futures, il semble nécessaire de faire évoluer ces mécanismes de communications. Les axes d'études sont l'implémentation de possibilités de séquencement plus élaborées aussi bien au niveau réception et émission. Il faut également une indépendance entre les contrôleurs de communication.

2.4.3 Contrôle de l'application et modèle de programmation

Le réseau sur puce FAUST a été conçu dans le but d'implémenter un circuit réalisant des fonctions de traitements numériques pour un système de télécommunication. Les chaînes de traitement sont décomposées en unités distribuées sur le réseau. Les traitements et les communications entre ces unités sont assez indépendants des données dans ce type d'application (cf. paragraphe 2.2.3).

Le modèle de programmation de FAUST est basé sur un processeur (CPU cf. Figure 2.15). Ce processeur coordonne les traitements entre toutes les ressources du réseau.

Nous allons décrire ces mécanismes en faisant référence à la Figure 2.16. Dans un premier temps, les données à traiter sont écrites dans la mémoire interne (1). En émission, il peut s'agir de données reçues par le lien Ethernet. En réception, ce sont les données numérisées en sortie de l'étage radiofréquence.

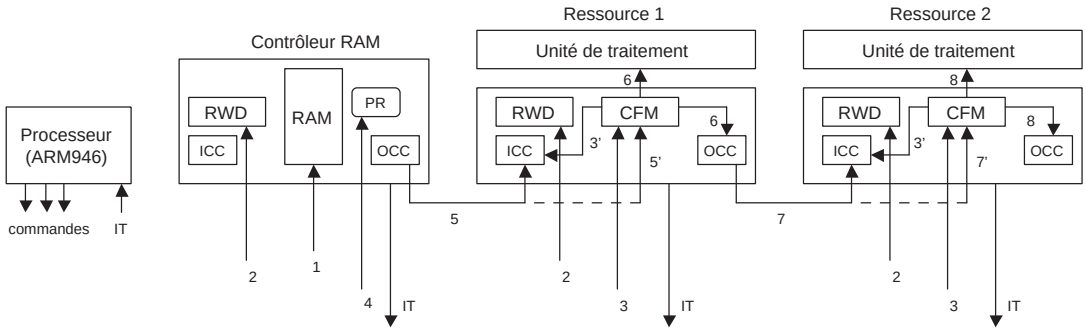


FIG. 2.16 – Scénario de configuration et de contrôle d'un flot de données dans FAUST

Les ressources mémoires (contrôleur RAM) ont un mode de fonctionnement particulier. Leur fonction est de stocker des données mais également de les réordonner ou de les brasser selon des algorithmes déterminés. Ce type de traitement est très utile à différentes étapes des chaînes de modulations en particulier pour l'entrelacement des données. Un processus de lecture (PR) microprogrammé effectue ces différentes manipulations.

Le processeur envoie ensuite, via le réseau, les paramètres de configurations dans les registres du RWD de chaque ressource (2). Ces configurations correspondent aux paramètres des contrôleurs de communications (ICC et OCC) et également aux unités de traitement. Il faut ensuite envoyer les signaux de démarrage (3) pour le contrôleur RAM et pour les ICC via le séquenceur du CFM (3'). Le contrôleur de RAM reçoit le programme pour le processus de lecture (4). Les ICC envoient alors les crédits vers les ressources sources et les données sont transmises (5). Le premier paquet du bloc de donnée est associé à une commande INIT_WRITE (5'). Cette commande indique au CFM qu'il faut charger une nouvelle configuration de traitement et d'OCC (6). Le numéro de configuration est également transmis dans l'en-tête du paquet. Une fois les données traitées par la Ressource 1, elles sont envoyées à la Ressource 2 (7). De nouveau, la commande INIT_WRITE (7') permet de configurer la ressource (8). Lorsque les configurations sont terminés, des interruptions (IT) peuvent être envoyées au processeur pour qu'il reconfigure les ressources.

Ce scénario de présentation est volontairement simplifié. En pratique, le contrôleur

RAM peut gérer jusqu'à quatre processus de lecture et les interfaces réseaux peuvent contenir plusieurs ICC et OCC.

Ce mode de programmation présente certains inconvénients. Tout d'abord, le séquencement des traitements n'est pas très souple. En effet, on ne peut pas le dissocier de la configuration des OCC. De plus le fait de propager les commandes INIT_WRITE d'une ressource à l'autre introduit des découpages en sous-blocs de données qui ne correspondent pas toujours à la granularité du traitement réalisé par la ressource.

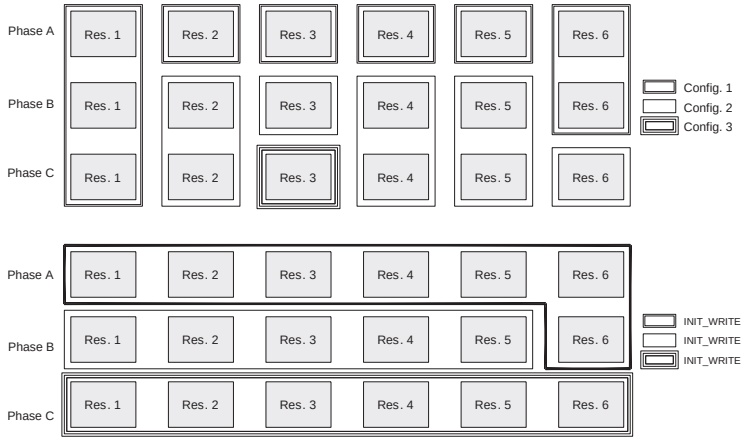


FIG. 2.17 – Configuration des unités de traitement par le mécanisme INIT_WRITE

Pour illustrer ce phénomène, nous pouvons prendre l'exemple de la Figure 2.17. On imagine un traitement réparti sur 6 ressources. Dans la partie supérieure de la Figure, on indique le nombre de configurations correspondant à des traitements différents pour chaque ressource. On constate que la ressource 1 effectue toujours le même traitement alors que la ressource 3 utilise 3 configurations distinctes.

Avec le mécanisme actuel (partie inférieure de la Figure), l'application doit être découpée en 3 phases de manière globale. Les commandes INIT_WRITE doivent se propager de ressources en ressources et imposer des changements de configurations. En effet, les ressources ne peuvent pas générer en interne une commande INIT_WRITE. Donc, pour pouvoir faire les changements de configurations au niveau de la ressource 3, il faut que les commandes INIT_WRITE soient générées au niveau de la ressource 1. Ainsi la ressource 1 doit changer de configuration 3 fois de suite. Cela peut poser problème si la taille des blocs de données n'est pas un multiple de 3. Il est par contre possible de bloquer les commandes INIT_WRITE, comme cela est fait pour la ressource 5.

Pour faire évoluer ce mode de programmation, un axe d'étude pourrait être le

remplacement du mécanisme de chargement des configurations basé sur la commande INIT_WRITE par un système de séquencement programmable au niveau de l'interface. On pourrait également rendre indépendamment la configuration de l'unité de traitement et des OCC.

2.5 Conclusion

Ce second chapitre nous a permis de détailler le contexte des systèmes de télécommunications des projets MATRICE et 4MORE. Nous nous sommes en particulier intéressé à l'organisation des chaînes de modulation et démodulation en bande de base au niveau de la couche physique afin de mettre en avant les problématiques d'implémentation matérielle de ces systèmes en ce qui concerne les communications et les paramètres de configuration. Nous avons ensuite présenté l'architecture du réseau sur puce FAUST. Cette architecture sert de plate-forme pour intégrer les unités de traitement d'un modem 4G. Le choix d'un NoC se justifie surtout par les contraintes importantes en bande-passante et les nombreux flots de données à gérer en parallèle. Pour finir nous avons abordé les différents enjeux liés à la conception d'un NoC dans le contexte de FAUST.

Les chapitres suivant vont être consacrés à l'exposé des contributions de cette thèse. Nous verrons d'abord un environnement de modélisation pour les applications de télécommunication qui seront réalisées avec le circuit FAUST. Ensuite nous présenterons une architecture de contrôle et de configuration des traitements et des communications dans un système NoC qui constitue une amélioration par rapport aux solutions utilisées dans FAUST.

Chapitre 3

Modélisation et analyse des performances pour les réseaux sur puce

Dans le premier chapitre nous avons présenté les architectures de réseau sur puce dans un cadre général. Nous avons vu quels avantages elles apportent mais également quelles sont les nouvelles problématiques auxquelles il faut répondre. Au cours du second chapitre, nous nous sommes recentrés sur une architecture de réseau plus spécifique. Il s'agit du projet FAUST qui a pour objectif de proposer une plate forme d'intégration pour des systèmes de télécommunication sans-fil.

Dans le chapitre que nous abordons à présent, nous allons présenter quelles approches nous avons suivies et quels outils logiciels nous avons mis en oeuvre pour proposer un environnement de modélisation bien adapté au réseau FAUST et aux applications qui lui sont associées.

Ce chapitre comporte cinq sections. Dans un premier temps, nous nous intéresserons à la modélisation des réseaux sur puces et nous exposerons les choix que nous avons effectués lors de nos travaux. Les deux sections suivantes seront consacrées aux environnements de simulation développés d'une part avec l'outil NS-2 et d'autre part avec le langage SystemC. Nous verrons ensuite quels types de simulation ont été réalisées. Pour finir nous présenterons et analyserons les résultats obtenus.

3.1 La modélisation des NoC

3.1.1 Objectifs des travaux de modélisations

Au cours du chapitre 1, nous avons identifié la nécessité de pouvoir anticiper, dans le flot de conception, les performances d'une architecture de réseau sur puce afin de déterminer au plus juste un certain nombre de paramètres d'implémentation. Différents critères ont été exposés plus précisément au paragraphe 1.4.1.

Dans le chapitre 2 nous avons introduit l'architecture de réseau FAUST conçu pour l'implémentation des traitements en bande de base de la couche physique d'applications de télécommunication.

L'objectif est maintenant de proposer une méthode pertinente permettant de montrer que la structure de communication FAUST supporte bien les échanges de données générés lors de l'exécution d'un traitement répartie sur plusieurs ressources. Nous considérons donc que d'une part nous avons une architecture de NoC prédéfinie avec un certain nombre de paramètres que l'on souhaite ajuster et d'autre part une chaîne de traitement fonctionnelle. La problématique apparaît alors lorsque l'on réunit ces deux entités. En particulier, il ne s'agit donc pas d'étudier une structure de réseau sur puce dans le cas général mais de valider une implémentation concrète d'un système de traitement.

3.1.2 Complexité liée au réseau sur puce

Dans une architecture de réseau sur puce telle que FAUST, chaque ressource de traitement a un comportement qui lui est propre. Celui-ci peut paraître localement simple à appréhender. Cependant la complexité globale augmente rapidement avec le nombre de ressources du fait des interactions qui existent entre elles. L'analyse statique reste limitée. De plus, un certain nombre de paramètres dont dépendent les performances restent difficile à déterminer :

- De paramètres du réseau

Ce sont des paramètres qui interviennent au niveau de l'organisation structurelle (topologie) du réseau. Il s'agit de déterminer le nombre de noeuds et de liens ainsi que la position des différentes ressources. Le nombre de noeuds et de liens permet d'augmenter la bande-passante. On peut par exemple doubler les liens pour supporter un débit plus important. La contrepartie est le coût matériel. Pour une topologie de réseau donnée, la position des ressources de traitement va aussi avoir une influence sur les

performances. Pour réduire la latence, on peut rapprocher les ressources qui communiquent fréquemment entre elles. Il faut aussi éviter de former des zones de congestion. Pour cela, une bonne répartition du trafic sur l'ensemble des liens de communication est souhaitable.

- De paramètres des ressources

Chaque unité de traitement communique grâce à une interface réseau (cf. paragraphe 1.2.2.3). Avec l'architecture FAUST, ces interfaces intègrent des mémoires FIFO qui rendent l'unité de traitement indépendante des mécanismes de transferts (cf. paragraphe 2.3.3). Tout se passe comme si la communication se faisait par une connexion directe entre les unités au travers d'une mémoire partagée. La façon d'organiser les transferts est directement liée au nombre de mémoire. Plusieurs mémoires permettent des transferts en parallèle, au contraire une unique mémoire en entrée ou en sortie de l'unité impose de séquencer les communications. Il faut également déterminer la taille de ces mémoires. Plus la taille est grande, plus les transferts sont fluides. Cet espace de stockage permet aussi d'anticiper les transferts ce qui a pour effet de masquer les latences. Cependant, que ce soit au niveau du nombre ou de leur taille, ces mémoires ont un coût matériel, il est donc impératif d'optimiser ces paramètres, pour chaque ressource, en fonction des contraintes du système que l'on va réaliser.

- De paramètres de communication

Le protocole de communication de FAUST offre des degrés de liberté pour la configuration des communications. Cela concerne en particulier la taille des paquets. Dans le cas d'une commutation de paquet de type *wormhole*, les paquets avec peu de flits sont préférables car ils occupent les noeuds sur une période plus courte ce qui réduit la congestion. Cependant, chaque paquet possède un flit d'en-tête. Multiplier le nombre de paquet en utilisant une granularité plus fine augmente donc le trafic sur le réseau. Le seuil d'envoi des crédit est un second paramètre qui intervient sur les performances des transferts (cf. paragraphe 2.3.3.2). Les crédits doivent être envoyés régulièrement pour fluidifier les transferts sans pour autant générer un trafic trop important. Enfin, il est également possible d'agir sur les chemins de routage ou le choix du canal virtuel utilisé. En tenant compte des contraintes des applications visées, il faut donc se donner des outils pour ajuster ces paramètres.

3.1.3 Nécessité d'un environnement de modélisation et synthèse des travaux antérieurs

Une analyse statique des flots de données permet d'évaluer l'état moyen du système mais entraîne un lissage des comportements dynamiques. Dans la section précédente, nous avons identifié des paramètres sur lesquels il est nécessaire de travailler pour que les unités de traitement exploitent au mieux le réseau de communication. Dans ce but, le recours à un environnement de modélisation apparaît comme nécessaire.

Dans la suite, nous allons présenter un aperçu de travaux antérieurs sur la modélisation des NoC. Il est possible de distinguer deux niveaux de modélisation dont les objectifs sont complémentaires. Le premier niveau consiste a modéliser la structure d'interconnexion. Le second prend en charge le trafic réseau.

3.1.3.1 Modélisation de la structure de communication

Il s'agit de modéliser le fonctionnement du réseau de communication indépendamment des unités de traitement qui vont exploiter ce réseau. Plusieurs approches sont possibles :

- Utiliser un environnement de modélisation haut-niveau

L'objectif est de développer rapidement un modèle fonctionnel sans avoir le niveau de détail nécessaire à une implémentation physique. Ces environnements fournissent un ensemble de primitives permettant de construire un système et de le simuler. On peut citer des travaux réaliser avec PtolemyII [PTOL06] comme [SSMK01],[ZhMa02] ou avec SDL [SpDL06] (*Specification and Description Language*) pour [HoHK03] et [AnKu04]. Globalement, ces environnements sont moins utilisés car ils sont peu connus dans le milieu de la conception.

- Utiliser un outil de modélisation de réseau

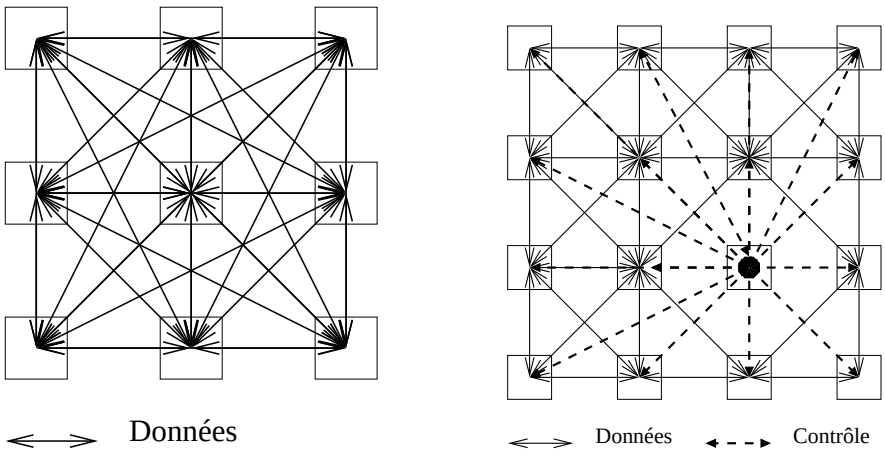
Une solution pour simuler une architecture de réseau sur puce est de reprendre des outils standards développés pour la modélisation de réseaux non-intégrés. Plusieurs simulateurs de réseaux existent et il est intéressant d'étudier comment ils peuvent être adaptés aux problématiques des NoC. Cette démarche a été mené avec l'outil OPNET [OPNE06] par [XWHC04] et [BCGK04]. OPNET fournit un environnement de simulation rapide, basé sur de la communication par paquet. Un autre simulateur utilisé pour la recherche sur les réseaux est NS-2 [NS206]. Nous présentons en détails cet outil en Annexe C. La première utilisation de NS-2 dans le cadre des NoC est publiée dans

[SuKJ02]. Par la suite, d'autres travaux ont été présentés [HEMK05] et [NgCh05]. Que ce soit avec OPNET ou NS-2, ces approches permettent une exploration rapide de différentes topologies de réseau. Les modèles sont ajustés pour donner de bons ordres de grandeurs au niveau des latences et des débits.

- Développer un modèle spécifique

L'approche la plus souvent choisie pour simuler le fonctionnement d'un réseau sur puce est de développer un modèle spécifique de l'architecture que l'on vise à partir de langages de description ou de programmation standards dans le but d'avoir un modèle précis au niveau cycle ou même bit. Dans cet optique, il existe des propositions de modèles VHDL haut-niveau comme [STNu02] et [BaVC04] qui ont l'avantage de pouvoir être aisément raffinés pour devenir synthétisables. Pour avoir une vitesse de simulation plus importante, certains NoC sont modélisés en langage C++ comme pour [WiSL04] et [HuMa04]. De plus en plus de travaux présentent des solutions basées sur le langage SystemC (cf. Annexe B). Des méthodologies complètes sont mises en place spécifiquement pour l'étude des NoC comme [KDWG03] ou [PRRG04] pour le réseau Æthereal (cf. paragraphe 1.3.2) ou [CCGM04] avec OCCN (*On-Chip Communication Network*).

3.1.3.2 Modélisation du trafic réseau



(a) Traffic aléatoire (b) Traffic local et contrôle

FIG. 3.1 – Modélisation du trafic de donnée sur un NoC

Nous avons vu dans le paragraphe précédent différentes méthodes pour modéliser un réseau en tant que structure de communication. Il s'agit maintenant de simuler des échanges de données. Dans ce but, deux approches sont possibles :

- Évaluer les performances du NoC

Dans ce type d'étude, on cherche à caractériser le réseau. Les résultats sont présentés de manière statistique. Il s'agit par exemple de la latence moyenne en fonction de la charge du réseau. Pour obtenir de ce type de résultat, les modèles font appel à des générateurs de trafics aléatoires. Différentes distributions statistiques sont possibles pour modifier la répartition temporelle des paquets, leurs tailles, leurs destinations. Le modèle le plus simple consiste en une equirépartition des paquets (Figure 3.1(a)), ce qui correspond à un réseau où les ressources ont toutes des fonctions équivalentes. Dans [SuKJ02], la répartition est biaisée de manière à ce que 75% du trafic reste local (proche voisin du point de vue de la topologie). Dans [WiSL04], on distingue un traffic de donnée fréquent et local, correspondant aux communications entre ressources dans un circuit de traitement, auquel on associe un trafic orienté vers certaines ressources spécifiques qui jouent le rôle de systèmes de contrôle (Figure 3.1(b)).

- Étudier le NoC dans un cadre applicatif

Les méthodes de génération de trafic aléatoire permettent de caractériser une architecture de réseau mais ne valident pas une application. En fait, il existe encore assez peu d'architectures NoC qui ont été présentées dans un cadre applicatif précis (cf. paragraphe 1.4.3). La solution [XWHC04] pour un système de traitement vidéo consiste à générer des traces de communication pour chacune des ressources en les faisant fonctionner dans un réseau idéal . Ensuite ces traces sont réinjectées pour chacune des ressources lors de la simulation globale du réseau. Le réseau SoCBUS [WiLi05] a été simulé dans le cadre d'une application de télécommunication (station de base pour un système 3G). Les unités de traitement sont modélisées fidèlement au niveau des communications sans pour autant réaliser l'intégralité des calculs.

3.1.4 Proposition d'une nouvelle approche pour la modélisation des NoC

Nous avons montré dans les paragraphes précédents les difficultés spécifiques liées à la modélisation et à l'évaluation d'une architecture de réseau sur puce. Dans le cadre du projet FAUST, nous nous proposons de mettre en place un environnement de modélisation spécifique aux applications visées.

Pour modéliser le réseau de communication, nous avons choisi deux approches :

– Une modélisation haut-niveau. Cette modélisation est basée sur le simulateur de réseau NS-2. Il nous a paru intéressant d'utiliser cet outil largement répandu

dans le milieu de la recherche sur les réseaux informatiques car il offrait un environnement de simulation gratuit, modulaire (programmation orientée objet) et personnalisable (possibilité d'éditer le code source et de recompiler le coeur du simulateur). Nous réduisons ainsi le temps de développement par rapport à un simulateur complet. Cependant, contrairement aux travaux cités précédemment (paragraphe 3.1.3.1), nous allons modifier certaines briques de base du simulateur de manière à mieux décrire les mécanismes spécifiques du réseau FAUST et ainsi augmenter la précision du simulateur. Nous exposerons cette démarche dans la section 3.2.

– Une modélisation au niveau TLM. Pour le circuit FAUST, un environnement de simulation SystemC a été développé parallèlement à notre étude avec NS-2 (cf. paragraphe 2.4.1). Par la suite, nous avons donc repris ces travaux avec SystemC qui nous apportaient ainsi un second modèle pour le réseau de communication. Ce modèle est décrit à la section 3.3.

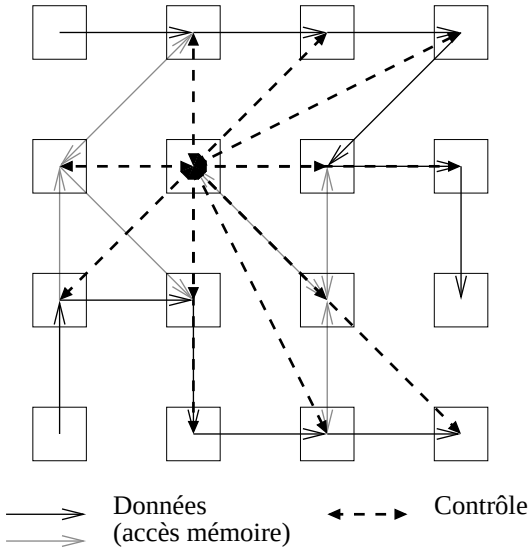


FIG. 3.2 – Exemple de flots de données à modéliser

En s'appuyant sur ces deux environnements de simulation du réseau, nous avons étudié les performances du réseau FAUST en modélisant les flots de données correspondant aux chaînes de modulation et démodulation MC-CDMA du projet MATRICE (cf. section 2.2). Les flots de données s'organisent comme un pipeline plus ou moins ramifié auquel se mêlent du trafic transverse qui correspond à des signaux de contrôle des ressources et à des accès aux ressources mémoires (Figure 3.2).

Nous ne cherchons pas à évaluer les performances du réseau FAUST dans un cas

général. Il ne s'agit pas non plus d'avoir une simulation complète dans les moindres détails du fonctionnement du circuit. En effet, nous voulons faire des choix de manière anticipée par rapport au flot de conception. Nous nous plaçons donc à des étapes où les détails d'implémentation de chacune des ressources ne sont pas encore connus ou figés. De surcroît, la complexité d'un simulateur complet et donc le temps d'exécution des simulation est un facteur limitant dans une phase d'exploration et d'étude de nombreux paramètres.

Nous avons donc opté pour une modélisation détaillée du trafic réseau sans opération réelle sur les données. En d'autres termes, la modélisation des communications est précise car les flots de données sont très proches de ceux du système final, mais la simulation est rapide à mettre en place et à exécuter car aucune donnée applicative n'est réellement traitée.

3.2 Réalisation d'un environnement de modélisation de réseau sur puce avec l'outil NS-2

Dans cette section, nous présentons les travaux de modélisation entrepris pour adapter le simulateur de réseau NS-2 [NS206] au réseau sur puce FAUST. Nous exposons d'abord notre démarche puis nous détaillons successivement le modèle du réseau et celui d'une ressource de traitement. Les détails sur l'outil NS-2 sont présentés en Annexe C.

3.2.1 Objectifs de la modélisation NS-2

Le fonctionnement d'un NoC est basé sur un protocole de communication qui génère de nombreux événements dont les interactions sont complexes à analyser. Avec NS-2, nous souhaitons proposer un environnement de modélisation à événement discret au niveau des communications sans entrer dans le détail de la gestion des données. L'objectif est d'avoir un modèle précis au niveau des échanges de données sur le réseau en tenant compte des spécificités de l'application sans pour autant réaliser une simulation du système complet. Le simulateur NS-2 permet de construire un modèle à partir d'une bibliothèque d'éléments préexistants tout en offrant une structure ouverte permettant la modification ou l'ajout de nouveaux modules. Les travaux avec NS-2 s'articulent autour de trois axes :

- Application. Dans un système NoC, on regroupe sur une même structure de communication des unités de traitement aux comportements hétérogènes. Pour uniformiser

la programmation de chaque unité tout en rendant compte des comportements variés, nous souhaitons donc de développer une unité de traitement générique et paramétrable.

- Communication. Il s'agit de mettre en place un système permettant de modéliser le protocole de communication sur le réseau.

- Outil. L'objectif est de montrer comment l'outil NS-2 conçu à l'origine pour des réseaux non-intégrés peut être exploité dans le cadre d'une architecture de réseau sur puce.

La démarche de modélisation consiste à construire un modèle de réseau proche d'une implémentation possible en gardant à l'esprit le respect de contraintes de topologie, de latence, d'espace mémoire ou de complexité. A partir de cette structure d'interconnexion, il s'agit ensuite d'établir des scénarios d'échanges de données représentatifs des applications visées (en particulier la chaîne de traitement MATRICE présentée au paragraphe 2.2.2). La modélisation se base sur trois composants clés :

- Le réseau. C'est la structure d'interconnexion et le support d'échange des informations. La mise en place du réseau répond à l'objectif *Outil*.

- Les interfaces réseaux. Elles permettent l'utilisation du réseau par les unités de traitement en faisant abstraction du protocole (cf. 1.2.2.3). Les interfaces réseaux correspondent à l'objectif *Communication*.

- Les unités de traitement. Ces modules vont générer les flux de données en fonction de l'application modélisée (objectif *Application*).

3.2.2 Développement d'un modèle de réseau FAUST avec NS-2

Dans cette section, nous allons présenter comment nous avons mis en place une modélisation du réseau FAUST à partir de l'environnement NS-2. De manière général, les composants réseaux ont pu être adaptés à l'architecture FAUST sans faire de modification majeur sur le code source. Il a par contre été nécessaire de créer de nouveaux Agents et Applications pour modéliser les ressources. Il faut noter également que NS-2 fonctionne avec des temps caractéristiques de réseau non-intégrés de l'ordre de la milliseconde (ms) contrairement à un NoC qui travaille au niveau de la nanoseconde (ns) pour des fréquences de plusieurs dizaines de MHz. Au niveau du paramétrage des composants NS-2, on va donc dilater le temps en utilisant les correspondances suivantes : 1ms équivaut à 1ns et 1Gbps équivaut à 1kbps.

3.2.2.1 Topologie

La topologie du réseau FAUST est de type maillée à deux dimensions. A chaque noeud du réseau est associé une ressource de traitement. En utilisant les structures de type *node* et *link*, on peut aisément construire de telles topologies avec NS-2. La création de la topologie se fait par un script OTcl. Pour l'instanciation des noeuds on utilise des objets de type *node*. Pour les ressources, on utilise également un objet de type *node* auquel on associe un ou plusieurs Agents ainsi qu'une Application. La Figure 3.3 représente un exemple de topologie de réseau visualisé avec l'outil NAM associé à NS-2 (cf. Annexe C). Les noeuds « ronds » représentent les noeuds du réseau et les noeuds « carrés » sont les points d'accès des ressources.

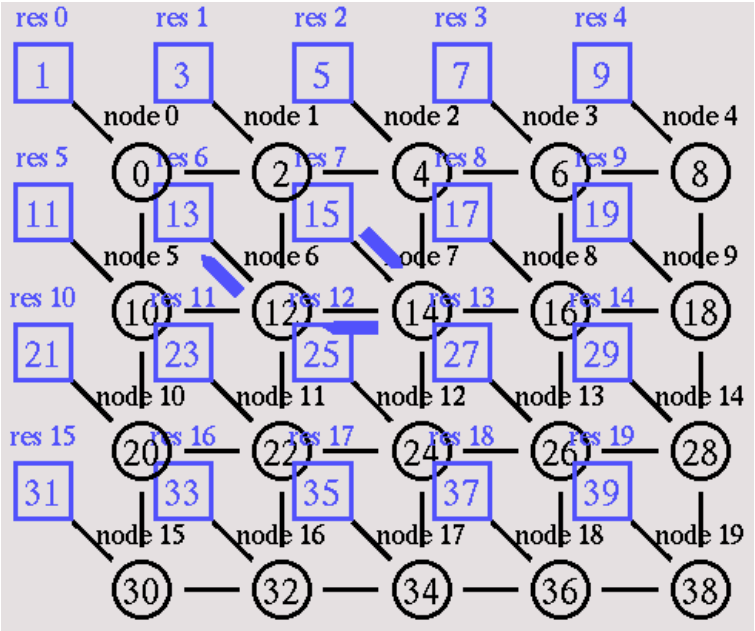


FIG. 3.3 – Topologie du réseau FAUST modélisé

Les noeuds sont reliés entre eux par des liens NS-2 bidirectionnels. La bande-passante est fixé à 3,2Gbps dans chaque direction. Cette bande-passante correspond au transfert d'un flit de 32 bits toutes les 10ns (fréquence de fonctionnement de 100MHz).

3.2.2.2 Mécanismes de communication

FAUST fonctionne en mode non-connecté par paquet. NS-2 est bien adapté à ce mode de communication puisqu'il intègre la notion de paquet. Dans un mode non-

connecté, chaque paquet contient l'ensemble des informations pour arriver à destination. Ces informations sont stockées dans un en-tête qui contiennent des données de routage. NS-2 propose différente structure d'en-tête qu'il est possible d'assembler pour former l'en-tête d'un paquet. Nous utilisons les structures suivantes :

- L'en-tête *hdr_cmn* (*common header*). Cet en-tête est obligatoire au fonctionnement du simulateur. Il permet en particulier de spécifier la taille du paquet en nombre d'octet.
- L'en-tête *hdr_ip*. Cet en-tête est utilisé pour le routage du paquet. Il contient les champs d'adresses source et destination.
- L'en-tête *hdr_noc*. Il s'agit d'un nouvel en-tête permettant de stocker les paramètres spécifiques à FAUST pour distinguer par exemple les données des crédits.

En ce qui concerne l'algorithme de routage, nous utilisons le routage statique utilisé par défaut dans NS-2. Un mécanisme de routage source permet de modifier manuellement les chemins de routage lorsque ceux proposés par l'algorithme ne conviennent pas.

3.2.2.3 Mode de commutation

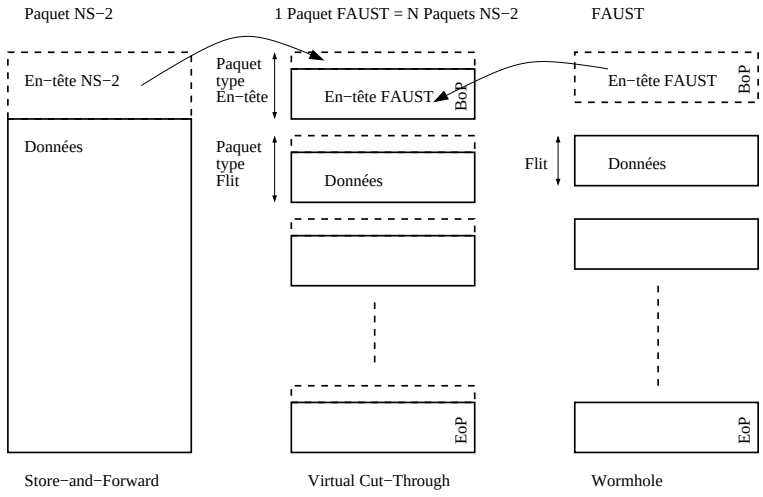


FIG. 3.4 – Mode de commutation NoC et NS-2

FAUST utilise un mode de commutation *wormhole*. Les paquets sont découpés en flits qui circulent d'un noeud à l'autre (cf. paragraphe 1.2.1.2). Dans NS-2 la commutation est de type *store-and-forward*. Un noeud reçoit la totalité d'un paquet avant de le transmettre au noeud suivant. Nous avons donc fait un compromis pour tenir compte

de cette différence. Une analyse des délais de transmission permet de proposer un mode de commutation de type *virtual cut-through*.

En effet, avec une commutation *wormhole*, le délai de transmission d'un paquet ($delai_{worm}$) de n_{flit} flits entre n_{lien} liens est donné par l'équation 3.1.

$$delai_{worm} = n_{lien} \times T + (n_{flit} - 1) \times T \tag{3.1}$$

T est le temps de passage d'un flit dans un noeud. Ainsi le premier terme de la somme correspond au temps pour que le premier flit traverse les n_{lien} liens et le second terme le temps pour que les flits restants arrivent à destination.

Dans le simulateur NS-2, le délai ($delai_{ns2}$) d'un paquet de taille $taille_{paquet}$ est fonction des paramètres des liens. Il s'agit du délai du lien ($delai_{lien}$) et de la bande-passante du lien (bp_{lien}). On obtient donc l'expression 3.2.

$$delai_{ns2} = n_{lien} \times delai_{lien} + n_{lien} \times \frac{taille_{paquet}}{bp_{lien}} \tag{3.2}$$

Or par définition de T, la bande-passante correspond à 3.3.

$$bp_{lien} = \frac{taille_{flit}}{T} \tag{3.3}$$

On obtient alors 3.4.

$$delai_{ns2} = n_{lien} \times delai_{lien} + n_{lien} \times n_{flit} \times T \tag{3.4}$$

Pour obtenir des délais égaux nous réglons le paramètre $delai_{lien}$ à 0 dans NS-2 et nous utilisons les paquets NS-2 pour modéliser les flits FAUST. Ainsi avec $n_{flit} = 1$, le délai d'un paquet d'un flit dans NS-2 devient 3.5

$$delai_{ns2} = n_{lien} \times T \tag{3.5}$$

Il faut ensuite envoyer les paquets NS-2 avec une période T. Nous modélisons ainsi les paquets FAUST comme un ensemble de paquets NS-2 (Figure 3.4). En l'absence de contention, ces deux modes sont équivalents, notamment au niveau des latences. Si des conflits entre paquets se produisent, les comportements sont différents mais l'approximation reste acceptable, le débit global n'est pas modifié. Chaque lien est équipé d'une

file d'attente de type *DropTail* qui se comporte comme une mémoire FIFO. La taille de cette file sera suffisamment importante en nombre de flit pour éviter que des flits (paquets NS-2) soient perdus pour cause de saturation. Cette taille est fixée à 10 flits lors des simulations.

3.2.3 Développement d'un modèle de ressource de traitement

Les composants proposés par NS-2 ne correspondent pas au protocole spécifique de NoC que nous voulons modéliser. Nous avons donc décidé de développer de nouveaux composants. Comme vu au paragraphe 3.2.2.1, une ressource correspond à un Noeud associé à un Agent connecté à une Application. Dans notre modèle, l'association Noeud-Agent correspond à l'interface réseau et l'application à l'unité de traitement (Figure 3.5).

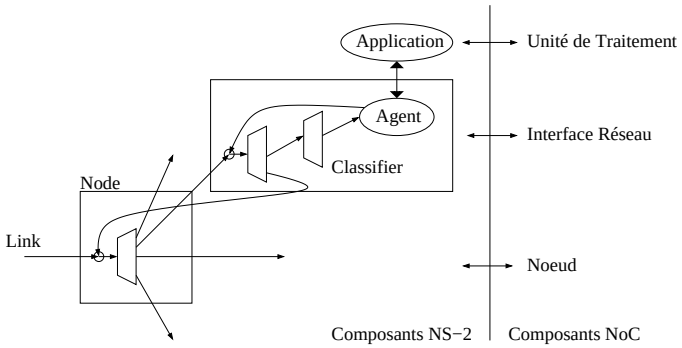


FIG. 3.5 – Modèle NS-2 de l'architecture FAUST

3.2.3.1 Développement d'un modèle d'interface réseau

L'interface réseau est un élément essentiel au fonctionnement du réseau FAUST. Ce système prend en charge toute la gestion du protocole de communication. Un nouvel Agent a été développé pour gérer ces mécanismes (classe *AgentNoC*). L'Agent reçoit des paquets, extrait des données applicatives et les transmet à l'Application. A l'inverse lorsque l'Application désire émettre des données, elle les fournit à l'Agent qui se charge de les mettre au format réseau sous forme de paquets. Ces flux de données entrant et sortant sont régulés par des mémoires de type FIFO. Plusieurs paramètres permettent de configurer ces nouveaux Agents. Le principe est d'avoir un modèle d'Agent générique que l'on instancie pour chaque unité de traitement en le configurant de manière spécifique (Figure 3.6).

- Contrôle de flux

Pour éviter une saturation du réseau, le trafic est régulé au niveau des agents. Deux techniques de contrôle de flux sont utilisées. La première agit de manière statique. Pour chaque agent, on spécifie au moment de leur instanciation la période d'envoi des flits. Ainsi, en indiquant une période plus importante, on peut limiter le débit sur le lien sortant d'un agent. La seconde technique fonctionne de manière plus dynamique. Elle est basée sur le mécanisme de crédit utilisé dans FAUST (cf. paragraphe 2.3.2) et qui est modélisé par les *AgentNoC*. Une étude de ce mécanisme est présentée dans [LeLJ05].

- Configuration de l'agent

Tout d'abord, différents paramètres sont fixés à l'instanciation de l'agent. Ces paramètres sont indiqués dans un script OTcl. Il s'agit principalement de fixer la taille des mémoires FIFO et la période d'envoi des flits (pour le contrôle de flux). Une fois l'agent crée, il existe deux types de configuration qui peuvent être modifiées dynamiquement au cours de la simulation. Ces configurations sont transmises sous forme de paquets et reçues par l'agent. Il s'agit des configurations pour l'envoi des données (équivalent aux configurations OCC dans FAUST) : taille des paquets, chemin de routage, nombre total de données (cf. paragraphe 2.3.3.2). L'autre type est la configuration des crédits (comme pour les ICC) : seuil d'envoi des crédits, chemin de routage, nombre total de crédit. Un mécanisme de file d'attente permet à l'Agent de recevoir des configurations et de les stocker en attendant la fin de l'exécution de la configuration précédente.

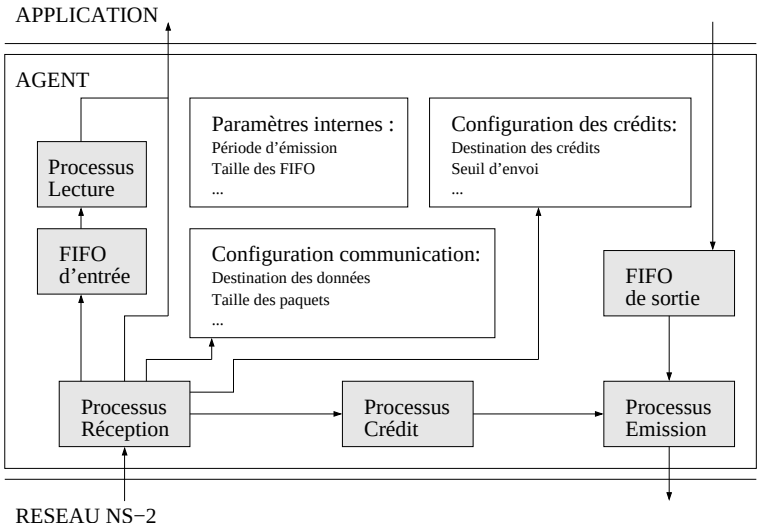


FIG. 3.6 – Structure de l'*AgentNoC*

3.2.3.2 Développement d'un modèle d'unité de traitement

Pour développer un modèle d'unité de traitement, nous utilisons la classe *Application*. Notre objectif n'est pas de réaliser les traitements réels des données mais de générer des trafics réseaux et de modéliser des latences de traitement.

- Principe de fonctionnement

La modélisation des traitements repose sur trois étapes : la réception de données (message) nécessaires au traitement, le temps de traitement et l'émission des données (message) produites par le traitement. Ces trois éléments sont paramétrables et sont ajustés de manière à obtenir un comportement proche de ce que serait une implémentation réelle de l'unité de traitement. La Figure 3.7 présente la structure d'un module Application.

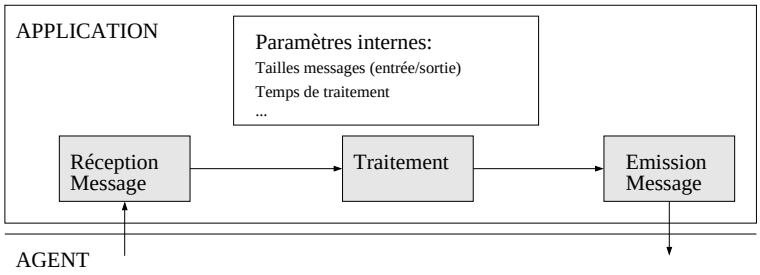


FIG. 3.7 – Structure d'un module Application

- Variantes spécifiques

Nous avons développé la classe *AppNoCstd* dérivée de la classe *Application* pour modéliser une unité de traitement. Deux variantes ont également été créées pour des comportements plus spécifiques. Il s'agit de :

- *AppNoCdma*. Ce module est utilisé pour modéliser les contrôleurs mémoire (RAM CTRL) utilisés dans FAUST. Nous les utilisons comme générateur de trafic. Il est possible de configurer la granularité d'envoi des données, la quantité totale de données à envoyer, l'intervalle de temps entre deux envois.
- *AppNoCequ*. Ce module permet de modéliser une unité de traitement qui reçoit simultanément deux flux de données. Il est utilisé pour la modélisation de l'unité d'Égalisation (EQU) dans la chaîne de réception MC-CDMA. Ce module fonctionne avec deux instances de la classe *AgentNoC*.

3.3 Modélisation d'une application basée sur le modèle SystemC du réseau FAUST

Comme expliqué au paragraphe 3.1.4, notre démarche de modélisation des NoC est basée sur deux environnements : l'outil NS-2 que nous venons de présenter et un modèle TLM/SystemC qui a été réalisé dans le cadre du projet FAUST (cf. paragraphe 2.4.1) et qui fait l'objet de cette nouvelle section.

3.3.1 Objectifs des travaux

Dans le cadre de la thèse, il était intéressant d'exploiter le modèle TLM/SystemC de FAUST en complément des travaux préalablement effectués avec NS-2. Ce modèle, présenté au paragraphe 2.4.1, permet de réaliser des simulations précises aux niveaux des mécanismes de communication du réseau. A partir de cette structure de communication, l'objectif était de construire un modèle équivalent à NS-2 au niveau de l'application en reprenant les même principes de modélisation. En pratique, les trafics de données générés sont semblables entre le deux modèles, la différence se faisant au niveau de la modélisation du réseau.

Nous pouvons ainsi avoir une comparaison entre les résultats obtenus avec les modèles NS-2 et SystemC. Ceci doit permettre de valider que les approximations faites avec NS-2 au niveau des mécanismes du réseau n'entraînent pas de trop grandes variations sur les résultats de performance donnés par la simulation.

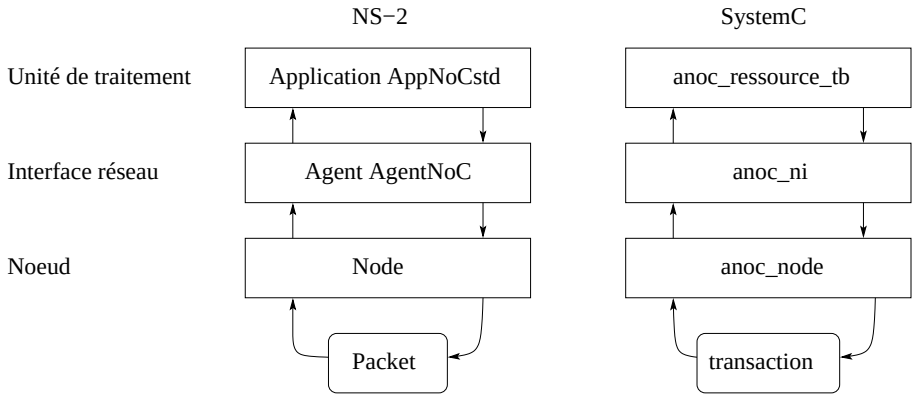


FIG. 3.8 – Parallèle entre la modélisation NS-2 et SystemC

Du point de vue structurelle, nous pouvons établir un parallèle entre les modules

utilisés avec NS-2 et avec SystemC (Figure 3.8).

3.3.2 Modélisation d'une unité de traitement générique

Un modèle d'unité de traitement (*anoc_ resource_ tb.h/.cpp*) a été développé en SystemC. Il exploite le modèle d'interface réseau (*anoc_ ni*) précédemment présenté (cf. paragraphe 2.4.1.2). La connexion entre l'unité de traitement et l'interface réseau se fait par l'intermédiaire de mémoires FIFO (*sc_fifo*). Les données sont synchronisées grâce aux mécanismes de lectures et d'écritures bloquantes. De la même manière que pour NS-2, nous modélisons seulement des échanges de données et nous n'implémentons pas le détail des calculs sur les données. Le modèle est générique et il est configuré pour chaque ressource. Les paramètres sont la taille des messages en entrée, en sortie et le temps de traitement. On retrouve ainsi la même structure que celle illustrée Figure 3.7. Il est également possible de choisir un mode de fonctionnement avec deux mémoires FIFO en entrée de manière à recevoir deux flux en parallèle. Un modèle dérivé (*anoc_ resource_ gen.h/.cpp*) de cette unité de traitement est utilisé pour réaliser des générateurs de trafics de données sur le réseau. Ces générateurs de données sont programmables et ils peuvent séquencer plusieurs configurations au niveau de l'interface réseau pour orienter les flux de donnée vers différentes destinations.

3.3.3 Création d'un modèle de réseau

Le réseau est créé en assemblant les différents noeuds et les ressources de traitement par l'appel de fonctions de connexion entre modules dans un fichier principal. Un modèle de processeur (*anoc_ resource_ cpu.h/.cpp*) associé à une interface réseau spécifique (*anoc_ ni_ cpu*) permet d'envoyer sur le réseau des commandes de configuration pour les interfaces réseau et de contrôle tels que les commandes démarrage pour les ICC (cf. paragraphe 2.4.3).

3.4 Modélisation d'une chaîne de traitement en bande de base pour la modulation et démodulation de type MC-CDMA

Maintenant que nous disposons de deux modèles, nous allons pouvoir les utiliser dans le cadre de l'intégration d'un système applicatif en particulier. Nous nous intéressons

à la modélisation du trafic généré dans un circuit implémentant un modem bande de base sur une architecture NoC de type FAUST (cf. paragraphe 2.3.4). Nous modélisons les ressources de traitement utilisées dans les chaînes de modulation et démodulation MC-CDMA mises en place dans le cadre du projet MATRICE (cf. section 2.2). Le but est de vérifier que les fonctionnalités du réseau et ses performances sont compatibles avec les besoins de l'application.

3.4.1 Présentation de l'architecture modélisée

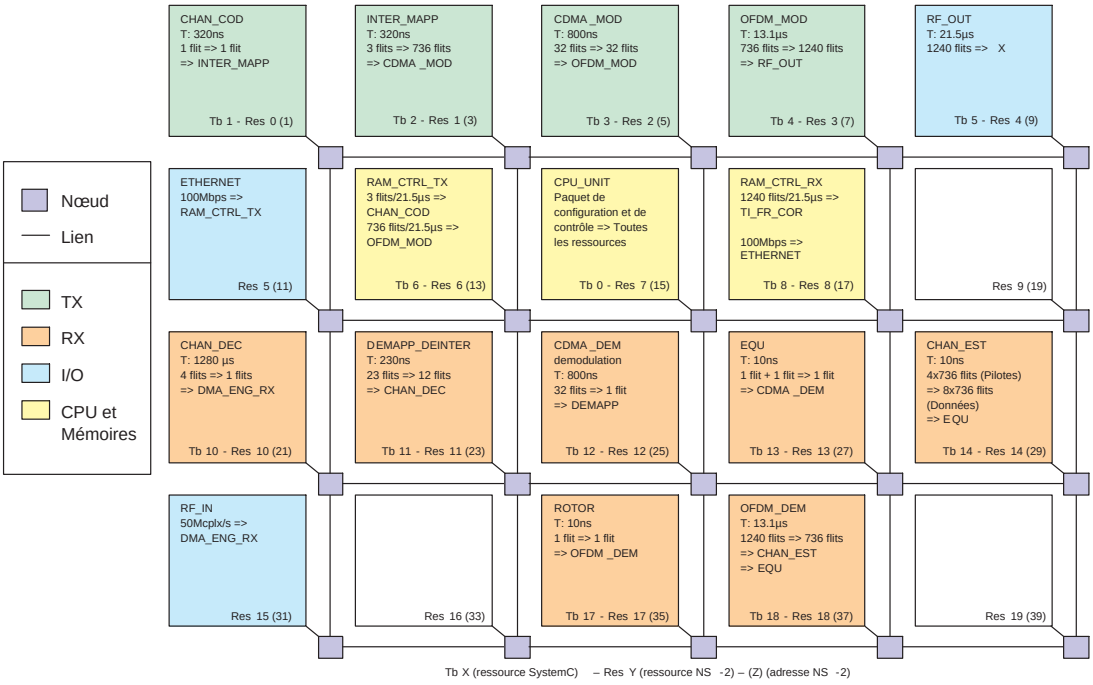


FIG. 3.9 – Architecture du réseau et ressources de traitement modélisées

Le NoC modélisé est un réseau composé de 20 noeuds et de 17 ressources. Cette architecture est proche du circuit FAUST présenté au chapitre 2. Il faut cependant noter que le systèmes MATRICE a subi des évolutions parallèlement à l'avancement de la thèse c'est pourquoi il existe certaines différences. Le système est géré par une ressource processeur (CPU_UNIT) qui envoie des paquets contenants des configurations et des commandes de contrôle. 2 ressources modélisent les contrôleurs mémoires. Les chaînes d'émission et de réception sont composées respectivement de 4 et 7 ressources. 2 ressources font l'interface entre le circuit et le front-end radiofréquence (RF_IN et

RF_OUT). La ressource ETHERNET permet de gérer des échanges de données avec les couches supérieures du système. La Figure 3.9 représente l'architecture modélisée. Pour chaque ressource, le temps de traitement, le nombre de flit consommés et produits à chaque traitement et la destination des flits produits sont indiqués.

TAB. 3.1 – Paramètres de modélisation des unités de traitement

	Temps de traitement (en ns)	Taille des messages (en flits)		Nombre de traitements par trame
		Entrée	Sortie	
CHAN_COD	320	1	1	72
INTER_MAPP	320	3	736	24
CDMA_MOD	800	32	32	552
OFDM_MOD	13100	736	1240	28
RF_OUT	21500	1240	#	28
ROTOR	10	1	1	34720
OFDM_DEM	13100	1240	736	28
CHAN_EST	10	2944	17664	1
EQU	10	1	1	17664
CDMA_DEM	800	32	1	552
DEMAPP_DEINTER	230	23	12	24
CHAN_DEC	128	4	1	72

Nous ne détaillons pas la nature des traitements réalisés par chacune des ressources car cela n'a pas d'influence sur le fonctionnement du modèle. L'analyse des chaînes de modulation et de démodulation MC-CDMA mis en place dans le cadre du projet MATRICE nous a permis d'extraire des paramètres qui sont utilisés pour la configuration des ressources. Les paramètres de configuration de chaque unité de traitement sont récapitulés dans le Tableau 3.1.

3.4.2 Scénarios de simulation

Une fois l'architecture définie, il faut établir des scénarios de simulation. Ces scénarios doivent permettre de valider qu'il n'existe pas de blocages fonctionnels liés au protocole de communication et que le système respecte bien les contraintes temporelles de l'application. Nous nous focalisons sur des modes de fonctionnement spécifiques lorsque

le réseau est le plus sollicité. En effet, il n'est pas utile de simuler l'application dans son intégralité puisque durant certaines étapes le trafic est faible ou concerne peu de ressources. C'est pourquoi, entre autres, les phases de synchronisation du récepteur ne sont pas modélisées.

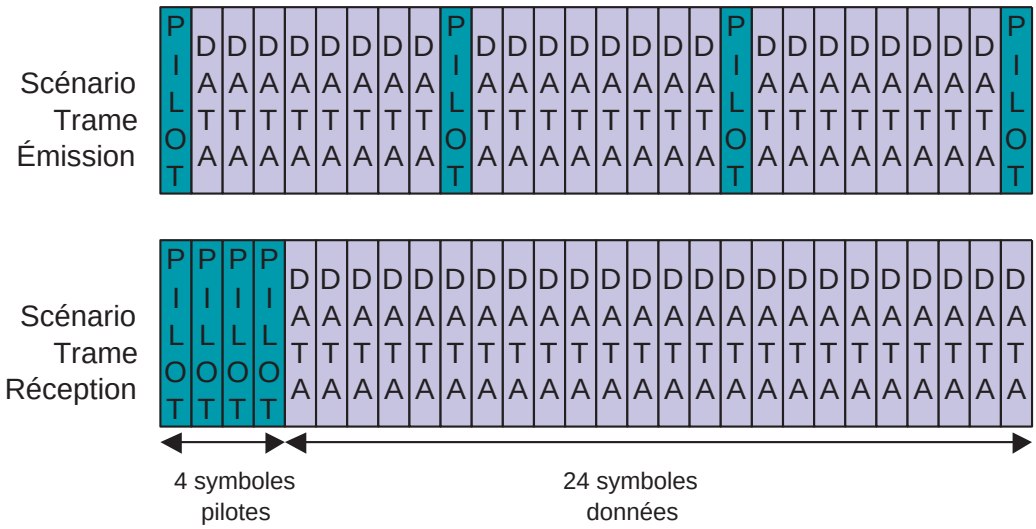


FIG. 3.10 – Structure des trames MC-CDMA modélisées en TX et RX

Nous avons choisi de simuler simultanément le trafic correspondant à l'émission et à la réception de trames de données. Le protocole de télécommunication associé au système MATRICE n'est pas prévu pour être *full-duplex* mais les deux modes peuvent cohabiter ponctuellement lorsque le modem commute d'une phase de réception à une phase d'émission et qu'il doit donc commencer les calculs sur la trame à émettre avant d'avoir terminé les traitements sur la trame reçue. Il s'agit d'une situation où le trafic de donnée est le plus important ce qui correspond bien aux objectifs recherchés.

Nous utilisons les structures de trame présentées sur la Figure 3.10. Une trame est composée de 28 symboles OFDM (24 pour les données utilisateurs et 4 pour les pilotes).

Pour l'émetteur, la structure de trame est identique à celle présentée pour MATRICE exeception faite des symboles de synchronisation. Le contrôleur mémoire (RAM_CTRL_TX) envoie alternativement les données correspondant aux pilotes vers la ressource de modulation OFDM (OFDM_MOD) et les données utilisateur vers la ressource codage canal (CHAN_COD). Ceci correspond aux flots **1** et **2** sur la Figure 3.11

En réception, on suppose que la trame a été reçue, synchronisée et stockée en mémoire (RAM_CTRL_RX), c'est pourquoi la position des symboles pilotes est diffé-

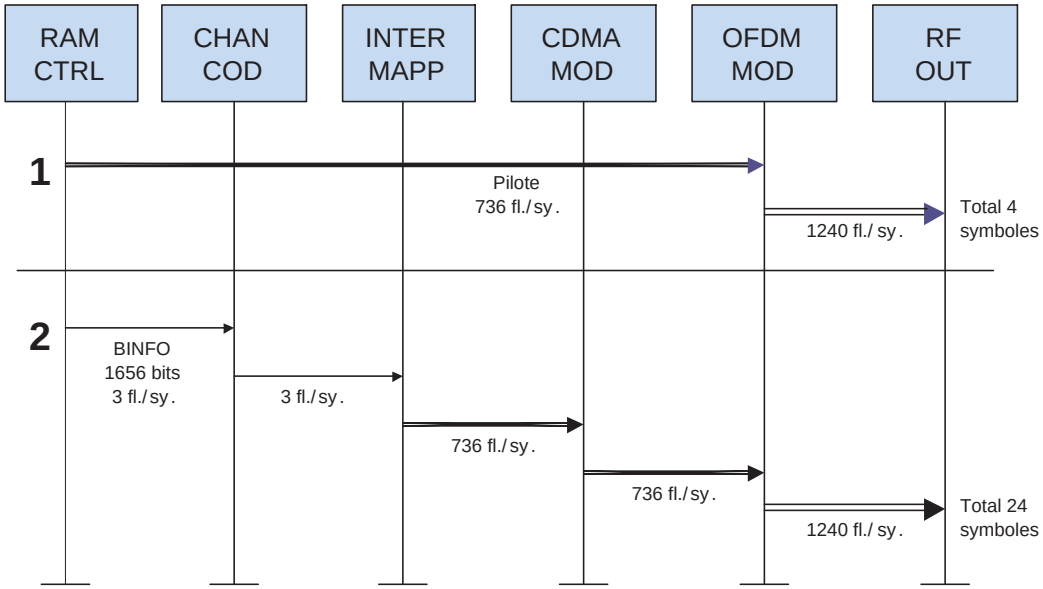


FIG. 3.11 – Flots de communication de la chaîne d'émission MATRICE

rentes par rapport à la structure de trame d'émission. Les données sont envoyées vers le démodulateur OFDM (flot **1** sur la Figure 3.12). Les 4 symboles pilotes sont envoyés vers l'estimateur de canal (CHAN_EST, flot **2**). L'ensemble des symboles de donnée sont ensuite démodulés en utilisant les coefficients de correction calculés à partir des pilotes (flot **3**).

En complément des flots de données pour les chaînes d'émission et de réception, nous simulons le transfert des échantillons numérisés vers la mémoire interne ce qui crée un flux à 1,6Gbps (fréquence d'échantillonnage à 50 MHz) entre les ressources RF_IN et RAM_CTRL_RX. Nous créons également un trafic à 100Mbps entre les ressources ETHERNET et RAM_CTRL_TX et RAM_CTRL_RX et ETHERNET ce qui correspond à un utilisateur qui envoie et reçoit des données.

Pour avoir un régime établi, il faut faire tourner la simulation sur un nombre suffisant de symboles. Il faut cependant trouver un compromis pour avoir un temps de simulation raisonnable. Le choix s'est donc porté sur la modulation et la démodulation de 2 trames complètes.

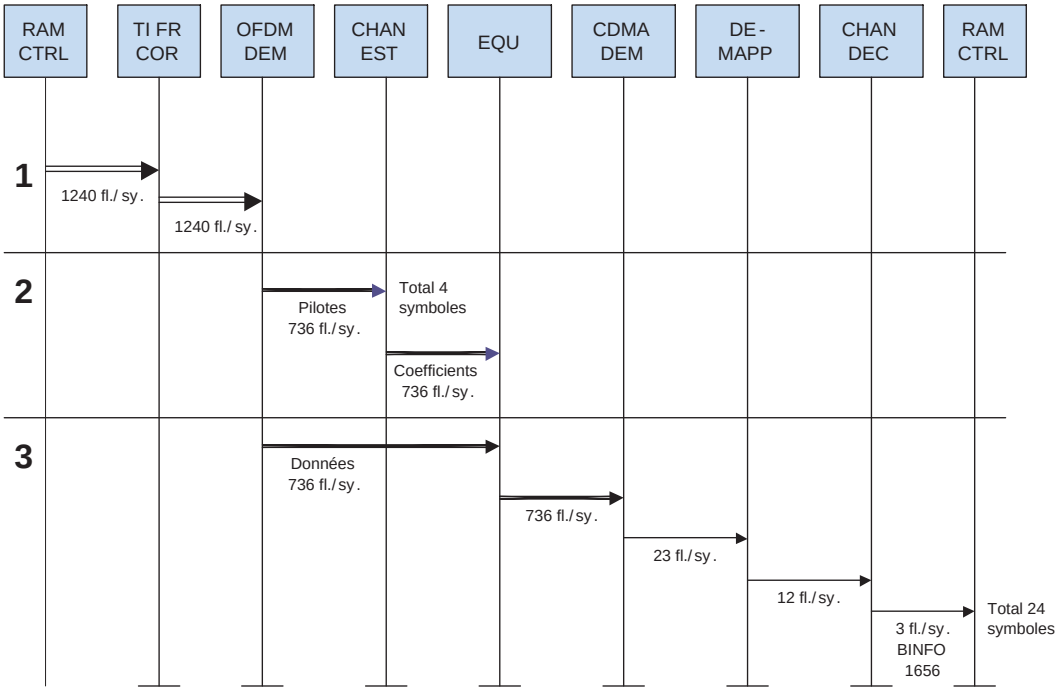


FIG. 3.12 – Flots de communication de la chaîne de réception MATRICE

3.5 Analyses des résultats obtenus avec les modèles NS-2 et SystemC et comparaison des deux approches

Dans cette section, nous allons présenter les résultats de simulation obtenus à partir des scénarios exposés précédemment. Nous verrons d'abord comment le modèle NS-2 nous permet de valider l'architecture avec un trafic réel correspondant à une chaîne de modulation et de démodulation MC-CDMA, ensuite nous comparerons les résultats obtenus avec les modèles NS-2 et SystemC. Dans une seconde partie, nous évaluerons les avantages et limitations liés à l'utilisation de NS-2 dans le contexte des NoC.

3.5.1 Résultats de simulation NS-2

Les résultats de simulation valident le bon fonctionnement de la chaîne de modulation et de démodulation MC-CDMA distribuée sur une architecture de NoC similaire au réseau FAUST. Le protocole de communication n'entraîne pas de blocage fonctionnel. Les fichiers de trace d'exécution montrent que toutes les données sont traitées et qu'aucun flit n'est perdu. Les paramètres de configuration sur la taille des paquets et le seuil

d'envoi des crédits sont bien compatibles avec la taille des mémoires FIFO des interfaces réseaux. En sortie de la chaîne d'émission, on peut vérifier que les symboles peuvent être émis sans interruption tout au long de la trame. Les contraintes temporelles sont respectées : en émission et réception, le système permet de traiter au moins un symbole OFDM par temps symbole (fixé 21,5 μs dans le cas de MATRICE).

- Débit global

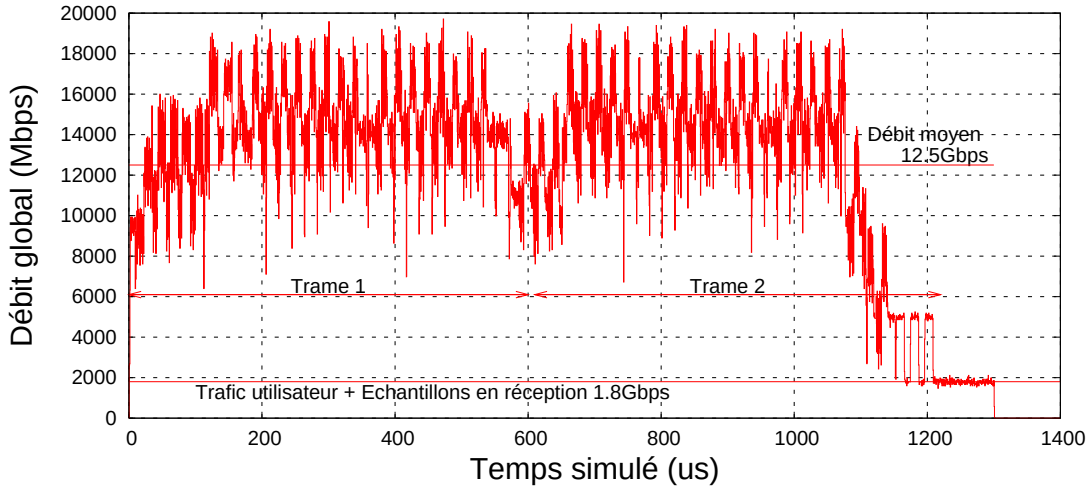
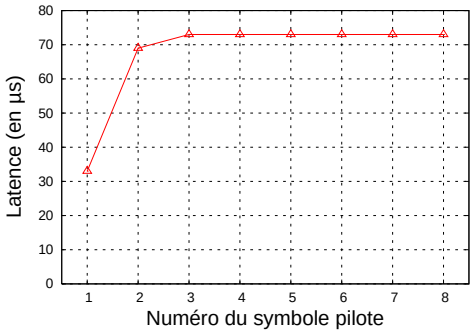


FIG. 3.13 – Débit de réception cumulé sur l'ensemble ressources (NS-2)

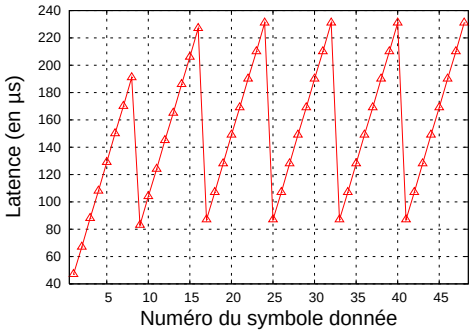
Nous avons mesuré le débit global sur le réseau au cours de la simulation. La méthode de calcul utilisée est présentée au paragraphe 1.4.1. Il est possible de connecter 20 ressources au réseau. Les liens ont une bande-passante de 3,2Gbps (cf. paragraphe 3.2.2.1), le débit global maximal est donc théoriquement de 64Gbps. Les résultats de simulation (Figure 3.13) montre un débit maximal de 20Gbps avec un débit moyen de 12,5Gbps sur un temps simulé de 1,3ms. Ces résultats donnent un bon ordre de grandeur du volume de données qui doit être échangé dans un tel système, ce qui justifie d'autant plus l'utilisation d'un NoC par rapport à un structure d'interconnexion plus classique de type bus.

- Latence symbole

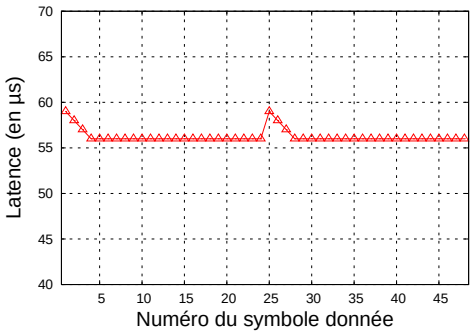
Pour caractériser le réseau, le temps de traitement pour chaque symbole a été mesuré. Cette latence correspond au temps écoulé entre l'envoi du premier flit du symbole en début de chaîne et la réception du dernier flit en fin de chaîne (ce raisonnement est valable en émission et en réception). Les latences sont supérieures au temps symbole



(a) Émission pilotes



(b) Émission données



(c) Réception données

FIG. 3.14 – Latence sur les symboles

mais ceci est normal dans un système en pipeline.

Pour l'émission, nous distinguons la latence pour l'émission des symboles de pilotes et des symboles de données (Figures 3.14(a) et 3.14(b)). La latence des symboles de données est plus longue que celle des symboles de pilotes car il y a davantage de ressources à parcourir. On constate que l'on n'atteint pas un temps de latence stabilisé puisque tous les 8 symboles de données, on intercale un symbole de pilote que ce qui entraîne une reconfiguration du pipeline.

Le temps de démodulation par symbole de donnée est présenté sur la Figure 3.14(c). La latence est légèrement supérieure pour les premiers symboles. Ceci est dû au temps de calcul des premiers coefficients d'égalisation par l'estimateur de canal. On constate que la latence symbole en réception est inférieure à celle en émission. On peut l'expliquer par le fait que la sortie de l'émetteur se fait à débit limité ce qui ralentit les ressources précédentes. Au contraire en réception, les données démodulées sont envoyées en mé-

moire sans limitation de débit. La chaîne de démodulation est donc cadencée à une vitesse supérieure. En fait, la cadence de fonctionnement est moins critique en émission puisqu'il n'est pas utile de produire des symboles plus rapidement que leur fréquence d'envoi alors qu'en réception il est intéressant de pouvoir démoduler les symboles le plus rapidement possible pour pouvoir exploiter les données qu'ils contiennent.

- Cadence des unités de traitement

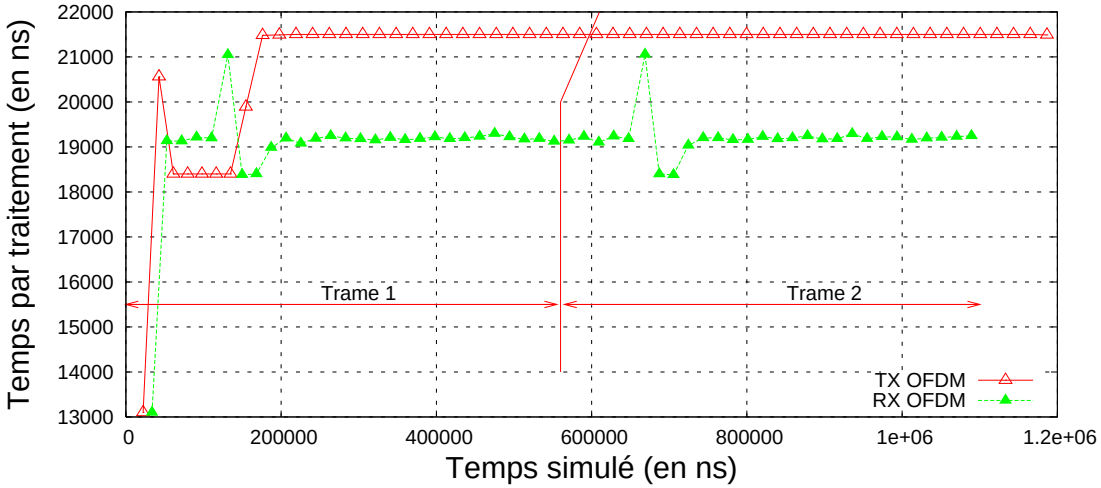


FIG. 3.15 – Temps par traitement pour les unités OFDM (émission et réception)

Le problème du dimensionnement de la puissance de calcul des unités de traitement a également été abordé. La Figure 3.15 présente les temps par traitement pour les unités OFDM (OFDM_MOD et OFDM_DEM, cf. section 3.4.1). Le paramètre temps de traitement du modèle est configuré à 13,1μs. Le temps par traitement correspond au temps écoulé entre deux traitement consécutifs. Ce temps inclut donc le temps de calcul mais également le temps pour transférer les donnée entre l'unité de traitement et l'interface réseau. On constate que le temps par traitement augmente puis se stabilise lorsque le pipeline est chargé. Les unités OFDM fonctionnent avec une granularité au niveau symbole. On vérifie donc que le temps par traitement est bien inférieur au temps d'un symbole ce qui garantie que le système respecte la cadence symbole.

3.5.2 Étude comparative entre les modèles NS-2 et SystemC

Le trafic de données correspondant à une chaîne de modulation et démodulation MC-CDMA a été simulé sur les modèles NS-2 et SystemC. Le scénario a été réduit à

la modulation et à la démodulation simultanée d'une trame MC-CDMA. Avec les deux modèles, le système est fonctionnel et les configurations des ressources sont validées.

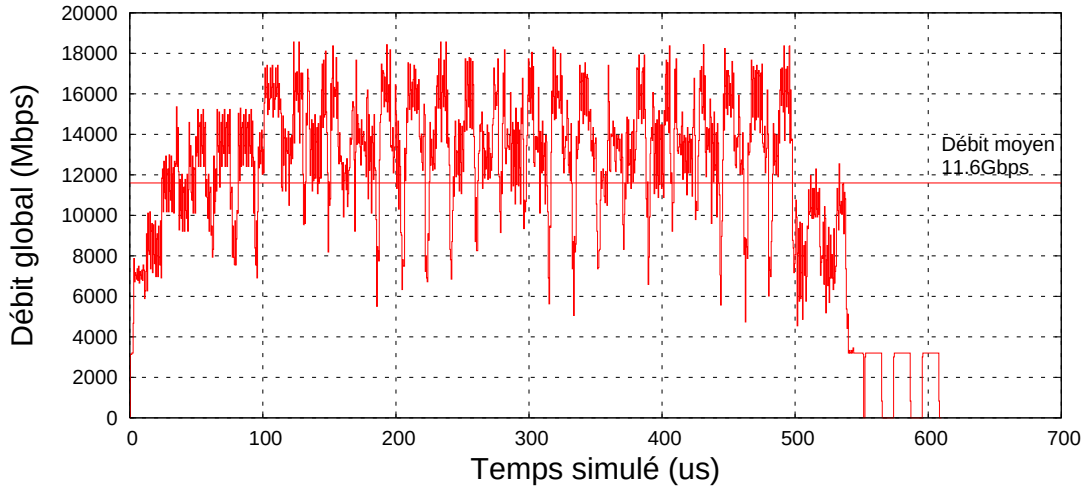


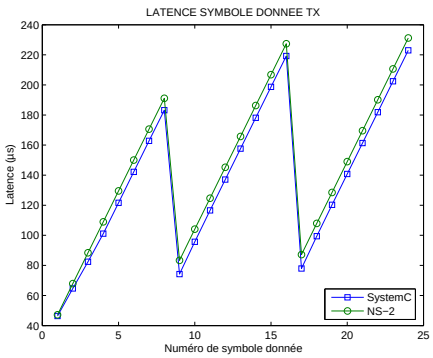
FIG. 3.16 – Débit de réception cumulé sur l'ensemble ressources (SystemC)

Si l'on observe le débit global sur le réseau modélisé en SystemC (Figure 3.16), on obtient les mêmes ordres de grandeur qu'avec NS-2, que ce soit au niveau du débit global moyen et du débit maximal. Le débit obtenu avec SystemC est légèrement inférieur car les trafics complémentaires correspondant aux échantillons en réception et à l'utilisateur sur le lien Ethernet n'ont pas été simulé car nous n'avons pas implémenté les générateurs adéquates avec SystemC.

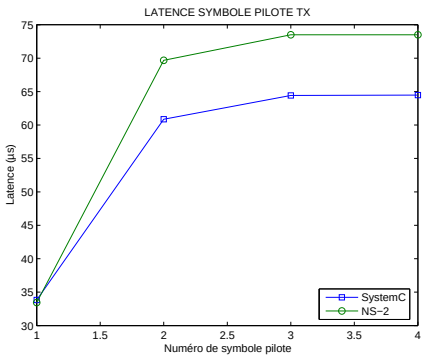
Au niveau de la latence des symboles pour l'émission et la réception, les résultats sont également semblables (Figure 3.17).

3.5.3 Évaluation de l'outil NS-2 et comparaison avec le modèle SystemC

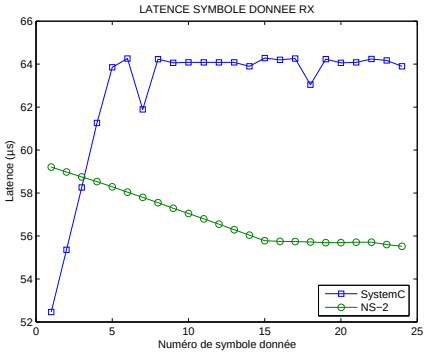
Nous avons présenté des résultats de modélisation pour le trafic réseau correspondant à la couche physique d'un système de télécommunication basé sur une modulation MC-CDMA. Cette modélisation a été réalisée en adaptant l'outil NS-2 aux spécificités du réseau sur puce FAUST et également en utilisant un modèle SystemC de ce réseau. Nous allons maintenant faire un bilan sur l'utilisation de NS-2 ainsi qu'une comparaison avec l'approche SystemC.



(a) Émission données



(b) Émission pilotes



(c) Réception données

FIG. 3.17 – Comparaison des latences émetteur et récepteur entre les modèles NS-2 et SystemC

3.5.3.1 Limitations du modèle NS-2

L'outil NS-2 a été choisi comme support pour construire rapidement un environnement de modélisation pour un NoC. Cette démarche était possible dans la mesure où un nombre important de modules de NS-2 pouvaient être exploités. La problématique résidait dans le fait que NS-2 a été développé dans le cadre de réseaux non-intégrés qui ont des spécificités assez éloignées de celles des NoC. Un travail d'adaptation a donc été nécessaire. Pour ce qui concerne les ressources, la structure de NS-2 se prêtait bien à la création de nouveaux Agent et Application. Par contre, au niveau des mécanismes internes du réseau, des compromis ont dû être faits, sans quoi le temps de développement aurait annulé les bénéfices de l'utilisation de NS-2.

- Mécanismes de commutation de paquet

La modélisation du mécanisme de commutation présentée au paragraphe 3.2.2.3 est précise lorsque les paquets circulent sans conflit entre eux. Dans le cas contraire, le modèle atteint ses limites. Avec une réelle commutation de type *wormhole*, lorsque plusieurs paquets doivent utiliser le même lien, le noeud effectue un arbitrage pour sélectionner celui des deux qui est le premier à passer. Par la suite, l'ensemble des flits du paquets suivent. En effet, les paquets ne peuvent pas être divisés car seul l'en-tête contient les informations nécessaires pour qu'ils arrivent à destination. Avec NS-2, les paquets sont utilisés comme des flits. Les informations d'en-tête sont donc recopiés dans chaque flit - mais ne sont pas comptabilisées dans la taille du flit. Au niveau du noeud, plusieurs paquets peuvent donc avoir leurs flits imbriqués (Figure 3.18). Du point de vue des performances, la bande-passante du lien étant fixée, la latence totale n'est pas changée, par contre la latence individuelle par paquet est modifiée.

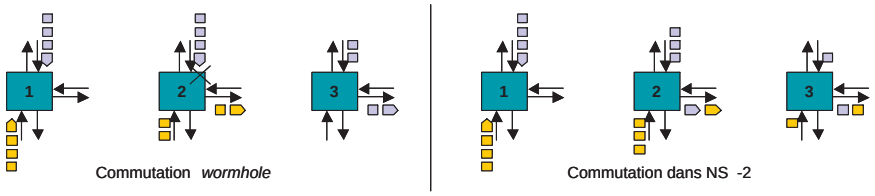


FIG. 3.18 – Comparaison entre la commutation *wormhole* et le modèle NS-2 (1/2)

- Contrôle de flux au niveau réseau

Un second point de limitation se situe au niveau du blocage des paquets sur les noeuds. La commutation *wormhole* permet d'avoir un faible espace mémoire au niveau de chaque noeud. Lorsqu'un paquet est momentanément bloqué au niveau d'un noeud, suite à un arbitrage en sa défaveur, les flits qui le constituent vont être stockés sur les noeuds précédents dans le chemin de routage. Dans NS-2, il n'existe pas de lien direct entre les noeuds ce qui empêche d'avoir un contrôle de flux au niveau des noeuds. C'est pourquoi nous utilisons un mécanisme de type *store-and-forward* avec lequel les noeuds sont complètement indépendants. Il faut alors pouvoir stocker la totalité d'un paquet au niveau d'un noeud car lorsque l'en-tête est bloqué, aucune mécanisme n'empêche l'arrivée de nouveaux flits. Il en résulte que le modèle NS-2 ne permet pas de simuler le blocage d'un paquet sur plusieurs noeuds (Figure 3.19). Pour une latence globale respectée, l'écoulement des paquets va donc avoir tendance à être lissé en supprimant des phénomènes de blocage et déblocage comme c'est le cas avec une commutation *wormhole*.

- Bilan de l'étude sur l'outil NS-2

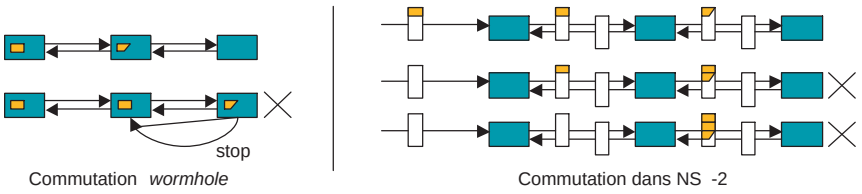


FIG. 3.19 – Comparaison entre la commutation *wormhole* et le modèle NS-2 (2/2)

Les travaux avaient pour objectifs d'étudier les possibilités de modélisation d'un réseau NoC avec l'outil NS-2 et ensuite de valider une architecture de système de télécommunication par la modélisation du trafic de données. NS-2 permet de modéliser les architectures de réseau à haut niveau, il ne propose donc pas beaucoup d'option pour le paramétrage des couches basses. Il en résulte que la simulation d'un réseau NoC ne peut pas rendre compte fidèlement des mécanismes proches de l'implémentation matérielle. Le tableau 3.2 récapitule les différents points d'évaluation de l'outil NS-2.

TAB. 3.2 – Éléments d'évaluation de l'outil NS-2

Avantages	**Inconvénients**
Moteur de simulation de réseau directement utilisable	Faible flexibilité sur la mécanique interne du réseau
Nombreux éléments de réseau (noeud, lien) permettant de développer une architecture NoC	Éléments pas nécessairement bien adaptés aux spécificités des NoC
Modification des scénarios sans compilation (interface OTcl)	Temps d'exécution des simulations lent comparé à du code compilé
Visualisation graphique du réseau et des paquets (outil NAM)	Pas de visualisation chronologique (type transaction)

Malgré les restrictions au niveau des mécanismes bas-niveau, le modèle NS-2 permet de simuler tous les événements du protocole de communication au niveau réseau ce qui autorise une validation fonctionnelle du système. Il fournit de bons ordres de grandeurs sur les débits et les latences ce qui permet d'explorer différentes architectures et d'observer l'influence de divers paramètres. Compte tenu des relations complexes qui existent entre les différentes ressources du réseaux, la possibilité de modéliser de façon précise les échanges de données permet de faire apparaître certains phénomènes difficilement prévisibles statiquement. Par exemple, la cadence de traitement des unités de traitement dépend à la fois de la réception des données à traiter mais également de

la possibilité d'émettre les données une fois traitées. Ces interactions entre ressources sont bien modélisées. L'outil NS-2 remplie donc bien les objectifs fixés au départ mais il manque de précision lorsqu'on souhaite se rapprocher d'une solution plus matérielle.

3.5.3.2 Comparaison entre l'approche NS-2 et SystemC

Les résultats comparatifs présentés au paragraphe 3.5.2 ont montré que les modèles SystemC et NS-2 donnaient des résultats du même ordre de grandeur. Cependant, compte tenu des limitations exposées précédemment, nous pouvons dire que NS-2 est un outil pertinent lorsque l'on veut étudier une première version d'un protocole de communication et explorer une architecture nouvelle de réseau mais son champ d'action est limité par rapport à un modèle SystemC spécifique au réseau FAUST. Plusieurs points de comparaison sont présentés dans le tableau 3.3. En ce qui concerne la vitesse de simulation, pour un même scénario, le modèle SystemC est 3,5 fois plus rapide que NS-2 qui est ralenti par les commandes OTcl qui ne sont pas compilées.

Tab. 3.3 – Comparaison des approches NS-2 et SystemC

	Approche NS-2	**Approche SystemC**
Temps de développement	☑	☒
Rapidité de simulation	☒	☑
Interface utilisateur	☑	☒
Précision du modèle	☒	☑
Exploration d'architecture	☑	☒
Intégration dans un flot de conception	☒	☑

3.6 Conclusion

Dans ce chapitre, nous venons de présenter les résultats de nos travaux sur la modélisation des réseaux sur puce dans le cadre de l'architecture FAUST. Notre but était de générer un trafic de données correspondant à une chaîne de modulation et démodulation de type MC-CDMA et de simuler ce trafic sur une architecture NoC en respectant le protocole de communication. Deux environnements de modélisation ont été utilisés, l'un fonctionnant avec l'outil NS-2, le second étant basé sur un modèle SystemC au niveau TLM.

Les deux environnements nous donnent des résultats comparables et permettent de valider les performances du réseau en ce qui concerne sa capacité à supporter le trafic spécifique à l'implémentation d'un système de télécommunication.

Cette étude nous permet de conclure que l'outil NS-2 est utile pour une exploration rapide du réseau : topologie, placement des ressources, paramètres des communication. Une fois cette étape validée, le modèle SystemC apporte plus de précision au niveau des mécanismes de communication. Nous verrons ainsi au Chapitre 4 comment le modèle SystemC a été exploité pour modéliser une nouvelle architecture de contrôle et de configuration pour les ressources de traitement distribuées sur le réseau.

Chapitre 4

Conception d'un système de contrôle et de configuration pour un réseau sur puce

Une architecture de réseau sur puce crée une structure de communication performante pour échanger une grande quantité d'information entre différentes ressources de traitement. Cependant un réseau sur puce est, par construction, un système distribué. Il n'y a pas de connexions directes entre les ressources. Ceci apporte une grande flexibilité au niveau de l'organisation des communications et permet d'avoir des systèmes reconfigurables. Pourtant lorsque l'on cherche à implémenter une application sur un tel système, il faut faire face à de nouvelles problématiques : comment faire fonctionner et contrôler une application distribuée sur un réseau ? comment assurer la gestion des communications ? comment piloter des unités reconfigurables ? Nous avons identifié deux problématiques en particulier au chapitre 1 : la gestion des flots de données (cf. paragraphe 1.4.2) et le contrôle global de l'exécution de l'application (cf. paragraphe 1.4.3).

Dans ce chapitre, nous allons tout d'abord exposer comment nous nous positionnons pour aborder ces problématiques. Nous présenterons ensuite notre approche pour le contrôle d'un NoC et nous détaillerons une architecture d'interface réseau. Ensuite, nous nous intéresserons à la modélisation VHDL de cette interface et aux résultats de synthèse auxquels nous avons abouti. Pour finir, nous évaluerons notre approche dans sa globalité grâce à une modélisation complète du système basée sur un environnement de cosimulation SystemC et VHDL.

4.1 Contrôle d'une application sur un NoC et gestion des communications associées

Cette section a pour objectif de présenter comment nous abordons les problématiques de gestion des communications et du contrôle des traitements pour des ressources distribuées sur un réseau d'interconnexion sur puce. Tout d'abord nous présentons une architecture de ressource généralisant la structure utilisée dans le circuit FAUST. A partir de cette architecture, nous détaillons des notions liées au déroulement temporel d'une application avant de nous intéresser à certaines contraintes de fonctionnement dans le cadre d'applications précises.

4.1.1 Positionnement du problème

Nous travaillons sur des applications de type *flot de données* dans lesquelles différentes unités de traitement s'échangent des données. Une unité de traitement reçoit des données par différents canaux d'entrée. Elle envoie des données par des canaux de sortie. Son fonctionnement est géré par l'intermédiaire de signaux de contrôle. En ayant recours à une structure de réseau sur puce, nous avons fait le choix d'une architecture distribuée. Le réseau de communication est utilisé pour échanger l'ensemble des informations, que ce soit des données à traiter ou des données pour le contrôle des unités.

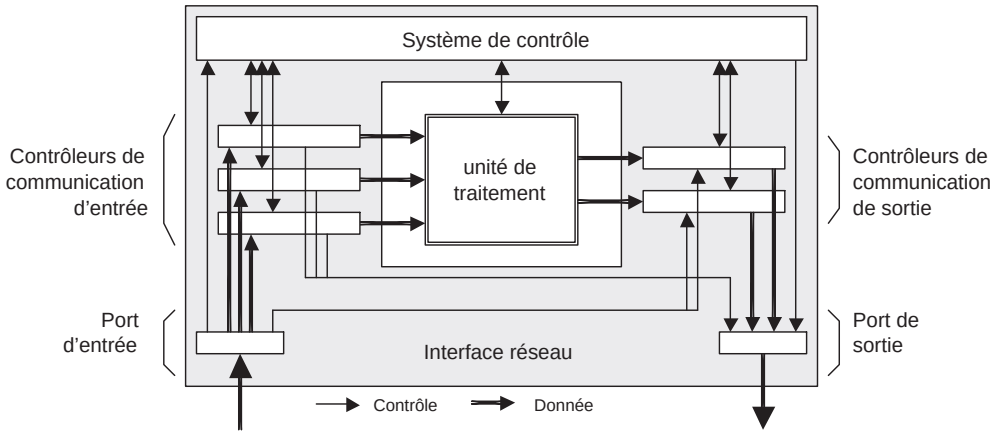


FIG. 4.1 – Architecture générique d'une ressource

Pour communiquer avec le réseau, les unités de traitement sont associées à une *interface réseau*. Cette interface a pour objectif d'abstraire le réseau du point de vue de l'unité de traitement (cf. paragraphe 1.4.2). Dans notre architecture NoC, une *ressource*

correspond au couple formé par l'unité de traitement et l'interface réseau. La Figure 4.1 présente une architecture générique d'une ressource. Les canaux d'entrée et de sortie de l'unité de traitement sont connectés au réseau par le biais de contrôleurs de communication qui prennent en charge la gestion du protocole. Un système de contrôle gère les signaux de contrôle de l'unité de traitement. Des ports d'entrée et de sortie orientent les différents flux entrants et sortants.

L'interface réseau du circuit FAUST est construite sur ce principe. Nous avons introduit aux paragraphes 2.4.2 et 2.4.3 certains points que nous souhaitons faire progresser au niveau de la gestion des flux de données et du contrôle des traitement.

4.1.2 Déroulement temporel d'une application

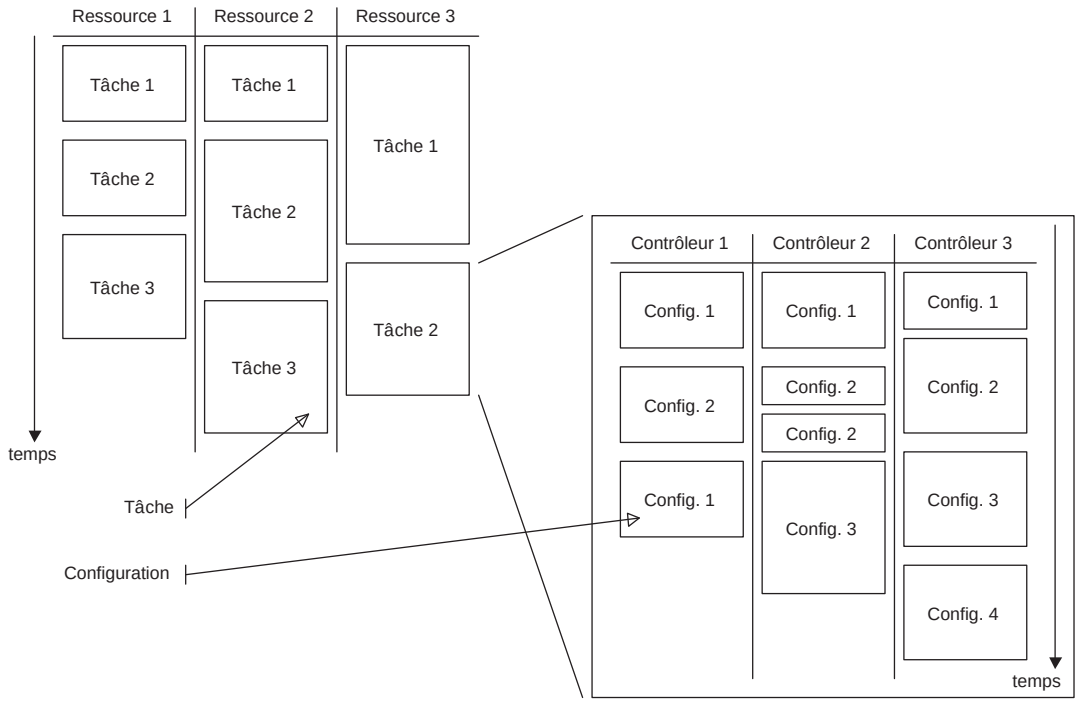


FIG. 4.2 – Déroulement temporel d'une application

Nous allons définir certaines notions que nous utilisons pour décrire le déroulement temporel d'une application. Le terme *application* correspond à la fonction globale du circuit. Dans une architecture de NoC, cette fonction est réalisée par différents sous-systèmes de traitement que nous appelons *ressources*. Ces ressources sont distribuées et communiquent grâce au réseau. Au cours du déroulement de l'application, chaque

ressource va avoir à réaliser différentes *tâches* de traitement. La Figure 4.2 donne un exemple de ce fonctionnement pour un réseau composé de trois ressources.

Au niveau d'une ressource, une tâche correspond à une série de calculs au niveau de l'unité de traitement et/ou à des transferts de données au niveau de l'interface réseau. Ces opérations sont pilotées par différents contrôleurs que nous avons décrit avec l'architecture générique de ressource (Figure 4.1) présentée précédemment. Plus précisément, une tâche correspond donc à l'exécution d'un certain nombre de *configurations* pour chaque élément de contrôle de l'interface. Une configuration est constituée de paramètres qui permettent l'exécution d'un calcul (pour l'unité de traitement) ou d'un transfert de données (pour un contrôleur de communication). Dans la suite le terme contrôleurs sera également utilisé pour faire référence à la partie contrôle de l'unité de traitement. La Figure 4.2 détaille pour la Ressource 3 les différentes configurations de ses trois contrôleurs pour la tâche 2.

Le découpage en tâches est assez arbitraire. Il peut être fait de manière à permettre un découpage dans le temps à un moment où tous les contrôleurs d'une ressource ont terminé d'exécuter leurs configurations. Il est maintenant possible de distinguer deux niveaux de séquencement dans le déroulement d'une application :

- Le *séquencement global* : gestion de l'enchaînement des tâches au niveau de chaque ressource.
- Le *séquencement local* : gestion de l'enchaînement des configurations pour chaque contrôleur au sein d'une tâche.

4.1.3 Analyse des contraintes de fonctionnement dans le cadre d'applications précises

Pour préparer la nouvelle approche que nous présentons dans la section suivante (4.2), nous avons analysé les modes de fonctionnements des applications liées aux projets MATRICE [MATR06] et 4MORE [4MOR06] étudiées dans le cadre du projet FAUST (cf. chapitre 2). Ces applications nous ont servies de support de réflexion. Elles viennent en complément de ce qui a été présenté sur l'évolution des systèmes de télécommunications sans-fil et sur les techniques de modulation (cf. section 2.1). Le but est de pouvoir anticiper les besoins futurs des architectures de contrôle de tels circuits.

4.1.3.1 Configurations des flux d'entrée et de sortie

Les flux d'entrée et de sortie, gérés par les contrôleurs de communication des interfaces réseau (nommés ICC et OCC, cf. paragraphe 2.3.3.1) correspondent à une certaine quantité de données à transférer (bloc de données). En fonction de nombreux paramètres comme le choix de la modulation (choix de la constellation[1] et du rapport de codage[2]) ou du nombre de codes d'étalement de spectre utilisés, la taille des blocs de données varient.

Nous avons évalué le nombre de modes de fonctionnement possibles, en terme de taille de données, pour les deux applications de télécommunication étudiées dans le cadre du projet FAUST (cf. section 2.2). Ce nombre de mode est directement lié au nombre de configurations des contrôleurs même si des possibilités de séquencement permettent de réutiliser des configurations. Par exemple, si une ressource doit transférer des blocs de 8 flits ou de 24 flits en fonction de la modulation. La configuration permettant de transférer 8 flits peut être utilisée 3 fois de suite pour transférer 24 flits. On peut ainsi limiter le nombre de configuration à condition de reconfigurer le séquencement.

TAB. 4.1 – Évaluation du nombre de modes de fonctionnement

MATRICE	**UL ou DL**	Nombre de modulations	4
		Nombre de codes d'étalement	1 à 32
		Nombre de modes	**128**
4MORE	**UL**	Nombre de modulations	12
		Nombre de codes d'étalement	1 à 8
		Nombre de sous-bandes utilisées	1 à 24
		Nombre de modes	**2304**
	DL	Nombre de modulations	15
		Nombre de codes d'étalement	1 à 32
		Nombre de modes	**480**

Le nombre de modes de fonctionnement donne donc un ordre de grandeur de la complexité de la configuration des unités de traitement. Cette complexité se répartit entre des paramètres de configuration et des paramètres de séquencement en fonction de l'implémentation qui est faite. Le Tableau 4.1 montre que pour pouvoir faire fonctionner les standards étudiés dans les projets MATRICE et 4MORE, il faut pouvoir mettre en

[1] par exemple : QPSK, 8PSK, 16QAM...
[2] par exemple : 1/2, 2/3, 3/4...

place des systèmes de contrôle et de configuration capables de supporter un grand nombre de modes de fonctionnement.

4.1.3.2 Séquencement des transferts

Nous venons de voir que, suivant les modes de fonctionnements, la taille des blocs de données est un paramètre pris en compte dans la gestion des configurations des interfaces réseau. En fait, les chaînes de traitement ne sont pas linéaires. Pour un mode de fonctionnement donné, une ressource peut être amenée à gérer plusieurs flux de données simultanément ou séquentiellement. Au niveau de l'entrée/sortie d'une ressource, les données peuvent avoir des sources/destinations semblables ou différentes et être de même nature ou pas.

TAB. 4.2 – Évaluation de la complexité des flots de données

		MATRICE				**4MORE**			
		Entrée		**Sortie**		**Entrée**		**Sortie**	
	Ressource	Flux Donnée	**Flux Ressource**	Flux Donnée	**Flux Ressource**	Flux Donnée	**Flux Ressource**	Flux Donnée	**Flux Ressource**
UL	CHAN COD	1	**1**	1	**1**	1	**1**	1	**1**
	INTER	1	**1**	1	**1**	1	**1**	1	**1**
	MAPP	1	**1**	1	**1**	1	**1**	1	**1**
	CDMA MOD	1	**1**	1	**1**	1	**1**	1	**1**
	ALAM COD					1	**1**	1	**1**
	OFDM MOD	(1) 1	**(1) 2**	1	**1**	(1,2) 3	**(1) 2**	1	**1**
	RF OUT	(1) 2	**(1) 2**			(2) 4	**(2) 4**		
DL	RF IN			1	**1**			(2) 2	**(2) 2**
	SYNCHRO	1	**1**			(2) 3	**(2) 3**	(2) 2	**(2) 2**
	ROTOR	1	**1**	1	**1**	1	**1**	1	**1**
	OFDM DEM	1	**1**	(1) 2	**(1) 2**	1	**1**	(1,2) 3	**(1,2) 3**
	CHAN EST	1	**1**	(2) 2	**(2) 2**	(2) 2	**(2) 2**	(4) 4	**1**
	MIMO DECOD					(6) 6	**(3) 3**	(3) 3	**(2) 2**
	EQU	(2) 2	**(2) 2**	1	**1**	(2) 2	**1**	1	**1**
	CDMA DEM	1	**1**	1	**1**	1	**1**	1	**1**
	DEMAPP	(2) 2	**1**	1	**1**	2 (2)	**2**	1	**1**
	DEINTER	1	**1**	1	**1**	1	**1**	1	**1**
	CHAN DEC	1	**1**	1	**1**	(1) 2	**(1) 2**	1	**1**

(flux simultanés) flux totaux

Le Tableau 4.2 illustre les différents flux de communication pour les systèmes MATRICE et 4MORE. Pour chaque ressource, on sépare les flux entrants et sortants. Pour chaque flux, il peut y avoir des données de nature différentes (flux donnée) ou en provenance ou destination de ressources différentes (flux ressource). On constate qu'un certain nombre de ressources fonctionne avec un seul flux en entrée et en sortie. Pour un mode de fonctionnement donnée, un séquencement des flux n'est donc pas nécessaire. D'autres ressources doivent par contre gérer simultanément plusieurs flux. On peut prendre pour

exemple le démodulateur OFDM (OFDM DEM) dans le cas 4MORE (Figure 4.3). En entrée, il reçoit les échantillons temporels du signal à démoduler. En sortie, on distingue deux phases. Pour les symboles pilotes, toutes les porteuses correspondent à des pilotes et sont envoyées vers l'estimateur de canal (CHAN EST) ce qui fait 1 flux données et 1 flux ressource. Pour les symboles données, les porteuses peuvent être de type donnée ou pilote. Elles sont envoyées vers le décodeur MIMO (MIMO DECOD) ou vers l'unité de synchronisation temporelle et fréquentielle (SYNCHRO). Il y a alors 2 flux données et 2 flux ressource. Un système de séquencement doit donc permettre d'enchaîner ces deux phases et de basculer entre les deux destinations lors de la seconde phase.

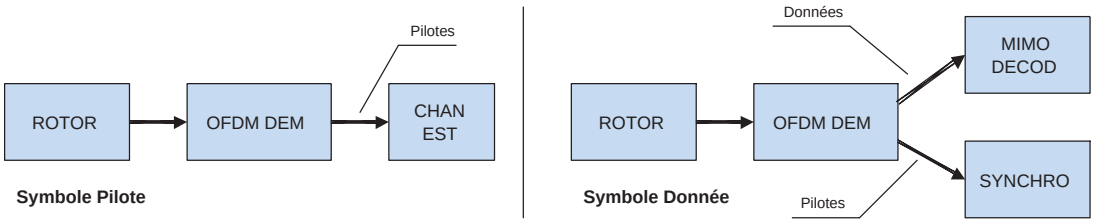


FIG. 4.3 – Exemple du démodulateur OFDM pour la gestion des flux de données

4.2 Nouvelle approche pour le contrôle et la configuration d'une architecture NoC

Après avoir présenté les problématiques auxquelles nous voulons répondre concernant le contrôle et la configuration d'une architecture de réseau sur puce, nous consacrons à présent cette section à l'exposé de notre proposition d'architecture et de son principe de fonctionnement.

4.2.1 Description de l'approche suivie

4.2.1.1 Un système de contrôle semi-distribué

Nous avons choisi de mettre en place une structure de contrôle que nous qualifions de *semi-distribuée* et *hiérarchique*. Nous cherchons ainsi un compromis entre d'une part une solution entièrement centralisée et d'autre part complètement distribuée. Avec une approche centralisée, le système de contrôle doit faire face à une charge de tâches très importante pour pouvoir gérer l'exécution des traitements sur toutes les ressources. De

plus, ce mode de fonctionnement génère un trafic réseau important à cause des nombreux messages de contrôle qu'il faut échanger entre le système central et les ressources à chaque étape de traitement. A l'opposé, dans un système complètement distribué, chaque ressource doit être autonome ce qui augmente sa complexité. Ce choix est valable dans un système massivement parallèle dans lequel on intègre des processeurs génériques qui ont tous des positions équivalentes dans la répartition des calculs mais plus difficile à mettre en oeuvre dans le cas de l'intégration d'un modem bande de base avec des ressources de traitement spécialisées.

Nous proposons donc une structure de contrôle semi-distribuée qui s'organise de la façon suivante :

- Un système de contrôle central

Ce système peut être implémenté sous la forme d'un processeur. Ce processeur pilote et séquence l'application à haut niveau. Il contrôle donc le séquencement global, c'est à dire l'organisation des différentes tâches sur chaque ressource. Par la suite nous utiliserons aussi bien le terme de *système de contrôle central* que de *processeur*.

- Plusieurs systèmes de contrôle locaux (SCL)

Ces systèmes sont implémentés au niveau de chaque ressource de traitement. Il réalisent le séquencement local, c'est à dire l'enchaînement des configurations sur chacun des contrôleurs (contrôleur de communication ou unité de traitement).

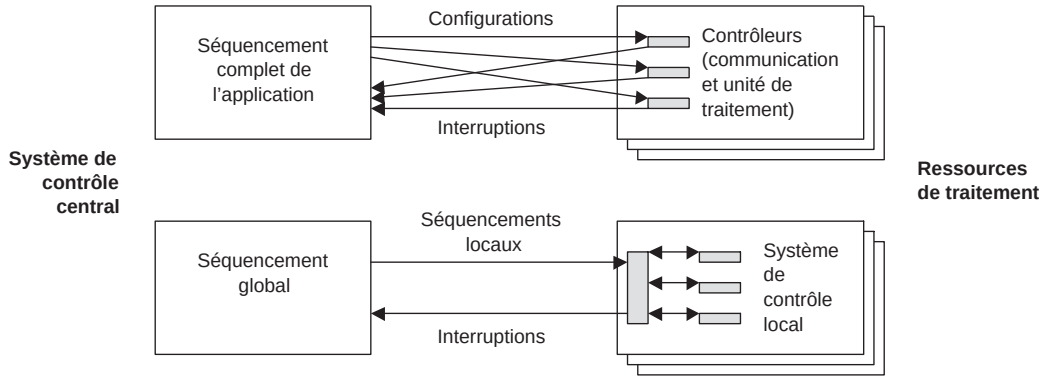


FIG. 4.4 – Comparaison entre un contrôle centralisé et semi-distribué

Le système de contrôle central peut ainsi répartir le contrôle de l'application sur les différents systèmes de contrôle locaux (Figure 4.4). Pour réduire la complexité du système de contrôle central, le système local doit être rendu le plus autonome possible. En effet, le système central est ainsi moins souvent sollicité par les systèmes locaux (par

exemple, par des messages d'interruptions). L'application peut s'exécuter plus rapidement et de manière plus fluide.

Le découpage entre séquencement global et local est possible car nous travaillons sur des applications de type flot de données pour lesquelles ce n'est pas la ressource qui prend une décision sur l'origine ou la destination des données. En d'autres termes, les applications visées ont un schéma de fonctionnement semi-automatique : pour chaque tâche, une ressource est configurée et fonctionne de façon autonome jusqu'à la tâche suivante.

4.2.1.2 Un système de gestion des configurations

Pour exécuter une tâche, le système de contrôle local doit disposer de deux éléments de programmation :

- La série des configurations à exécuter

Une tâche correspond à l'exécution d'une série de configurations sur les différents contrôleurs de la ressource. Il est donc nécessaire de connaître la séquence (le scénario) des configurations pour chacun des contrôleurs.

- Le contenu des configurations

Pour chacun des contrôleurs, une configuration va correspondre à différents paramètres d'exécution. Pour un contrôleur de communication de sortie il s'agit par exemple de la taille des paquets, du chemin de routage vers la destination, de la taille du bloc de donnée.

Comme expliqué précédemment, nous voulons que chaque ressource ait une autonomie importante dans la gestion du séquencement local. Cela revient à dire que le scénario doit contenir un nombre significatif de configurations. Le problème qui se pose alors est celui de la gestion du stockage des paramètres de configuration. En effet, il faut être en mesure de stocker au niveau de la ressource à la fois la série de configurations mais également le contenu des configurations. Cela implique au niveau de chaque ressource un espace mémoire important. Ceci est coûteux en terme de surface de silicium d'autant plus qu'il faut dupliquer cet espace mémoire sur chacune des ressources.

Pour palier ce problème, notre approche consiste à traiter de façon indépendante le chargement des séquences de configurations et le chargement du contenu des configurations. Le chargement des séquences de configurations est réalisé par le système de contrôle central introduit précédemment. Ce système ne manipule pas directement le

contenu des configurations mais seulement une référence. On peut dire qu'il manipule les configurations de manière virtuelle (ou indirecte). En pratique, cela permet de gérer des scénarios beaucoup plus compacts en terme d'informations contenues.

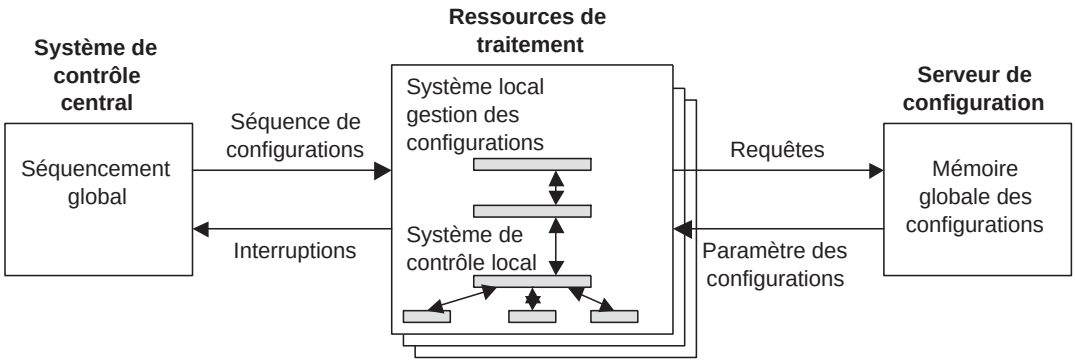


FIG. 4.5 – Contrôle semi-distribué avec serveur de configuration

Pour ce qui concerne la gestion du contenu des configurations, nous introduisons une nouvelle ressource qui joue le rôle de *système central de gestion des configurations* ou plus concisement de *serveur de configurations* (Figure 4.5). Ce serveur est capable de fournir le contenu des configurations nécessaires aux contrôleurs de l'ensemble des ressources.

Au niveau de chaque ressource, nous ajoutons un *système local de gestion des configurations* (SLGC). Ce système va pouvoir communiquer avec le serveur de configurations via le réseau. Pour une tâche donnée, seul le séquencement des configurations sera stocké dans son intégralité au niveau de la ressource. Les configurations stockées localement évolueront au cours de l'exécution des séquences et des requêtes vers le serveur de configuration.

4.2.2 Principes de fonctionnement du protocole de contrôle et de configuration

Nous allons maintenant détailler le fonctionnement du protocole mis en place et évaluer les avantages d'une telle approche. Les éléments principaux sont présentés Figure 4.6.

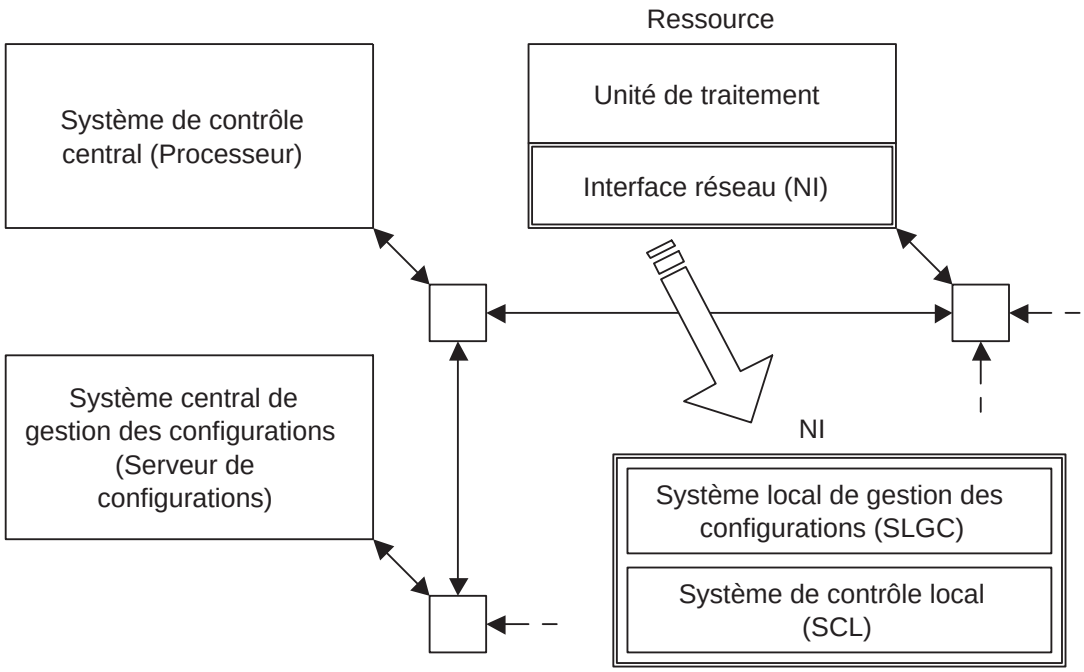


FIG. 4.6 – Acteurs principaux du système de contrôle et de configuration

4.2.2.1 Organisation fonctionnelle

Au fur est à mesure de l'exécution des tâches sur chacune des ressources, le processeur, va charger des séquences de configurations dans chacun des SCL. Chaque SCL exécute les séquences de configurations correspondants aux différents contrôleurs qui lui sont associés. Le SCL envoie des requêtes au système local de gestion des configurations (SLGC). Deux cas de figures peuvent alors se présenter :

- Les paramètres de configuration ne sont pas présents localement. Le SLGC va envoyer une requête de configuration au serveur de configuration. En retour de cette requête, il obtiendra le contenu de la configuration qu'il pourra directement fournir au SLC. Le SLC pourra alors configurer le contrôleur auquel la configuration est destinée.
- La configuration a déjà été chargée. Elle est encore présente dans la mémoire du SLGC. La configuration peut donc être directement chargée dans le contrôleur sans avoir à faire de requête sur le réseau comme précédemment.

4.2.2.2 Avantages de cette approche

Les principaux avantages de cette approche sont :

- En provenance du système central, nous limitons les transferts sur le réseau puisque le contenu des configurations n'est plus envoyé. Au niveau des ressources, les configurations sont quand même reçues mais ces réceptions peuvent se faire de manière plus espacée dans le temps et ainsi s'insérer de façon plus fluide dans le trafic du réseau.
- La taille des mémoires pour stocker les paramètres de configurations dans les ressources est limitée. Il ne s'agit que d'une mémoire cache, la totalité des configurations se trouve dans le système central de gestion des configurations.
- La charge du système central est maîtrisée puisqu'une grande partie du séquencement des configurations est gérée localement et il n'intervient plus dans la gestion des paramètres de configuration.

De manière plus générale, cette approche ne remet pas en cause l'aspect entièrement distribué du système. En particulier, toutes les communications continuent à se faire en utilisant le réseau d'interconnexion. De plus les systèmes introduits sont par la nature des fonctionnalités présentées configurables (et reconfigurables) ce qui maintient un degré important de souplesse et de flexibilité dans la programmation du système.

4.3 Présentation détaillée de l'architecture d'interface réseau

L'organisation du contrôle et de la reconfiguration des ressources de traitement s'appuie sur une architecture d'interface réseau (NI) que nous allons décrire dans cette section. La Figure 4.7 présente une vue globale de l'architecture. Elle réutilise les mêmes modules que ceux intégrés dans le circuit FAUST pour les chemins de données. Pour la communication du réseau vers l'unité de traitement, il s'agit du *port d'entrée* (IP) qui oriente les paquets vers leur cible et des mémoires FIFO gérées par les *contrôleurs de communication d'entrée* (ICC). Pour les communications de l'unité de traitement vers le réseau, les données sont mises en paquets par le *contrôleur de communication de sortie* (OCC), les paquets sont arbitrés dans le *port de sortie* (OP). Ces modules ont été décrits dans le paragraphe 2.3.3.

Dans la suite de cette section nous allons détailler le fonctionnement du SCL (paragraphe 4.3.1), la gestion du séquencement des configurations (paragraphe 4.3.2), le

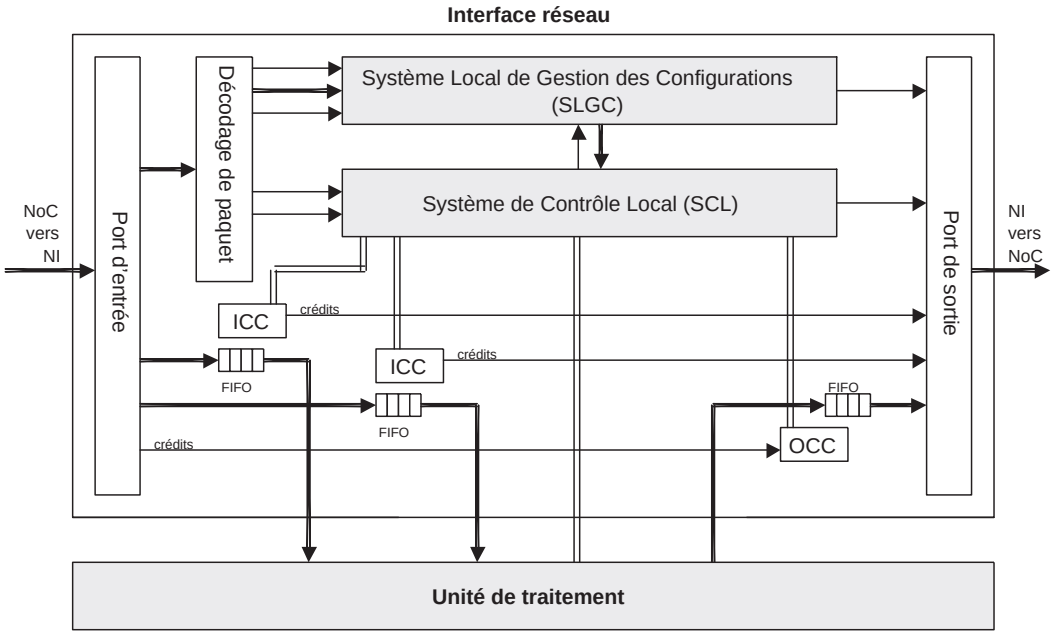


FIG. 4.7 – Architecture de l'interface réseau

fonctionnement du SLGC (paragraphe 4.3.3) et le format des paquets de contrôle (paragraphe 4.3.4)

4.3.1 SCL : Système de Contrôle Local

La gestion du séquencement local est assurée par le *système de contrôle local* (SCL). Ce système gère le chargement des configurations pour les contrôleurs de communication de l'interface réseau (NI) et pour l'unité de traitement. Le processeur (système de contrôle global) envoie par le réseau les séquences de configuration pour chacun des contrôleurs du NI. Nous appelons *contrôleurs du NI*, les contrôleurs de communication (ICC, OCC) et aussi l'unité de traitement qui du point de vue du SCL sont gérés de façon équivalente.

Pour chaque contrôleur du NI, la séquence de configurations se traduit par une série d'instructions. Le SCL est composé d'un *séquenceur d'instructions* (SeqInstr). Pour le SeqInstr, chaque contrôleur est considéré comme un *contexte* différent. Le SeqInstr va donc gérer les séquences de configurations pour les différents contextes qu'il contrôle. Nous aurions pu utiliser un séquenceur d'instruction dédié pour chacun des contrôleurs, c'est à dire utiliser un séquenceur par contexte. En fait nous avons pris pour hypothèse que le temps d'exécution d'une configuration est long par rapport au temps de

changement de contexte. Cette hypothèse a été validée a posteriori. Il est donc plus économique en terme de complexité de mutualiser les ressources matérielles d'un seul séquenceur sur plusieurs contrôleurs sans que cela pénalise les performances du système. Un *gestionnaire de contexte* indique au SeqInstr quel est le contexte courant.

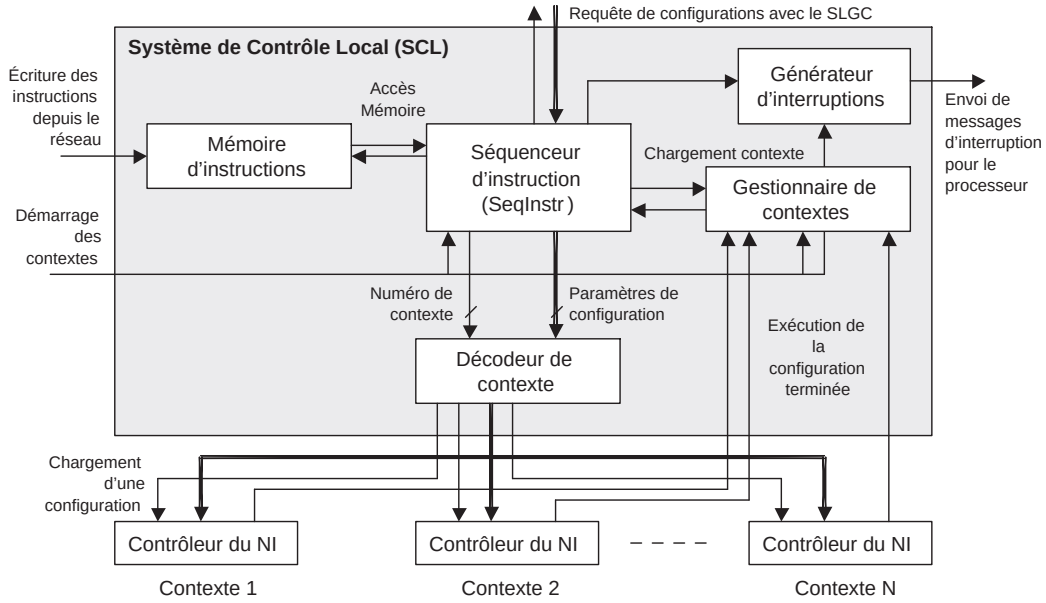


FIG. 4.8 – Structure du SCL et de son interface avec les contrôleurs du NI

Pour stocker la série d'instruction, le SCL contient également une *mémoire d'instructions*. Cette mémoire est écrite directement depuis le réseau. Le processeur charge donc les séquences d'instructions dans les mémoires des SCL en envoyant des paquets sur le réseau. Nous détaillerons au paragraphe 4.3.4 le format de ces paquets. L'espace mémoire va donc être divisé en sous ensembles correspondant à des séquences d'instructions codant pour le chargement de configuration de chaque contexte.

Pour chaque contexte, le séquenceur d'instruction va fournir une série de configurations à un contrôleur du NI. Un *décodeur de contexte* envoie un signal de chargement de configuration au contrôleur concerné. Les paramètres des configurations sont fournis par le *système local de gestion des configurations* (SLGC). Le gestionnaire de contexte permet au SCL de commuter d'un contexte à l'autre. Lorsque la séquence de configurations associés à un contexte est terminée, le SCL peut envoyer une interruption (si elle n'est pas masquée) au processeur via un *générateur d'interruptions*.

4.3.2 Séquencement des configurations

Code Instruction	Valeur binaire (3 bits)	Paramètre (5 bits)	Commentaires
RCF	000	CFG	Chargement de la configuration identifiée par CFG
RCR	001	CFG	Chargement de la configuration identifiée par CFG et mémorisation de l'adresse courante dans le registre d'adresse de boucle (`R_ADD_REG`)
LLO	010	NBL	Boucle locale. Répétion NBL fois
GLO	011	NBL	Boucle globale. Répétition NBL fois
STP	011	0	STOP. Fin de la séquence

FIG. 4.9 – Instructions pour le séquencement des configurations

Chaque interface réseau comporte un *système de contrôle local* (SCL). Le SCL intègre une mémoire permettant de stocker des séquences de configurations pour les contrôleurs de communication de l'interface et pour l'unité de traitement. Ces séquences correspondent à une série d'instructions. Actuellement, les instructions sont codées sur 8 bits et il existe 5 instructions (Figure 4.9). Les configurations sont identifiées par un numéro (CFG). Avec 5 bits de paramètres, on peut ainsi utiliser 32 configurations pour chaque contexte. L'instruction principale (RCF) permet de coder une requête pour un numéro de configuration. Le terme requête est employé car lorsque le SCL décode l'instruction, il ne dispose que d'un numéro et non des paramètres de la configuration. Il va donc envoyer une requête au SGLC pour obtenir le contenu de la configuration. Des instructions supplémentaires (LLO et GLO) ont été ajoutés pour pouvoir coder deux niveaux de boucles de répétition. Ainsi on distingue la boucle locale (LLO) qui permet de revenir en arrière dans la lecture des instructions et la boucle globale qui permet de rejouer la série d'instruction. L'instruction RCR permet de mémoriser l'adresse mémoire courante pour indiquer le point de retour à l'instruction de boucle locale. Enfin l'instruction STP (boucle globale avec le paramètre 0) indique la fin de la séquence.

Pour illustrer de manière plus concrète l'utilisation de ces instructions de séquencement, nous allons prendre l'exemple du contenu de la mémoire d'instruction présenté Figure 4.10. Trois séquences d'instructions ont été mémorisées aux adresses `0`, `8` et `17`. Le processeur envoie des commandes pour démarrer l'exécution des séquences. Pour cela il indique un numéro de contexte et une adresse de début dans la mémoire d'instruction. Pour chaque contexte, le SCL dispose d'un registre `S_ADD_REG` qui indique la position de la première instruction de la séquence en mémoire. Ce registre permet de reprendre la séquence depuis le début avec l'instruction boucle globale GLO. Un second registre

```
 0 RCF 1
 1 RCF 1
 2 RCF 2
 3 STP 0
 4
 5
 6
 7
 8 RCF 2
 9 RCR 1
10 RCF 2
11 LLO 3
12 STP 0
13
14
15
```

```
16
17 RCF 2
18 RCF 3
19 RCR 2
20 RCF 4
21 LLO 4
22 RCR 1
23 LLO 2
24 GLO 2
25
26
27
28
29
30
31
```

FIG. 4.10 – Exemple du contenu de la mémoire d'instructions

`C_ADD_REG` (initialisé avec l'adresse de début) contient l'adresse courante. A chaque instruction, le séquenceur incrémente ce registre. L'instruction RCR mémorise l'adresse courante dans un troisième registrer d'adresse (`R_ADD_REG`) pour les rebouclages locaux. La Figure 4.11 traduit de manière graphique l'exécution des séquences d'instructions.

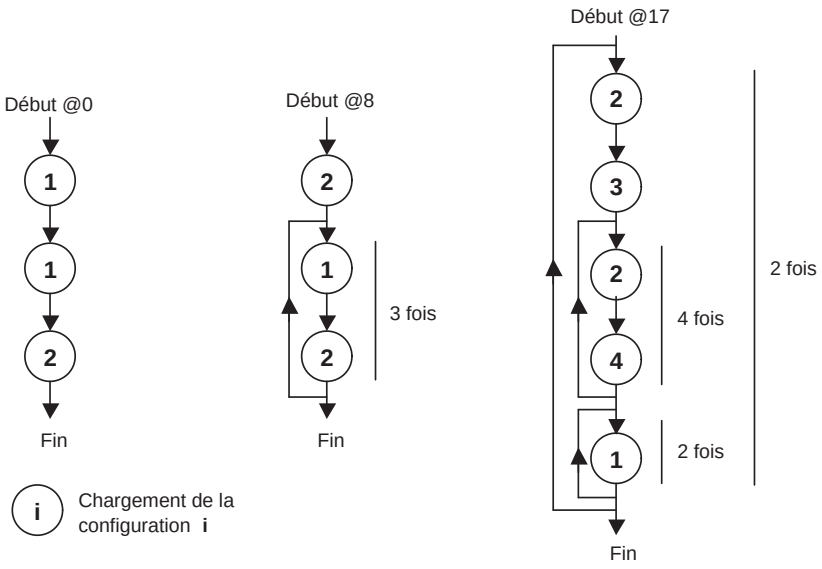


FIG. 4.11 – Exemple de séquences de chargement de configurations

4.3.3 SLGC : Système Local de Gestion des Configurations

Notre approche consiste à regrouper les paramètres de configuration dans une mémoire accessible via le réseau et qui joue le rôle de serveur. Au niveau d'une ressource, lorsqu'une même configuration va être exécutées plusieurs fois au cours du déroulement de l'application, il est intéressant en terme de performance d'avoir une copie locale de la configuration pour pouvoir la charger plus rapidement. Ces mécanismes sont gérés par le SLGC (Figure 4.12)

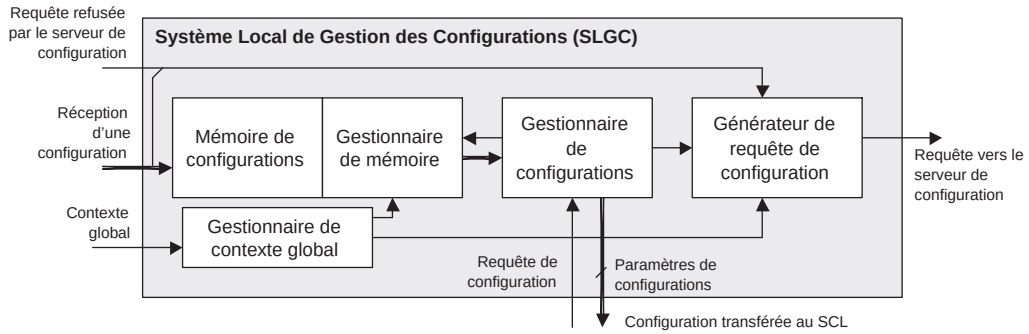


FIG. 4.12 – Structure du SLGC

Le SLGC contient une *mémoire de configurations* permettant de stocker les paramètres de configurations. Nous utilisons des configurations de taille fixe (64 bits). Un *gestionnaire de mémoire* permet à cette mémoire de fonctionner comme une mémoire cache associative. Lorsque le SCL émets une requête de configuration, il fournit le numéro de configuration (CFG) indiqué par l'instruction et le numéro de contexte (CTX) correspondant à cette configuration. La concaténation de ces deux champs (CTX+CFG) est utilisée comme clé d'interrogation pour le gestionnaire de mémoire. Si la configuration associée à la clé est présente, la mémoire est lue. Dans le cas contraire le *gestionnaire de configurations* demande l'envoi d'une requête sur le réseau. Le *générateur de requête de configuration* génère un paquet correspondant à la requête. En complément des champs CTX et CFG, la configuration est identifiée par le numéro de la ressource (RES) et un numéro de contexte global (GCX) pour la ressource.

Le champ GCX est fourni par le *gestionnaire de contexte global*. Sa valeur est chargée par le processeur. Le changement de contexte global entraîne la suppression de toutes les configurations stockées localement. Ce contexte global permet d'utiliser les mêmes séquences de configurations avec des paramètres différents. Un tel mécanisme est utile lorsque le l'on veut par exemple changer le volume de données traitées sans modifier les flots de communication.

En réponse à une requête, les paramètres de configuration sont stockées dans la mémoire de configuration. La politique de gestion du cache est de type FIFO. Les configurations les plus anciennement chargées sont les premières à être remplacées. En cas de saturation, le serveur de configuration peut envoyer un paquet indiquant que la requête doit être réitérée.

4.3.4 Formats des paquets de contrôle

Nous présentons à présent le format des paquets qui permettent l'échange d'informations de contrôle sur le réseau (Figures 4.13 et 4.14). Pour chaque paquet, nous donnons les paramètres d'en-tête correspondant au format des paquets FAUST (cf. paragraphe chap2 :sec :protcommfaust). Il s'agit de l'*adresse cible* dans l'interface (TGTID), du champ *commande* (CMD) et du champ *paramètre* (PARAM). Le contenu des flits (composé de 4 octets) est ensuite indiqué.

4.3.4.1 Paquets reçus par l'interface réseau

En réception, le port d'entrée oriente les paquets en fonction du champ TGTID. Les paquets de contrôle sont reçus avec l'adresse cible 0. En fonction du paramètre de la commande WRITE, un module de décodage de paquets (Figure 4.7) extrait les informations contenues dans les flits de données et les transmets aux modules de l'interface.

Pour l'écriture des instructions, il est possible de mettre plusieurs flits dans un même paquet pour écrire un série d'instructions. L'*écriture mémoire* est prioritaire et suspend les accès internes. Ceci garantit qu'un paquet d'écriture reçu par l'interface ne sera pas bloqué ou perdu. L'*écriture des configurations* se fait en indiquant le numéro de contexte (CTX) et de configuration (CFG). Les configurations font 64 bits et sont transmises sur 2 flits. Pour *démarrer* l'exécution d'un *contexte* au niveau du séquenceur d'instruction, il faut indiquer l'*adresse* de départ pour la lecture des instructions. L'octet *Masque IT* permet de coder si une interruption à destination du processeur sera envoyée en fin d'exécution du contexte (et également le canal virtuel utilisé pour l'envoi du paquet d'interruption). Le paquet *Requête configuration refusée* indique que la requête de configuration en cours n'a pas pu aboutir et qu'elle doit donc être envoyée de nouveau. Le paquet *Changement de contexte global* met à jour le champ GCX de l'interface et vide la mémoire des configurations (qui ne sont plus valables pour cette nouvelle valeur de contexte global).

			TGTID	CMD	PARAM
Écriture instruction			0	WRITE	1
	-	-		Adresse	Instruction
Écriture configuration			0	WRITE	2
	-	-	-		CTX+CFG
	Configuration (32 bits MSB)				
	Configuration (32 bits LSB)				
Démarrage contexte			0	WRITE	3
	-	Masque IT		Adresse	Contexte
Requête config. refusée			0	WRITE	4
Changement contexte global			0	WRITE	5
	-	-	-		GCX

FIG. 4.13 – Formats des paquets de contrôle reçus par l'interface

4.3.4.2 Paquets émis par l'interface réseau

L'interface peut émettre deux types de paquets. Dans l'implémentation actuelle, les en-têtes de ces paquets sont écrits dans l'interface et ne peuvent pas être modifiés dynamiquement. A la fin de l'exécution d'une séquence d'instruction, l'interface peut émettre une *interruption* si celle-ci n'est pas masquée. Elle indique le numéro d'identification de la ressource (RES) et le contexte qui vient de se terminer (CTX).

Pour obtenir les paramètres d'une configuration, l'interface envoie un paquet de *requête* à destination du serveur de configurations. Le format du paquet est lié au modèle actuel du serveur qui sera présenté dans le paragraphe 4.5.1.2 (explication du chiffre 3 inscrit dans le premier flit). Il correspond à un paquet de lecture mémoire. Le champ RPTT[3] indique le chemin de routage retour pour les configurations. Pour identifier la configuration au niveau du serveur, l'interface doit transmettre le contexte global (GCX), le numéro d'identification de la ressource (RES), de contexte (CTX) et de configuration (CFG).

4.4 Mise en oeuvre de l'interface réseau et caractérisation

Pour valider l'architecture de l'interface réseau précédemment décrite, nous avons développé un modèle VHDL. Dans cette section, nous ferons une analyse des temps

[3] *Return Path To Target*

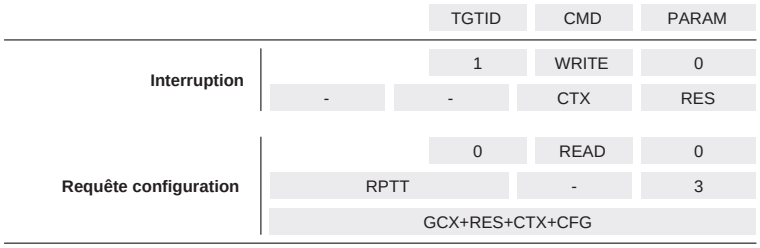


FIG. 4.14 – Formats des paquets de contrôle envoyés par l'interface

caractéristiques de fonctionnement de cette interface puis nous donnerons des résultats obtenus après synthèse de la description VHDL.

4.4.1 Description VHDL de l'interface réseau

L'interface réseau est décrite comme un système synchrone. Les ports d'entrée et sortie sont communs à toutes les ressources du réseau. Outre le signal d'horloge et de remise à zéro, il s'agit des signaux *data* (flit de 32 bits avec 2 bits de contrôle), *send* et *accept* (un par canal virtuel en provenance et vers le réseau). L'architecture de l'interface est modulaire. Les blocs de base sont assemblés en fonction des besoins de l'unité de traitement (nombre de mémoires FIFO d'entrée et de sortie). Les systèmes de contrôle locaux (SCL) et les systèmes locaux de gestion des configurations (SLGC) peuvent être adaptés en fonction de paramètres décrits au paragraphe 4.4.3.1. Ces deux systèmes implémentent des machines à états finis de type Moore. La Figure 4.15 présente le graphe d'état du séquenceur d'instructions du SCL.

4.4.2 Analyse des différentes latences de fonctionnement

Le modèle VHDL a été simulé pour caractériser les performances temporelles de l'interface. Le fonctionnement du séquenceur d'instruction, du gestionnaire de configuration et du gestionnaire de contexte sont régis par des machines à états finis. Leurs comportements dépendent à la fois de signaux internes à l'interface, comme la fin de l'exécution d'une configuration par exemple ou de signaux externes comme la réception d'une commande de démarrage pour un contexte. En fonction de la séquence et de la concomitance de ces événements, différents comportements temporels sont observés.

La Figure 4.16 présente un chronogramme construit sur la base des simulations effectuées. Nous mesurons le nombre de cycles d'horloge (période T) correspondant au traitement d'un contexte par le séquenceur d'instruction. Lorsque les paramètres

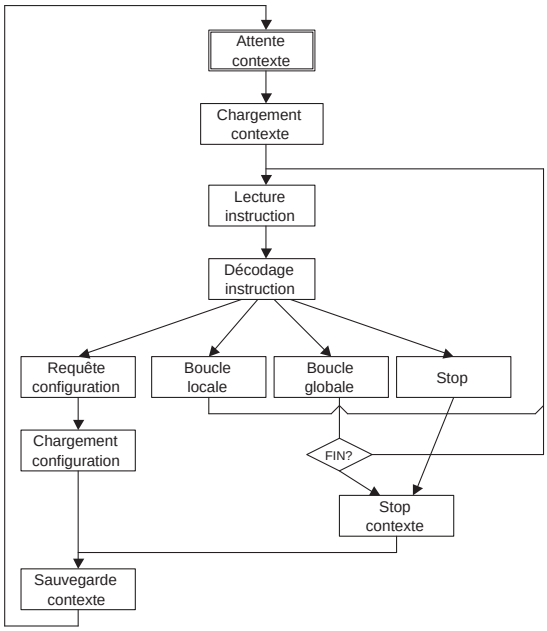


FIG. 4.15 – Graphe d'états du séquenceur d'instruction du SCL

de configurations (64 bits par configuration) sont déjà présents dans la mémoire de configuration du SLGC et que la séquence d'instruction ne contient par de boucle, il suffit de 12 cycles d'horloge pour charger le contexte, lire la configuration, la charger dans le contrôleur du NI correspondant au contexte et sauvegarder le contexte. Avec une horloge à 200 MHz, la fréquence de chargement des configurations est donc supérieure à 16 MHz. La gestion des instructions de boucle ajoutent quelques cycles puisqu'il faut lire plusieurs instructions avant d'obtenir une instruction de chargement de configuration. Le gestionnaire de contexte effectue un arbitrage de type *round-robin* pour sélectionner le prochain contexte à charger. Ce temps d'arbitrage peut varier de 3 à 6 cycles en fonction du nombre de contexte en cours d'exécution.

S'il est nécessaire d'envoyer une requête au serveur de configuration, le temps de chargement d'un contexte est plus long. Il dépend du temps de réponse du serveur de configuration et de la latence du réseau. Lorsque qu'une ressource est adjacente au serveur (dans le cas d'une topologie maillée à deux dimensions), un minimum de 32 cycles est nécessaire. Ce temps dépend de l'implémentation du serveur de configuration. Nous y reviendrons au paragraphe 4.5.1.2.

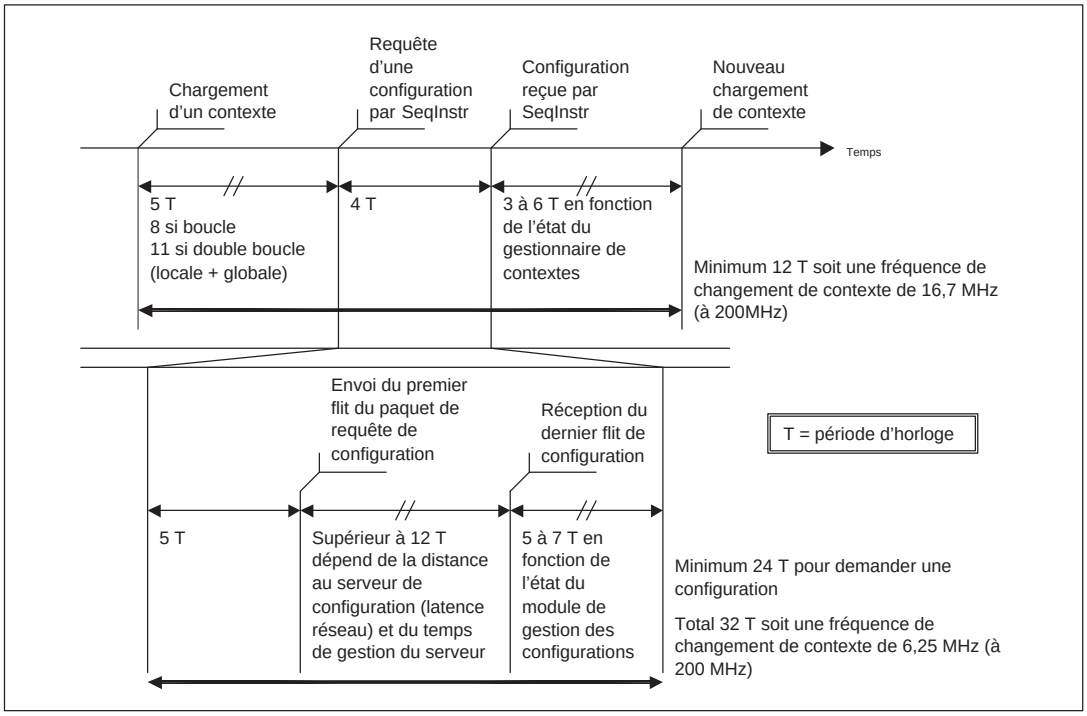


FIG. 4.16 – Chronogramme de fonctionnement de l'interface réseau

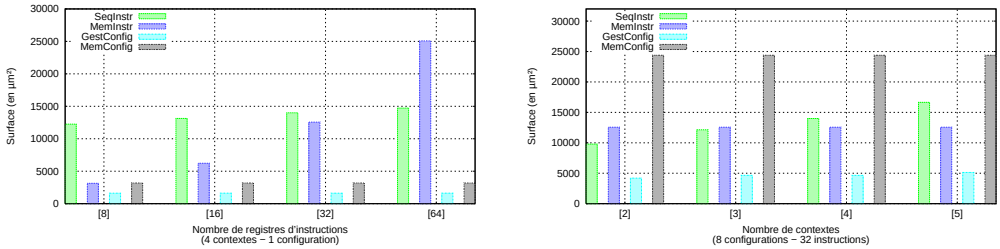
4.4.3 Résultats de synthèse physique

Nous avons choisi de développer un modèle VHDL de l'interface réseau de manière à évaluer de manière précise les conditions d'implémentation de cette architecture. Les modules VHDL ont été synthétisés avec une technologie CMOS 0,13μm de STMicroelectronics (HCMOS9). L'outil de synthèse est BuildGates Synthesis de Cadence. Nous verrons d'abord l'influence des paramètres d'instanciation sur la complexité du système, ensuite nous présenterons le coût de l'interface puis nous comparerons cette architecture d'interface à celle qui existe dans le circuit FAUST.

4.4.3.1 Étude de la complexité du système en fonction des paramètres d'instanciation

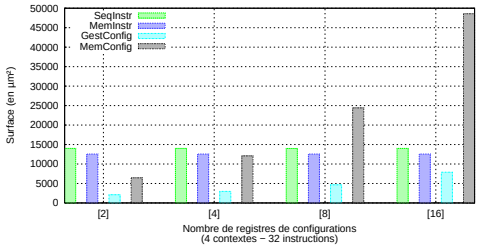
Le modèle VHDL de l'interface réseau a été développé de manière à offrir une grande modularité lors de la phase d'instanciation. Cela se traduit par différents paramètres génériques qui peuvent être ajustés en fonction des besoins de l'interface. Pour cette étude, nous prenons en compte trois paramètres :

- **Nombre de contextes**. Cette valeur est directement liée au nombre de contrôleurs du NI que l'on veut utiliser. Par exemple, pour une unité de traitement avec 2 flux d'entrée et 3 flux de sortie, il faut gérer 6 contextes (2 ICC, 3 OCC et l'unité de traitement).
- **Nombre d'instructions**. Il s'agit de la taille de la mémoire stockant les instructions de séquencement. Plus les scénarios réalisés par la ressource sont complexes, plus il faut pouvoir stocker d'instructions. Au niveau matériel, augmenter l'espace mémoire implique également une augmentation de la taille des bus d'adresse et aussi des registres de stockage d'adresse dans le séquenceur d'instruction (SeqInstr).
- **Nombre de configurations**. C'est le nombre de configurations qui peuvent être stockées dans la mémoire cache du SLGC. Les configurations ont une taille fixée à 64 bits. Le nombre de configurations correspond au nombre de registres de 64 bits dans la mémoire du SLGC. La détermination du nombre de registres répond à un compromis entre la limitation de l'espace mémoire et le nombre d'accès au cache infructueux qui entraîne une requête et affecte donc les performances temporelles du système.



(a) Nombre d'instructions

(b) Nombre de contextes



(c) Nombre de configurations

FIG. 4.17 – Résultats de synthèse pour différentes valeurs de paramètres génériques

La Figure 4.17 présente les résultats de synthèse pour différentes valeurs des paramètres précédemment cités. Nous nous intéressons aux modules principaux du système

de contrôle et de configuration que sont le séquenceur d'instruction (SeqInstr), la mémoire d'instruction (MemInstr), le gestionnaire de configuration associé au gestionnaire de mémoire de configuration (GestConfig) et la mémoire de configuration (MemConfig).

Comme nous pouvons voir sur la Figure 4.17(b), le nombre de contexte n'a pas une grande influence sur la surface des différents modules. Le principe de mutualiser les ressources d'un unique séquenceur pour plusieurs contextes s'avère donc bien un choix intéressant. En fait les Figures 4.17(a) et 4.17(c) font comprendre que le coût matériels est dominé par la surface des mémoires. Pour réduire la surface d'une interface, il faut donc déterminer avec précision la taille des mémoires. Le choix d'utiliser un serveur de configuration est donc également un bon choix pour limiter la surface des interfaces.

4.4.3.2 Étude en fréquence du système de contrôle et de configuration

Le système de contrôle et de configuration (SCL, SLGC et le décodeur de paquet) a été synthétisé avec différentes périodes d'horloge cibles afin de caractériser les performances temporelles. Les paramètres pour cette série de synthèses sont : 4 contextes, 6 registres de configuration et 24 instructions. Les résultats présentés sur le tableau 4.3 indiquent que le système fonctionne jusqu'à 500MHz. On peut donc considérer que le système de contrôle et de configuration ne sera pas un facteur limitant en fréquence lors de l'intégration complète des interfaces réseau avec une unité de traitement.

Tab. 4.3 – Résultats de synthèse pour différentes fréquences cibles

Surface totale (μm^2)	45907	46123	46496	51523
Période cible (ns)	5	4	3	2
Fréquence cible (MHz)	200,00	250,00	333,33	500,00
Marge temporelle[4] (ns)	1,03	0,03	0,01	**-0,02**
Fréquence réelle (MHz)	251,89	251,89	334,45	**495,05**

4.4.3.3 Synthèse de l'interface réseau complète

Dans cette section, nous présentons des résultats de synthèse détaillés pour une interface réseau complète. Les mémoires sont câblées avec des registres. La synthèse a été effectuée avec une fréquence cible de 300 MHz. L'architecture de l'interface est modulaire et paramétrable, l'idée étant de construire une interface spécifique à l'unité de traitement à laquelle elle se rattache. En fonction d'une unité de traitement, la

complexité de l'interface peut ainsi varier. Les résultats de synthèse correspondent donc à un cas particulier c'est pourquoi nous avons cherché à synthétiser une interface assez représentative de ce qui pourrait être implémenté dans un circuit. Nous avons construit une interface supportant 2 flux d'entrée et 1 flux de sortie, la connexion avec l'unité de traitement se faisant avec 3 mémoires FIFO (de 16 flits chacune). Le SCL gère 4 contextes (3 contrôleurs de communication et une unité de traitement) et peut stocker 30 instructions. Le SLGC contient 4 registres de configuration. Les résultats de synthèse pour le système de contrôle et de configuration (SCL et SLGC) sont donnés par le Tableau 4.4.

TAB. 4.4 – Résultats de synthèse le système de contrôle et de configuration

Modules	Surface (µm²)		Détails
SCL	32350	14580	Mémoire d'instructions
		15381	Séquenceur d'instruction
		1793	Générateur d'interruption
		596	Gestionnaire de contextes + Décodage
SLGC	18422	13334	Mémoire de configurations
		2772	Gestionnaire de mémoire
		1496	Gestionnaire de configuration
		734	Générateur de requête
		86	Gestionnaire de contexte global
Décodage de paquet	3948		
Divers	24	4 contextes, 4 registres de configurations, 30 instructions	
TOTAL	54744		

Le Tableau 4.5 présente les résultats au niveau de l'interface complète. Il montre très clairement que les mémoires FIFO représentent une part très importante dans la complexité de l'interface avec près de 45% de la surface. L'ensemble du système de controle et configuration ne représente pas plus d'un tiers de la surface. Avec une technologie $0{,}13\mu m$ comme celle utilisée pour le circuit FAUST (cf. paragraphe 2.3.4), la complexité de l'interface peut être estimée à 25k portes avec une densité de 128k portes/mm^2. Dans FAUST, un noeud réseau aysnchrone associé à une interface asynchrone/synchrone représente 35k portes (100k portes/mm^2). Avec des unités de traitement de taille moyenne de l'ordre de 300k portes, le coût des fonctions de contrôle, de configuration et de communication est proche de 20% (60k portes) de la surface de chaque ressource.

4.4.3.4 Comparaison avec l'interface FAUST

La comparaison entre l'interface mise en place dans le projet FAUST (cf. paragraphe 2.3.3.1) et cette nouvelle architecture d'interface réseau n'est pas aisée. En effet, il ne

TAB. 4.5 – Résultat de synthèse pour l'interface réseau

Modules	*Surfaces (μm^2)*	*% Surface*
SCL + SLGC + Décodage paquet	54744	31,92%
3 Mémoires FIFO	75988	44.31%
2 ICC	9243	5.39%
OCC	8154	4.75%
IP	10491	6.12%
OP	5971	3.48%
Divers	6897	4.02%
Total	**171488**	**100.00%**

s'agit pas d'une optimisation d'implémentation mais de modifications assez importantes en terme de fonctionnalités. Nous présentons différents axes de comparaison dans le Tableau 4.6.

TAB. 4.6 – Comparaison entre l'interface réseau FAUST et la nouvelle architecture

	NI FAUST	**Nouvelle NI**
Séquencement des communications et des traitements	−	+
Surface de silicium	+	−
Autonomie par rapport au processeur	−	+
Indépendance des flots de communication et des traitements	−	+

Les modules *Configurations Manager* (CFM) et *Read Write Decoder* (RWD) ne sont plus utilisés. Ils sont remplacés par le SCL et le SLGC. En terme de fonctionnalités, le CFM permet de séquencer seulement deux configurations pour les ICC. Les OCC et l'unité de traitement ne sont pas indépendants, le CFM charge les configurations en fonction des numéros de configurations associés aux commandes INIT_WRITE contenues dans l'en-tête des paquets de donnée. Les résultats de synthèse donnent une surface du CFM de $4381\mu m^2$ pour la gestion d'un ICC et $6037\mu m^2$ lorsqu'il y en a deux. Avec notre nouvelle architecture, le SCL permet un séquencement indépendant des configurations sur les ICC, OCC et l'unité de traitement. Cela représente une complexité bien plus importante, comme nous voyons sur le Tableau 4.4, en particulier due aux registres d'instructions. Cette augmentation de la complexité est en partie compensée par le

SLGC qui permet d'économiser des registres de configuration par rapport au RWD. En effet, il est maintenant possible de télécharger dynamiquement les paramètres de configuration et il n'est plus nécessaire de les stocker intégralement au niveau de l'interface.

4.5 Expérimentation : modélisation globale et validation du système de contrôle et de configuration

Depuis le début du chapitre nous avons décrit une approche pour contrôler et configurer des ressources de traitement interconnectées avec un réseau sur puce. Après avoir exposé les principes de fonctionnement, nous avons, dans la section précédente, décrit et caractérisé une architecture d'interface réseau qui est l'élément clé dans l'organisation du système. Dans cette section, nous allons maintenant présenter un environnement de simulation intégrant plusieurs ressources utilisant cette nouvelle interface. Cette expérimentation permet de valider le système dans son ensemble et d'évaluer ses performances.

Nous décrirons tout d'abord l'environnement de simulation, ensuite nous présenterons un scénario d'expérimentation puis nous discuterons des résultats obtenus.

4.5.1 Description de l'environnement de simulation mis en place

L'objectif recherché est de créer un environnement de simulation permettant de valider sur un scénario concret les différents mécanismes introduit dans la section 4.2 : chargement de séquences d'instruction et gestion de contextes au niveau des interfaces réseau, téléchargement de paramètres de configuration à partir d'un serveur...

Pour le réseau, nous utilisons le modèle TLM/SystemC de FAUST (cf. paragraphe 2.4.1). Le scénario de simulation est écrit dans un fichier sous forme de commandes qui sont lues et exécutées par une ressource modélisant le processeur central de contrôle. Ces commandes permettent de charger des séquences d'instruction pour les interfaces réseau et de démarrer les contextes. Elles gèrent également la réception d'interruptions de fin d'exécution pour un contexte donné.

Les ressources de traitement sont modélisées en VHDL. Des coquilles (*wrappers*) d'adaptation permettent de réaliser la cosimulation entre le réseau TLM/SystemC et les ressources VHDL. Pour la cosimulation, nous utilisons de préférence l'outil Modelsim de Mentor Graphics [MODE06] pour travailler sur les signaux VHDL mais également Incisive de Cadence [CADE06].

Un exemple de l'architecture modélisée est présenté sur la Figure 4.18. Les ressources de traitement sont représentées par RES1, RES2 et RES3. Les ressources GEN1 et GEN2 sont des générateurs de données pour créer du trafic vers les ressources de traitement.

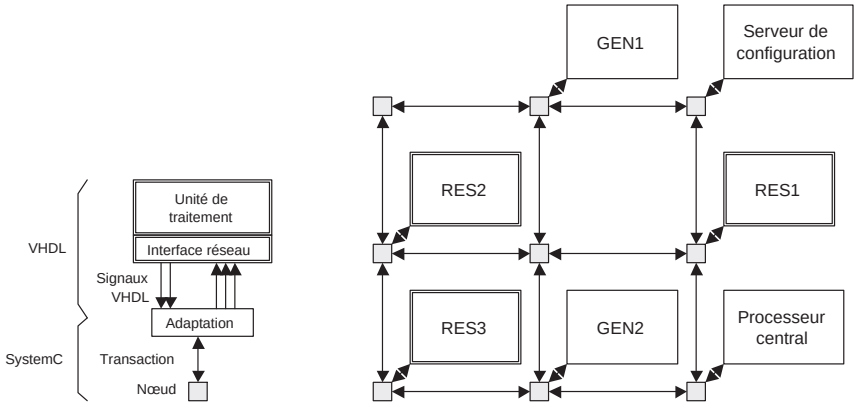


FIG. 4.18 – Architecture du réseau pour la modélisation globale

Dans la suite, nous allons détaillés la modélisation des ressources de traitement, le modèle du serveur de configuration et la configuration des interfaces réseau.

4.5.1.1 Modélisation des ressources de traitement

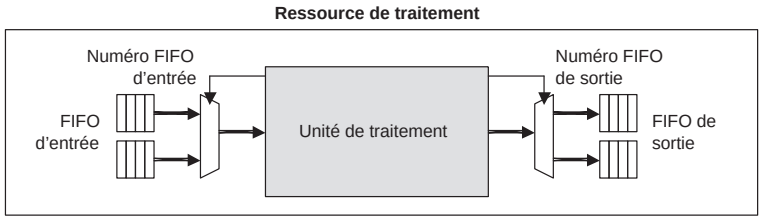


FIG. 4.19 – Structure des mémoires FIFO d'entrée et de sortie de la ressource de traitement

Pour la modélisation d'une ressource de traitement, nous devons associer les interfaces réseaux à des unités de traitement. Les différents modules de l'interface réseau sont décrits en VHDL (cf. paragraphe 4.4.1). Nous avons donc développé un modèle VHDL d'une unité de traitement générique pour pouvoir les connecter. Nous reprenons le principe de modélisation des unités de traitement du Chapitre 3. Seuls les flux de données sont modélisés et aucune donnée n'est réellement traitée. L'unité de traitement communique avec l'interface réseau au travers de mémoires FIFO (Figure 4.19). Les paramètres de configuration sont données par la Figure 4.20. Pour configurer l'unité, on

indique : le temps de traitement (en nombre de cycle), le nombre de données nécessaires au traitement, le nombre de données produites par le traitement. Pour la lecture et l'écriture des données, un numéro permet de sélectionner la mémoire source et destination. Les cadences de lecture et d'écriture peuvent être indépendamment ralenties à une fois tous les N cycles avec N comme paramètre de configuration.

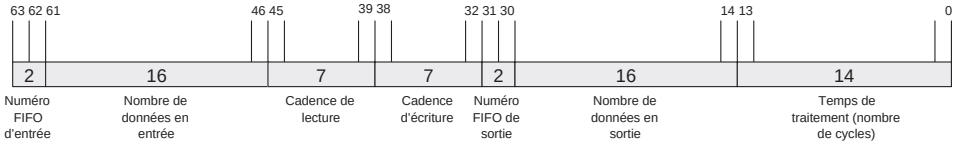


FIG. 4.20 – Paramètres de configuration des unités de traitement

4.5.1.2 Modèle du serveur de configuration

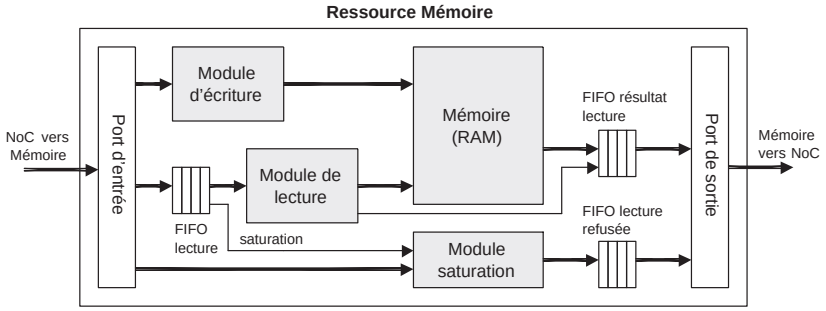


FIG. 4.21 – Structure de la ressource mémoire utilisée comme serveur de configuration

Le stockage des paramètres de configuration est géré par un serveur centralisé (cf. paragraphe 4.2.1.2). Pour modéliser ce serveur, nous avons développé en SystemC une ressource mémoire accessible en lecture et en écriture depuis le réseau. Le format des paquets correspondants est présenté sur la Figure 4.22 (il s'agit des mêmes notations décrites au paragraphe 4.3.4). Le champ RPTT[5] indique le chemin de routage vers la ressource ayant demandé la lecture. Le champ N précise le nombre de mots mémoire à lire. Le champ ADDR contient l'adresse de lecture ou d'écriture.

L'architecture de la mémoire est illustré sur la Figure 4.21. L'écriture mémoire est prioritaire sur la lecture. Le *module d'écriture* écrit donc les mots mémoire au fur est à mesure de leur réception ce qui garantit qu'il n'y a pas de blocage sur le réseau. Le *module de lecture* gère les requêtes de lecture qui sont stockées dans une mémoire

[5] *Return Path To Target*

FIFO. En cas de saturation de cette FIFO, les requêtes de lecture ne sont plus prises en compte. Le *module saturation* envoie des paquets pour indiquer aux ressources qui ont fait une requête de lecture que celle-ci est annulée.

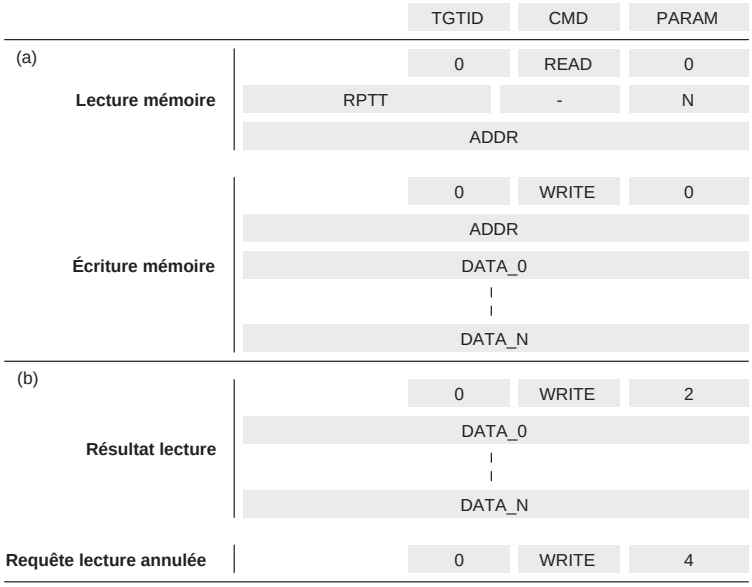


FIG. 4.22 – Format des paquets reçus (a) et émis (b) par la ressource mémoire

Pour utiliser cette ressource mémoire comme serveur de configuration, nous utilisons les requêtes de lecture comme requêtes de configuration. Le résultat de lecture renvoie alors les paramètres de la configuration. Les configurations sont précalculées et enregistrées dans un fichier. A l'initialisation de la simulation, ce fichier est chargée dans la ressource mémoire. Les mots mémoire font 32 bits. Une configuration de 64 bits correspond donc à la lecture de deux mots successifs. En fait, un troisième mot mémoire est utilisé pour stocker le numéro d'identification de la configuration CTX+CFG[6]. La configuration est stockée à l'adresse mémoire correspondant aux champs GCX+RES+CTX+CFG[7].

4.5.1.3 Configuration des interfaces réseau

C'est le processeur qui envoie les séquences d'instruction dans chacune des interfaces. Les instructions font références à des numéros de configuration qui sont stockés dans le

[6] cf. paragraphe 4.3.4
[7] cf. paragraphe 4.3.4

serveur de configuration. Le serveur contient des configurations pour les contrôleurs de communication (ICC et OCC) et pour les unités de traitement.

Pour modéliser l'interface réseau, nous avons utilisé directement les contrôleurs de communication de l'interface FAUST (cf. paragraphe 2.3.3.2). Les registres de configuration des ICC et des OCC sont chargés à partir d'un mot de configuration sur 64 bits comme présenté sur la Figure 4.23. Le registre correspond au nombre de boucles pour l'OCC est fixé à 1, les boucles étant maintenant gérées par le séquenceur d'instruction. Le format des configurations pour l'unité de traitement a été présenté précédemment (cf. paragraphe 4.5.1.1).

Configuration	63 ,,, 32	31 ,,, 0
ICC	reg_icc_lo1	reg_icc_if1
OCC	reg_occ_conf	reg_occ_header

FIG. 4.23 – Configuration des contrôleurs de communication

4.5.2 Validation globale du système pour la gestion des communications et le contrôle de l'application

Nous allons à présent proposer un scénario pour valider le fonctionnement de l'architecture et présenter des résultats de simulation.

4.5.2.1 Architecture et scénarios de validation

Nous utilisons l'architecture de réseau présenté sur la Figure 4.18. Nous avons défini un scénario expérimental d'échange de données entre les ressources illustré par la Figure 4.24. Les simples flèches représentent les crédits et les doubles flèches les données. A partir de la ressource RES1, les nombres indiquent la quantité de crédits ou données échangés avec respectivement entre parenthèse le seuil d'envoi des crédits ou la taille des paquets. Ce scénario permet de tester l'ensemble des instructions de séquencement.

Pour chaque contexte, de chaque ressource, une séquence d'instruction correspondant au chargement des configurations est précalculée. Pour ce scénario, le serveur de configuration doit gérer 15 configurations différentes. L'exécution du scénario représente l'échange et le traitement de 750 flits de données. A titre d'exemple, la Figure 4.25 présente la séquence d'instruction pour le contexte associé au contrôleur de communication d'entrée (ICC) de la ressource RES1. La configuration 1 correspond à l'envoi

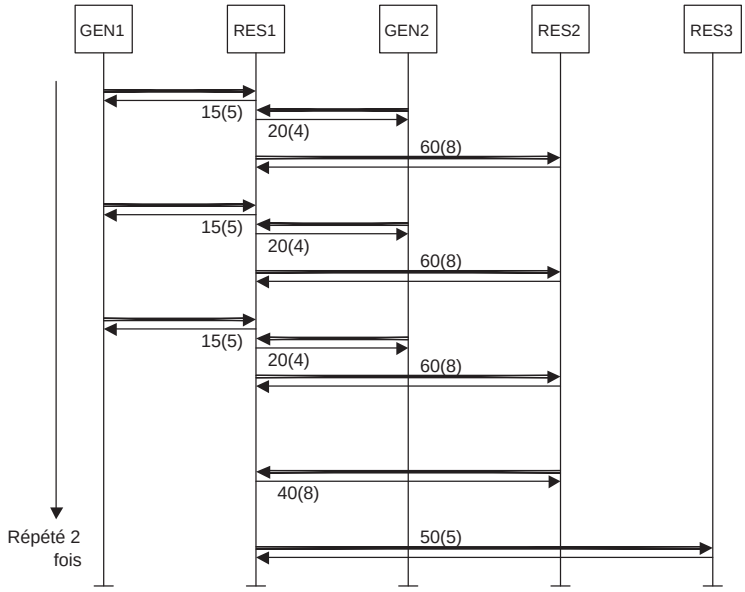


FIG. 4.24 – Scénario de simulation

de 15 crédits vers la ressource GEN1, la configuration 2 envoie 20 crédits vers GEN2 et la configuration 3 code pour 40 crédits vers RES2.

```
0 RCR 1
1 RCF 2
2 LLO 3
3 RCF 3
4 GLO 2
```

Chargement de la configuration 1, sauvegarde de l'adresse de retour
Chargement de la configuration 2
Boucle locale : 3 fois
Chargement de la configuration 3
Boucle globale : 2 fois

FIG. 4.25 – Exemple de séquence d'instructions pour l'ICC de la ressource RES1

4.5.2.2 Résultats de simulation

Le scénario a été simulé sur la base d'une horloge cadencée à 200MHz. Le système est fonctionnel, l'ensemble des paquets ont bien été échangés. Dans la suite, nous analysons les résultats selon trois points : le nombre de registres de configuration, le débit sur le réseau et la charge du serveur de configuration.

- Étude de l'influence du nombre de configurations stockées localement sur les performances du système

Les simulations ont permis d'étudier l'influence du nombre de registres de configuration dans le SLGC[8] sur le temps d'exécution du scénario. Ce nombre de registres

[8]Système Local de Gestion des Configurations

d'une taille fixée à 64 bits correspond directement au nombre de configurations que peut stocker la mémoire cache du SLGC. La problématique consiste à trouver un compromis entre une taille de mémoire cache réduite et un nombre d'accès limité (faible trafic réseau) aux serveur de configuration. Nous avons effectué des simulations en faisant varier le nombre de registres de 1 à 8. Les résultats sont donnés par la Figure 4.26. Avec 8 registres, toutes les configurations de chaque ressource peuvent être stockées localement. Les requêtes de configuration n'ont donc lieu que lors de leur premier chargement. Par la suite, lorsque l'on réduit le nombre de registre, la mémoire cache est plus souvent rafraîchie, il y a donc plus souvent des configurations à télécharger depuis le serveur, ce qui a pour conséquence d'augmenter le temps d'exécution du scénario. Cependant le temps nécessaire pour demander et recevoir une configuration est rapide par rapport au temps de son temps d'exécution : une configuration correspond au transfert de quelques flits sur le réseau alors qu'une configuration gère de plus grands nombres de flits. C'est pourquoi le temps d'exécution du scénario augmente de façon limité : dans l'exemple simulé, il est 25% plus long. Ce type de résultat permet d'avoir des informations utiles pour paramétrer les interfaces réseau.

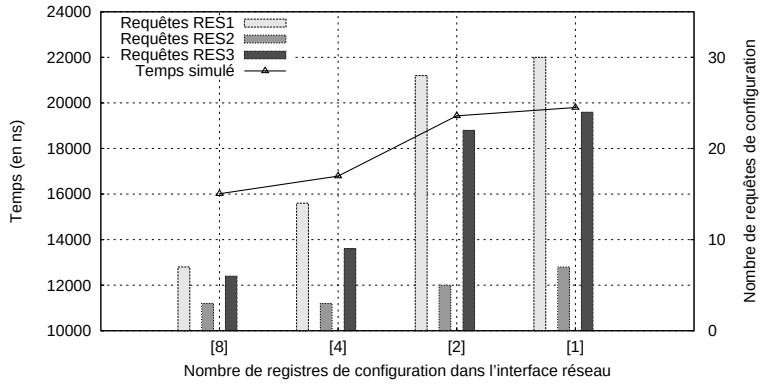


FIG. 4.26 – Temps simulé et nombre de requêtes de configuration en fonction du nombre de registres de configuration dans les interfaces réseau

- Étude du trafic réseau

Les simulations ont également été exploitées pour visualiser le trafic réseau. Nous présentons les résultats dans le cas où il y a un registre de configuration par ressource (Figure 4.27). Le débit correspond au calcul de débit décrit au paragraphe 1.4.1.1. La surcharge de trafic générée par les requêtes et les transferts de configuration est assez faible et bien répartie tout au long de la simulation ce qui limite les phénomènes de congestion. Le pic en début de simulation est dû au fait que tous les contextes

sont démarrés à quelques cycles d'intervalle ce qui concentre le nombre de requêtes de configuration.

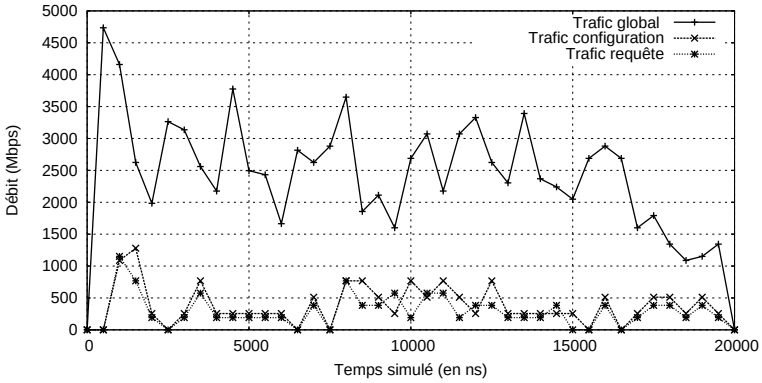


FIG. 4.27 – Trafics des requêtes et des configurations comparés au trafic global

- Étude de la charge du serveur de configuration

Un point également étudié lors des simulations est la charge du serveur de configuration. En effet, le serveur doit pouvoir répondre aux requêtes en provenance de toutes les ressources. Pour évaluer la charge du serveur, nous calculons son taux d'inactivité qui correspond au rapport entre le temps d'attente entre l'envoi de deux configurations et le temps écoulé entre c'est deux envois. Le modèle SystemC utilisé permet d'envoyer une configuration toutes les 80ns. si par exemple, le serveur envoie une configuration toutes les 100ns, son taux d'inactivité est donc de 20%. On constate sur la Figure 4.28 qu'avec le scénario utilisé le serveur peut largement répondre à la demande.

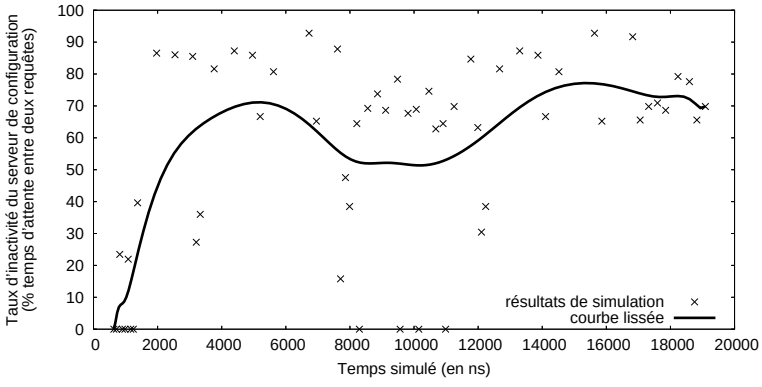


FIG. 4.28 – Étude de la charge du serveur de configuration

Dans le cas d'un réseau plus grand avec un nombre de ressources plus important, la

charge du serveur de configuration ne sera pas un goulot d'étranglement. En effet, les mécanismes actuels permettent un fonctionnement avec plusieurs serveurs de configuration ce qui permet de répartir la charge. Ainsi on peut imaginer pour l'implémentation d'un modem, un serveur pour les configurations de la chaîne d'émission et un second serveur pour la chaîne de réception. Ceci permet de plus de modifier les paramètres de configuration de l'émetteur sans perturber le récepteur ou inversement.

4.6 Conclusion

Nous avons présenté une structure de contrôle hiérarchique (semi-distribuée) qui répond à notre problématique de gestion d'une application et des communications associées pour un système de traitement construit autour d'un réseau sur puce. Cette structure s'appuie sur différents éléments matériels avec en particulier une architecture d'interface réseau. La charge du système de contrôle central est réduite grâce aux mécanismes de séquencements locaux qui augmentent l'autonomie des interfaces. Un serveur de configuration qui stocke les paramètres de configuration permet de limiter la surface des interfaces réseau. La structure de l'interface est modulaire pour optimiser l'intégration de ressources dans un SoC hétérogène. La solution est parfaitement compatible avec une approche NoC puisqu'elle n'impose pas de nouvelles structures de communication. Les résultats de synthèse montrent que notre architecture d'interface réseau est intégrable avec un coût raisonnable. Une modélisation globale du système valide les mécanismes de fonctionnement du systèmes et fournissent des résultats en terme de performances et de dimensionnement des paramètres d'instanciation.

Conclusions et perspectives

Aujourd'hui, la possibilité de concevoir un système sur puce complexe et multi-application est étroitement liée à la capacité d'intégrer une structure de communication performante associée à des méthodologies de conception et de programmation efficaces. Le concept de réseau sur puce (NoC), rendu possible par l'évolution de la technologie silicium, est en passe de devenir une solution privilégiée pour simplifier l'intégration d'IP pour des systèmes sur puce (SoC) devant supporter un trafic de données important.

Cependant, comme toute nouvelle architecture, les NoC requièrent des efforts de recherche importants. Les concepteurs doivent faire face à de nouvelles problématiques d'implémentation : protocole de communication, topologie, niveau de qualité de service. De plus, les réseaux sont des systèmes distribués ce qui introduit de nouvelles contraintes au niveau de la synchronisation des échanges de données, du contrôle des traitements et de la prévisions des performances.

Au LETI, les architectures NoC sont étudiées dans l'objectif de proposer une plate-forme flexible et reconfigurable pour implémenter des systèmes de télécommunication de quatrième génération (4G) notamment au travers des projets MATRICE et 4MORE. Cette démarche structurée autour du projet FAUST a abouti à la réalisation d'un premier circuit basé sur un NoC asynchrone.

Les travaux de thèse que nous venons de présenter s'inscrivent également dans ce double contexte : les architectures de réseaux sur puce et les applications de télécommunication sans-fil haut-débit. L'étude des concepts d'implémentation associés aux NoC, le réseau FAUST, le cas spécifique des applications de télécommunication traitées dans le cadre du projet FAUST nous ont permis d'aborder de nouvelles problématiques concernant la modélisation des NoC, la gestion des communications et le contrôle des traitements.

En matière de modélisation de réseau sur puce, notre contribution est basée sur l'outil de simulation NS-2 adapté aux mécanismes de communication du réseau FAUST.

Nous avons ainsi pu simuler le trafic de données sur le réseau pour une chaîne de traitement MC-CDMA [LCDL05]. Ceci nous a permis de valider les performances du réseau au niveau débit et latence et ainsi de démontrer la capacité d'un NoC à supporter une application complexe au niveau des flots de communication. Nous avons en particulier montré des pics de débit à 20Gbps sur le réseau. Les résultats ont été confirmés par une seconde étude réalisée à partir d'un modèle SystemC du réseau FAUST.

Pour la gestion des communications et le contrôle des traitements, nous proposons une approche qui s'articule autour d'une architecture nouvelle d'interface réseau [LDLJ06]. Notre contribution repose sur l'implémentation d'un séquenceur d'instruction au niveau de chaque interface réseau permettant une reconfiguration dynamique des flots de communication et des traitements applicatifs. Les paramètres de configuration sont obtenus de façon autonome par les interfaces qui les téléchargent à partir d'un serveur connecté au réseau. Un système centralisé répartit les tâches de communication et de traitement au niveau de chaque ressource en programmant les instructions de chargement de configuration dans les interfaces. Une description VHDL de l'interface permet d'obtenir des résultats de synthèse en fonction de différents paramètres d'instanciation. L'ensemble de cette approche a été modélisé et simulé dans un environnement mixte VHDL et SystemC.

Notre travail de thèse permet une avancée dans le contexte de l'implémentation d'applications de télécommunication sur une architecture de réseau sur puce. Notre volonté était de répondre aux mieux aux contraintes des standards en cours de développement tout en anticipant les évolutions futures.

Ainsi comme perspectives pour nos travaux, nous souhaitons dans un premier temps valider notre système de contrôle des traitements et de gestion des communications dans le cas d'une application complète comme 4MORE mais également IEEE 802.16e et 802.11n par exemple. Ensuite, nous envisagerons des applications nécessitant des flots de données plus complexes. Nous avons pu constater que le passage d'un mode SISO à un mode MIMO (à deux antennes) entre MATRICE et 4MORE a multiplié les flots de données. Il est fort probable qu'à l'avenir un plus grand nombre d'antennes soit utilisé ce qui engendrera des chaînes de traitement encore plus ramifiées. Dès à présent, nous pourrions simuler comment notre approche se comporterait avec ce type de contraintes.

Au niveau de l'architecture de l'interface réseau différentes pistes d'évolution sont dès à présent possibles. La gestion du stockage des configurations peut être optimisée par une politique de mémoire cache tenant compte du contexte et des instructions de boucle. Il est également possible d'ajouter des instructions de séquencement pour réali-

ser des scénarios plus complexes. Pour le moment, les configurations sont de tailles fixes. certaines unités de traitement ont besoins de nombreux paramètres pour être configurées. Des mécanismes permettant de supporter des tailles de configurations variables sont donc envisageables.

En ce qui concerne le serveur de configuration. Il faut à présent définir son architecture et passer d'un modèle SystemC haut-niveau à une implémentation matérielle. La gestion des adresses mémoires peut être optimisée. On peut également imaginer un serveur intelligent dans lequel les configurations ne sont pas stockés en mémoire mais générées de manière dynamique en fonction de paramètres globaux de l'application (modulation, nombre de codes, nombre de sous-bandes, etc.).

Avec le système actuel, il est possible (par reconfiguration) de gérer plusieurs applications séquentiellement mais pas simultanément. Il serait intéressant d'étudier des mécanismes permettant de faire cohabiter plusieurs chaînes de traitement en parallèle avec un partage des ressources pour rendre le système multi-applicatif. Dans ce but, chaque ressource devra être en mesure de basculer d'une application à l'autre en fonction des données à traiter disponibles. Il faudra également enrichir le protocole pour permettre aux ressources de synchroniser leurs états.

A plus long terme, nous pourrions aborder d'autres domaines applicatifs (traitement d'image, vidéo, etc) et étudier comment notre système pourrait être adapté.

Annexe A

Les NoC comparés aux structures d'interconnexion classiques

Dans cette annexe, nous présentons des éléments pour comparer les architectures de réseaux sur puces d'une part aux réseaux non intégrés et d'autre part aux solutions à base de bus partagés.

A.1 Réseaux sur puce et réseaux non intégrés

Les travaux antérieurs sur les réseaux informatiques peuvent être utiles aux concepteurs de NoC cependant la plupart des contraintes qui déterminent les choix d'architecture sont différentes.

Une première distinction apparaît au niveau de l'organisation topologique du réseau. Un réseau non-intégré n'est pas connu à l'avance. Il est conçu pour pouvoir ajouter ou retirer des noeuds ou des ressources de traitement. Ses mécanismes de fonctionnement doivent donc prendre en compte une topologie variable. Au contraire, un réseau sur puce est entièrement déterminé au moment de sa fabrication. Son architecture est figée dans le silicium. Cela a pour conséquence d'apporter des simplifications au protocole et ainsi d'alléger l'implémentation en terme de ressources matérielles et logicielles et également de diminuer la latence des communications [WiLi03].

Du point de vue implémentation les contraintes sont également différentes : au niveau des noeuds, les connexions ne sont pas limitées par une mise en boîtier des circuits, cela rend possible des liens de 256 bits en parallèle contrairement aux 8 à 16 bits des réseaux

non-intégré [DaTo01]. A l'inverse les ressources mémoires sont plus limitées et il faut choisir des algorithmes qui réduisent le stockage de données.

Dans les réseaux informatiques, les informations sur l'état de congestion d'un noeud ne peuvent pas être transmises de façon instantanée à ses proches voisins puisqu'il faut également utiliser des paquets [YeBM03b]. Au contraire dans un NoC il est possible de transmettre des informations de congestion en utilisant des fils dédiés comme c'est le cas avec le mécanisme de *Proximity Congestion Awareness* (PCA) [NMOJ03].

Au niveau des interfaces réseaux la différence est également importante. En effet, sur les réseaux classiques, un processeur dédié est souvent en charge de la gestion du protocole de communication. Pour un NoC, cela est plus difficilement envisageable car la taille de l'interface réseau serait alors comparable à celle des unités de traitement. Il faut donc plutôt avoir recours à une solution matérielle pour exécuter les fonctions du réseau.

Nous présentons maintenant cinq aspects problématiques qui entrent en compte dans le coût du réseau et qui sont discutés dans [RaGo02] :

- Fiabilité des communications : Dans un NoC, les liens physiques sont beaucoup plus fiables que dans les réseaux traditionnels. Il est donc possible d'assurer l'intégrité des données au niveau de la couche liaison de donnée par des mécanismes peu coûteux.
- Interblocage : Étant donné que la topologie des NoC est fixée dans le silicium, les interblocages peuvent être évités en utilisant un algorithme de routage adapté sans avoir recours à la suppression de paquets comme dans les réseaux traditionnels.
- Ordonnancement des données : Contrairement aux réseaux non-intégrés où il est typique de réordonner les données à la réception, il est plus pertinent d'imposer que les paquets soient reçus dans l'ordre pour réduire les mémoires et les ressources matérielles.
- Mode de commutation et stratégie de stockage : Pour limiter la taille des mémoires dans les noeuds, il est intéressant d'utiliser des commutations de type *wormhole* avec file d'attente en entrée alors que des techniques plus coûteuses sont plus facilement utilisées sur des réseaux classiques.
- Garanties temporelles : Les réseaux non-intégrés offrent des services basés sur la commutation de paquets et ne peuvent assurer un débit qu'en ayant recours à des mécanismes coûteux en mémorisation. Dans les NoC, la commutation de circuit couplée à du multiplexage temporel est une solution pour garantir un débit et une latence à un coût moindre.

A.2 L'approche NoC comparée au bus

Les réseaux sur puce sont souvent présentés comme une alternative aux interconnexions traditionnelles utilisant un système de bus. Différents éléments de comparaison entre bus et NoC sont régulièrement invoqués dans les publications [GuGr00] [WiGo02] [BeMi02]. Le Tableau A.1 présente une synthèse des arguments contre ou en faveur des réseaux sur puce d'un point de vue qualitatif.

Tab. A.1 – Comparaison des approches Bus et NoC

	BUS		NoC	
Implémentation Physique				
Géométrie	-	Les fils d'interconnexion sont globaux, longs et parasités par chaque ressource de traitement. Ceci entraîne des difficultés au routage et une fréquence de fonctionnement limitée	+	La géométrie est régulière. Les propriétés physiques et électriques sont prévisibles. Les liaisons courtes point à point permettent un fonctionnement à fréquence élevée.
Énergie	-	Les données doivent parvenir à tous les récepteurs potentiels ce qui entraîne inutilement une consommation importante d'énergie	+	Il est possible de couper l'alimentation de certaines zones du réseau en pause
Structure de communication et de contrôle				
Temps	+	Temps de transferts constants et relativement courts	-	Les temps de transfert augmentent proportionnellement avec le nombre de liens et de nœuds entre les modules
	-	Temps d'accès proportionnel au nombre de maîtres reliés au bus	+	Le temps d'accès au nœud est court
Arbitrage	-	Les mécanismes d'arbitrage dégradent les performances du bus	+	Les décisions de routage sont distribuées et basées sur des informations locales
	-	Le délai d'arbitrage augmente avec le nombre de modules connectés	+	Le même nœud de routage peut être instancié quelle que soit la taille du réseau
Concepts	+	Simples et bien compris. Différents standards permettent d'intégrer de nombreuses IP compatibles	-	Les équipes de conception doivent s'habituer aux nouveaux concepts amenés par les NoC
Qualité de service				
Blocage	+	Il n'y a pas de problèmes d'interblocages	-	La gestion des interblocages est à prendre en compte
Bande-passante	-	La bande-passante est partagée entre toutes les ressources. Elle n'augmente pas lorsque le système grandit	+	La bande-passante globale augmente avec la taille du réseau

Peu d'études proposent une approche quantitative pour comparer les performances d'une architecture basée sur un bus avec une architecture NoC. De plus les résultats sont souvent biaisés car ils dépendent beaucoup d'hypothèses de départ portant sur le choix des architectures de bus et de NoC. [BCGK03] présente une modélisation mathématique fournissant des fonctions de coût en terme de surface, consommation et fréquence de fonctionnement qui tendent à démontrer que le NoC est plus économique. Dans [ZKCS02], une autre modélisation compare un bus générique avec un modèle de

réseau en tore. Il apparaît que le NoC est plus performant qu'un bus pour des systèmes comportant plus de deux douzaines de ressources ou lorsque les applications échangent des informations localement (messages parcourant moins de trois liens en moyenne). Une comparaison du même type montrant les avantages du NoC en terme de fréquence de fonctionnement et de performances est faite dans [Arte06]. Une simulation comparative entre trois architectures : un bus, un bus hiérarchisé et un NoC est présentée dans [AMCB06]. La modèle de trafic utilisé n'est pas explicité ce qui limite la pertinence des résultats en particulier en ce qui concerne les latences de transferts.

Annexe B

Introduction au langage SystemC et à la Modélisation TLM

Cette annexe a pour objectif de présenter une rapide introduction au langage SystemC et à la modélisation TLM.

B.1 Présentation du langage SystemC

SystemC est un langage de description matériel qui a pour objectif de modéliser des systèmes numériques matériels et logiciels. L'OSCI (*Open SystemC Initiative*) a pour but de diffuser, promouvoir et rédiger les spécifications de SytemC. Cette organisation rassemble de nombreuses sociétés et laboratoire de recherche. Une description complète de SystemC est présenté dans le manuel de référence [OSCI06]. Dans le cadre de cette thèse, nous avons travaillé sur la version 2.0.1, la version 2.1 est maintenant disponible.

SystemC est une extension du langage C++ qui se présente sous la forme d'une bibliothèque de classes contenant différentes briques de base pour modéliser un système matériel (Tableau B.1). Cette bibliothèque contient également un moteur de simulation événementiel ce qui permet de se passer d'un simulateur externe. Un modèle SystemC s'exécute donc comme un programme C++ après compilation. Les types de données usuels C++ ont été enrichis par des types de données adaptées au matériel (logique binaire, décimaux virgule fixe...). Les autres éléments de la bibliothèque se répartissent entre les éléments structurels qui représentent la structure des systèmes matériels (modules, ports) et les éléments de communication entre processus et modules (événements, canaux).

TAB. B.1 – Organisation de SystemC d'après [GLMS02]

Canaux primaires Signal, FIFO, Mutex, Sémaphore, etc	
Éléments structurels Modules, Ports, Interfaces, Canaux, Évènements	**Types de données** Logique 4 états (01XZ), Bits, Vecteurs de bits, Nombres virgule fixe,
Moteur de simulation événementiel	
C++ Langage Standard	

B.2 Introduction à la modélisation TLM

La modélisation TLM (*Transaction Level Modeling*) se définit comme un certain niveau d'abstraction dans la modélisation d'un système matériel. Ces différents niveaux d'abstraction coïncident généralement avec le flot de conception et sont présentés sur la Figure B.1. Nous allons décrire ces différents niveaux du plus élevé (abstrait) au plus bas (concret).

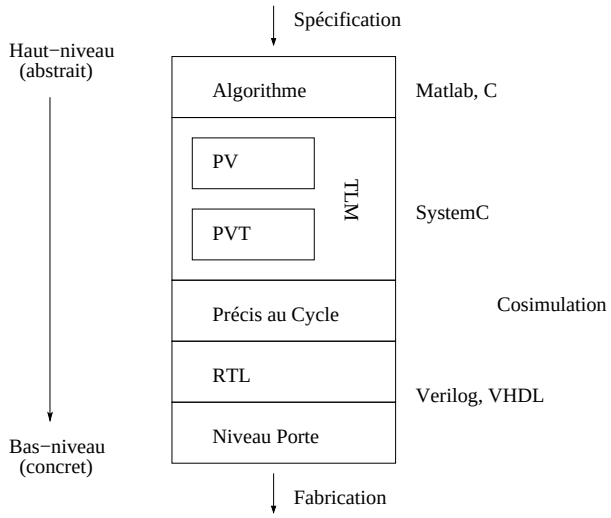


FIG. B.1 – Différents niveaux d'abstraction pour la conception d'un SoC

- Algorithme. C'est le niveau d'abstraction le plus élevé. Seul les algorithmes sont décrits sans référence à une architecture matérielle ou logicielle. Il n'y a pas de traitements en parallèle.

- TLM : *Transaction Level Modeling*. A ce niveau, le système est modélisé comme une plate-forme composée de plusieurs modules. La communication entre les modules est réalisée avec un modèle d'interconnexion constitué de canaux de communication dont le rôle est de conduire un élément de donnée d'un module à l'autre. Cet échange unitaire de donnée constitue une *Transaction*. On distingue deux niveaux de modélisation TLM : *Programmer View* (PV) et *Programmer View plus Timing* (PVT). Dans le premier cas, aucune information temporelle n'est prise en compte. Le modèle est purement fonctionnel. Chaque module exécute des tâches. Les transactions sont utilisé à la fois pour échanger des données mais également pour synchroniser les tâches. Dans le second cas, des informations temporelles sont ajoutées sur les traitements et les transferts. La fonctionnalité entre les modèles PV et PVT n'est pas changée par contre le second modèle va pouvoir donner de bonnes approximations sur le fonctionnement temporel du système. Pour travailler au niveau TLM, des bibliothèques additionnelles ont été développés en particulier par STMicroelectronics [CGJM03].

- Précis au Cycle (*Cycle-accurate* - CA). L'étape suivante dans le flot de conception d'un système synchrone est d'ajouter l'information d'horloge. Un modèle précis au niveau cycle décrit ce qui se passe à chaque cycle d'horloge.

- RTL : *Register Transfer Level*. Le niveau RTL est le premier niveau d'abstraction assez précis dans le flot de conception pour permettre une synthèse automatique. Il est précis au niveau cycle et également dans l'affectation des bits pour chacun des signaux (*bit-accurate*). A ce niveau d'abstraction, les langages du type VHDL ou Verilog sont généralement préférés au SystemC. Cependant SystemC permet la cosimulation ce qui permet de remplacer progressivement des modules du modèle TLM par leurs équivalents RTL et ainsi d'assurer la continuité du flot de conception.

- Niveau Porte (*Gate Level*). Il s'agit de la suite classique du flot de conception qui consiste à synthétiser le modèle RTL pour obtenir une liste des connexions (*netlist*) des portes de base fournies par la bibliothèque de la technologie utilisée.

Annexe C

Présentation du simulateur NS-2

Dans cette annexe, nous présentons l'outils de simulation de réseau NS-2.

C.1 Introduction

NS-2 (*Network Simulator* - version 2 - [NS206]) est un simulateur à événement discret orienté objet traditionnellement utilisé dans le domaine de la recherche sur les réseaux informatiques. Il permet en particulier des définir des protocoles de communication et d'étudier le trafic sur les différents éléments d'un réseau (filaire ou non). Il est fourni avec l'outil NAM (*Network ANimator*) qui donne une visualisation graphique de la topologie du réseau et des données échangées. NS-2 a été distribué pour la première fois en 1996. Il repose sur les travaux de développement des simulateurs REAL [REAL06] et NS-1. NS-2 intègre des changements d'architecture important par rapport à NS-1, il est maintenant basé sur l'association des deux langages de programmation C++ et OTcl. OTcl [OTcl06] est une extension du langage de commande Tcl/Tk pour la programmation objet.

Le développement de NS-2 est financé sur des fonds américains. Dans un premier temps, il s'agissait du projet projet VINT (*Virtual InterNetwork Testbed*) initié par la DARPA[1] et réunissant l'USC/ISI[2], Xerox PARC, Lawrence Berkeley National Laboratory et UC Berkeley. Plus récemment, les travaux sur NS-2 ont été poursuivi à l'USC/ISI avec le projet SAMAN (*Simulation Augmented by Measurement and Analysis for Networks*) de la DARPA et le projet CONSER (*Collaborative Simulation for Education and Research*) de la NSF[3]. NS-2 a également reçu des contributions de nom-

[1] Defense Advanced Research Projects Agency
[2] University of Southern California / Information Sciences Institute
[3] National Science Foundation

breux autres groupes de recherche dans le domaine. NS-2 est un logiciel libre, qui peut être utilisé gratuitement pour des travaux de recherche. Il fonctionne sur de nombreuses plate-forme (FreeBSD, Linux, SunOS, Solaris). Dans le cadre des travaux présentés dans ce mémoire, nous avons travaillé sur la version 2.26 (datant du 26 février 2003). Au moment où nous écrivons ces lignes, la version 2.30 est sur le point d'être disponible.

C.2 Organisation du simulateur

Le simulateur NS-2 repose sur l'utilisation de deux langages de programmation C++ et OTcl. Du point de vue de l'utilisateur, NS-2 est un interpréteur de commandes OTcl qui comporte un ordonnanceur d'événements et une bibliothèque des composants réseaux. L'utilisation de NS-2 se fait en trois temps (Figure C.1) :

- L'utilisateur écrit un script OTcl pour définir la topologie du réseau, pour instancier les différents modules du réseau, pour décrire des scénarios de trafic.
- NS-2 interprète le script et exécute la simulation. Les résultats sont stockés dans des fichiers en fonction des commandes du simulateur utilisées.
- Une fois la simulation terminée, les résultats peuvent être analysés directement avec l'outil de visualisation NAM ou avec des outils additionnels.

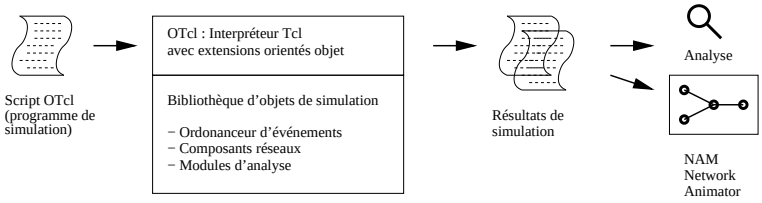


FIG. C.1 – Flot de simulation avec NS-2

Les composants du réseau et l'ordonnanceur d'événements sont codés et compilés en langage C++ ce qui permet une meilleure rapidité d'exécution. Ces objets compilés sont accessibles à partir du script OTcl grâce à une interface de liaison écrite en Tclcl (Tcl avec prise en charge des classes C++). Les objets C++ peuvent donc être directement contrôlés par l'utilisateur sans qu'il ait à modifier le code source. C'est l'un des grands atouts de NS-2 : fournir une bibliothèque de composants compilés que l'on peut exploiter de manière très souple à partir de scripts de commandes.

C.3 Architecture du réseau

NS-2 permet de construire des réseaux fonctionnant sur le principe de la commutation de paquets. Le paquet est l'élément d'information qui circule sur un réseau. Il comporte un en-tête avec les paramètres spécifiques au protocole utilisé. Il contient également un espace qui modélise les données échangées entre les composants du réseau. Pour définir un réseau, il faut assembler différents composants que nous allons décrire dans les paragraphes qui suivent.

- Noeud

Le noeud (classe OTcl : *Node*) est composé de *Classifiers* et d'*Agents*. Les *Classifiers* démultiplexent les flux des paquets. Les Agents gèrent le protocole. Ils sont les producteurs et les consommateurs de paquets. NS-2 fournit des Agents modélisant de nombreux protocole (TCP, UDP...). Lorsqu'un paquet arrive dans le noeud, il est orienté vers un des liens raccordés au noeud en fonction de son adresse de destination. Chaque noeud possède une adresse (*id_*). Si l'adresse est celle du noeud, le paquet est reçu par l'Agent dont le port correspond. Il n'y a pas de durée modélisée pour ces traitements au niveau du noeud.

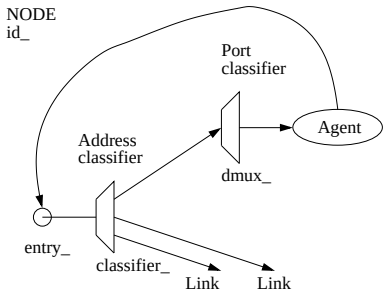


FIG. C.2 – Composant d'un noeud NS-2

La Figure C.2 illustre l'organisation d'un noeud. Les noms suivis du caractère « _ » correspondent à des variables internes de la classe *Node*. *Entry_* pointe sur le premier *Classifier* à l'entrée du noeud. Cette entrée est utilisée à la fois par les paquets venant d'un autre noeud et par les Agents raccordés au noeud.

- Lien

Les liens servent à raccorder les noeuds entre eux. Les caractéristiques principales d'un lien sont sa bande passante et le délai de propagation (géré par l'élément *link_*). Un lien contient également (cf. Figure C.3) une file d'attente (*queue_*) avec différentes

politiques de gestion possible, un élément pour gérer les pertes de paquets (*drophead_* en fonction de l'état de la file d'attente et un élément pour gérer la durée de vie des paquets (*ttl_*).

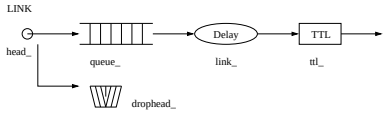


FIG. C.3 – Composant d'un lien NS-2

- Application

La classe *Application* modélise les sources et les puits de trafic. Une interface standard permet de connecter un objet Application à un Agent. L'Agent gère l'encapsulation en paquets des données générées par l'Application. NS-2 propose des modèles d'Application réseaux classiques (Telnet, FTP...) et des générateurs de trafic aléatoire (distribution CBR[4], Exponentielle...).

C.4 Principes d'utilisation

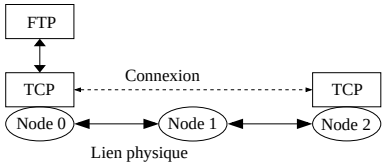
Pour clore cette présentation du simulateur NS-2, nous allons présenter un exemple simple de modélisation d'un réseau. L'objectif est de montrer comment à partir d'un script OTcl, il est possible de définir un réseau, de générer un trafic et de lancer une simulation. Les quelques lignes de code reproduites ci-dessous permettent de réaliser un réseau à trois noeuds. Deux Agents prenant en charge une connexion TCP sont associés aux noeuds des extrémités du réseau. Une Application de type FTP génère un trafic de données. Le scénario consiste à faire fonctionner cette source de trafic pendant une seconde.

```
# Création d'une instance du simulateur
$set ns [new Simulator]

# Création de 3 noeuds
$set n0 [$ns node]
$set n1 [$ns node]
$set n2 [$ns node]

# Création des liens entre les noeuds
$ns duplex-link $n0 $n1 1Mb 50ms DropTail
```

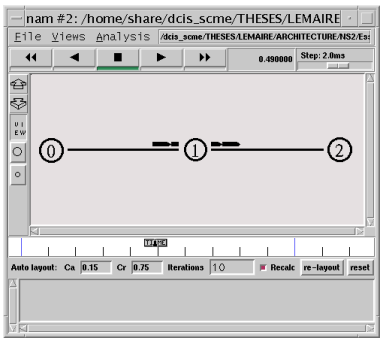
[4]Constant Bit Rate

```
$ns duplex-link $n1 $n2 1Mb 50ms DropTail

# Création d'agents TCP sur les noeuds
$set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
$set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n2 $tcpsink0

# Connexion des agents
$ns connect $tcp0 $tcpsink0

# Création d'une source de trafic
$set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

# Création d'un scénario de simulation
$ns at 0.0 "$ftp0 start"
$ns at 1.0 "$ftp0 stop"
$ns run
```

La Figure C.4 donne une vision schématique du réseau ainsi modélisé ainsi qu'une capture d'écran de l'outil d'animation NAM. Les liens sont configurés pour offrir une bande-passante de 1Mbps avec un délai de 50ms.



(a) Architecture du réseau



(b) Capture d'écran NAM

FIG. C.4 – Exemple de modélisation NS-2

Bibliographie

[4MOR06] « 4MORE » – `http://www.ist-4more.org`.

[ACGM03] A. Adriahantenaina, H. Charlery, A. G. L. Mortiez et C. A. Zeferino – « SPIN : a scalable, packet switched, on-chip micro-network », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Mars 2003.

[AHKB00] V. Agarwal, M. S. Hrishikesh, S. Keckler et D. Burger – « Clock rate vs. IPC : The end of the road for conventional microprocessors », *Proc. 27th Annual International Symposium on Computer Architecture*, Juin 2000, p. 248–259.

[AMBA06] « AMBA Home Page » – 2006, `http://www.arm.com`.

[AMCB06] F. Angiolini, P. Meloni, S. Carta, L. Benini et L. Raffo – « Contrasting a noc and a traditional interconnect fabric with layout awareness », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Mars 2006, p. 124–129.

[ARM06] « ARM946E-S » – 2006, `http://www.arm.com`.

[Alam98] S. Alamouti – « A simple transmit diversity technique for wireless communications », *IEEE Journal on Selected Areas in Communications* **16** (1998), p. 1451–1458.

[AnGr03] A. Andriahantenaina et A. Greiner – « Micro-network for SoC : Implementation of a 32-port SPIN network », *Proc. Design Automation and Test in Europe Conference and Exhibition*, Mars 2003, p. 1128–1129.

[AnKu04] D. Andreasson et S. Kumar – « On improving best-effort throughput by better utilization of guaranteed throughput channels in an on-chip communication system », *Proc. 22nd IEEE Norchip Conference*, Novembre 2004, p. 265–268.

[Arte06] Arteris S.A. – « A comparison of network-on-chip and busses », `http://www.arteris.com`.

[BCGK03] E. Bolotin, I. Cidon, R. Ginosar et A. Kolodny – « Cost considerations in network on chip », *Integration - The VLSI Journal, special issue on Networks on Chip* **38** (2003), p. 19–42.

[BCGK04] — , « QNoC : QoS architecture and design process for network on chip », *Journal of Systems Architecture, special issue on Network on Chip* **50** (2004), p. 105–128.

[BCVC05] E. Beigne, F. Clermidy, P. Vivet, A. Clouard et M. Renaudin – « An asynchronous noc architecture providing low latency service and its multi-level design framework », *Proc. 11th IEEE International Symposium on Asynchronous Circuits and Systems*, Mars 2005, p. 54–63.

[BLUE06] « The official Bluetooth wireless info site » – 2006, `http://www.bluetooth.com`.

[BaFu02] J. Bainbridge et S. Furber – « Chain : a delay-insensitive chip area interconnect », *IEEE Micro* **22** (2002), p. 16–23.

[BaPF04] W. Bainbridge, L. Plana et S. Furber – « The design and test of a smartcard chip using a chain self-timed network-on-chip », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 3, Février 2004, p. 274–279.

[BaVC04] N. Banerjee, P. Vellanki et K. Chatha – « A power and performance model for network-on-chip architectures », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, Février 2004, p. 1250–1255.

[BeBe04] D. Bertozzi et L. Benini – « Xpipes : a network-on-chip architecture for gigascale systems-on-chip », *IEEE Circuits and Systems Magazine* **4** (2004), p. 18–31.

[BeMi01] L. Benini et G. D. Micheli – « Powering networks on chips », *Proc. 14th International Symposium on Systems Synthesis*, Octobre 2001, p. 33–38.

[BeMi02] — , « Networks on chips : a new SoC paradigm », *Computer* **35** (2002), p. 70–78.

[Bern04] C. Bernard – « Mécanismes de communication », Document interne, CEA-LETI Grenoble, Avril 2004.

[BhMa03] P. Bhojwani et R. Mahapatra – « Interfacing cores with on-chip packet-switched networks », *Proc. 16th International Conference on VLSI Design*, Janvier 2003, p. 382–387.

[BjSp05] T. Bjerregaard et J. Sparso – « A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip », *Proc. Design, Automation and Test in Europe*, vol. 2, Mars 2005, p. 1226–1231.

[BlGa01] D. Blaauw et K. Gala – « Deep-submicron issues in high-performance design », *Proc. International Workshop on Power And Timing Modeling, Optimization and Simulation*, Septembre 2001.

[CADE06] « Cadence » – `http://www.cadence.com`.

[CCGM04] M. Coppola, S. Curaba, M. D. Grammatikakis, G. Maruccia et F. Papariello – « OCCN : a network-on-chip modeling and simulation

framework », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 3, Février 2004, p. 174–179.

[CCHM99] H. Chang, L. Cook, M. Hunt, G. Martin, A. McNelly et L. Todd – *Surviving the SoC revolution : A guide to platform-based design*, Kluwer Academic Publishers, 1999.

[CGJM03] A. Clouard, F. Ghenassia, K. Jain, L. Maillet-Contoz et J. P. Strassen – « Using transaction-level models in a SoC design flow », SystemC : Methodologies and Applications (W. Muller, W. Rosenstiel et J. Ruf, éds.), Kluwer Academic Publishers, 2003, p. 29–63.

[CTCH04] A. Chun, E. Tsui, I. Chen, H. Honary et J. Lin – « Application of the Intel© Reconfigurable Communications Architecture to 802.11a, 3G and 4G standards », *Proc. 6th CAS Symposium on Emerging Technologies*, Mai-Juin 2004, p. 659–662.

[CoBM02] P. Coussy, A. Baganne et E. Martin – « Analyse fonctionnelle des moyens de communication proposés dans les systèmes sur silicium », *Proc. Journées Francophones sur l'Adéquation Algorithme Architecture*, Décembre 2002.

[Cord99] B. Cordan – « An efficient bus architecture for system-on-chip design », *Proc. IEEE 1999 Custom Integrated Circuits Conference*, Mai 1999, p. 623–626.

[CuSG99] D. J. Culler, J. P. Singh et A. Gupta – *Parallel computer architecture : A hardware/software approach*, Morgan Kaufmann Publishers, 1999.

[DRGR03] A. Radulescu, K. Goossens et E. Rijpkema – « Concepts and implementation of the philips network-on-chip », *Proc. International Workshop on IP Based System-on-Chip Design*, Novembre 2003.

[DaTo01] W. J. Dally et B. Towles – « Route packets, not wires : on-chip interconnection networks », *Proc. Design Automation Conference*, Juin 2001, p. 684–689.

[Dall90] W. J. Dally – « Virtual-channel flow control », *Proc. 17th Annual International Symposium Computer Architecture*, Mai 1990, p. 60–68.

[DuBL05] Y. Durand, C. Bernard et D. Lattard – « FAUST : on-chip distributed architecture for a 4G baseband modem SoC », *Proc. Design and Reuse IP-SOC*, Décembre 2005, p. 51–55.

[FaKa03] K. Fazel et S. Kaiser – *Multi-carrier and spread spectrum systems*, Wiley Publishers, Septembre 2003.

[Flyn97] D. Flynn – « Amba : enabling reusable on-chip designs », *IEEE Micro* **17** (1997), p. 20–27.

[GART06] « Press release » – Février 2006, `http://www.gartner.com`.

[GDMP03] K. Goossens, J. Dielissen, J. van Meerbergen, P. Poplavko, A. Radulescu, E. Rijpkema, E. Waterlander et P. Wielage –

« Guaranteeing the quality of services in networks on chip », Networks on Chip (A. Jantsch et H. Tenhunen, éds.), Kluwer, 2003, p. 61–82.

[GLMS02] T. Grötker, S. Lia, G. Martin et S. Swan – *System design with SystemC*, Kluwer Academic Publishers, 2002.

[GPDG05] K. Goossens, S. Gonzalez Pestana, J. Dielissen, O. P. Gangwal, J. van Meerbergen, A. Rădulescu, E. Rijpkema et P. Wielage – « Service-based design of systems on chip and networks on chip », Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices (P. van der Stok, éd.), Philips Research Book Series, vol. 3, Springer, 2005, p. 37–60.

[GPIS04] C. Grecu, P. P. Pande, A. Ivanov et R. Saleh – « Structured interconnect architecture : A solution for the non-scalability of bus-based socs », *Proc. Great Lakes Symposium VLSI*, Avril 2004, p. 192–195.

[GlNi94] C. Glass et L. Ni – « The turn model for adaptive routing », *Journal of the Association for Computing Machinery* **41** (1994), p. 874–902.

[GuGr00] P. Guerrier et A. Greiner – « A generic architecture for on-chip packet-switched interconnections », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Mars 2000, p. 250–256.

[HEMK05] A. Hegedus, G. M. Maggio et L. Kocarev – « A ns-2 simulator utilizing chaotic maps for network-on-chip traffic analysis », *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, Mai 2005, p. 3375–3378.

[HJKP00] A. Hemani, A. Jansch, S. Kumar, A. Postula, J. Orberg, M. Millberg et D. Lindqvist. – « Network on chip : An architecture for billion transistor era », *Proc. 18th IEEE Norchip Conference*, Novembre 2000.

[HoDr02] R. Hofmann et B. Drerup – « Next generation CoreConnect™ processor local bus architecture », *Proc. 15th Annual IEEE International ASIC/SOC Conference*, Septembre 2002, p. 221–225.

[HoHK03] R. Holsmark, M. Hogberg et S. Kumar – « Modelling and evaluation of a network on chip architecture using sdl », *Proc. 11th SDL Forum*, Juillet 2003.

[HoMH01] R. Ho, K. Mai et M. A. Horowitz – « The future of wires », *IEEE* **89** (2001), p. 490–504.

[Holt05] K. Holt – « Wireless LAN : Past, present, and future », *Proc. Design, Automation and Test in Europe*, vol. 3, 2005, p. 92–93.

[Hong05] H. Yang – « A road to future broadband wireless access : MIMO-OFDM-based air interface », *IEEE Communications Magazine* **43** (2005), p. 53–60.

[HuMa04] J. Hu et R. Marculescu – « Dyad - smart routing for networks-on-chip », *Proc. 41st Design Automation Conference*, Juin 2004, p. 260–263.

[ICC04] C. Bernard et D. Varreau – « Spécification du bloc Input Communication Controler d'une unité HW », Document interne, CEA-LETI Grenoble, Janvier 2004.

[IP03] — , « Spécification du bloc Input d'une unité HW », Document interne, CEA-LETI Grenoble, Novembre 2003.

[ITM04] — , « Spécification du bloc IT manager d'une unité HW », Document interne, CEA-LETI Grenoble, Mars 2004.

[ITRS05] « International roadmap for semiconductors » – `http://public.itrs.net`.

[JVLZ05] M. Juntti, M. Vehkapera, J. Leinonen, V. Zexian, D. Tujkovic, S. Tsumura et S. Hara – « MIMO MC-CDMA communications for future cellular systems », *IEEE Communications Magazine* **43** (2005), p. 118–124.

[JeTW05] A. Jerraya, H. Tenhunen et W. Wolf – « Guest editors' introduction : Multiprocessor systems-on-chips », *Computer* **38** (2005), p. 36–40.

[KDWG03] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr et S. Goossens – « A modular simulation framework for architectural exploration of on-chip interconnection networks », *Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Octobre 2003, p. 7–12.

[KJSF02] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja et A. Hemani – « A network on chip architecture and design methodology », *Proc. IEEE Computer Society Annual Symposium on VLSI*, Avril 2002, p. 105–112.

[KNDR01] F. Karim, A. Nguyen, S. Dey et R. Rao – « On-chip communication architecture for OC-768 network processors », *Proc. Design Automation Conference*, 2001, p. 678–683.

[KNRS00] K. Keutzer, A. Newton, J. Rabaey et A. Sangiovanni-Vincentelli – « System-level design : orthogonalization of concerns and platform-based design », *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **19** (2000), p. 1523–1543.

[KaHM87] M. Karol, M. Hluchyj et S. Morgan – « Input versus output queueing on a space-division packet switch », *IEEE Transactions on Communications* **35** (1987), p. 1347–1356.

[KaND02] F. Karim, A. Nguyen et S. Dey – « An interconnect architecture for networking systems on chips », *IEEE Micro* **22** (2002), p. 36–45.

[LCDL05] R. Lemaire, F. Clermidy, Y. Durand, D. Lattard et A. A. Jerraya – « Performance evaluation of a NoC-based design for MC-CDMA telecommunications using NS-2 », *Proc. 16th IEEE International Workshop on Rapid System Prototyping*, Juin 2005.

[LDLJ06] R. Lemaire, Y. Durand, D. Lattard et A. A. Jerraya – « A semi-distributed control system for application management in a NoC-based architecture », *Proc. 24th IEEE Norchip Conference*, Novembre 2006.

[LLCB06] R. Lemaire, D. Lattard, F. Clermidy et C. Bernard – « Système sur puce à controle semi-distribué », Mars 2006, Brevet Numéro E.N. : 06-50892.

[LaRD01] K. Lahiri, A. Raghunathan et S. Dey – « Evaluation of the traffic-performance characteristics of system-on-chip communication architectures », *Proc. 14th International Conference on VLSI Design*, Janvier 2001, p. 29–35.

[LeLJ05] R. Lemaire, D. Lattard et A. A. Jerraya – « Évaluation des performances de transferts de données sur un noc régulé par un mécanisme de contrôle de flux », *Proc. Journées Nationales du Réseau de Doctorants en Microélectronique*, Mai 2005.

[LiST00] J. Liang, S. Swaminathan et R. Tessier – « aSOC : A scalable, single-chip communications architecture. », *Proc. IEEE International Conference on Parallel Architectures and Compilation Techniques*, Octobre 2000, p. 37–46.

[MATR06] « MATRICE - MC-CDMA transmission techniques for integrated broadband cellular systems » – `http://www.ist-matrice.org`.

[MBVV02] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde et R. Lauwereins – « Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs », *Proc. the reconfigurable computing is going mainstream, 12th International Conference on Field-Programmable Logic and Applications*, Septembre 2002, p. 795–805.

[MHKE99] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, T. Olsson, P. Nilsson, D. Lindqvist et H. Tenhunen – « Globally asynchronous locally synchronous architecture for large high-performance ASICs », *Proc. IEEE International Symposium on Circuits and Systems*, vol. 2, Mai-Juin 1999, p. 512–515.

[MMBM03] T. Marescaux, J.-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde et R. Lauwereins – « Networks on chip as hardware components of an OS for reconfigurable systems », *Proc. 13th International Conference on Field Programmable Logic and Applications*, Septembre 2003.

[MMMO03] F. Moraes, A. Mello, L. Möller, L. Ost et N. Calazans – « Low area overhead packet-switched network on chip : Architecture and prototyping », *Proc. IFIP International Conference on Very Large Scale Integration*, Décembre 2003.

[MNTJ04] M. Millberg, E. Nilsson, R. Thid et A. Jantsch – « Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip », *Proc. Design Automation and Test Europe Conference*, Février 2004.

[MNTK04] M. Millberg, E. Nilsson, R. Thid, S. Kumar et A. A. Jantsch – « The Nostrum backbone-a communication protocol stack for Networks on

Chip », *Proc. 17th International Conference on VLSI Design*, 2004, p. 693–696.

[MODE06] « ModelSim - Mentor Graphics » – `http://www.model.com`.

[MTCM05] A. Mello, L. Tedesco, N. Calazans et F. Moraes – « Virtual channels in networks on chip : implementation and evaluation on Hermes NoC », *18th annual symposium on integrated circuits and system*, 2005, p. 178–183.

[MVKF99] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber et W. Fichtner – « Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems », *Proc. 12th IEEE International ASIC/SOC Conference*, Septembre 1999, p. 317–321.

[Moor65] G. E. Moore – « Cramming more components onto integrated circuits », *Electronics Magazine* **43** (1965), p. 114Ű–117.

[MuRB06] N. Muralimanohar, K. Ramani et R. Balasubramonian – « Power efficient resource scaling in partitioned architectures through dynamic heterogeneity », *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, Mars 2006, p. 100–111.

[NMAV05] V. Nollet, T. Marescaux, P. Avasare, D. Verkest et J.-Y. Mignolet – « Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles », *Proc. Design, Automation and Test in Europe*, vol. 1, 2005, p. 234–239.

[NMOJ03] E. Nilsson, M. Millberg, J. Oberg et A. Jantsch – « Load distribution with the proximity congestion awareness in a network on chip », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, 2003, p. 1126–1127.

[NS206] « The network simulator - NS-2 » – `http://www.isi.edu/nsnam/ns`.

[NgCh05] V.-D. Ngo et H.-W. Choi – « On chip network : topology design and evaluation using NS2 », *Proc. 7th International Conference on Advanced Communication Technology*, vol. 2, Février 2005, p. 1292–1295.

[Nils02] E. Nilsson – *Design and implementation of a hot-potato switch in a network on chip*, Mémoire, Departement of Microelectronics and Information Technology, Royal Institute of Technology, Juin 2002.

[NoMV04] V. Nollet, T. Marescaux et D. Verkest – « Operating-system controlled network on chip », *Proc. Design Automation Conference*, Juin 2004, p. 256–259.

[OCC04] C. Bernard et D. Varreau – « Spécification du bloc Output Communication Controler d'une unité HW », Document interne, CEA-LETI Grenoble, Juin 2004.

[OCPI06] « Open Core Protocol - International Partnership » – `http://www.ocp-ip.org`.

[OP03] C. Bernard et D. Varreau – « Spécification du bloc Output d'une unité HW », Document interne, CEA-LETI Grenoble, Octobre 2003.

[OPNE06] « OPNET » – `http://www.opnet.com`.

[OSCI06] « Open SystemC Initiative. SystemC v2.0.1 » – `http://www.systemc.org`.

[OTcl06] « OTcl - Object Tcl extensions » – `http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl`.

[PGIS03] P. Pande, C. Grecu, A. Ivanov et R. Saleh – « Design of a switch for network on chip applications », *Proc. International Symposium on Circuits and Systems*, vol. 5, Mai 2003, p. 217–220.

[PGJI05] P. P. Pande, C. Grecu, M. Jones, A. Ivanov et R. Saleh – « Performance evaluation and design trade-offs for network-on-chip interconnect architectures », *IEEE Transactions on Computers* **54** (2005), p. 1025–1040.

[PIBU95] « PI-Bus Systems Toolkit » – `http://cordis.europa.eu/esprit/src/results/pages/infoind/infind24.htm`.

[PPBL04] P. Paulin, C. Pilkington, E. Bensoudane, M. Langevin et D. Lyonnard – « Application of a multi-processor SoC platform to high-speed packet forwarding », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 3, Février 2004, p. 58–63.

[PRRG04] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens et O. P. Gangwal – « Cost-performance trade-offs in networks on chip : a simulation-based approach », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, Février 2004, p. 764–769.

[PTOL06] « The Ptolemy project » – `http://ptolemy.eecs.berkeley.edu`.

[RDPG05] A. Radulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage et K. Goossens – « An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration », *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **24** (2005), p. 4–17.

[REAL06] « REAL 5.0 overview » – `http://www.cs.cornell.edu/skeshav/real/overview.html`.

[RGRD03] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage et E. Waterlander – « Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Mars 2003, p. 350–355.

[RWD04] C. Bernard – « RWD : Read Write Decoder », Document interne, CEA-LETI Grenoble, Septembre 2004.

[RaGo02] A. Radulescu et K. Goossens – « Communication services for networks on chip », Domain-specific Embedded Multiprocessors (S. Bhattacharyya, E. Deprettere et J. Teich, éds.), Dekker, 2003.

[RiGW01] E. Rijpkema, K. Goossens et P. Wielage – « A router architecture for networks on silicon », *Proc. 2nd Workshop on Embedded Systems*, Novembre 2001, p. 181–188.

[RoSV97] A. Rowson, J.A. ; Sangiovanni-Vincentelli – « Interface-based design », *Proc. Design Automation Conference*, Juin 1997, p. 178–183.

[SSMK01] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey et A. Sangiovanni-Vincentelli – « Addressing the system-on-a-chip interconnect woes through communication-based design », *Proc. Design Automation Conference*, Juin 2001, p. 667–672.

[STBU03] STMicroelectronics – « STBus communication system : Concept and definitions », Tech. report, STMicroelectronics, 2003.

[STNu02] D. Siguenza-Tortosa et J. Nurmi – « VHDL-based simulation environment for Proteo NoC », *Proc. 7th IEEE High-Level Design Validation and Test Workshop*, Octobre 2002, p. 1–6.

[STRR05] G. Sassatelli, L. Torres, S. Riso, M. Robert et F. Moraes – « A mesh based Network on Chip characterization : A GALS approach », *Proc. 20th International Conference on Design of Circuits and Integrated Systems*, 2005.

[ScRF04] A. Scherrer, T. Risset et A. Fraboulet – « Hardware wrapper classification and requirements for on-chip interconnets », *Proc. Signaux, Circuits et Systèmes*, Mars 2004.

[SpDL06] « SDL forum society » – `http://www.sdl-forum.org`.

[SuKJ02] Y.-R. Sun, S. Kumar et A. Jantsch – « Simulation and evaluation for a network on chip architecture using ns-2 », *Proc. 20th IEEE Norchip Conference*, Novembre 2002.

[TLVI05] T. Theocharides, G. Link, N. Vijaykrishnan et M. Irwin – « Implementing ldpc decoding on network-on-chip », *Proc. 18th International Conference on VLSI Design*, Janvier 2005, p. 134–137.

[Tane96] A. S. Tanenbaum – *Computer networks*, Prentice Hall, 1996.

[UWB06] « uwb forum » – 2006, `http://www.uwbforum.org`.

[VIRT06] « Virtex-4 Multi-Platform FPGA » – 2006, `http://www.xilinx.com`.

[VSIA06] « Virtual Socket Interface Alliance » – `http://www.vsi.org`.

[Vulm04] A. Vulmiere – « Les standards et les normes », *Veille Technologique* (2004), p. 27–34.

[WIM06] « The WIMAX forum » – 2006, `http://www.wimaxforum.org`.

[WISH02] OpenCores.org – « Wishbone, revision b.3 specification », Tech. report, OpenCores.org, 2002.

[WiGo02] P. Wielage et K. Goossens – « Networks on silicon : blessing or nightmare ? », *Proc. Euromicro Symposium on Digital System Design*, Septembre 2002, p. 196–200.

[WiLi03] D. Wiklund et D. Liu ; – « SoCBUS : switched network on chip for hard real time embedded systems », *Proc. International Parallel and Distributed Processing Symposium*, Avril 2003.

[WiLi05] D. Wiklund et D. Liu – « Design, mapping, and simulations of a 3G WCDMA/FDD basestation using network on chip », *Proc. International Workshop on SoC for real-time applications*, Juillet 2005.

[WiSL04] D. Wiklund, S. Sathe et D. Liu – « Network on chip simulations for benchmarking », *Proc. 4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, Juillet 2004, p. 269–274.

[Wing01] D. Wingard – « Micronetwork-based integration for SOCs », *Proc. Design Automation Conference*, Juin 2001, p. 18–22.

[XWHC04] J. Xu, W. Wolf, J. Henkel, S. Chakradhar et T. Lv – « A case study in networks-on-chip design for embedded video », *Proc. Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, Février 2004, p. 770–775.

[YeBM03b] T. Ye, L. Benini et G. De Micheli – « Packetization and routing analysis of on-chip multiprocessor networks », *Journal of Systems Architecture* **50** (2004), p. 81–104.

[YeLF93] G. F. N. Yee, J.P. Linnartz – « Multicarrier CDMA in indoor wireless radio networks », *IEEE Personal Indoor and Mobile Radio Communications* (1993), p. 109–113.

[ZKCS02] C. A. Zeferino, M. E. Kreutz, L. Carro et A. A. Susin – « A study on communication issues for systems-on-chip », *Proc. 15th Symposium on Integrated Circuits and Systems Design*, Septembre 2002.

[ZeSu03] C. A. Zeferino et A. A. Susin – « SoCIN : a parametric and scalable network-on-chip », *Proc. 16th Symposium on Integrated Circuits and Systems Design*, Septembre 2003, p. 169–174.

[ZhMa02] X. Zhu et S. Malik – « A hierarchical modeling framework for on-chip communication architectures », *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Novembre 2002, p. 663–670.

[Zimm80] H. Zimmermann – « OSI reference model–the ISO model of architecture for open systems interconnection », *IEEE Transactions on Communications* **28** (1980), p. 425–432.

Table des figures

Liste des tableaux

Glossaire

4G : Quatrième Génération

4MORE : 4G MC-CDMA Multiple Antenna System-on-Chip for Radio Enhancements

BOP : *Begining Of Packet*

CDMA : *Code Division Multiple Access*

CFM : *Configuration Manager*

DL : *Down Link - Download*

EOP : *End Of Packet*

FAUST : *Flexible Architecture of Unified Systems for Telecom*

FFT : *Fast Fourier Transform*

FHT : *Fast Hadamard Transform*

GALS : Globalement Asynchrone Localement Synchrone

ICC : *Input Communication Controler*

IP : *Intellectual Property - Input Port (dans le contexte de FAUST)*

IST : *Information Technologies Society*

ITM : *Interrupt Manager*

ITRS : *International Technology Roadmap for Semiconductors*

LAN : *Local Area Network*

LDPC : *Low Density Parity Check*

MAC : *Media Access Control*

MAN : *Metropolitan Area Network*

MATRICE : *Multicarrier CDMA Transmission Techniques for Integrated Broadband CEllular Systems*

MC-CDMA : *MultiCarrier - Code Division Multiple Access*

MIMO : *Multiple-Input Multiple-Output*

MPSoC : *Multi-Processor System-on-Chip*

MT : *Mobile Terminal*

NI : *Network Interface*

NoC : *Network-on-Chip*

NS-2 : *Network Simulator - version 2*

OFDM : *Orthogonal Frequency Division Multiplexing*

OP : *Output Port*

OCC : *Output Communication Controler*

PAN : *Personal Area Network*

RPTT : *Return Path To Target*

RWD : *Read Write Decoder*

RX : *Receiver*

SCL : Système de Contrôle Local

SISO : *Single-Input Single-Output*

SLGC : Système Local de Gestion des Configurations

SoC : *System-on-Chip*

SS-MC-MA : *Spread Spectrum - MultiCarrier - Multipe Access*

TDD : *Time Division Duplex*

TLM : *Transaction Level Modeling*

TX : *Transmitter*

UL : *Up Link - Upload*

UMTS : *Universal Mobile Telecommunications Systems*

UWB : *Ultra Wide Band*

WAN : *Wide Area Network*

WLAN : *Wireless Local Area Network*

Index

RESUME
Ces travaux de thèse ont pour cadre l'architecture de systèmes sur puce (SoC) implémentant des fonctionnalités de la couche physique d'applications de télécommunication. Ces systèmes exigent des structures de communication performantes et nous avons étudié une architecture de réseau sur puce (NoC) en nous basant sur la plate-forme FAUST développée au LETI.
Dans ce contexte, les contributions de la thèse se focalisent sur la conception et la modélisation d'une architecture de contrôle pour la gestion d'une application complexe distribuée sur un NoC.
Nous présentons d'abord un environnement de modélisation basé sur le simulateur NS-2. Cet environnement logiciel nous fournit des outils pour analyser les besoins en communication des applications cibles et dimensionner les paramètres du réseau. Il a permis de valider l'architecture FAUST dans le cas d'un trafic de données pour une chaîne de traitement MC-CDMA (multiporteuse, accès multiple par code). Les résultats sont confirmés par un modèle SystemC du réseau FAUST.
Ensuite, nous exposons nos travaux sur le développement d'une architecture pour la gestion des communications et le contrôle des traitements sur un NoC. La solution proposée utilise une architecture d'interface réseau reconfigurable associée à chaque unité de traitement. Un processeur central programme les interfaces avec des séquences d'instructions correspondant à des scénarios de chargement de configurations qui sont stockées dans un serveur. Les performances cette approche ont été validées avec un environnement de simulation mixte VHDL et SystemC.

MOTS-CLÉS
Réseau sur puce, NoC, système sur puce, SoC, interface réseau, NI, modélisation de réseau, application de télécommunication, architecture de communication, structure d'interconnexion, système de contrôle hiérarchique.

TITRE EN ANGLAIS
Design and modelling of a control system for telecommunication applications with a Network-on-Chip (NoC) architecture

ABSTRACT
This thesis deals with the architecture of Systems-on-Chip (SoC) which integrate physical layer functionalities of telecommunication applications. Such systems require high-performance communication structures and we have studied a Network-on-Chip (NoC) architecture based on the FAUST framework developed by the LETI.
In this context, the thesis contributions focus on the design and modelling of a control architecture for management of complex applications distributed on a NoC.
First, we present a modelling environment based on the NS-2 simulator. This software environment gives us tools to analyse communication needs of targeted applications and to adjust network parameters. The FAUST architecture has been validated in the case of data traffic for a MC-CDMA (multi-carrier, code division multiple access) processing chain. Results are confirmed by a SystemC model of the FAUST network.
Then, we expose our works on the development of an architecture for communication management and processing control on a NoC. The proposed solution uses a reconfigurable network interface architecture combined with each processing unit. A central processor programs instructions sequences in the interfaces corresponding to loading scenarios of configurations that are stored in a server. The performances of the whole approach have been validated with a VHDL/SystemC mixed simulation environment.

KEY WORDS
Network-on-Chip, NoC, System-on-Chip, SoC, Network Interface, NI, network modelling, telecommunication application, communication architecture, interconnection structure, hierarchical control system.

SPECIALITE
Micro et Nano Électronique