Réseau sur puce sécurisé pour applications cryptographiques sur FPGA

Rémy Druyer

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Systèmes Automatiques et Micro-Electroniques (SYAM)

École doctorale I2S

Unité de recherche LIRMM CNRS UMR5506

Réseau sur puce sécurisé pour applications cryptographiques sur FPGA

Présentée par Rémy DRUYER

Le 26 Octobre 2017

Sous la direction de Lionel TORRES et Pascal BENOIT

Devant le jury composé de

Bertrand GRANADO, Professeur des Universités, UPMC - LIP6	Président
Jean-Philippe DIGUET, Directeur de recherche, CNRS, LAB-STICC	Rapporteur
Fernando MORAES, Professeur des Universités, PUCRS	Rapporteur
Lionel TORRES, Professeur des Universités, LIRMM – Université de Montpellier	Directeur de thèse
Pascal BENOIT, Maître de Conférences, HDR, LIRMM – Université de Montpellier	Codirecteur de thèse
Patrick LE-QUERE, Ingénieur, BULL	Examinateur
Paul-Vincent BONZOM, Ingénieur, CEA TECH	Membre invité

UNIVERSITÉ DE MONTPELLIER

Table des matières

Liste des figures et des tableaux

Glossaire

AES : Advanced **E**ncryption **S**tandard

ALM : Adaptive **L**ogic **M**odule

ANSSI : **A**gence **N**ationale de la **S**écurité des **S**ystèmes d'**I**nformation.

ASIC : Application-Specific **I**ntegrated **C**ircuit : Circuit intégré non reprogrammable dédié à une application donnée.

BURST : Echange de données composées de plusieurs mots consécutifs.

CLB : **C**onfigurable **L**ogic **B**loc

CPU : **C**entral **P**rocessing **U**nit

FPGA : **F**ield **P**rogrammable **G**ate **A**rray : Circuit logique reprogrammable.

FLIT : **FL**ow control un**IT** : Dans le domaine des NoC, il s'agit de la plus petite unité de constituante d'un paquet pouvant être échangée à chaque cycle de transmission de données.

HSM : **H**ardware **S**ecurity **M**odule

IP (Intellectual Property) : Ce terme est utilisé dans le domaine de la conception sur FPGA et sur ASIC pour désigner toute partie pouvant être réutilisée dans le développement d'un nouveau système. Dans le cadre de cette thèse, son utilisation est restreinte à la dénomination des cœurs de propriété intellectuelles, qui manipulent ou stockent des données et composent les systèmes sur puce programmables.

MPSoC : Multi **P**rocessor **S**ystem-**o**n-**C**hip

NoC (**N**etwork-**o**n-**C**hip) : Réseau sur puce (*traduction Française*).

NIST : National **I**nstitute of **S**tandards and **T**echnology

LUT : **L**ookup **T**able

RAM : **R**andom **A**ccess **M**emory

RTL : **R**egister **T**ransfer **L**evel

TCB : **T**rusted **B**ase **C**omputing

TPM : **T**rusted **P**latform Module

VHDL : **V**HSIC **H**ardware **D**escription **L**anguage.

VHSIC : Very High Speed Integrated Circuit

Remerciements

Premièrement, je tiens à remercier chaleureusement mon directeur de thèse Lionel Torres, ainsi que mon codirecteur de thèse Pascal Benoit, qui ont su m'accompagner durant plus de trois années. Ils se sont toujours montrés présents et attentifs, et je les remercie pour toute l'aide qu'ils m'ont apportée.

Je remercie également, Paul-Vincent Bonzom et Patrick Le-Quéré, mes deux encadrants du côté industriel, qui ont été essentiels dans cette collaboration.

Je tiens aussi à remercier le personnel et les doctorants du LIRMM que j'ai pu côtoyer durant ces deux années passées au laboratoire, et qui participent quotidiennement à la bonne ambiance qu'il y règne.

Je remercie tous mes collègues du service Trustway que j'avais initialement rejoint pour terminer mon doctorat, mais avec qui je vais prolonger mon aventure professionnelle pendant au moins quelques temps.

Enfin, je remercie mes parents, ma famille, et mes amis proches, qui ont toujours fait preuve de soutien à mon égard.

Introduction générale

Les systèmes sur puce (*System-on-Chip* ou *SoC)* ont vu leur utilisation largement se répandre durant ces quinze dernières années. Ce phénomène s'explique notamment par leur large adoption dans les systèmes embarqués grand public tels que les tablettes ou les smartphones, et ceci pour plusieurs raisons.

Pour commencer, en les comparant aux systèmes basés sur des CPU qui équipent la plupart des ordinateurs, leur consommation en énergie est faible, tout en délivrant des puissances de calculs élevées. Le fait qu'un ou plusieurs processeurs, puissent partager la même puce en silicium que des mémoires et des périphériques, réduit drastiquement les pertes en énergie autrement dues à la présence de bus externes reliant différents circuits. Cette caractéristique commence d'ailleurs à attirer l'attention d'acteurs d'un domaine où jusqu'ici les SoC étaient très minoritaires et où les problématiques de consommation sont d'importance capitales, il s'agit des supercalculateurs. En effet, soit le Japonais Fujitsu [SICa], soit les constructeurs Bull et Cray [SICb] pour l'Europe, ont annoncé la conception de nouvelles architectures de calculs à hautes performances basées sur des SoC ARM.

Une autre raison qui explique la large adoption des systèmes sur puce, est leur encombrement spatial réduit au regard du nombre de fonctions qu'ils peuvent contenir. Cela concerne entre autres, les protocoles de communication sans-fils (3G/4G, Wifi, Bluetooth, NFC), la gestion l'audio (micro, haut-parleur), de la vidéo (écran, processeur graphique), ou des cartes SIM et cartes mémoires externes. Grâce à l'amélioration des technologies de conception, le nombre et la complexité des fonctions embarquées s'amplifient, tout en minimisant les surfaces qu'occupent les circuits.

Désormais, les architectures capables d'interconnecter les éléments au sein des systèmes sur puce doivent suivre la même évolution en performances. Des bus sont habituellement utilisés dans l'optique de supporter les communications, cependant leur bande-passante se trouve limitée lorsque le nombre d'éléments communicants est excessif. Partant de ce constat, des personnes ont cherché des architectures, dit « scalable », c'est-à-dire, pouvant supporter une augmentation des besoins en bande-passante, sans engendrer une dégradation de leurs autres performances. Pour trouver des solutions, ces personnes se sont tournées vers un domaine où ces problématiques se sont déjà posées, il s'agit des réseaux informatiques. D'où l'idée de porter des architectures se reposant sur des réseaux de routeur dans les circuits électroniques [BEN02]. Ce type de système d'interconnexions est appelé « réseau sur puce » (*Network-on-Chip* ou *NoC*).

Les systèmes sur puce sont majoritairement intégrés dans des circuits ASIC (*Application Specific Integrated Circuit*), non reprogrammables et ayant des coûts initiaux de conception et de production très élevés. Ce n'est qu'en produisant des volumes de circuits conséquents que le montant de l'étape de conception devient négligeable par rapport au coût total, et que le prix unitaire d'un circuit s'effondre. Les phases de tests et de validations sont très importantes puisque les erreurs de conception ne peuvent pas être corrigées une fois le circuit fabriqué. Les circuits FPGA (*Field Programmable Gate Array*) sont notamment utilisés lors du prototypage des circuits ASIC, grâce à la possibilité qu'ils offrent d'être reprogrammés. Cependant, leur utilisation ne se cantonne pas qu'à cette fonction, puisqu'ils peuvent tout aussi bien être utilisés

dans des produits finis, particulièrement dans des secteurs comme la défense, l'avionique, l'aérospatial ou le médical. Les systèmes implémentés sur FPGA sont appelés systèmes sur puce reprogrammables.

Les architectures matérielles des ASIC et des FPGA sont très différentes. Les circuits ASIC sont pour ainsi dire, « taillés sur mesure » pour l'application qu'ils fournissent. Pour les FPGA, ce n'est pas le cas puisque les constructeurs ne peuvent pas savoir par avance, quels types d'applications vont être implémentés dans le circuit. La quantité et la disposition des ressources logiques à l'intérieur de la puce sont ainsi prévues pour une certaine polyvalence. Comme les systèmes d'interconnexions sont tributaires des architectures des circuits sur lesquels ils sont utilisés, et que les recherches concernant les réseaux sur puce se sont surtout intéressées aux implémentations sur ASIC, sur plateforme FPGA il faut chercher à tirer parti au mieux des ressources disponibles en adaptant le système d'interconnexions.

Un aspect à ne surtout pas négliger lors de la conception des systèmes numériques modernes est la sécurité. Les SoC par nature, fournissent un niveau certain niveau de protection contre les attaques matérielles, puisque quasiment tous les composants du système sont intégrés dans une seule puce de silicium. Cela empêche un attaquant de positionner une sonde au niveau d'un bus externe dans le but d'espionner les données qui transitent dessus, voire de les modifier. Cependant, la sécurisation des systèmes numériques est un très vaste sujet et de nombreuses failles sont potentiellement présentes au niveau matériel ou logiciel. Par exemple, lors de l'exécution du système d'exploitation, des environnements cloisonnés et sécurisés doivent être fournis aux applications pour éviter qu'un processus malicieux (virus, ver, cheval de troie...) puisse accéder au contenu privé ou sensible, manipulé par une autre application.

Au niveau matériel, il est théoriquement possible qu'un des acteurs ayant un rôle dans le flot de conception puisse ajouter délibérément ou par négligence, une faille ou une porte dérobée au circuit [SKO12]. Lorsque cette action est délibérée, on dit qu'il s'agit d'un cheval de Troie matériel (*Hardware Trojan*) [TEH10]. Les délais de mise sur le marché des produits tendent à être les plus courts possibles, et dans le but de les minimiser, la réutilisation d'un certain nombre de fonctions d'un produit à l'autre est quasiment obligatoire. Ces fonctions, pouvant être aussi bien matérielles que logicielles, sont appelées cœur de propriété intellectuelle ou plus couramment IP (*Intellectual Property*). Les IP se vendent et s'achètent entre tiers, elles sont assemblées et intégrées pour former des systèmes complexes. Il n'est pas exclu qu'une de ces IP contiennent une faille ou un processus malicieux, menant à des attaques pouvant bloquer le système ou bien divulguer des données qu'il manipule. Dans des domaines comme la défense, où les systèmes cryptographiques sont très utilisés, la confidentialité des données est capitale.

Sur ce point, l'ajout de mécanismes de sécurité aux systèmes d'interconnexions peut être une aubaine pour la sécurisation matérielle et logicielle du système sur puce. En effet, la position centrale du réseau gérant les communications, peut lui permettre de contrôler les échanges entre les différents cœurs et d'identifier d'éventuelles tentatives d'intrusion. Des contremesures peuvent ensuite être appliquées pour empêcher qu'une n'attaque atteigne ses objectifs.

Plan du manuscrit

Le chapitre 1 s'attache à détailler le contexte, les problématiques, et les objectifs de ce travail de thèse. Il revient sur les défis qui touchent les architectures qui gèrent les communications au sein des systèmes sur puce, ainsi que sur les questions relatives à la sécurité auxquelles ces derniers peuvent être soumis.

Le chapitre 2 a pour but de dresser un état de l'art des réseaux sur puce, et plus particulièrement, de ceux qui sont destinés aux circuits FPGA. Pour cela, il aborde certaines spécificités de l'architecture matérielle des circuits reconfigurables. Ensuite, il explore différentes métriques qui peuvent être employées dans le but d'établir une comparaison entre les réseaux sur puce. Enfin, il étudie, et compare une dizaine de travaux de recherche de NoC dédiés aux plateformes FPGA.

Le chapitre 3 présente le portage sur FPGA du réseau sur puce Hermes que nous avons réalisé. Dans un premier temps, il détaille l'architecture générique, et polyvalente de ce réseau. Ensuite, il expose les développements effectués lors de ce portage. Enfin, une fois les résultats de l'implémentation de ce NoC sur FPGA exposés, nous explorons les raisons qui expliquent pourquoi il ne permet pas de répondre aux contraintes applicatives que nous nous sommes fixées.

Le chapitre 4 aborde la conception de TrustNoC, un réseau sur puce performant et à faible latence pour circuit FPGA. Les résultats obtenus avec le portage d'Hermes permet d'investiguer sur des choix de conception à effectuer, dans le but d'atteindre de meilleures performances avec un réseau sur circuit reconfigurable. Ce chapitre décrit ensuite l'architecture de TrustNoC. Puis, il se conclut avec la présentation des résultats et des performances obtenus avec ce réseau sur puce, et leurs comparaisons avec nos contraintes applicatives.

Le chapitre 5 traite de l'implémentation de moniteurs de sécurité dans TrustNoC. Il est composé d'un état de l'art de la sécurité des systèmes d'interconnexions, et des systèmes multiprocesseurs sur puce. Il présente le modèle de menace choisi lors de l'élaboration des mécanismes de sécurité reconfigurables pour le réseau sur puce TrustNoC. Il se poursuit sur la description de l'architecture des moniteurs. Enfin, il se termine sur la présentation des résultats de l'implémentation des moniteurs.

Le chapitre 6 conclut ce manuscrit, et présente les perspectives ainsi que les pistes ouvertes par ces travaux de thèse.

Chapitre 1) Contexte et problématique

Ce chapitre explore les motivations qui ont conduit à l'élaboration de ce travail de thèse CIFRE. Celles-ci proviennent en partie du contexte industriel, mais aussi de problématiques plus globales qui touchent les architectures d'interconnexions des systèmes sur puce, et leur sécurité.

Avant d'exposer les différentes problématiques qui ont nourri nos réflexions durant toute la durée de la préparation de cette thèse, il est nécessaire de dresser un portrait clair du contexte dans lequel nous évoluons. Ce chapitre débute par une présentation succincte des enjeux industriel. Il se poursuit avec la présentation les principaux types de systèmes d'interconnexions, avec pour chacun d'entre eux, les bénéfices et les limitations. Puis, il aborde les menaces qui remettent en question la sécurité des systèmes sur puce. Et enfin, il énonce les objectifs de ce travail de thèse.

Sommaire du Chapitre 1

1.1. Contexte industriel et académique

Ce travail de thèse est effectué dans le cadre d'une convention industrielle de formation par la recherche (CIFRE). Cette CIFRE est le fruit d'un partenariat entre le laboratoire d'informatique, de robotique et de microélectronique de Montpellier (le LIRMM) et le service Trustway appartenant à la société Bull-Atos.

Le service Trustway a pour rôle le développement de produits de sécurité, tels que le disque dur sécurisé Globull ainsi qu'une gamme de modules de sécurité matériels (*HSM* pour *Hardware Security Module*). Les HSM sont des ressources matérielles fournissant un ensemble de services cryptographiques tels que du chiffrement symétrique, asymétrique, des signatures, et des algorithmes de hachage. Ces modules sont évalués par des cabinets indépendants afin d'obtenir des certifications (telles que la FIPS-140 ou les Critères Communs EAL+) attestant que leur conception respecte certaines règles de sécurité. Ces règles sont définies par des agences gouvernementales tels que l'ANSSI pour la France ou NIST pour les Etats-Unis [ANS], [NIS].

L'utilisation de plateformes matérielles telles que les HSM, est largement préconisée pour la réalisation de produits électroniques, où la sécurité est une des préoccupations premières, notamment en ce qui concerne la génération et le stockage des clés cryptographiques ([TPM], [TCG11] et [FIS11]). Pour concevoir ses HSM, la société Bull emploie des circuits FPGA. Ils ont pour principale charge, la gestion des clés cryptographique et l'exécution des différents algorithmes de chiffrement et de hachage. Ce type de circuit est adapté à la fabrication de quantités faibles ou moyennes de produits, contrairement à une utilisation d'ASIC pour effectuer les mêmes tâches. Ces derniers ont des coûts de conception et de validation très élevés, et qui ne peuvent être rentabilisés que par d'importants volumes de production. En outre, l'avantage des FPGA en tant que plateformes reconfigurables, est de pouvoir supporter des mises à jour de leur programme via des patchs correctifs, résolvant des problèmes fonctionnels ou des failles de sécurité.

On peut s'interroger sur la possibilité qu'une plateforme logicielle supportée par un CPU, remplace les circuits FPGA pour l'exécution des calculs cryptographiques. Cependant, pour beaucoup de ces algorithmes, une implémentation matérielle permet d'obtenir de bien meilleures performances. Bien que la fréquence d'horloge puisse être 10 fois moins élevée sur circuit reprogrammable que sur CPU, la parallélisation de certains calculs sur FPGA, augmente drastiquement les débits de certains algorithmes de cryptographie.

Les systèmes sur puce conçus par le service Trustway comportent de plus en plus de cœurs, par conséquent, les systèmes d'interconnexions jusqu'ici utilisés sont confrontés à un problème de manque évolutivité. D'où l'intérêt que porte ce département aux systèmes sur puce optimisés pour circuits FPGA, dont notamment les réseaux sur puce. La sécurité matérielle et logicielle des produits que le service conçoit est également capitale, il est primordial de s'assurer que l'emploi d'un réseau sur puce n'ouvre pas de nouvelles vulnérabilités. Pour pallier ce risque, l'étude des mécanismes de sécurité pouvant être ajoutés au système d'interconnexions est nécessaire.

Dans un premier temps, nous allons donc rapidement étudier les problématiques qui touchent aux systèmes d'interconnexion

1.2. Systèmes d'interconnexions

Les systèmes sur puce sont réalisés grâce à un assemblage d'IP, dans lequel chacune joue un rôle bien particulier. Généralement, on peut classer les IP dans deux catégories. La première catégorie d'IP qui initie les requêtes est appelée maître. La seconde qui reçoit ces requêtes est appelée esclave. Il peut s'agir de processeurs exécutant du code logiciel, des accélérateurs cryptographiques, des interfaces de communications, des mémoires, des unités de traitement audio ou vidéo, ainsi que divers périphériques. Pour que le système sur puce fonctionne avec des performances suffisantes, toutes ces IP doivent pouvoir échanger des informations à des débits minimaux, et des latences maximales.

Pour remplir cette fonction, de nombreux types de systèmes de communications sont envisageables. Cependant, ils sont tributaires de beaucoup de caractéristiques, comme le nombre de nœuds à faire communiquer, la largeur des liens, la taille et la fréquence d'émission des messages, le débit, la latence, et la plateforme matérielle ciblée.

1.2.1. Bus

Le bus est sûrement l'architecture la plus commune. Comme l'illustre la **Figure 1.1**, tous les nœuds sont connectés au même medium, composé au minimum, d'un bus de contrôle ou de requête, d'un bus d'adresse et d'un bus de données. Les esclaves utilisent le bus d'acquittement pour répondre aux requêtes provenant des maîtres. Lorsqu'un nœud émet un message, il est diffusé à tous les autres, mais auparavant l'arbitre central régissant les accès devra avoir autorisé la transmission.

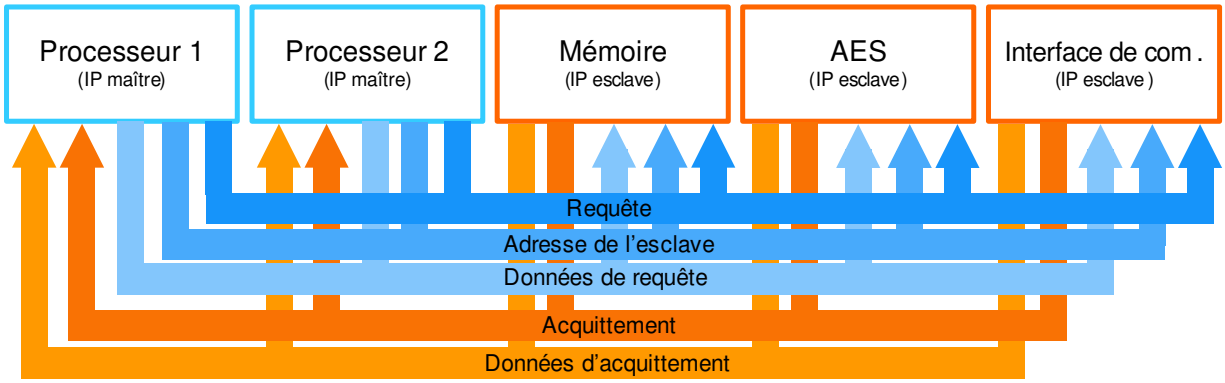


Figure 1.1 : Composition d'un bus de communication sur puce

La principale limitation de ce type d'architecture réside dans la bande-passante qui est partagée entre tous les nœuds émetteurs. Comme il n'y a qu'un seul nœud qui peut accéder au bus à la fois, lorsque le nombre de nœuds est trop important, la bande-passante n'est plus suffisante pour pouvoir satisfaire les performances du système.

Il existe de nombreuses de variations et de spécifications différentes de bus, certaines offrant la possibilité d'effectuer des requêtes en « *burst* », ciblant plusieurs mots d'un registre mémoire, ou bien encore intégrant des mécanismes de gestion d'erreur, avec des réémissions de messages en cas de détection d'une faute intervenant lors de la transmission. Pour donner quelques exemples de bus très utilisés, on peut citer le bus AMBA - AXI proposé par ARM [XIL15], le bus CoreConnect d'IBM [IBM99] ou encore le bus open source Wishbone [WIS10].

Pour pallier le problème de limitation de bande-passante, des architectures basées sur des hiérarchies de bus ont été proposées [KHA16], il s'agit de plusieurs bus connectés par des ponts. Cependant elles se révèlent très vite limitées, puisque lorsqu'une IP souhaite communiquer avec une IP connectée à un autre bus, elle va devoir monopoliser le bus pour accéder au pont, puis monopoliser tous les bus intermédiaires jusqu'à atteindre l'IP destinataire.

1.2.2. Crossbar

Ce type d'architecture a l'avantage de supporter plusieurs connexions en parallèle, dans la mesure où les IP initiatrices et réceptrices ne sont pas déjà impliquées dans un échange. Sur la **Figure 1.2** les bus de requête, d'adresse et de données ont été simplifiés en simples flèches dans un souci de lisibilité.

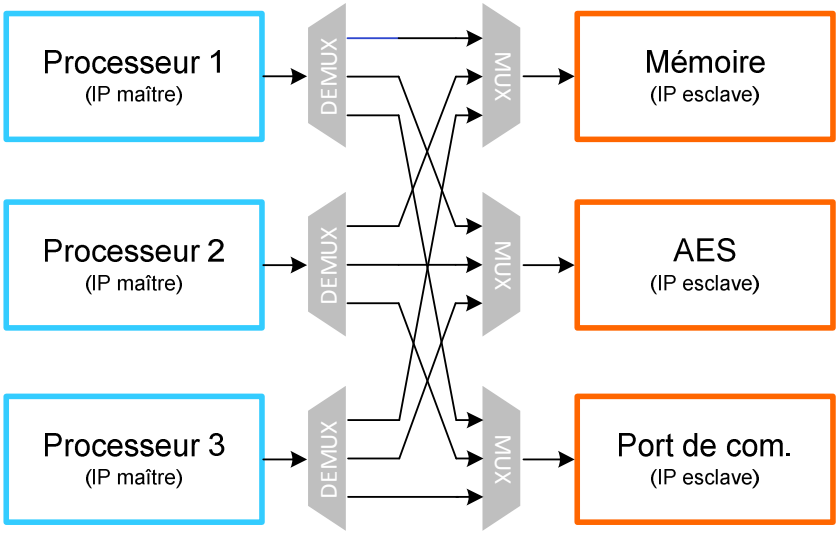


Figure 1.2 : Architecture des multiplexeurs dans un crossbar

Le problème d'évolutivité rencontré lors de l'augmentation du nombre de nœuds se retrouve aussi dans les architectures en crossbar. Contrairement au bus, la limitation n'est pas causée par la bande-passante, car il y a plusieurs médias partagés entre toutes les IP. Par contre, la complexité et la surface des multiplexeurs et des démultiplexeurs augmentent de façon très importante avec l'accroissement du nombre d'entrées et de sorties.

1.2.3. Système d'interconnexions propriétaires

Comme nous avons pour objectif d'implémenter le système d'interconnexions sur FPGA, nous avons étudié les options disponibles dans les outils de conception des vendeurs de ces circuits. Il s'agit par exemple, du bus AXI4 pour Xilinx [XIL15], ou Qsys pour Altera/Intel FPGA [QSY]. Ces solutions sont potentiellement très performantes car elles sont développées par les entreprises qui réalisent les architectures matérielles des FPGA, et donc qui peuvent exploiter leur connaissance totale du matériel pour optimiser les systèmes d'interconnexions, notamment à travers les algorithmes de placements-routages.

Cependant, la principale limitation de ces systèmes propriétaires réside dans leur opacité. En effet, comme les codes sources de ces architectures sont secrets et chiffrés, il est donc impossible d'effectuer des recherches sur la présence éventuelle de failles de sécurité. Cette opacité est sans doute premièrement motivée par la protection du secret industriel, cependant, elle rentre souvent en contradiction avec les objectifs relatifs à la sécurit

1.2.4. Réseau sur puce (Network-on-Chip / NoC)

Les réseaux sur puce ou NoC semblent être une des principales solutions possibles pour résoudre le problème d'évolutivité des systèmes d'interconnexions. La structure des NoC reposent sur un réseau de routeurs, par l'intermédiaire desquels les IP échangent des messages. Pour de multiples raisons que nous détaillerons par la suite, c'est ce type d'interconnexions que nous avons choisi d'étudier plus particulièrement. Notamment, les NoC ne rencontrent pas la principale limitation à laquelle sont soumis les bus, car le medium de communication peut être prolongé par l'augmentation du nombre de routeurs. Les NoC ne sont pas non plus confrontés aux problèmes d'extensions qui touchent les crossbars, car le nombre d'entrées/sorties par routeur peut être limité. La **Figure 1.3** présente un exemple de réseau sur puce à topologie maillée à deux dimensions.

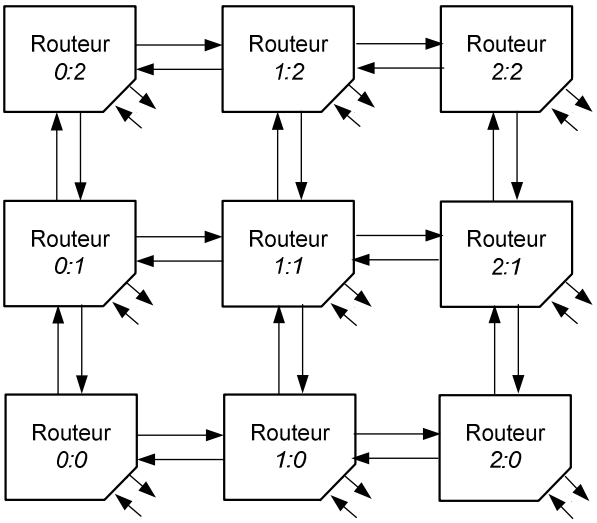


Figure 1.3 : Réseau sur puce de topologie maillée à deux dimensions

Il existe de très nombreuses déclinaisons de réseau sur puce. Pour les caractériser, on utilise un certain nombre de propriétés, tels que la topologie, la qualité de service, le type de commutation ou bien encore l'algorithme de routage, que nous allons décrire dans la suite.

1.2.4.1. Topologie

Cette caractéristique définit comment les routeurs sont connectés entre eux (**Figure 1.4**). Le choix d'une topologie relève souvent d'un compromis entre la distance moyenne entre les routeurs (aussi appelé *hop*) et la complexité d'implémentation.

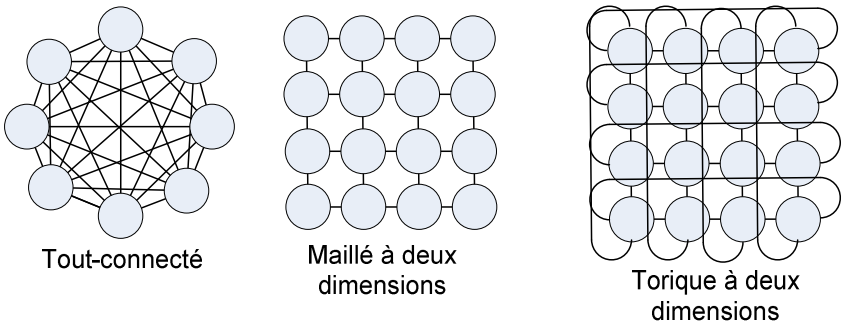


Figure 1.4 : Topologies possibles de NoC

1.2.4.2. Qualité de service des communications

La qualité de service des communications garantit que les transmissions dans le réseau sont effectuées avec le respect d'un certain nombre de contraintes en performances. Ces contraintes peuvent être exprimées en matière de latence ou bien de débit. Pour l'utilisateur cela se traduit par exemple, par le délai maximal de prise en compte de l'information émanant d'une entrée du système comme une interface de saisie. La qualité de service est aussi très importante pour les flux audios, et vidéos. Des variations de latence ou de débit dans ces flux peuvent engendrer des distorsions plus ou moins importantes de l'image ou du son émis par le système. La qualité de service des communications est indispensable pour les applications soumises à des contraintes temps réel.

1.2.4.3. Commutation de circuits

Avec ce mécanisme, une connexion est ouverte au préalable d'un échange entre deux IP. L'établissement de cette connexion réserve les ressources nécessaires à la réalisation de l'échange d'information, et prévient d'autres entités de pouvoir utiliser ces mêmes ressources tant que la connexion est ouverte (**Figure 1.5 - A)**). L'avantage de processus est de pouvoir garantir que la latence de transmission du message ne dépassera pas une valeur maximale, et ne variera pas de manière trop importante ; cela permettant donc d'offrir une certaine qualité de service.

1.2.4.4. Commutation de paquets

Avec la commutation de paquets, les informations sont transmises sous la forme de sous-blocs de données (**Figure 1.5 - B)**). Elle permet d'obtenir d'une bande-passante plus importante qu'avec la commutation de circuits, car la charge peut être répartie de manière plus homogène dans le réseau. Cependant la latence des échanges, et donc la qualité de service sont facilement perturbées par la congestion si aucun mécanisme tels que les canaux virtuels, la gestion des priorités, ainsi qu'un algorithme de routage adaptatif ne sont pas employés.

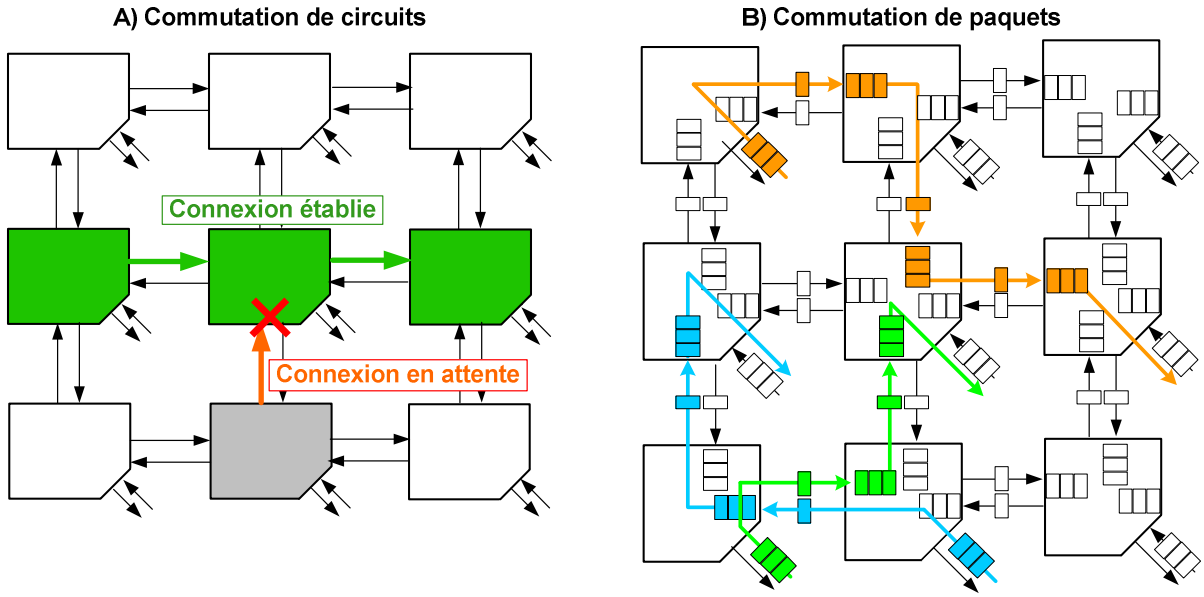


Figure 1.5 : Réseaux à commutation de circuits et à commutation de paquets.

Dans le cadre de la commutation de paquets, on dénombre trois façons de transmettre les flits (plus petite unité constituante d'un paquet). Chacune de ces façons réalise un compromis entre la complexité de mise en œuvre et la latence.

a. Store & Forward

Avec ce mécanisme, chaque paquet doit être complètement reçu par un routeur avant qu'il ne soit transmis au routeur suivant. Par conséquent, la taille des mémoires tampons positionnées dans les ports des routeurs doit être au minimum égale à la taille maximale des paquets. Cette méthode est très peu utilisée car elle limite la taille des paquets sous peine de devoir augmenter la taille des mémoires tampons. Elle impose aussi une latence importante.

b. Virtual Cut-Through

Il s'agit en quelque sorte, d'une version améliorée du mécanisme *Store & Forward*, puisque son fonctionnement repose sur le même principe, excepté qu'un routeur peut commencer à transmettre un paquet dès que le routeur receveur a assez d'espace pour le stocker dans son intégralité. Les auteurs de [MAI15] prétendent que l'implémentation de ce mécanisme est moins lourde que celle du mécanisme *wormhole*, et qu'il résiste mieux à la charge.

c. Wormhole

Dans cette méthode chaque paquet possède un en-tête qui contient toutes les informations nécessaires au routage. Celui-ci est transmis de routeur en routeur, tant qu'il y a assez d'espace pour le stocker. Le passage de l'en-tête dans un routeur réserve ce dernier pour la réception et la transmission du reste des flits qui compose le paquet. Le mécanisme *wormhole* est le plus communément utilisé dans les NoC évoqués dans la littérature, puisqu'il représente les meilleures performances en termes de latence, sans pour autant demander une grande complexité de mise en œuvre. Son deuxième avantage est qu'il ne nécessite pas la définition de tailles de mémoires tampons supérieures à la taille du paquet, ce qui permet d'échanger des paquets plus longs et d'employer des mémoires tampons plus compactes.

1.2.4.5. Algorithme de routage

Les algorithmes de routage ont pour rôle de déterminer les chemins que vont emprunter les paquets à l'intérieur du réseau pour se rendre à leur destination. Les nombreux algorithmes existants peuvent être classés en fonction de plusieurs caractéristiques comme leur capacité à **s'adapter ou non à la congestion** du réseau [RAN06]. Les **décisions** concernant le routage peuvent être **centralisées** et intégralement prises à l'émission des paquets, ou bien **distribuées** et réalisées au fur et mesure de leur cheminement dans le medium de communications.

Les algorithmes **non-adaptatifs d'ordre dimensionnel** (dont le « XY ») sont ceux qui sont les plus communément utilisés dans les articles de l'état de l'art des réseaux sur puce pour FPGA. La principale raison étant que leurs implémentations requièrent peu de ressources logiques.

Un des points principaux à retenir à propos des algorithmes de routage est que leur choix résulte la majorité du temps en un compromis entre la résistance des performances à la congestion, et la complexité d'implémentation en occupation de ressources, fréquence de fonctionnement et consommation d'énergie.

1.2.4.6. Conflits de routage ou d'arbitrage

Avec l'augmentation du niveau de charge dans un réseau ainsi que l'utilisation de certains algorithmes de routage ou d'arbitrage, plusieurs conflits risquent de se produire.

a. Deadlock

C'est le blocage définitif d'un ou plusieurs messages à un endroit du réseau. Cela peut se produire lorsque qu'un paquet cherche à occuper une ressource qui est elle-même occupée par un autre paquet, ainsi de suite, et ceci sans qu'aucune de ces ressources ne soit jamais libérée.

b. Livelock

Ce conflit se produit lorsqu'un paquet se retrouve dans une boucle infinie, sans jamais pouvoir atteindre sa destination. Les algorithmes XY et à restriction de virage (*Turn-Model*) ne peuvent pas rencontrer ce genre de collision.

c. Starvation

Cela se traduit par le fait qu'un paquet ne peut jamais avoir accès à une ressource demeure dans cet état indéfiniment. Ce conflit peut être causé par une logique d'arbitrage mauvaise ou inadaptée. Les NoC employant le multiplexage temporel *(Time Division Multiplexing* ou *TDM)*, c'est-à-dire, où chaque paquet se voit attribué une fenêtre temporelle propre, ne sont pas susceptibles de subir ce genre de conflit.

1.3. Sécurité des systèmes sur puce

1.3.1. Contexte de la sécurité numérique

Il apparaît comme une évidence que la sécurité est une pierre angulaire de notre société numérique. L'utilisateur lambda souhaite pouvoir contrôler la diffusion sur la toile de ses données personnelles, que ce soient ses photos de vacances, ou bien des informations plus critiques comme des données bancaires, ou relatives à sa santé. Les entreprises et organismes publics doivent se protéger des attaques visant la disponibilité de leurs infrastructures, ainsi que du cyber-espionnage. L'informatique fait partie intégrante de l'organisation de notre société de manière critique, à travers des domaines comme l'énergie, la défense, la santé, et l'économie.

Le développement de nouvelles technologies lance quotidiennement de nouveaux de défis du point de vue de la sécurité, avec entre autres l'explosion du secteur de l'internet des objets.

La **Figure 1.6** regroupe un ensemble d'attaques qui ciblent les systèmes sur puce [BOS08]. Il est difficile d'en établir une liste exhaustive, tant leur diversité et leur complexité sont grandes. La plupart des attaques citées compose de vastes domaines d'études qui ne cessent de connaître des évolutions. C'est le cas, par exemple, des chevaux de Troie matériels, des attaques en fautes, des attaques par canaux cachés, des attaques logicielles ou de la rétro-ingénierie. De plus, ces différentes techniques peuvent être combinées pour faire apparaître de nouvelles failles de sécurité. L'étude de l'architecture d'un circuit par rétro-ingénierie, permet par exemple, de réaliser des attaques en fautes, de manière beaucoup plus précise et efficace. Ceci-dit, de telles techniques peuvent être très complexes, donc il n'est pas nécessaire de développer des mécanismes de sécurité pour protéger des systèmes peu critiques contre des attaques très onéreuses.

Mis à part les attaques par cheval de Troie matériel, qui nécessitent un accès à la chaîne de production du circuit, ou l'emploi, par le développeur du système d'outils de conception malveillants, les attaques à distance sont quasi-exclusivement de type logiciel. C'est pour cette raison que les implémentations matérielles sécurisées sont très employées dans des domaines critiques, comme la monétique ou la défense. Une bonne implémentation matérielle permet de sécuriser des fonctions critiques, comme la génération et le stockage des clés cryptographiques.

Afin de sécuriser un système sur puce, il est donc nécessaire de ne négliger aucun des aspects de la couche matérielle, jusqu'au niveau logiciel. Lors de la conception et la fabrication du circuit, il faut veiller à ne pas avoir laissé la possibilité à des processus malveillants d'y être intégrés. En ce qui concerne les interfaces utilisées pour le test des circuits, elles doivent être désactivées avant le déploiement. Ensuite, les couches qui réalisent la liaison entre le niveau matériel et logiciel doivent toutes faire l'objet de la même attention du point de vue la sécurité. Concernant la couche logicielle, l'utilisation d'environnements cloisonnés est une manière de protéger les données manipulées par chaque processus applicatif. Il est important de noter qu'un ensemble de contre-mesures a fait l'objet de recherches et de développements, dans le but de lutter contre des menaces matérielles comme les attaques par canaux cachés, et les attaques par injection de fautes. Enfin, la sécurisation du système d'interconnexions est une manière efficace de contrer certaines menaces, principalement logicielles.

1.3.2. Taxonomie non exhaustive des attaques sur les systèmes sur puce

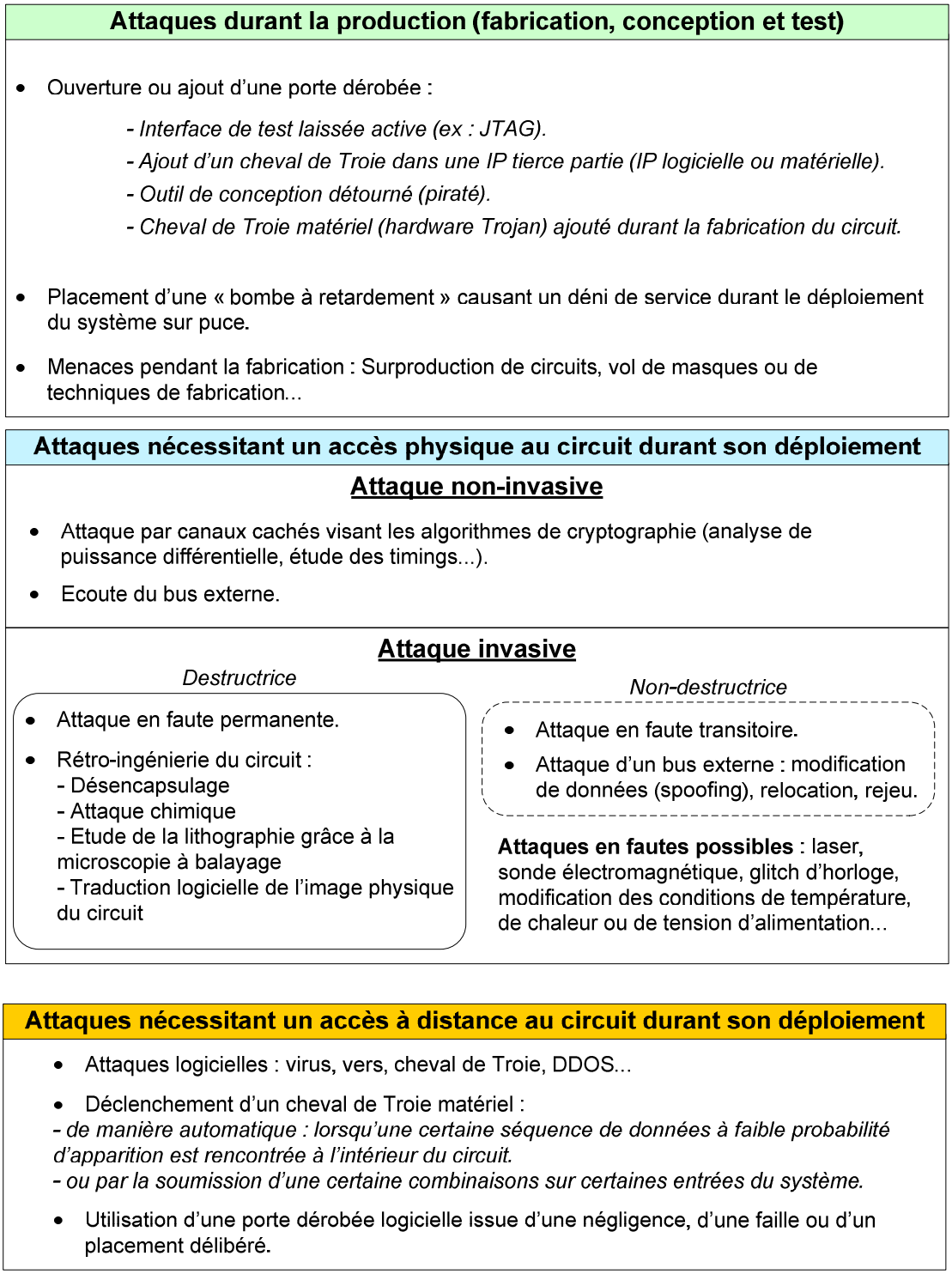


Figure 1.6 : Taxonomie non exhaustive des attaques pouvant viser les systèmes sur puce.

1.4. Problématiques et objectifs

Nous avons débuté ce chapitre en présentant le contexte académique et industriel de ce travail de thèse, et en évoquant les problématiques rencontrées à propos du choix d'un système d'interconnexions performant et sécurisé pour circuit FPGA.

En effet, le choix des circuits reconfigurables pour l'intégration d'un système sur puce est principalement motivé par des coûts de production plus faibles que les ASIC, pour des volumes réduits. Le fait que les FPGA soient reconfigurables peut faciliter les étapes de développement et de validation, puisqu'il est toujours possible de corriger un programme, et de le recharger dans le circuit. Les plateformes matérielles telles que les ASIC et FPGA ont aussi l'avantage de bénéficier de performances plus élevées que les CPU pour l'exécution de certaines tâches, comme des algorithmes de cryptographie.

Grâce à l'amélioration des technologies de conception et de fabrication, les systèmes sur puce sont capables de contenir de plus en plus d'IP, et de réaliser un plus grand nombre de fonctions. Ceci dit, il est primordial que le système d'interconnexions fournisse suffisamment de bande-passante pour soutenir les communications de toutes les IP. Il est d'égale importance que la latence, et le coût en surface du réseau utilisé soient minimes. C'est dans ce contexte que les réseaux sur puce semblent être une des solutions les plus viables de par le large spectre des possibilités qu'ils offrent. Il nous reste à chercher et à identifier l'architecture de réseau sur puce, la plus apte à répondre à nos contraintes applicatives.

Bien que les circuits FPGA et ASIC soient similaires sur de nombreux aspects, leurs spécificités jouent un rôle crucial dans la conception d'un réseau sur puce. En effet, les ASIC sont taillés sur mesure pour exécuter une application donnée, alors que les ressources logiques des circuits reconfigurables sont disposées et quantifiées avant que l'application soit connue. C'est donc dans une certaine mesure, à l'architecture du réseau à s'adapter à celle du FPGA, afin de pouvoir atteindre les meilleures performances possibles. **Un de nos objectifs est donc d'identifier à travers un état de l'art, ainsi que des développements, une architecture performante de réseaux sur puce pour FPGA, et répondant au mieux à nos contraintes applicatives.**

Comme nous l'avons déjà mentionné, la sécurité touche tous les niveaux de l'architecture d'un système sur puce, aussi bien matériels que logiciels. Les NoC sont intégrés au cœur des systèmes sur puce, gérant les échanges de données entre les processeurs, les mémoires, les interfaces de communication et les processeurs dédiés. Il est donc fondamental de s'interroger sur le fait que l'intégration d'un réseau sur puce puisse potentiellement créer de nouvelles failles de sécurité, et trouver des manières de les combler. De plus, **nous considérons l'ajout de mécanismes de sécurité au niveau matériel pour contrôler les échanges de données. Ceci dans le but de parer les effets d'attaques matérielles ou logicielles sur le système sur puce.**

Dans l'optique de proposer des mécanismes de sécurité efficaces, notre premier objectif est donc de rechercher les attaques qui peuvent cibler un système sur puce programmable, et son architecture de communications. A partir de ces recherches, nous dressons un modèle de menaces afin de concevoir et développer des contre-mesures adéquates, et ayant un surcoût limité sur les performances.

1.5. Contraintes applicatives et objectifs en performances

C'est la confrontation du service Trustway (BULL) aux verrous technologiques durant ses conceptions d'équipements de sécurité, qui a motivé ce projet de recherche. Dans ce cadre, un certain nombre de contraintes et d'objectifs ont pu être définis pour juger de la viabilité des solutions proposées pendant cette préparation de thèse.

Un impératif de confidentialité nous empêche de décrire précisément les architectures matérielles des produits développés par Trustway. On peut tout de même indiquer que les systèmes d'interconnexions habituellement employés par ce service montrent leur limite lors de leur utilisation sur FPGA pour le support des communications d'un très grand nombre d'IP.

Les modules matériels de sécurité sont concernés par des impératifs de performances en matière de débit et de latence. Ils doivent également pouvoir fournir une grande variété de services cryptographiques, ce qui explique la complexité de leur architecture.

Par conséquent, nous cherchons à répondre à plusieurs objectifs, comme celui de proposer une structure supportant les communications d'une centaine d'IP de manière sécurisée. En outre, elle doit pouvoir être intégrée dans le FPGA Altera de plus grande capacité de la famille ARRIA 10 SoC. L'interconnexion ne pouvant occuper la majorité des ressources logiques du circuit, sa consommation ne peut excéder 30% de celles-ci, afin de laisser suffisamment d'espace au reste du système sur puce.

Pour des questions de compatibilité, les interfaces du réseau doivent dans un premier temps pouvoir supporter des IP avec des largeurs de données de 32 bits. Par conséquent, dans l'optique de fournir une bande-passante suffisante aux IP, la fréquence du réseau doit avoisiner une valeur minimale de 150 MHz.

Concernant le motif des communications, l'application du module de sécurité fournit à la fois des services de cryptographies symétriques et asymétriques. Pour les architectures visées par nos recherches, la cryptographie symétrique qui a tendance à solliciter fortement les communications, n'est utilisée que ponctuellement pour traiter des faibles quantités de données. Quant à la cryptographie asymétrique, elle sollicite plus fortement les capacités de calculs que celles des communications.

Le schéma d'échanges des données étant donc composé d'une alternance de commandes, et de paquets de données de 256 bits à 8192 bits (selon la taille de la clef cryptographique), le support des bursts est nécessaire afin de maximiser les performances en débit.

Enfin, afin de faciliter l'intégration de notre éventuelle solution, nous voulons éviter la proposition de plusieurs interconnexions hétérogènes. L'idéal étant de pouvoir proposer un seul système d'interconnexion commun à toute les IP, et qui peut offrir à certains groupes d'IP la possibilité d'échanger des données avec une latence minimale de quelques cycles d'horloges. Le cas typique concerné par cet objectif est celui des communications entre des cœurs de calcul et leurs mémoires de travail.

Chapitre 2) Etat de l'art

Le chapitre précédent a permis d'exposer les limites actuelles des systèmes d'interconnexions classiques, et d'évoquer à travers les réseaux sur puce, une possible solution à ces problématiques. Les questions de sécurité autour des systèmes numériques modernes ont également été abordées, en effet, elles sont d'une importance capitale dans la conception des systèmes sur puce programmable et de leurs architectures de communications.

Nous allons maintenant détailler l'avancement actuel des travaux de recherche de la communauté scientifique, à propos des réseaux sur puce. En particulier, nous passerons en revue la littérature à propos de la conception adaptée aux plateformes FPGA qui tente de tire au mieux parti de leur architecture matérielle pour atteindre les meilleures performances possibles.

Sommaire du Chapitre 2

2.1. Spécificités des circuits FPGA

Les systèmes sur puce sont généralement conçus grâce à un langage RTL *(Register Transfer Level)* comme le VHDL ou le Verilog. Le comportement interne d'un circuit peut être décrit grâce à ces langages, par la connexion de registres, de signaux, et de fonctions logiques combinatoires ou séquentielles, ceci dans le but de réaliser de nouvelles fonctions pouvant être très complexes.

La composition interne d'un circuit en langage RTL peut être ensuite employée pour fabriquer un ASIC, ou bien pour configurer un FPGA. Ces derniers sont des circuits reconfigurables, composés de blocs de granularité plus ou moins importante. Les blocs de logique et les interconnexions reprogrammables sont ceux de plus fine granularité, puis des blocs dédiés occupant plus d'espace, sont positionnées sur toute la surface du circuit FPGA. Ces derniers peuvent être des multiplieurs ou des mémoires RAM ; étant utilisés par la majorité des applications configurées sur FPGA, les fonctions de traitement numérique de signal ou le stockage de quantités importantes de données est plus performant avec des blocs dédiés qu'avec des blocs de logique programmables.

Dans l'optique de configurer un FPGA avec l'architecture d'un circuit décrite en langage RTL, un outil conception permettant de réaliser une étape **synthèse**, de **placement**, de **routage** et de **génération du bitstream**, doit être employé. Durant l'étape de **synthèse,** la cohérence de la syntaxe du code RTL est d'abord vérifiée, puis celui-ci est traduite dans un langage de plus bas niveau, décrivant les différents types et nombre de ressources matérielles du FPGA devant être configurés. Si la synthèse est réalisée avec succès, les étapes de **placement** et de **routage** des ressources logiques du circuit vont être exécutées conjointement, dans le but de répondre aux contraintes fixées par le concepteur du système.

Il s'agit de deux étapes itératives, se répétant plusieurs fois, jusqu'à satisfaire au mieux toutes les contraintes spécifiées. Le concepteur du système peut déterminer ces contraintes, comme la fréquence de fonctionnement à atteindre. Il peut aussi faire des choix d'optimisations, notamment en favorisant l'optimisation de la fréquence de fonctionnement au détriment du nombre de ressources logiques occupées. Le temps de recherche de la meilleure solution pour les algorithmes de placement-routage peut-être plus ou moins long, sachant que plus le temps de recherche est long, moins les optimisations de performances seront significatives. Enfin, l'étape de **génération du bitstream** produit le fichier de configuration du FPGA.

Le composant matériel essentiel des FPGA est la LUT (*lookup table*), elle peut être combinée pour réaliser des fonctions logiques complexes (**Figure 2.1**).

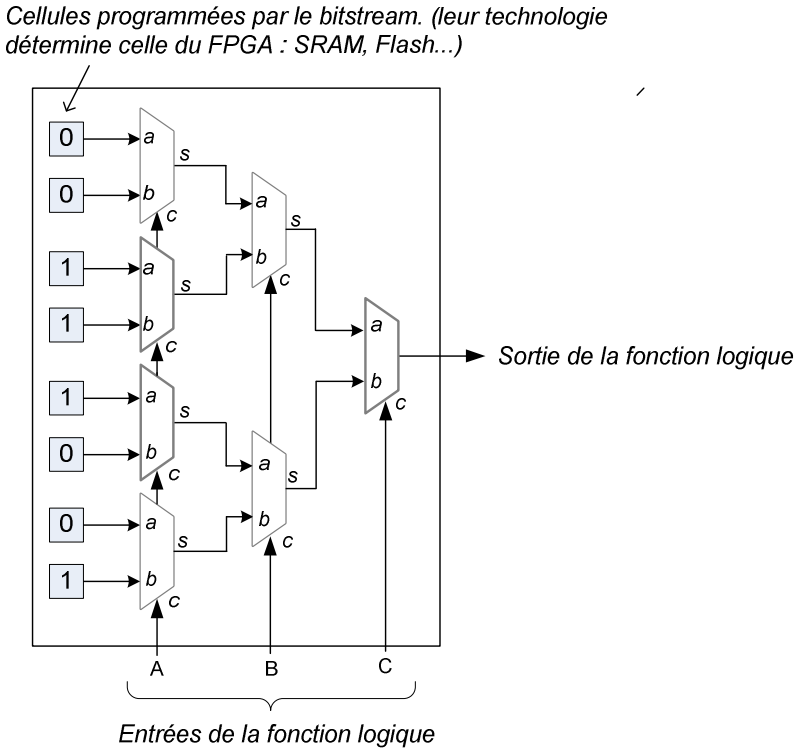


Figure 2.1 : Exemple d'architecture d'une LUT à trois entrées et une sortie.

La présence de LUT à elle seule, ne permet pas de réaliser des systèmes très complexes puisqu'il manque encore la notion d'éléments « mémoires » de la logique séquentielle. Par conséquent, les blocs logiques qui composent la majorité des FPGA actuels comportent en plus des LUT, de la mémoire RAM sous forme de registres synchrones, des chaînes de propagation de retenues pour les additionneurs logiques ainsi que des multiplexeurs.

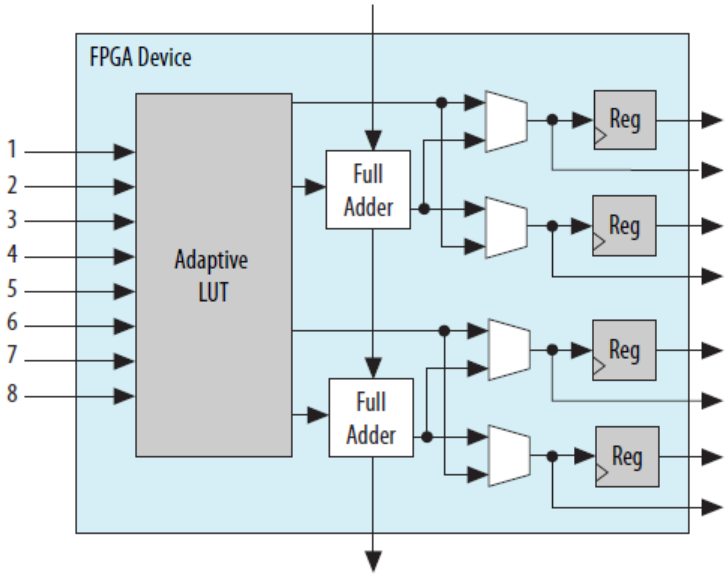


Figure 2.2 : Composition des ALM des FPGA Arria 10 (extrait de [ALT10]).

Les blocs logiques différent selon les concepteurs de FPGA et même selon les diverses familles de circuit. Le vendeur Altera les nomme « ALM » *(Adaptive Logic Module)*, alors que pour Xilinx il s'agit des « CLB » *(Configurable Logic Bloc)*. Les singularités entre les plusieurs types de blocs logiques peuvent résider dans le nombre et la taille des LUT présentes ; mais aussi dans la manière dont les blocs s'interconnectent entre eux.

La **Figure 2.2** illustre la composition du bloc logique présent dans les FPGA ARRIA10 du vendeur Altera. Dans cet exemple, la LUT employée possède 8 entrées et elle est dite « adaptive » car elle peut être divisée pour réaliser différentes combinaisons de plusieurs LUT, ayant chacune de 2 à 6 entrées selon les cas.

Le compromis entre la complexité de mise en œuvre et les performances, explique que les concepteurs des architectures des FPGA ne souhaitent pas augmenter le nombre d'entrées des LUT au-delà d'un certain seuil. La combinaison de plusieurs LUT appartenant à des blocs logiques différents, est actuellement suffisante pour implémenter des fonctions complexes avec des performances satisfaisantes.

La présence des registres à l'intérieur des blocs logiques, apporte la possibilité de synchroniser via une horloge, les fonctions combinatoires configurées dans les LUT. C'est par ce biais que l'intégration de la logique séquentielle dans les FPGA est possible. Un même FPGA peut comporter plusieurs horloges ayant chacune sa propre fréquence. Une horloge est transportée jusqu'aux registres grâce à un « arbre d'horloge ».

Le deuxième élément étant initialisé par fichier de configuration, est le réseau d'interconnexions. Une fois que les blocs de logique sont configurés, il reste à définir la manière dont ils sont connectés les uns avec les autres, c'est le rôle joué par le réseau d'interconnexions. Aux carrefours des interconnexions, se trouvent des matrices reprogrammables configurées par le bitstream, qui définissent la manière dont tous les blocs sont connectés entre eux.

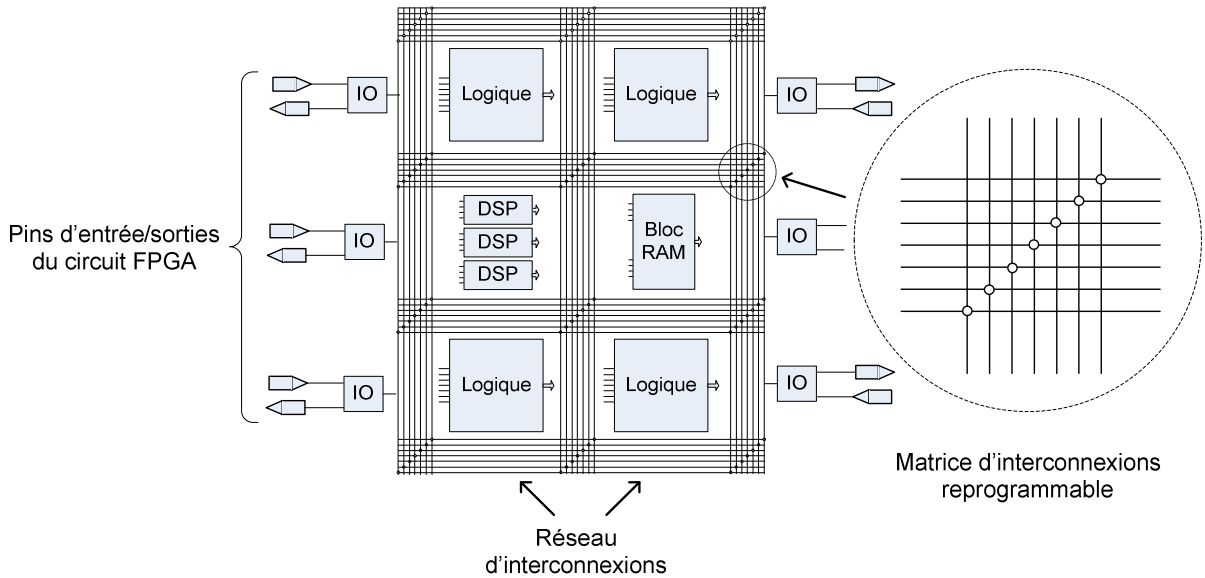


Figure 2.3 : Représentation simplifiée de l'architecture interne d'un FPGA

La **Figure 2.3** représente la manière dont une architecture matérielle interne d'un FPGA peut être constituée, avec un réseau d'interconnexions qui entoure et connecte les différents blocs logiques et blocs dédiés.

Une des différences majeures avec les ASIC, **qui a un impact important sur la conception des réseaux sur puce**, c'est que la quantité et le positionnement des ressources physiques sont prédéterminés. Les architectes qui conçoivent les circuits FPGA, ne peuvent savoir par avance quelle application va être programmée dans la puce. Ils doivent donc dimensionner les différentes ressources internes, de façon à ce que le circuit soit suffisamment polyvalent et performant ; contrairement à un ASIC qui est taillé sur mesure pour l'application qu'il supporte. Réciproquement, cela signifie que le fait d'adapter un système pour l'architecture d'une puce FPGA permet d'atteindre de meilleures performances.

Il existe différentes technologies de FPGA qui définissent en réalité, la technologie des cellules mémoires, à l'intérieur des LUT et des matrices d'interconnexions, configurées par le bitstream. La technologie SRAM est celle qui est très majoritairement utilisée actuellement. Il s'agit d'une technologie volatile, ce qui signifie que le circuit perd sa configuration lorsqu'il est mis hors tension. C'est aussi la technologie reconfigurable la plus performante en termes de vitesse et d'occupation en ressources, puisque son procédé de fabrication a été l'un des plus employés et améliorés durant ces dernières années. Par contre elle assez consommatrice d'énergie. Le niveau d'intégration de la technologie SRAM est le plus important des technologies reconfigurables, à savoir, comptant le plus de cellules mémoires pour une surface donnée, ce qui est intéressant pour fabriquer des circuits FPGA de faible surface.

Bien qu'étant moins performantes que les mémoires SRAM, les mémoires flash ont l'avantage d'être plus résistantes aux radiations et de consommer moins de puissance que la SRAM. Leur non-volatilité, est intéressant du point de vue de la sécurité, car elle retire la nécessité de reprogrammer le circuit à chaque fois qu'il est mis hors tension, puis rallumé ; et donc cela empêche un attaquant de pouvoir intercepter le fichier de configuration envoyé au circuit, bien que ce dernier puisse être chiffré. Les circuits de cette catégorie, semblent plus adaptés pour les domaines de l'aéronautique ou du spatial, où les systèmes sont soumis à des niveaux de radiation élevés pouvant provoquer des erreurs logiques à l'intérieur du circuit.

Un autre genre de FPGA utilisable est celui de type *anti-fusible*. Le fonctionnement d'un anti-fusible est l'inverse de celui d'un fusible, c'est-à-dire que lorsqu'un courant dépassant un certain seuil le traverse, cela crée un court-circuit. Les FPGA de cette famille ne sont programmables qu'une seule fois, et donc ne sont pas reconfigurables. Ils sont également indiqués pour les applications nécessitant un niveau de résistance élevé aux radiations ou aux contraintes de sécurité très importantes, car leur configuration ne peut être modifiée.

2.2. Métriques de caractérisation des NoC

Les performances d'un réseau sur puce sont évaluables à travers un ensemble de métriques. Une fois mesurées, celles-ci permettent d'établir des comparaisons entre plusieurs réseaux sur puce, ou bien de déterminer si un NoC est capable de supporter les communications d'une application donnée.

2.2.1. Latence de transport des messages et résistance à la charge

La latence ou le délai de transmission définit l'intervalle de temps qui sépare l'envoi d'un message par un expéditeur, et sa réception par un destinataire. Elle est généralement mesurée en nombre de cycles d'horloge. Il est possible de considérer les variations de latences, ainsi que les valeurs minimales, maximales et moyennes. De manière générale, la latence est un facteur important de la **qualité de service des communications d'un réseau**, puisque si la latence fluctue de manière importante, le service rendu par l'application risque d'être dégradé. Pour garantir une certaine qualité de service, une solution est de s'assurer que la latence maximale de transmission des messages ne dépasse pas une certaine valeur seuil.

La résistance à la charge est une des métriques les plus importantes des réseaux sur puce. Il s'agit de l'évolution de la latence en fonction du taux d'occupation du réseau. Pour la majorité des NoC, l'augmentation du nombre de messages injectés dans le réseau va se traduire par des phénomènes de congestion, par exemple, lorsque plusieurs paquets cherchent à avoir accès à une ressource qui ne peut être utilisée que par un seul d'entre eux. Si l'accroissement du taux d'injection se poursuit, le réseau va finir par atteindre un point de saturation à partir duquel la latence moyenne de transmission des messages va augmenter de manière exponentielle. Cela se traduit dans les fait par le blocage des paquets, qui ne pourront donc ne jamais atteindre leur destination.

2.2.2. Débit

Le débit est la **quantité de données transmise durant un intervalle de temps spécifique**, exprimé en secondes ou en cycles d'horloge. La mesure de débit peut porter sur un seul échange entre un émetteur et un récepteur à charge de réseau nulle, ceci pour obtenir la bande-passante théorique maximale entre deux nœuds, ou bien, viser la quantité totale de données sortant du réseau lorsque ce dernier a atteint un état de stabilité à un niveau de charge donné.

Le fait d'augmenter la largeur des liens de données, par exemple de 32 bits à 64 bits, peut avoir l'effet d'augmenter drastiquement le débit. Il est par contre nécessaire, que les IP connectées au réseau puissent gérer cette largeur de données, et que cette augmentation n'ait pas pour effet de trop réduire la fréquence de fonctionnement du NoC.

2.2.3. Consommation d'énergie

La consommation totale d'un circuit est la somme de la **consommation statique**, c'est-à-dire le courant de fuite des transistors lorsque le circuit est au repos ; et de la **consommation dynamique**, lors de la commutation des transistors. De nombreux moyens ont été investigués pour tenter de réduire la consommation des systèmes sur puce, notamment en diminuant les valeurs des tensions d'alimentation, ou des fréquences de fonctionnement, voire en déconnectant certaines parties du système lorsqu'elles ne sont pas utiles.

Pour réduire la consommation en énergie d'un réseau sur puce, il faut tenter de réduire au maximum la valeur du rapport : **complexité d'implémentation** sur **performances**. Plus les algorithmes de routage, ou d'arbitrage sont complexes et occupent des ressources matérielles, plus ils sont consommateurs de puissance. Cependant, plus ils sont performants, plus ils sont susceptibles de réduire le temps de transport des messages et donc de réduire la consommation dynamique du circuit.

2.2.4. Surface et occupation en ressources

La métrique de surface concerne plutôt les circuits ASIC, alors que l'occupation des ressources est utilisée pour les FPGA. Plus la complexité d'un système est élevée, plus sa surface ou son occupation en ressources vont être grands. Pour les ASIC, un circuit de faible surface est synonyme d'un coût de fabrication moins important.

Comme nous l'avons vu, les FPGA sont composés d'un certain nombre de ressources de différentes natures. Après le placement-routage d'une application donnée ou d'un réseau sur puce, l'outil de conception rédige un rapport qui indique le nombre et le type de ressources nécessaires pour implémenter cette application dans le circuit. Evidemment, une modification des paramètres architecturaux du NoC comme la taille des paquets ou celle des mémoires tampons, entraîne une modification de l'occupation en ressources. Certaines estimations peuvent être appliquées, afin de traduire un coût en ressources sur FPGA, en une surface de circuit sur ASIC. Cependant, elles ne sont pas considérées comme très précises et sont très dépendantes de la génération et de la technologie des circuit comparés.

2.2.5. Fréquence de fonctionnement

Les préoccupations en termes de fréquence de fonctionnement sont différentes pour les circuits ASIC et FPGA. Pour un même système sur puce, la fréquence de fonctionnement d'une implémentation sur ASIC va être beaucoup plus élevée que sur FPGA. C'est logique, puisque l'ASIC est taillé sur mesure pour l'application, alors que ce n'est pas le cas sur FPGA, qui compte des éléments supplémentaires comme les canaux de configuration des ressources reprogrammables.

Pour évaluer un réseau sur puce, on va effectuer des placements-routages afin de déterminer quelle est la fréquence maximale que le réseau peut atteindre, pour des conditions de tension et de température données. Ces conditions sont généralement sélectionnées pour représenter le « pire cas », c'est-à-dire lorsque le NoC risque d'être le plus lent.

Un chemin comprend tous les éléments logiques combinatoires, comme les LUTs, entre deux registres synchrones, chacun d'eux rajoutant un certain délai de propagation. Ce délai de propagation peut être défini par le temps que va mettre la valeur de sortie d'un élément de logique combinatoire, à s'actualiser en fonction du changement des valeurs d'entrées. Pour chaque chemin, il suffit d'additionner la valeur du délai de propagation de chaque élément logique qui le compose, dans des conditions de température et de tension données, pour obtenir son délai de propagation total.

Un concepteur a parfois la possibilité d'augmenter la fréquence de fonctionnement d'un système en ajoutant un étage de synchronisation dans le chemin critique ; ce qui a pour effet de scinder ce dernier deux parties.

2.3. Etat de l'art des travaux sur les réseaux sur puce pour FPGA

2.3.1. Introduction

Les réseaux sur puce ont été initialement conçus pour les circuits ASIC afin de répondre à la problématique du support des communications d'un nombre élevé de nœuds de manière performante. Avec l'amélioration des technologies de conception, cette problématique a également fini par toucher les circuits reconfigurables, d'où la raison du développement des NoC sur FPGA.

Ceci-dit, bien qu'il y ait de nombreuses similarités entre ASIC et FPGA, chacune des deux plateformes possède des spécificités matérielles, ce qui explique qu'un réseau sur puce peut atteindre des performances supérieures à un autre sur l'une des des deux plateformes. En effet, les ASIC sont des circuits taillés sur mesure pour une application donnée, alors que les circuits FPGA sont conçus pour pouvoir en accueillir un large spectre. De ce fait, l'architecture du circuit reconfigurable se doit d'être polyvalente, ce qui modifie les contraintes autour de l'utilisation des mémoires et des interconnexions.

2.3.2. Support de la reconfiguration dynamique partielle

Une des différences majeures entre les FPGA les ASIC est la capacité qu'ils ont à être reprogrammés. Il est même possible pour certains modèles de modifier en partie leur programme durant leur fonctionnement, cela s'appelle la reconfiguration dynamique partielle. Cette technique offre un niveau de souplesse matériel proche de celui du logiciel, cependant, le réseau sur puce doit avoir été conçu pour pouvoir supporter de tels changements.

C'est notamment le cas de [MAR02] un des premiers travaux qui s'intéressent au développement d'un NoC spécifiquement pour circuit FPGA, et qui supporte la reconfiguration dynamique partielle. En effet, le retrait, l'ajout ou bien la modification d'un nœud du réseau a de grandes chances d'entraîner des différences importantes dans le fonctionnement des communications. Par ailleurs, le réseau présenté dans [MAR02] possède des caractéristiques d'un réseau sur puce pour ASIC assez classique, comme la commutation de paquets, le mode de transmission en *wormhole*, la topologie torique, et l'algorithme de routage XY.

C. Hilton et B. Nelson proposent également un réseau supportant la reconfiguration dynamique partielle, qui s'appelle PNOC [HIL06]. Ce réseau sur puce est pensé pour atteindre des performances élevées sur FPGA, grâce à une architecture employant la commutation de circuits, plus légère que la commutation de paquets. L'utilisation d'une connexion pour les communications a le bénéfice d'imposer une latence fixe lors des transferts de données. Cependant, avec la commutation de circuits, les ressources du réseau comme les routeurs sont partagées de manière moins homogène qu'avec la commutation de paquets, ce qui réduit la bande-passante générale du réseau. L'architecture des routeurs de PNOC est elle aussi très spécifique, puisqu'ils peuvent être connectés à jusqu'à 8 nœuds (**Figure 2.4**). Plusieurs canaux peuvent être employés dans une connexion entre deux routeurs afin de diminuer les phénomènes de goulot d'étranglement. Enfin, les nœuds connectés à un même routeur communiquent par paire de manière indépendante. Par rapport au NoC a commutation de paquets présenté dans [BAR03], les auteurs de [HIL06] indiquent obtenir une fréquence trois fois plus grande, et une consommation de ressources deux fois moins importante. Ceci avec le

même FPGA Virtex-II Pro ; pour un seul routeur PNOC possédant 8 ports, au lieu de 8 routeurs pour [BAR03]. Afin de compléter ces résultats, il serait intéressant d'observer l'évolution des performances lorsque l'on augmente le nombre de nœuds supporté par le réseau, obligeant l'utilisation de plusieurs routeurs pour le réseau PNOC. L'étude de la résistance à la charge de PNOC, et sa comparaison avec celle d'un réseau à commutation de paquets permettrait également de mieux cerner ces cas d'utilisation idéaux.

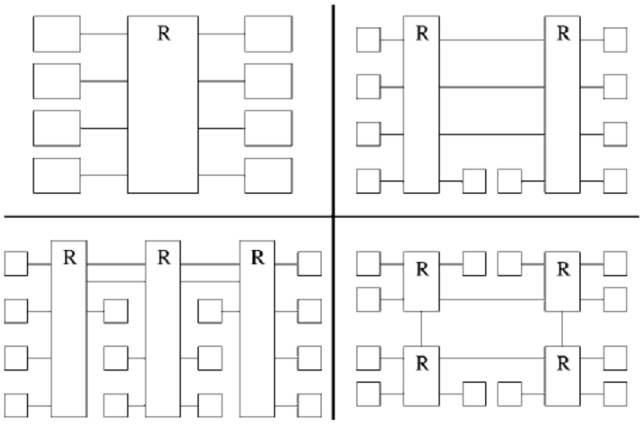


Figure 2.4 : Quatre topologies possibles de PNOC (schéma extrait de [HIL06]).

La philosophie du NoC supportant la reconfiguration dynamique présenté dans [MOL10] est radicalement différente de celle des deux réseaux présentés précédemment. En quelque sorte, il s'agit de leur combinaison, puisque la commutation de paquets est employée en parallèle de la commutation de circuits. Les paquets conviennent au transport de messages de faible taille, cependant, ils sont également utilisés pour la transmission de demandes d'établissements de connexions, dans le but de supporter des longues communications. La reconfiguration dynamique partielle sert à configurer les routeurs qui emploient la commutation de circuits. Grâce à cette technique, la complexité de ces routeurs est réduite au maximum puisqu'ils n'ont besoin d'aucune logique d'arbitrage et de routage. De ce fait, la latence qu'ils proposent est minimale, offrant une très bonne qualité de service.

Le désavantage de cette technique est d'être relativement complexe et lente à employer, notamment à cause du manque de maturité de la technologie de reconfiguration dynamique partielle. D'une part, il faut stocker en mémoire les 120 configurations de routeur possibles qui représentent toutes les combinaisons existantes de connexions entre les ports d'entrées et de sorties. Ces configurations sont employées pour modifier dynamiquement les routeurs lors de l'établissement d'une connexion.

D'autre part, lors de leurs expérimentations ces chercheurs ont estimé que le processus de reconfiguration des routeurs prenait environ 54ms, comprenant le temps de réaction du PC hôte, la recherche des fichiers de configurations adéquats, ainsi que le temps effectif de reconfiguration de 17 ms à travers l'interface JTAG. Une des améliorations proposées est de surcadencer la fréquence de l'interface de reconfiguration de 100 MHz à 300 MHz. Cette approche est intéressante, cependant elle ne sera réellement utilisable qu'une fois le processus de reconfiguration dynamique largement accéléré.

2.3.3. Intégration des mémoires tampons

Les mémoires tampons ou buffers, déterminent la manière dont sont stockés les paquets dans le routeur en attendant leur transmission. Leur taille et leur placement sont d'importance capitale puisqu'ils ont un très large impact sur les performances en latence, en résistance à la charge, en fréquence, et en occupation de ressources. Cet impact est d'autant plus grand sur circuit reconfigurable, que la mémoire interne se trouve en quantité limitée dans le FPGA, et est souvent responsable des chemins critiques.

Concernant l'intégration des mémoires tampons, les quatre solutions suivantes sont les plus souvent utilisées : **en entrée** du routeur, **en sortie** du routeur, comme **canaux virtuels**, ou **non utilisées** (*bufferless*). Aucune de ces possibilités n'est idéale, chacune présentant ses avantages et ses inconvénients, et pouvant mieux convenir à certaines applications qu'à d'autres.

La taille (ou la profondeur) des buffers définit la capacité qu'ont les routeurs à stocker un certain nombre de flits ou de paquets. Les **Figure 2.5** et **2.6** extraites de la thèse de Séverine Riso [RISO06], montre l'effet de la taille des buffers placés en entrée du routeur, sur la résistance à la charge du réseau.

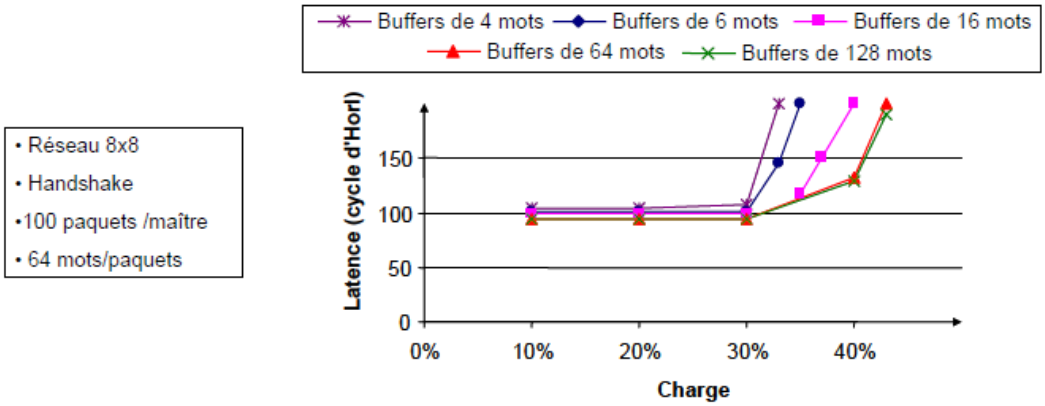


Figure 2.5 : Evolution de la latence pour différentes tailles de buffers (source : [RISO06])

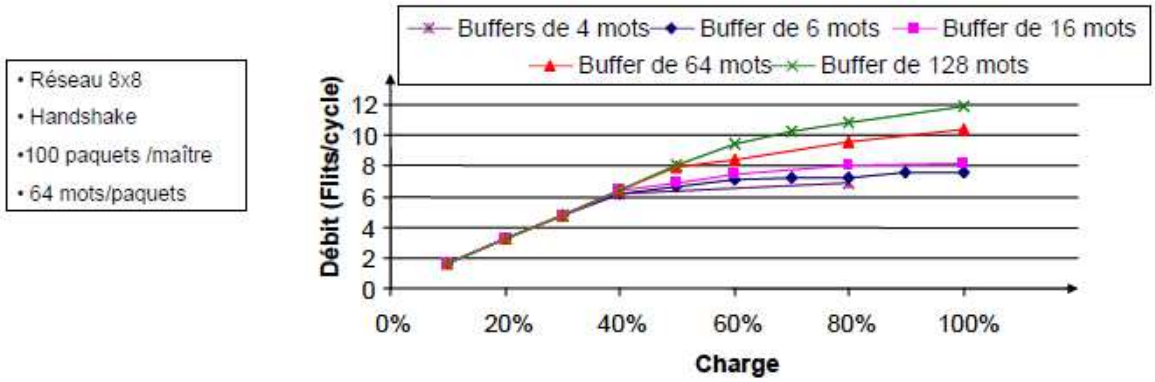


Figure 2.6 : Evolution du débit pour différentes tailles de buffers (source : [RISO06]).

Comme le montrent ces figures, l'accroissement de la taille des buffers retarde l'arrivée du point de saturation, et permet au réseau d'atteindre des débits supérieurs lorsque le niveau de charge est important. Que ce soit pour la latence et le débit, l'effet de la taille des buffers n'est pas perceptible jusqu'à ce qu'un certain niveau de charge ait été atteint. Pour être optimisé, ce paramètre doit donc être réglé en fonction du niveau de charge spécifique de l'application.

Le **Tableau 2.1** résume pour chaque article de notre état de l'art des NoC pour FPGA, la méthode employée pour la mémorisation des paquets.

Article	Types de mémorisation proposés
[MAR02]	• Deux **canaux virtuels en entrée** multiplexés temporellement.
[BAR03]	• Un buffer unique par **sortie**.
[SET05]	• Un buffer unique par **entrée** et par **sortie**.
[HIL06]	• **Aucun buffer** : commutation de circuits.
[KAP06]	• Un buffer à **l'entrée** de chaque primitive *split*, et un buffer de **canal** pour chaque entrée de la primitive *merge*.
[EHL07]	• Un buffer par **entrée**.
[SCH08]	• Deux **canaux virtuels en entrée**. *Ou* • Un buffer par **entrée**.
[YEL11]	• Un buffer par **entrée**.
[PAP12]	• Un buffer par **entrée**. *Ou* • Un buffer par **sortie**. *Ou* • **N canaux virtuels** par entrée.
[HUA12]	• Un buffer **en entrée** de chaque primitive split, et un buffer de **canal** en amont de chaque primitive merge.
[MAI15]	• Deux **canaux virtuels en entrée**.
[KAP15]	• Sans buffer *: bufferless*.
[CHE15]	Utilisation d'aucune, d'une seule, ou de ces deux solutions : • Un buffer **en entrée**, en amont de l'étage de routage (*split*). • Buffer de **canal** en amont de l'étage de Buffers en **entrée** positionnés après l'étage de routage *(split)* et avant l'étage d'arbitrage *(merge)*.

Tableau 2.1 : Structures des buffers utilisées dans notre l'état de l'art des NoC pour FPGA.

Il est intéressant de noter que certains types de mise en mémoire tampons sont combinées. Par exemple, [SET05] propose à la fois une mémorisation au niveau du port d'entrée, et du port de sortie. Il y a également[SCH08] qui propose d'utiliser à la fois le positionnement des buffers en entrée ainsi que les canaux virtuels.

- Positionnement d'un buffer par entrée :

La mise en tampon des données en entrée du routeur est la plus classique et la plus communément utilisée. Un exemple d'architecture de routeur employant ce type de mémorisation est représenté sur la **Figure 2.7**. La mise en tampon en entrée offre la possibilité d'utiliser une logique de routage centralisée ou décentralisée, c'est-à-dire, soit partagée entre toutes les entrées, soit dédiée à chacune d'entre elles. La logique décentralisée offre de meilleures latences mais augmente le coût en ressource logique.

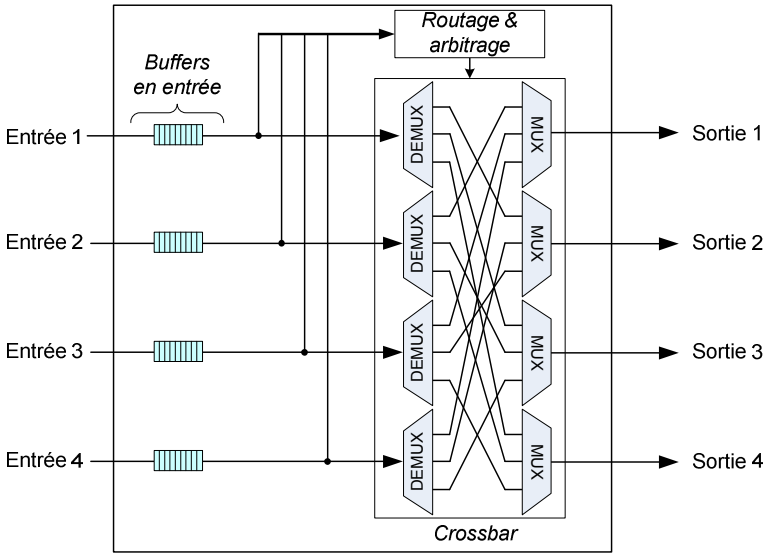


Figure 2.7 : Routeur employant un seul buffer en entrée.

Un inconvénient rencontré avec le positionnement d'un seul buffer en entrée du routeur est le blocage de type « *Head of Line* » ou *HoL*. Cela signifie que le paquet en tête de file bloque tous les autres paquets qui le suivent tant que celui-ci n'est pas routé (**Figure 2.8**).

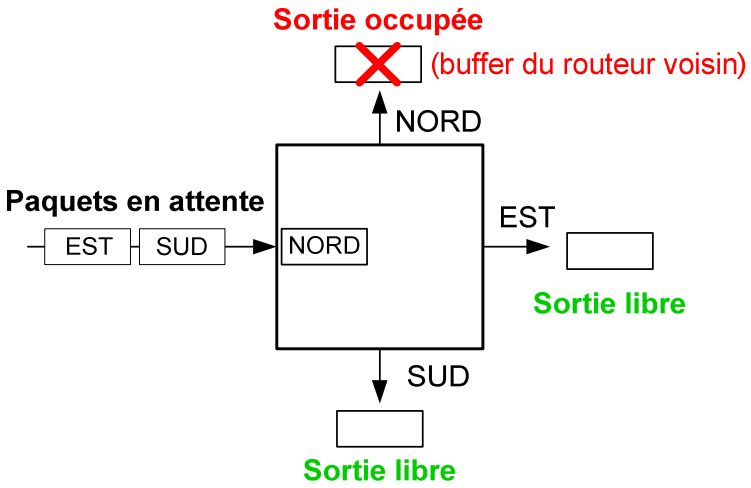


Figure 2.8 : Blocage « Head of Line ».

Pour pallier à ce conflit de routage l'utilisation de plusieurs buffers en entrée du routeur est préconisée **Figure 2.9** et **2.10**.

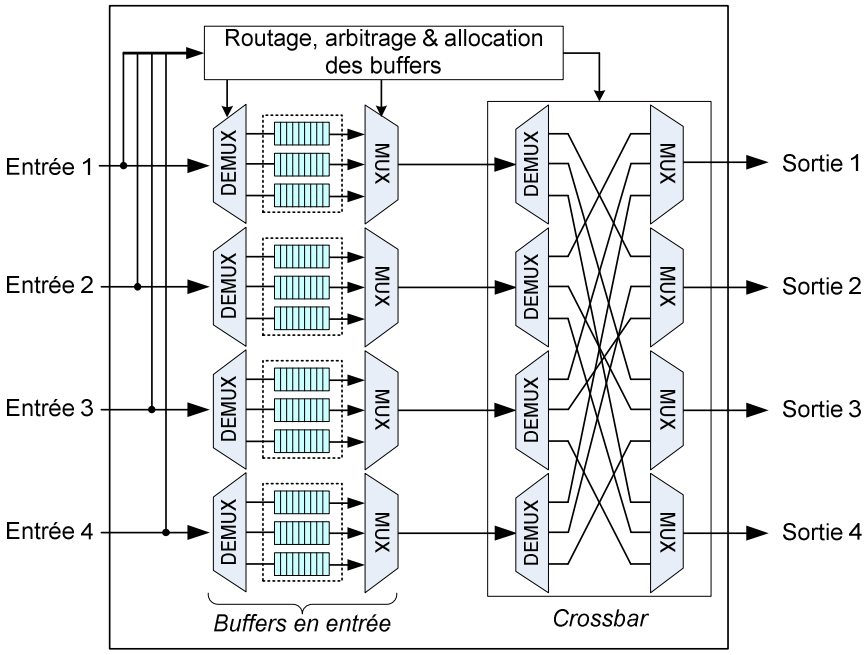


Figure 2.9 : Routeur employant plusieurs buffers en entrée.

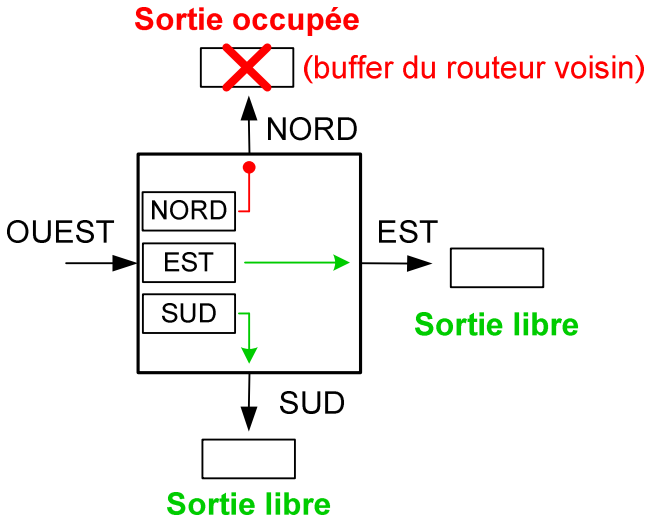


Figure 2.10 : Résolution de l'HoL grâce à l'utilisation de plusieurs buffers en entrée du routeur.

- Canaux virtuels

Les canaux virtuels offrent la possibilité de créer une notion de priorité entre certains types de paquets. Par exemple, on peut stocker les paquets signalés comme « urgent » dans un ou plusieurs canaux virtuels dédiés, et router ces paquets en premier.

Les canaux virtuels sont une méthode de mémorisation qui offre un très grand niveau de flexibilité puisque le choix de la manière d'allouer les paquets aux canaux virtuels est totalement libre. Comme nous l'avons vu, celle-ci peut être basée sur une notion de priorité, mais les canaux virtuels peuvent aussi permettre d'isoler plusieurs types de trafic, ou d'étendre l'espace de stockage global.

Le principal désavantage de ce procédé est d'être relativement lourd et d'imposer un surcoût en surface important, notamment sur circuit FPGA. Par rapport à la solution de positionnement de l'étage de mémorisation en entrée des routeurs, plusieurs modules supplémentaires sont nécessaires, notamment un étage d'aiguillage (DEMUX), un étage d'arbitrage (MUX), ainsi que la logique de contrôle qui leur est associée. Dans [SCH08] l'implémentation de quatre canaux virtuels multiplie la surface du NoC par un facteur compris entre 4 et 5.

Les travaux présentés dans [SCH08] comparent les performances de NoC avec et sans canal virtuel, ceci pour trois trafics issus d'applications ayant des comportements différents. Le **premier trafic** est communément employé pour la mesure de la résistance à la charge des réseaux sur puce, il s'agit du trafic à distribution aléatoire uniforme. Le **deuxième trafic** est issu d'une application cryptographique qui alterne entre une période de calculs d'une forte intensité, et une période de communications. Le **troisième trafic** provient d'une application de transmission sans-fil Wi-Fi qui sollicite le réseau sur puce de manière constante et plus intensive. Lors de l'évaluation des performances d'un réseau pour une des quatre applications, le trafic de celle-ci ne varie pas, cependant, le taux d'injection d'un trafic concomitant augmente. Ce deuxième trafic simule l'activité de processus exécutés en parallèle de l'application.

Pour la première application, la solution sans canal virtuel est plus performante en latence pour des très faibles niveaux de charge. La complexité de cette solution étant plus faible, elle est plus rapide tant qu'aucun phénomène de congestion ne se produit. Pour l'application cryptographique, les performances sont données en nombre de clés calculées par seconde. La solution sans canal virtuel est là-aussi plus performante pour des faibles niveaux de charge, et le demeure pour des niveaux de charge plus importants que pour le trafic à distribution aléatoire uniforme. Enfin, pour l'application de transmission en Wi-Fi, le débit du réseau employant les canaux virtuels est au minimum deux fois plus important que le réseau qui n'en emploie pas, et ceci pour n'importe quel niveau de charge. De plus, les performances avec les canaux virtuels ne sont quasiment pas dégradées par l'augmentation du taux d'injection.

Les résultats de cet article montrent que les performances des réseaux avec et sans canal virtuel varient fortement selon l'application qu'ils supportent. Afin d'obtenir de meilleurs débits pour des applications qui sollicitent beaucoup les communications, il semble préférable d'employer des canaux virtuels. Ceci-dit, il serait intéressant de compléter les résultats, notamment en mesurant les performances en latence pour l'application cryptographique, et l'application de transmission Wi-Fi.

- Placement des buffers en sortie

Deux principaux choix sont possibles quant au placement de l'étage de mémorisation des paquets en sortie du routeur. Comme le montre la **Figure 2.11**, **a)** soit un buffer unique est placé après l'étage d'arbitrage, **b)** soit un buffer est positionné à chaque entrée des arbitres.

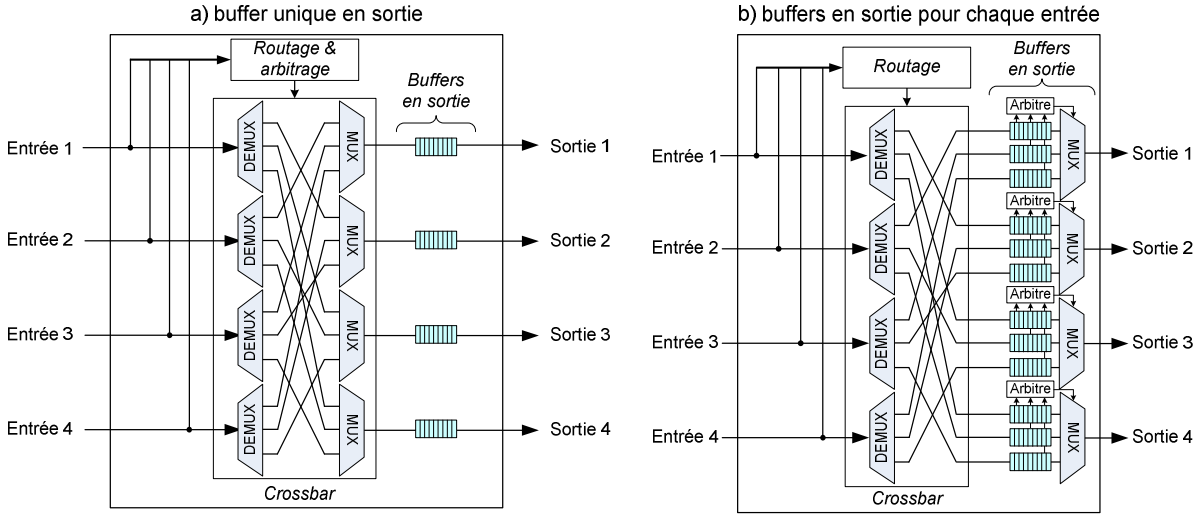


Figure 2.11 : Deux exemples d'étages de mémorisation placés en sortie du routeur.

Dans [RISO06], la résistance à la charge d'un réseau employant la mémorisation en entrée et un arbitre centralisé est comparée avec celle d'un réseau employant la mémorisation en sortie et un arbitrage décentralisé (comme l'architecture **a)** de la **Figure 2.11**). Les mesures montrent que pour un même espace mémoire, **la latence est inférieure de 30%** pour les **buffers placés en sortie** et **l'arbitrage décentralisé**. Et ce même réseau atteint **le point de saturation à 40% de charge au lieu de 30%** pour le réseau employant les buffers en entrée. Ces résultats s'expliquent notamment par l'élimination des conflits HoL. Le **surcoût de la décentralisation des arbitres** représente quant à lui **27%** de la surface totale du réseau.

Le placement des buffers en sortie selon l'architecture **b)** de la **Figure 2.11** est identique aux « *channel buffers* » présentés dans **[CHE15]**. Au niveau de chaque sortie, une mémoire tampon est dédiée à la réception des données provenant d'un port d'entrée. Puis, un arbitre par sortie gère l'accès des différentes mémoires tampons à celle-ci. De fait, cette structure est assez lourde puisqu'elle nécessite un nombre important de mémoires tampons et d'arbitres.

Du point de vue des performances, **[RISO06]** compare la résistance à la charge de l'architecture **a)** avec des tailles de buffers de **8 mots**, et l'architecture **b)** avec des tailles de buffers de **4 mots**. Le trafic employé est composé de paquets de **64 mots**, distribués de manière **aléatoire uniforme**. Leur **résistance respective est sensiblement identique**, à part le point de saturation qui est atteint légèrement plus tôt pour l'architecture **b)**. Par contre, la solution **b)** occupe **deux fois plus de surface** que la solution **a)**. Comme la taille des paquets injectés a un impact sur les résultats, il serait intéressant de mesurer la résistance des deux solutions pour des paquets de taille plus petite.

- Architecture sans mémoire tampon (*bufferless*)

Les solutions *bufferless* qui n'utilisent aucune mise en tampon ou limitée à un flit, emploient généralement l'algorithme *hot-potato* (« patate chaude ») pour empêcher les *deadlocks* et *livelocks* [KAP15]. Avec ce type d'algorithmes chaque paquet doit forcément quitter le module dans lequel il se trouve au prochain cycle d'horloge. Si sa destination est indisponible, le paquet est dévié de sa route initiale, jusqu'à ce qu'il puisse effectuer une nouvelle tentative d'accès. Pour qu'un algorithme *hot-potato* fonctionne, le nombre de sorties d'un routeur doit être au moins égal à son nombre d'entrées. A chaque cycle d'horloge, une sortie doit pouvoir être allouée à chaque paquet entrant.

Le réseau *bufferless* « Hoplite » présenté dans [KAP15] emploie une topologie torique à deux dimensions, et un algorithme de routage XY simplifié de type *hot-potato*. Les ports NORD et OUEST de chaque routeur servent uniquement comme entrée, et les ports SUD et EST, comme sortie. Un port bidirectionnel supplémentaire relie le routeur à une IP. Grâce à la réduction du nombre d'entrées et de sorties, la complexité du crossbar, des arbitres, et du module de routage est beaucoup plus faible que pour un routeur classique possédant 5 ports bidirectionnels. De plus, la mémorisation des paquets est réduite à 1 seul mot. Le **gain de surface** par rapport à un routeur classique est majeur, puisque qu'un routeur d'Hoplite ne représente que **25%** de celui-là.

Cependant, la principale faiblesse des réseaux *bufferless* est de faiblement résister à la charge. Pour un trafic composé de paquets d'un seul mot distribués de manière aléatoire et uniforme, **Hoplite atteint le point de saturation à partir de 5% de charge**. D'ailleurs, le point de saturation est sans doute atteint encore plus rapidement pour des paquets de plus larges tailles. **Les NoC *bufferless* semblent donc adaptés pour le transport d'un faible nombre de paquets de faibles tailles, tout en garantissant un coût en ressources minimal.** Concernant la fréquence de fonctionnement, elle est sans doute assez importante pour Hoplite, puisque la complexité des arbitres, du crossbar, et des modules de routage est minimale. De plus, une mémoire tampon resynchronise les données en entrée de chaque routeur. Un cas d'utilisation semblant idéal pour le réseau Hoplite serait d'être le réseau à commutation de paquets dans [MOL10], qui transporte les demandes de reconfiguration du réseau à commutation de circuits.

- Performances des mémorisations pour une architecture décentralisée

L'article [CHE15] traite d'une architecture de routeur décentralisée, où chaque port d'entrée possède son propre module de routage, et où un arbitre est positionné à chaque port de sortie. Des placements-routages et des mesures de résistance à la charge ont été effectués pour différentes configurations de mémorisation. **Sans mémorisation**, le réseau sature à 15% de charge. Avec des **buffers en entrée**, le réseau ne sature plus qu'à 30% de charge, sauf qu'il consomme un tiers de ressources en plus, et la fréquence est diminuée de 20%. Enfin, pour **les buffers positionnés en sortie** (à chaque entrée de l'arbitre), le point de saturation n'est atteint qu'à 35% de charge. Cependant, par rapport à la version sans mémorisation, la fréquence de fonctionnement est diminuée d'un quart, et la quantité de ressources consommées est multipliée par 3.

2.3.4. Utilisations des optimisations spécifiques aux circuits FPGA

Afin d'exploiter au mieux les caractéristiques spécifiques des circuits reconfigurables, les concepteurs peuvent faire appel à certaines ressources de manière intentionnelle. Premièrement, en plus des blocs de logiques programmables composées de LUT et de registres, les FPGA comptent un ensemble de blocs matériels dédiés à réaliser certaines fonctions de manière efficace. Deuxièmement, les outils de conception sur FPGA peuvent aussi être configurés pour maximiser certaines caractéristiques, comme la fréquence de fonctionnement ou la surface, ceci au détriment d'une ou plusieurs autres.

Tableau 2.2 : Optimisations employées dans notre l'état de l'art des NoC pour FPGA.

Article	Optimisation(s)
[MAR02]	• Synthèse des buffers en BRAM.
[BAR03]	• Synthèse des buffers en BRAM. • IP SelectRAM Xilinx utilisée pour la synthèse tables de routage.
[SET05]	• IP FIFO Logicore de Xilinx utilisée pour la synthèse des buffers.
[HIL06]	• Tables de routage synthétisées en BRAM.
[KAP06]	• Synthèse des buffers en SRL16 (configuration des LUTs en registre à décalage - Xilinx)
[EHL07]	• Synthèse des buffers en SRL16 (configuration des LUTs en registre à décalage - Xilinx)
[YEL11]	• Optimisation de la structure en pipeline du routeur pour la synthèse dans les ressources logiques du FPGA.
[PAP12]	• Maximisation de l'utilisation des ressources filaires : information de routage du paquet transporté par des fils dédiés. • Synthèse des buffers dans la RAM distribuée (LUT-RAM)
[HUA12]	• Synthèse des buffers en SRL16 (configuration des LUTs en registre à décalage - Xilinx)
[MAI15]	• Synthèse des canaux virtuels par paire dans une BRAM [KWA12].
[KAP15]	• Synthèse des buffers dans la RAM distribuée (LUT-RAM). • Optimisation de l'architecture du crossbar pour sa synthèse dans les LUT. • Utilisation de la contrainte « *RLOC* » dans l'outil de placement-routage afin de minimiser la surface de la conception.
[CHE15]	• Synthèse des buffers en SRL16 (configuration des LUTs en registre à décalage - Xilinx)

Comme on peut l'apercevoir dans le **Tableau 2.2**, la majorité des optimisations spécifiques aux FPGA pour les réseaux sur puce concernent la mémoire. Les registres intégrés dans les blocs de logiques programmables ne permettant pas de synthétiser de grands espaces de stockage, des

blocs RAM ont été ajoutés au sein des circuits reconfigurables. Ces blocs ont l'avantage d'être plus rapidement accessibles qu'une mémoire externe, et d'offrir un certain niveau de protection du point de vue matériel.

Les réseaux sur puce (non *bufferless*) nécessitent une certaine quantité de mémoire afin de mémoriser les paquets en cours de transmission. Un des premiers articles qui traitent de l'implémentation des NoC spécifiquement sur FPGA propose d'utiliser les blocs RAM pour la synthèse des mémoires tampons [BAR03]. Comme ce sont des blocs macroscopiques, en nombre de portes logiques ils représentent 97% du réseau sur puce. Cela provient essentiellement du fait que seule une infime partie de chaque bloc est utilisée.

Pour pallier ce problème et optimiser l'utilisation des BRAM lors de la synthèse des mémoires tampons, une solution proposée est de placer deux FIFO (d'entrée, de sortie ou de canaux virtuels) dans un seul BRAM [KWA12], [MAI15]. La contrainte, c'est que deux FIFO qui peuvent habituellement recevoir chacune une lecture et une écriture simultanées, doivent maintenant se les partager. La logique de régulation qui réalise ce partage favorise les accès en écriture aux accès en lectures, c'est-à-dire, la réception de données à la transmission. Afin de mesurer les performances de la proposition de partage des blocs RAM ou *BRS* (*Bloc RAM Sharing*), le réseau Booksim a servi de support [BOOKSIM]. Il s'agit d'un réseau sur puce qui n'a pas été spécialement conçu pour une utilisation sur FPGA, et qui a une architecture classique (routage XY, topologie maillée ou torique).

Pour un réseau 4x4 à 16 routeurs, la logique qui gère le partage des blocs RAM augmente le nombre de ressources consommées d'environ 5%, et diminue la fréquence de fonctionnement d'approximativement 4%. L'économie du nombre de blocs RAM est quant à lui de 40%. Concernant la latence des échanges, le surcoût est négligeable pour de faibles niveaux de charge, cependant, le taux de saturation est atteint plus rapidement pour le réseau qui emploie le partage des BRAM. La différence de résistance aux niveaux de charge élevés entre les deux réseaux n'est pas la même selon la topologie (torique ou maillée), et la distribution du trafic utilisée lors des mesures.

Certains réseaux proposent de se passer entièrement des blocs RAM, et d'utiliser à la place les registres contenus dans les blocs de logiques. Premièrement, les blocs RAM sont des ressources présentes en quantité relativement limitée dans les FPGA actuels, et il peut être plus intéressant pour certaines applications qu'ils soient entièrement dédiés au stockage du code des processeurs. Deuxièmement, les BRAM sont des éléments macroscopiques repartis de manière peu homogène sur la surface du circuit, donc leur utilisation requiert des fils de longueur assez importante, très susceptibles d'être des chemins critiques.

Dans [PAP12] les auteurs ont tenté d'identifier les points les plus importants sur lesquels se concentrer pour optimiser l'implémentation d'un NoC sur FPGA. Ils préconisent notamment d'utiliser les registres contenus dans les blocs logiques pour la synthèse des mémoires tampons, parce qu'il s'agit des éléments mémoires les mieux répartis sur la surface du circuit. Une alternative pour la synthèse des FIFO appelée SRL16, est seulement disponible dans les FPGA Xilinx. Il s'agit de configurer les blocs qui contiennent les LUT en registres à décalage [XIL08]. Cette solution est employée les travaux suivants [KAP06], [EHL07], [HUA12], et [CHE15].

2.3.5. Etages de traitement des routeurs

La structure des routeurs définit la manière dont les paquets sont traités, stockés, et transmis, et donc le nombre de cycles d'horloge que requiert le transport de chacun d'eux. Un nombre d'étages de traitement important dégrade les performances en latence, tandis qu'un nombre trop faible risque d'engendrer des chemins critiques, qui vont faire chuter la fréquence de fonctionnement du réseau sur puce (**Tableau 2.3**).

Tableau 2.3 : Décomposition du fonctionnement des routeurs de l'état de l'art.

Article	Structure de la chaîne de traitement des flits (latence initiale)
[BAR03]	• 5 cycles : - Stockage des flits du paquet en entrée dans la FIFO (*backpressure* et *wormhole*). - Calcul du routage XY dès la réception du flit d'entête. - Envoi d'une requête à l'arbitre. - Confirmation de la part de l'arbitre. - Transmission du flit d'entête ; puis des flits suivant à chaque cycle d'horloge.
[SET05]	• Cycles = 6 + 2 * nombre de flits dans le paquet : - Stockage de **tous** les flits du paquet en entrée dans la FIFO (*store & forward* et *handshake*). - Calcul du routage XY une fois le paquet complètement stocké. - Envoi d'une demande d'arbitrage à la FIFO de sortie. - Confirmation de disponibilité de la FIFO de sortie. - Transmission du paquet de la FIFO d'entrée à la FIFO (*store & forward).* - Transmission du paquet de la FIFO de sortie vers le routeur voisin en mode (*store & forward* et *handshake*).
[EHL07]	• 3 cycles : - Réception du paquet et calcul du routage pour le routeur suivant (*lookup*). - Arbitrage décentralisé (au niveau de chaque port de sortie). - Transmission du paquet au routeur suivant si le port de sortie est disponible.
[YEL11]	• 2 cycles : - Stockage du flit, calcul routage (*lookup*), arbitrage. - Traversée du crossbar, émission au routeur suivant.
[PAP12]	• 1 cycle : - Tous les traitements sont effectués durant le même cycle d'horloge.
[HUA12]	• 2 ou 4 cycles : - Stockage des flits du paquet en entrée dans la FIFO (*wormhole*). - *Cycle d'attente optionnel avant le routage décentralisé.* - Stockage des flits dans la FIFO de sortie (*wormhole*). - *Cycle d'attente optionnel avant l'arbitrage décentralisé et l'envoi du paquet au routeur voisin.*
[CHE15]	• 2 à 5 cycles : - Stockage optionnel des flits en FIFO d'entrée et routage. - Stockage optionnel des flits en FIFO de sortie, arbitrage et envoi des flits. (0 à 3 étages de pipeline pour le signal de contrôle de flux (chemin critique)).

La structure de routeur la plus communément utilisée est présentée dans [BAR03], pour laquelle le traitement de chaque flit d'entête est décomposé en 5 étages. Une fois que le routage et l'arbitrage ont été effectués pour le flit d'entête, chaque flit restant qui compose le paquet est transmis en un seul cycle d'horloge. La latence de 5 cycles d'horloge représente le cas idéal, où le paquet remporte l'arbitrage sans attendre qu'un autre paquet ne soit traité en premier.

[SET05] est le réseau sur puce avec la latence la plus importante que nous ayons rencontré durant la construction de notre état de l'art. La raison principale est qu'il emploie le mécanisme *store & forward* à la fois pour la transmission des paquets entre les routeurs, mais aussi, entre ses FIFO d'entrée et ses FIFO de sortie. Avec ce mécanisme, le paquet doit être entièrement stocké dans la mémoire tampon avant que sa transmission ne puisse se poursuivre. Donc plus la taille du paquet est grande, plus la latence de son transport est importante.

[EHL07] et [YEL11] propose une amélioration de la structure classique des routeurs à 5 étages. Celle-ci est appelée « *lookup routing* », et consiste dans le calcul du routage pour le routeur suivant. Ce calcul peut intervenir durant un autre traitement comme l'arbitrage, et permet donc de réduire la latence d'au moins un cycle.

Dans [PAP12] plusieurs propositions sont faites pour optimiser les performances des réseaux sur puce sur FPGA. Les auteurs prétendent notamment que le découpage du traitement des routeurs en plusieurs étages n'est pas bénéfique sur FPGA, de par la structure matérielle particulière de ce type circuit. C'est pour cette raison que tous les traitements dans les routeurs CONNECT sont effectués durant le même cycle d'horloge. Cependant, les travaux [HUA12] et [CHE15] prouvent que l'ajout d'étages de pipeline multiplie la fréquence de fonctionnement du réseau. Donc, cela a non seulement pour effet de compenser l'augmentation du nombre de cycles de latence, mais cela augmente aussi significativement le débit du NoC.

Pour l'article [HUA12], pour des tailles de liens 32 bits, l'implémentation de CONNECT atteint une fréquence d'environ 100 MHz, alors que le NoC proposé approche les 220 MHz pour seulement un cycle d'horloge supplémentaire. Puis, en ajoutant encore deux cycles d'horloge (ce qui porte le délai total de traversée du routeur à 4 cycles) la fréquence est environ de 300 MHz. Cette deuxième implémentation est plus intéressante du point de vue du débit, mais moins intéressante sur le plan de la latence, puisque l'augmentation de la fréquence ne suffit plus à compenser les cycles d'horloge supplémentaires nécessaires au traitement des flits.

Le réseau présenté [CHE15] propose une amélioration de l'architecture de [HUA12]. Le délai de traversée initial du routeur est seulement de 2 cycles : le premier est dédié au routage et le second à l'arbitrage. De plus, le signal de contrôle de flux peut être pipeliné afin d'améliorer la fréquence car il s'agit du chemin critique du réseau. Il est important de noter que le placement des FIFO en entrée et en sortie du routeur est optionnel. La configuration la plus lourde de ce réseau pour les placements-routages, c'est-à-dire, avec des buffers placés en entrée et en sortie, atteint une fréquence de 420 MHz lorsque le signal de contrôle de flux comprend 3 étages de pipeline. C'est la fréquence la plus élevée que nous ayons pu observer pour un réseau sur puce sur FPGA. Il s'agit également du réseau sur puce qui possède la meilleure résistance à la charge.

2.4. Conclusions

Finalement, nous ne pouvons que constater qu'à partir d'une synthèse de différents articles de la littérature scientifique qui traite des implémentations de réseaux sur puce sur FPGA, il peut être difficile d'identifier la stricte supériorité d'une solution par rapport une autre.

Par contre, nous notons que certains réseaux sont plus adaptés que d'autres pour soutenir les communications d'une application spécifique. Par exemple, avec le cas du NoC Hoplite [KAP15] qui a une structure allégée grâce à l'absence d'espaces de mémorisation, et qui garantit donc le transport de messages avec une faible latence tant que le niveau de charge demeure faible.

D'autre part, avec l'amélioration des technologies de FPGA, et la large variété des modèles existants, il peut être complexe de comparer les résultats de placements-routages effectués sur deux circuits différents. En effet, nous remarquons qu'entre les deux principaux vendeurs de FPGA Xilinx et Altera, la structure des blocs logiques est assez différente, et les LUT ne comportent pas le même nombre d'entrées. La famille récente de FPGA Virtex-7 de Xilinx emploie des LUT à 6 entrées, tandis que la famille Arria 10 d'Altera utilise des LUT à 8 entrées, décomposables en plusieurs LUT de plus petites tailles. Cela complexifie la comparaison du nombre de ressources utilisées, voire même, les fréquences de fonctionnement.

De plus, chaque vendeur propose plusieurs gammes de circuit FPGA avec des structures et des performances différentes. Pour répondre à cette préoccupation, les auteurs de [MAI15] proposent d'utiliser comme échelle de mise à niveau des résultats, la fréquence de fonctionnement des blocs de traitement du signal (DSP). Cependant, ils n'utilisent ce moyen que pour comparer les performances de deux familles appartenant au même vendeur.

Les auteurs des travaux [LEE09] et [LEE10] ont étudié les effets des paramètres architecturaux sur les résultats des placements-routages de NoC sur plusieurs familles de FPGA Xilinx. Ils ont observé pratiquement les mêmes tendances dans les résultats, c'est-à-dire que par exemple, l'écart relatif des performances en fréquence et en occupation de ressources est quasiment le même, lors du passage d'une topologie maillée à une topologie torique, pour deux familles différentes.

La résistance à la charge est une métrique très importante pour évaluer les performances d'un NoC, et elle a l'avantage de ne pas dépendre de la technologie et d'un modèle de circuit. Cependant, selon le type de trafic soumis au réseau, les performances peuvent largement varier, comme nous l'avons observé dans notre étude [SCH08].

La meilleure façon de comparer des NoC sur FPGA est donc de pouvoir effectuer des placements-routages sur le même circuit, et d'employer le trafic de l'application finale lors de l'évaluation de la résistance à la charge.

Malgré toutes les contraintes qui concernent la comparaison des performances de différents réseaux sur puce pour FPGA, on peut noter que les structures personnalisables telles que [HUA12] et [CHE15], qui emploient des modules de routage et d'arbitrage décentralisés soutiennent des hauts niveaux de charge, ainsi que des fréquences élevées, et ceci pour des niveaux de consommation de ressources modérés.

Chapitre 3) Portage du réseau Hermes sur FPGA

Afin d'avoir une idée précise des performances que l'on peut atteindre avec un réseau sur puce générique sur un circuit reconfigurable, nous avons décidé de réaliser un portage du NoC Hermes sur FPGA.

Avec ce portage, nous cherchons à satisfaire un certain nombre de contraintes applicatives qui proviennent d'objectifs industriels, comme le support d'IP qui communiquent dans un protocole propriétaire. Nous cherchons également à répondre à certaines exigences en matière de performances en latence, en fréquence et en taux d'occupation de ressources.

Sommaire du Chapitre 3

3.1. Introduction

Le premier objectif du portage d'Hermes [MOR04] sur FPGA, est de tenter de résoudre la difficulté qu'ont les systèmes d'interconnexions habituellement utilisés à faire communiquer un nombre important d'IP (cœur de propriété intellectuelle). Concernant le deuxième objectif de ce portage, il s'agit de tenter de répondre en partie, aux problématiques de sécurité auxquelles sont confrontés les systèmes numériques actuels, et que les interconnexions des logiciels de conception des principaux vendeurs de FPGA, ne résolvent pas. Cependant, l'exploration des mécanismes de sécurité est conditionnée par l'architecture du NoC sélectionné. En effet, il est vain de chercher à développer des solutions de sécurité spécifique pour un système d'interconnexions, qui ne saurait être utilisé dans un produit réel, à cause de ses performances insuffisantes.

Le choix d'Hermes pour cette implémentation sur puce programmable, est principalement motivé par sa polyvalence et ses caractéristiques génériques, communes à la majorité des réseaux sur puce qui sont l'objet d'articles scientifiques. Les résultats du portage d'Hermes donnent donc une idée de ce qu'il est possible d'attendre du portage de ce type de réseau sur puce sur FPGA, à savoir, un réseau du type commutation de paquet, à topologie maillée à deux dimensions, avec un routeur à 5 étages et utilisant un algorithme de routage XY. Le sélection d'Hermes pour ces travaux de thèse est aussi expliquée par le fait que son code source est en accès libre et permet donc de débuter les développements librement à partir d'une base qui a déjà été l'objet de nombreuses recherches.

Une alternative possible de réseau sur puce comme point de départ de nos travaux, est CONNECT, un réseau spécialement conçu pour tirer parti des spécificités de l'architecture des circuits FPGA [PAP12], alors qu'Hermes a été développé pour les circuits ASIC. Malheureusement la licence de CONNECT est limitée aux implémentations de type non-commerciales. Cette licence est suffisante pour effectuer des travaux de recherche, mais ne permet pas de remplir notre objectif qui est d'utiliser le portage du réseau dans une application cryptographique industrielle.

Nous allons insister sur l'étude des performances du portage d'Hermes sur FPGA, puisqu'elle nous permet d'en tirer des conclusions sur sa capacité, ainsi que celle des réseaux sur puce similaires, à être utilisé pour répondre à nos contraintes applicatives sur FPGA. Dans l'optique de réaliser une évaluation des performances du portage sur FPGA d'Hermes, se rapprochant au maximum de nos contraintes d'une implémentation dans un produit réel, il est nécessaire de développer les couches d'interfaces entre les routeurs et les modules communicants. Les cœurs de stockage ou de manipulation de données, aussi appelées IP, employés par le service Trustway, utilisent un protocole de communication spécifique. Celui-ci n'est pas du tout le même que celui du réseau Hermes. Ceci explique la nécessité de concevoir et de développer les couches d'interface entre les IP et le réseau Hermes, puisque ces couches ont un impact direct sur les performances de l'implémentation finale.

3.2. Caractéristiques générales d'Hermes

3.2.1. Topologies

Pour faire un rapide tour des caractéristiques du NoC Hermes [MOR04], son architecture est semblable à la majorité des réseaux sur puce qui sont l'objet d'articles scientifiques sur les réseaux sur puce. A savoir, une topologie maillée à deux dimensions, un algorithme de routage XY, arbitrage round-robin, et la commutation des paquets en *wormhole*.

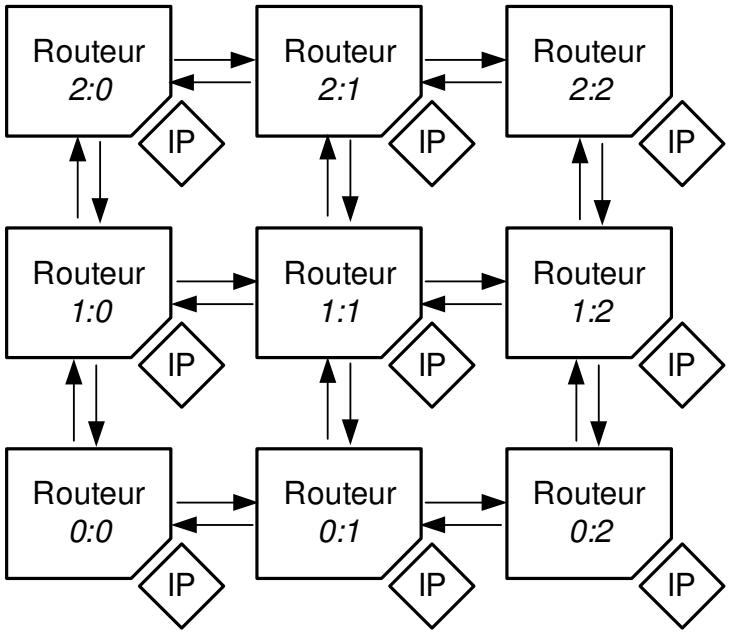


Figure 3.1 : Topologie d'Hermes 3 x 3 (9 routeurs).

La **Figure 3.1** représente une topologie maillée à deux dimensions, 3 sur 3, du réseau Hermes comptant 9 routeurs au total. Chaque routeur possède des liens bidirectionnels pour l'échange de paquets avec ses routeurs voisins. Un routeur compte entre 3 et 5 ports selon sa position dans le réseau, l'un d'eux est toujours dédié à la connexion d'une IP et les 2 à 4 ports restants sont employés pour la communication avec les autres routeurs. Chaque routeur possède sa propre adresse, un couple de coordonnées X et Y déterminées à partir de la position de ce dernier dans le réseau.

Il existe différentes options pour Hermes avec des choix d'algorithme de routage, d'arbitrage ou encore de contrôle d'erreur via un code à redondance cyclique (CRC). Ces variantes ont pour but d'améliorer la gestion du trafic des charges réseau importantes. Comme dans un premier temps notre objectif est d'aboutir aux meilleures performances en termes de latence, de fréquence de fonctionnement et d'occupation en ressources logiques, et que ces différentes options proposées ne permettent pas de les maximiser, nous avons fait le choix d'utiliser la version la plus simple d'Hermes, c'est-à-dire, sans option.

3.2.2. Contrôle de flux et synchronisation

Dans la version initiale d'Hermes deux modes de synchronisation des échanges sont utilisables : **handshake** ou **crédit**. Ces deux méthodes peuvent gérer des échanges entre des entités **synchrones** et **asynchrones**. L'avantage du mode handshake est de ne nécessiter que d'un seul signal dédié au contrôle de flux, cependant, par rapport au signal de crédit, il impose un cycle de latence supplémentaire lors de chaque échange.

Pour que la méthode avec le système de crédits puisse fonctionner pour deux entités asynchrones, l'horloge de l'entité émettrice doit être transmise à l'entité réceptrice afin de synchroniser la mise à jour du nombre de crédits. Les crédits représentent l'espace de réception de données disponible dans l'entité réceptrice. Tant que le signal de crédit provenant du récepteur est actif, l'émetteur considère que les données envoyées sont directement reçues, et il procède donc à l'envoi de nouvelles données dès le cycle d'horloge suivant. Tandis qu'avec le mode handshake, le récepteur doit faire varier la valeur du signal d'acquittement pour que l'émetteur considère ses données comme reçues.

Comme nous cherchons à obtenir les meilleures performances possibles en latence pour notre portage d'Hermes, nous avons tout intérêt à employer le contrôle de flux basé sur les crédits. En effet, dans notre cas, tous les routeurs sont synchronisés donc le signal de crédit garantit les meilleures performances en latence, sans augmenter le nombre de signaux nécessaires par rapport à la méthode handshake.

3.2.3. Nœuds maîtres, esclaves et requêtes associées

Afin d'aborder le fonctionnement des communications à l'intérieur du système sur puce, il est nécessaire de préciser les notions d'interfaces maîtres et esclaves tels que nous les concevons dans notre portage. La nature d'une interface définit la manière dont l'IP interagit avec les autres à travers le réseau.

Grâce à une interface maître, une IP peut soumettre des requêtes de lecture ou d'écriture vers l'interface esclave d'une autre IP. Pour notre portage, l'interface maître peut recevoir des paquets de réponse à ses requêtes de lecture, qui sont aussi appelés « paquets d'acquittement ».

Réciproquement, l'interface esclave réceptionne les paquets de requête provenant des interfaces maîtres, et renvoi des paquets de réponse aux requêtes de lecture.

Il important d'indiquer qu'une IP peut posséder plusieurs interfaces maîtres ou esclaves et chacune d'entre-elle est considérée comme un nœud distinct du point de vue du réseau.

Le **Tableau 3.1** donne des exemples d'IP et du type d'interface qui leur sont associé.

Tableau 3.1 : Exemples d'IP et type(s) d'interface(s) associé(s).

IP	Type(s) d'interface	Détails de l'IP, rôles des interfaces
Processeur	Maître	Procède à des calculs et effectue des lectures et des écritures vers des mémoires et des registres.
Mémoire/registre	Esclave	Accessible à travers une plage d'adresse qu'un maître utilise pour accéder à son contenu à travers des requêtes de lecture ou d'écriture.
DMA (Direct Media Access)	Maître	Configuré à travers une interface esclave et utilise une interface maître pour déplacer le contenu d'une mémoire vers une autre. Ce type d'IP est très utile pour décharger un processeur de tâches qui le monopoliserait pendant de nombreux cycles d'horloge.
Accélérateur crypto. (AES, RSA, SHA...)	Maître ou esclave	*(Se référer au paragraphe en dessous du tableau).*
Sortie vidéo (VGA, DVI, HDMI...)	Esclave	Unidirectionnel, reçoit les trames à travers l'interface esclave pour les afficher sur un périphérique vidéo.
Interface de communication (USB, PCI, UART,...)	Esclave	Reçoit des données à travers son interface esclave pour les transmettre vers l'extérieur du circuit. Lorsque des données arrivent de l'extérieur du circuit, une interruption est déclenchée afin d'avertir le processeur ou le module qui pilote l'interface de communication.

- Accélérateurs cryptographiques (AES, RSA, SHA...)

Deux versions des accélérateurs cryptographiques (AES, RSA, SHA...) sont généralement possibles. La première peut posséder une unique interface esclave grâce à laquelle lui sont transmises les données à chiffrer, et les commandes qui déclenchent les opérations cryptographiques. Une fois les calculs terminés (signalés par l'activation d'une interruption) les données chiffrées sont récupérables à travers cette même interface esclave.

La deuxième version emploie une interface esclave, et une interface maître. L'interface esclave offre la possibilité à un processeur d'accéder aux registres de configuration de l'accélérateur afin de lui indiquer à quelles adresses se situent les données à chiffrer, et les adresses où doit être écrit le résultat, une fois les opérations terminées. Lorsque la configuration de l'accélérateur cryptographique est prête, le processeur initialise ensuite un registre de commande afin de lancer les calculs. L'accélérateur utilise ensuite son interface maître pour écrire le résultat à l'endroit précédemment indiqué par le processeur.

3.2.4. Méthodes d'adressage

Dans notre portage d'Hermes deux niveaux d'adressage sont employés. Premièrement l'adresse réseau sur puce, autrement dit, l'adresse du routeur auquel est connecté l'interface maître ou esclave de l'IP. Et deuxièmement, l'adresse mémoire des registres internes à l'IP esclave

La façon dont fonctionne l'adressage NoC correspond à ce qui est présenté sur la **Figure 3.1**. Chaque routeur possède sa propre adresse, basée sur la paire de coordonnées horizontale (X) et verticale (Y) relative à son positionnement dans la topologie.

Un décodeur d'adresses est positionné à l'interface entre l'IP maître et le NoC, afin d'établir la correspondance entre les adresse mémoires des IP esclaves connues par l'IP maître et leur équivalent en adresse sur le réseau sur puce (**Tableau 3.2**). Il est intéressant de noter que chaque maître peut utiliser un plan d'adressage d'IP esclaves différent puisque les décodeurs d'adresses ne sont pas communs à toutes les IP maitres.

Tableau 3.2 : Exemple de contenu d'un décodeur d'adresses d'une interface IP maître.

Adresse mémoire de l'esclave (32 bits représentés en hexadécimal)		Adresse réseau sur puce
Basse	**Haute**	
00000000	00FFFFFF	X = 0 ; Y = 0
01000000	01FFFFFF	X = 1 ; Y = 0
02000000	02FFFFFF	X = 2 ; Y = 0
03000000	03FFFFFF	X = 0 ; Y = 1
04000000	04FFFFFF	X = 1 ; Y = 1
05000000	05FFFFFF	X = 2 ; Y = 1

L'adresse réseau sur puce issue du décodeur rentre dans la composition de l'entête du paquet de requête.

Le décodage d'adresse n'est pas nécessaire au niveau des interfaces IP esclaves car pour composer l'adresse de destination d'un paquet de réponse à une lecture, elles utilisent directement l'adresse source du paquet de requête provenant du maître.

3.2.5. Format des paquets

Le réseau sur puce Hermes emploie la commutation de paquets. Cette partie a pour but de détailler la composition des paquets selon la requête à partir de laquelle ils sont formés. Les IP maîtres soumettent des paquets de requête de lecture (**Figure 3.2**) et requête d'écriture (**Figure 3.3**), et les IP esclaves émettent des paquets de réponse aux requêtes de lecture (**Figure 3.4**).

Chaque paquet est composé de sous-unités appelées flits (flow control digits). Un flit est la plus petite unité d'information pouvant être échangée entre deux entités (interface, routeur, etc....) lors d'un front d'horloge.

Figure 3.2 : Composition des paquets de requête de lecture.

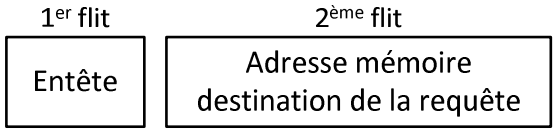


Figure 3.3 : Composition des paquets de requête d'écriture.

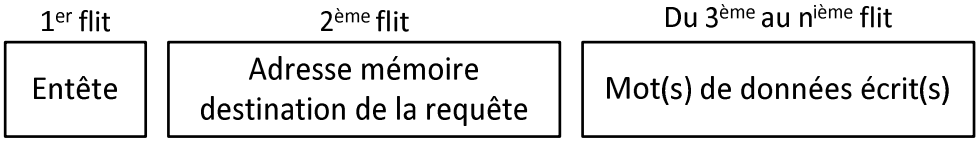
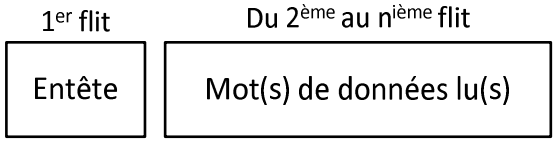
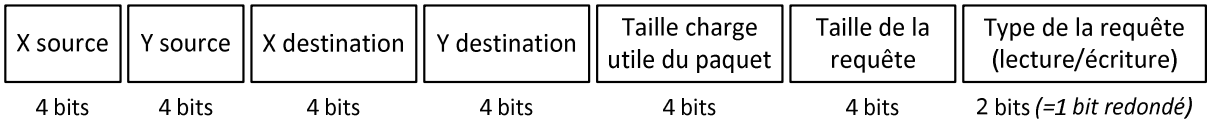


Figure 3.4 : Composition des paquets de réponse aux requêtes de lecture.



La **Figure 3.5** décrit la composition du flit d'en-tête qui contient toutes les informations nécessaires aux routeurs pour diriger le paquet à travers le réseau, à savoir l'adresse de destination et taille de la charge utile du paquet. Il contient aussi des informations sur la requête en elle-même, à savoir le nombre d'octets sur lequel celle-ci porte, ainsi que son type (écriture ou de lecture).

Figure 3.5 : Composition du flit d'en-tête des paquets (version à 32 bits).



Aucun paquet d'acquittement aux requêtes d'écriture n'est utilisé, cependant cela pourrait être le cas pour s'assurer que c'est bien l'esclave qui acquitte la requête de lecture et non pas directement l'interface maître, comme c'est le cas pour la version actuelle de notre portage d'Hermes. La contrepartie de l'ajout de ce paquet étant l'augmentation de la charge de trafic sur le réseau.

3.3. Architecture détaillée du routeur Hermes

3.3.1. Schéma modulaire

Un routeur Hermes standard est composé de 3 types de modules différents : des mémoires FIFO, un module d'arbitrage et de routage et un crossbar, son architecture est schématisée sur la **Figure 3.6** ci-dessous.

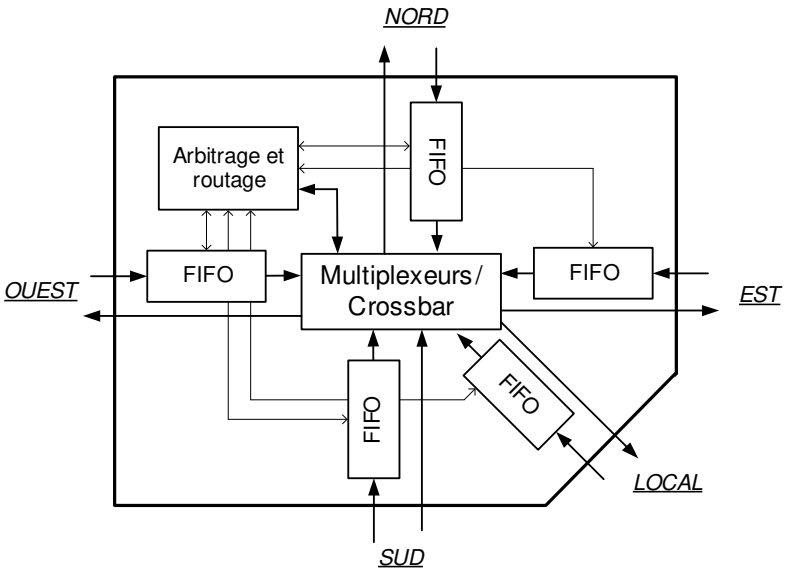


Figure 3.6 : Architecture modulaire d'un routeur d'Hermes.

3.3.2. Mémoire tampon FIFO

Un routeur possède de 3 à 5 mémoires tampons FIFO selon sa position dans la topologie du réseau. Le rôle des FIFO des ports NORD, EST, OUEST et SUD est de stocker les paquets en provenance des routeurs adjacents. La mémoire tampon du port LOCAL est dédiée à la réception des paquets provenant de l'IP. Lorsqu'une FIFO a un nouveau paquet à transmettre, elle informe le module d'arbitrage et de routage, et lui transmet toutes les informations nécessaires à l'exécution de l'algorithme qui détermine la direction que les flits du paquet vont devoir emprunter. Une fois que la FIFO a achevé la transmission du paquet à travers le crossbar, le module d'arbitrage et de routage en est informé afin qu'il mette à jour la liste des ports qu'il considère comme occupés et libres.

La mémoire FIFO est paramétrable en termes de nombre de flits (sous-unités composantes d'un paquet) qu'elle est capable de stocker. Cette taille de mémoire a un impact direct sur les performances générales du réseau. En effet, son augmentation améliore la capacité du NoC à résister à une surcharge du trafic. Elle optimise la capacité d'un réseau à transporter un certain nombre de messages en simultané avant que le nombre de messages ne soit trop important, est que le réseau atteigne le point de saturation où la latence moyenne des échanges augmente exponentiellement.

La contrepartie de l'augmentation de la taille des FIFO dans le but d'améliorer la résistance à la charge est la forte consommation de ressources que cela engendre. En effet, les mémoires font partie des éléments logiques qui occupent le plus d'espace et notamment sur circuit FPGA.

Lors de la synthèse d'un routeur de NoC de type Hermes, les mémoires vont représenter la majorité de l'espace occupé par le circuit. Cette surface a un impact direct sur la consommation en énergie ainsi que sur la fréquence de fonctionnement maximale. En prenant aussi en compte le fait que chaque routeur Hermes possède entre 3 et 5 mémoires FIFO, ce paramètre doit être dimensionné raisonnablement pour réaliser le meilleur compromis possible entre la résistance à la congestion de trafic et les performances des placements-routages.

3.3.3. Module d'arbitrage et de routage

Ce module a plusieurs rôles : il gère la priorité de transmission des paquets contenus dans les FIFO des ports d'entrée selon la disponibilité des ports de sortie, et en suivant la logique de l'algorithme *round-robin*. Puis, il exécute l'algorithme de routage XY et configure le crossbar selon le résultat.

Le fait que le calcul du routage soit centralisé et partagé entre toutes les FIFO, plutôt que divisé et distribué au niveau de chaque port d'entrée ou de sortie du routeur, permet d'économiser des ressources logiques. Cependant, la latence est plus importante pour l'architecture centralisée, puisque le routage ne peut être calculé que pour un seul paquet à la fois.

Une manière d'améliorer significativement les performances en latence d'Hermes serait d'implémenter le routage en mode « *look ahead* », de telle sorte, que le routage soit calculé au niveau de chaque port de sortie pour le routeur suivant. D'une part, cette architecture serait distribuée et donc permettrait d'éviter les files d'attente au niveau du calcul du routage. D'autre part, le calcul en mode *look ahead* serait effectué en parallèle d'une autre opération, ce qui réduirait d'au moins un cycle d'horloge la latence de traversée de chaque routeur.

3.3.4. Crossbar

Ce module établit les connexions physiques entre les ports d'entrée et de sortie du routeur à partir des informations qui lui sont envoyées par le module d'arbitrage et de routage. L'avantage d'employer un crossbar est de pouvoir établir plusieurs connexions simultanées entre différents ports d'entrée et de sortie.

3.3.5. Interfaces avec les IP maîtres et les IP esclaves

Durant notre portage d'Hermes nous avons tenté de nous approcher au maximum d'un cas d'utilisation réel et donc de répondre à certaines contraintes applicatives et matérielles. Une de ces contraintes est de pouvoir interfacer le réseau sur puce avec des IP maîtres et des IP esclaves qui communiquent dans un protocole propriétaire spécifique.

Au lieu de rajouter uniquement une couche d'interface entre le NoC et les IP, nous avons modifié l'architecture du routeur en remplaçant la mémoire tampon du port local par la logique de traduction du protocole des IP en paquets.

Le module de « paquétisation » de l'interface maître (**Figure 3.7**) construit le flit d'entête du paquet de requête en fonction de la longueur et du type de la requête émise par l'IP maître. Chaque flit suivant issu de la paquétisation est stocké dans une mémoire tampon en attente de sa transmission au réseau avec le mécanisme *wormhole*.

Pour éviter qu'un paquet issu de l'interface maître monopolise un des ports de sortie du routeur pendant une trop longue période, la taille des paquets est limitée à une valeur maximale. Si une requête est d'une longueur supérieure à cette valeur, celle-ci sera décomposée en plusieurs paquets.

L'opération de dépaquétisation est la réciproque de celle de la paquétisation, puisqu'il s'agit de traduire les paquets de réponse aux requêtes de lecture reçus en signaux d'acquittement, qui respectent le protocole propriétaire de l'IP.

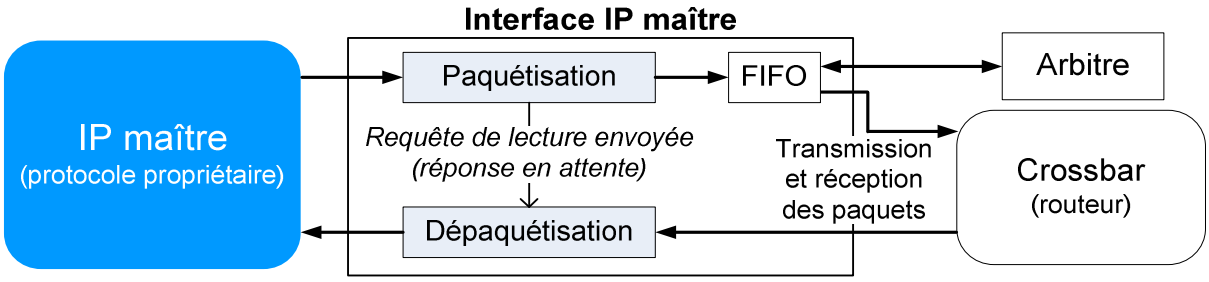


Figure 3.7 : Schéma structurel de l'interface IP maître.

L'interface esclave (**Figure 3.8**) a des rôles opposés à ceux de l'interface maître, puisqu'elle décompose les paquets de requêtes en signaux qui suivent le protocole propriétaire de l'IP, et elle compose les paquets de réponse aux requêtes de lecture à partir des signaux issus de l'IP.

Comme pour l'interface maître, les paquets qui sont émis ne peuvent pas dépasser une certaine taille. Lorsqu'une requête de lecture est de taille trop importante, sa réponse qui provient de l'IP esclave est traduite en plusieurs paquets.

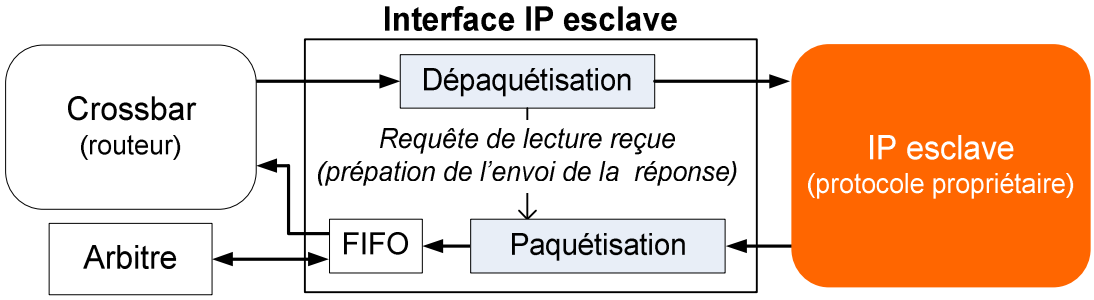


Figure 3.8 : Schéma structurel de l'interface IP esclave.

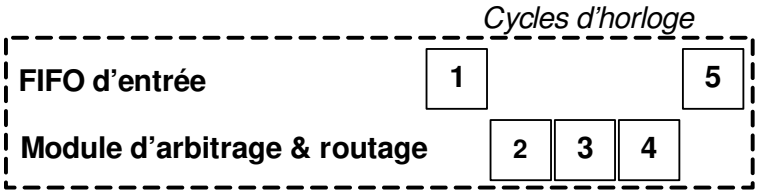
3.4. Performances en latence

Les **Figures 3.9** à **3.16** sont tracées pour un réseau sans congestion. Ainsi, aucun cycle de latence n'est causé par l'attente du traitement d'un autre paquet n'est représenté.

3.4.1. Latence par module

- Traversée d'un routeur

La latence minimale du transport d'un paquet à travers un routeur est de **5 cycles d'horloge** (**Figure 3.9**). Grâce à l'utilisation de la version synchrone d'Hermes et le contrôle de flux en **crédits**, chaque autre flit qui constitue le paquet ne prendra **qu'un seul cycle d'horloge** pour effectuer le trajet entre le port d'entrée et le port de sortie du routeur.



Détails des cycles :

1 Stockage du flit et envoi d'une demande d'arbitrage et de routage .

2 Réception de la demande d'arbitrage et de routage .

3 Calcul de l'arbitrage .

4 Calcul du routage XY et envoi d'un signal de confirmation à la FIFO .

5 Transmission des flits stockés .

Figure 3.9 : Traversée d'un routeur par un paquet.

- Traitement par l'interface d'une requête envoyée par l'IP maître

Tous les traitements effectués lors de la paquétisation d'une requête émise par l'IP maître sont effectués durant le même cycle d'horloge (**Figure 3.10**). En ajoutant le cycle d'horloge nécessaire au stockage du flit dans la FIFO, le délai de traversée de l'interface IP maître par la requête est donc de **2 cycles d'horloge**.

1 Lecture de la table de décodage d'adresse , composition du flit d'entête, et stockage du flit dans la FIFO de sortie .

2 à N+1 Début de transmission du paquet de N flit (s).

Figure 3.10 : Paquétisation et envoi d'une requête par l'interface IP maître.

- Traitement par l'interface IP maître d'un paquet envoyé par l'interface IP esclave

L'interface maître est amenée à recevoir des paquets (**Figure 3.11**). Il s'agit des réponses à ses requêtes de lecture. Le traitement de ces paquets est simple puisqu'il consiste seulement à transmettre tous les flits à l'exception du flit d'entête à l'IP maître. Par conséquent, la latence minimale de réception de l'interface IP maître est **d'un seul cycle d'horloge** (il faut compter un cycle en plus par mot de données lues).

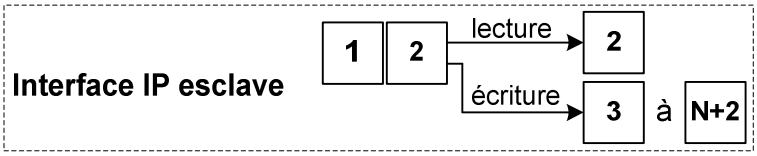
1 Filtrage de l'entête du paquet (non transmis à l'IP maître).

2 à N+1 Transmission des N flits restants qui composent le paquet en cours de réception.

Figure 3.11 : Dépaquétisation d'un paquet de réponse reçu par l'interface IP maître.

- Traitement par l'interface IP esclave d'un paquet de requête envoyé par l'interface IP maître

Les latences minimales pour que l'interface IP esclave reçoive et décode **les paquets de requêtes de lecture** ou **d'écriture** sont respectivement de **2** et **3 cycles d'horloge** (**Figure 3.12**).



1 Décodage des informations de la requête (taille, type ?) contenues dans l'entête du paquet reçu.

2 Réception de l'adresse destination de la requête (2ème flit).

lecture

2 Soumission de la requête de lecture.

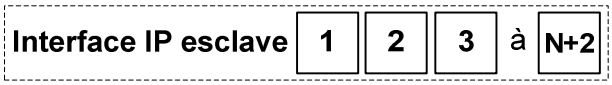
écriture

3 à N+2 Réception des N mot(s) de données à écrire et soumission de la requête.

Figure 3.12 : Dépaquétisation d'un paquet de requête reçu par l'interface IP esclave.

- Traitement par l'interface IP esclave des mots de données lues envoyés par l'IP esclave

La composition du flit d'entête du paquet de réponse est effectuée dès la réception du paquet de requête de lecture par l'interface esclave. Puis, les mots de données lues renvoyés par l'IP esclave doivent être tous être stockés dans la FIFO avant que la transmission du paquet de réponse ne débute (**Figure 3.13**). La latence de l'envoi de la réponse dépend donc à la fois de la taille de la requête de lecture, et du temps de réponse de l'esclave. Ceci dit, la latence minimale de transmission d'un paquet de réponse est de **3 cycles d'horloge**.



1 Stockage des données de lecture en provenance de l'IP esclave dans la FIFO.

2 Activation de la machine d'états de transmission des flits stockés dans la FIFO.

3 à N+2 Transmission des N mots de données lues (dépend entièrement du temps de réponse de l'IP esclave).

Figure 3.13 : Envoi d'un paquet de réponse par l'interface IP esclave.

3.4.2. Délai minimal d'un échange

La **Figure 3.14** décrit le délai de transmission d'une requête d'écriture d'une IP maître à une IP esclave. La latence minimale de transmission d'une requête d'écriture est de **15 cycles**.

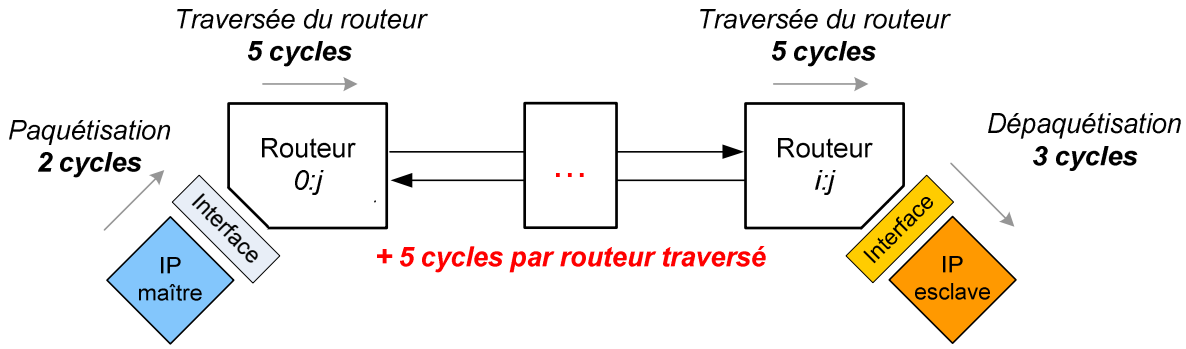


Figure 3.14 : Latence de transmission d'une requête d'écriture.

La **Figure 3.15** représente l'envoi d'une requête de lecture depuis une IP maître vers une IP esclave. La latence minimale pour l'envoi d'une requête d'écriture est de **14 cycles**.

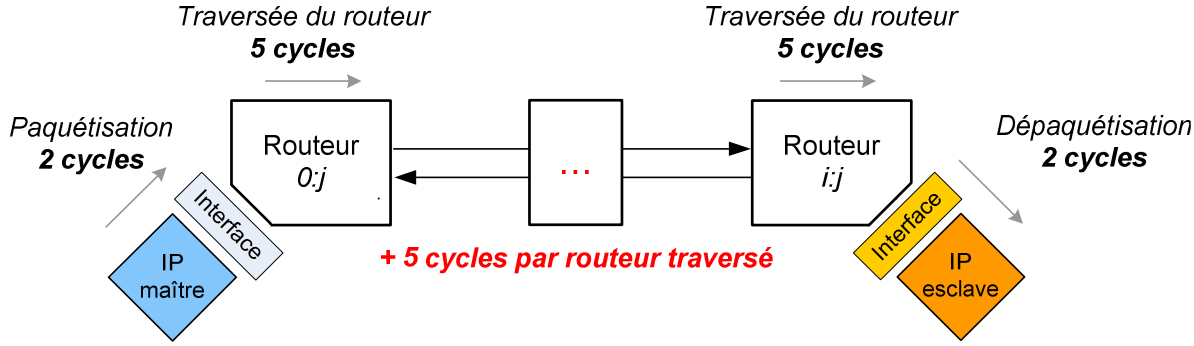


Figure 3.15 : Latence de transmission d'une requête de lecture.

La description des latences de transmission d'un paquet de réponse apparaît sur la **Figure 3.16**. La latence minimale du transport d'un paquet de réponse d'une IP esclave à une IP maître est de **9 cycles**.

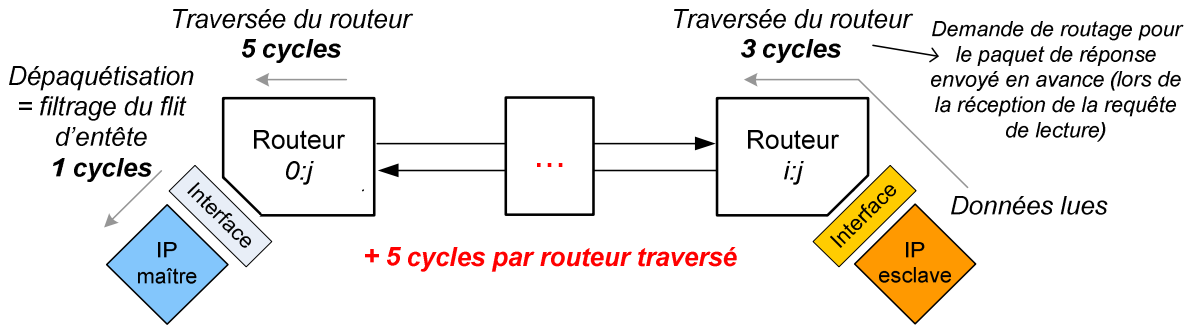


Figure 3.16 : Latence de transmission d'une réponse à une requête de lecture.

Le calcul de la latence minimale pour une requête de lecture complète, comprenant donc son émission et le retour de la réponse contenant les données lues à l'IP maître, donne un résultat de **23 cycles** d'horloge. De plus, il faut prendre en compte le délai supplémentaire de **10 cycles** pour la traversée de chaque routeur séparant les deux IP communicantes : **5 cycles** à l'aller, et **5 cycles** lors du retour.

3.4.3. Analyses des latences du réseau

La conception des interfaces des IP maîtres et des IP esclaves a été réalisée dans l'optique de favoriser les meilleures performances possibles en latence et en fréquence de fonctionnement, au détriment de la résistance du réseau à la charge. Comme les mesures de latences prennent en compte le délai des interfaces, cela signifie que les résultats obtenus sont conditionnés par nos contraintes applicatives et peuvent être différents pour d'autres types d'applications.

Pour rappel, la latence de transmission d'une requête d'écriture est de 15 cycles d'horloge, plus 5 cycles par routeur supplémentaire traversé. En ce qui concerne les requêtes de lecture, si l'on considère le délai d'envoi de la requête et du retour de la réponse, la latence minimale qui en résulte est de 23 cycles d'horloge, plus 10 cycles par routeur supplémentaire traversé.

La latence de transmission des paquets de réponse est d'autant plus importante, que tous les mots de données ciblés par la requête de lecture doivent avoir été renvoyés par l'IP esclave et stockés dans la FIFO de paquétisation de la réponse, avant que le paquet ne soit envoyé sur le réseau. Ceci, car le réseau Hermes ne peut pas mettre en pause la transmission d'un paquet en cours d'émission, et que la latence de réponse des esclaves aux requêtes de lecture n'est pas déterministe. Donc, la transmission d'un paquet de réponse à une requête de lecture ne peut pas démarrer avant que tous les mots de données lus n'aient été émis par l'IP esclave.

Nous constatons que ces latences sont trop importantes pour satisfaire les contraintes applicatives que nous avons fixées. Nous aimerions pouvoir employer un réseau puce sur FPGA se rapprochant au maximum des performances en latence des systèmes d'interconnexions de type crossbar ou bus, qui proposent généralement des latences minimales de transmission comprises entre 1 et 3 cycles d'horloge, pour les requêtes d'écriture ou de lecture.

Une des contraintes qui impacte le plus négativement les performances du réseau Hermes, est que le nombre de routeurs doit être au moins égal au nombre d'IP connectées au réseau. Ceci, ajouté au fait que la traversée d'un routeur par un paquet engendre un délai de 5 cycles, ça cause une augmentation drastique du délai moyen d'échange des messages.

Après avoir mis en évidence les faiblesses du réseau Hermes, il faut reconnaître certaines de ses qualités, notamment **sa résistance à la charge** que n'avons pas encore exploré car nous n'avons pas, pour l'instant, développé la plateforme de simulations adéquate. Toutefois, nous présumons cette résistance assez bonne, puisque la quantité d'espace de stockage des flits dans les routeurs est assez importante.

D'autre part, mis à part la latence minimale (ou initiale) de transmission de l'entête des paquets qui est assez importante, et qui est rédhibitoire pour la satisfaction de nos contraintes applicatives, **chaque autre mot qui compose les paquets ne met qu'un cycle d'horloge pour traverser un routeur ou une interface**. Cette caractéristique est très adaptée au transport de messages de très grande taille, et donc peut parfaitement convenir pour certaines applications.

3.5. Résultats des placements-routages

Après avoir abordé en détail les performances d'Hermes du point de vue de la latence, nous allons étudier les performances du portage en fréquence de fonctionnement et en occupation des ressources logiques. **Les objectifs que nous avons fixées pour les placements-routages sont d'atteindre un taux d'occupation des ressources logiques maximum de 30% du circuit FPGA, et une fréquence de fonctionnement d'au moins 150 MHz, ceci pour l'interconnexion d'une centaine d'IP grâce au réseau.**

3.5.1. Conditions expérimentales

Lors des placements-routages, des IP maîtres et esclaves de faible complexité ont été connectées aux interfaces, ceci afin de s'approcher des conditions d'une intégration réelle. Il est capital que les IP soient de faible complexité, afin que les chemins critiques se situent bien dans le réseau, et que la fréquence de fonctionnement représente les performances du NoC et non celles des IP. Pour les mesures du nombre de ressources logiques utilisées, seules les ressources du réseau et de ses interfaces sont prises en compte, excluant donc les ressources consommées par les IP.

Les développements et les mesures ont été réalisés avec la version 15.1.0 du logiciel de conception Quartus et pour le FPGA ALTERA ARRIA 10 SoC - 10AS066H4F34I3SGES.

Les quantités de ressources logiques consommées sont données en ALM (*Adaptive Logic Module*). Il s'agit des blocs de logiques programmables propriétaires d'Altera. Sachant que la taille des LUTs et la manière dont la synthèse est effectuée sont différentes pour chaque vendeur et le modèle du FPGA, il n'est pas aisé d'établir des équivalences ou de donner des résultats représentatifs pour tous les circuits. Nous revenons en détail sur cette problématique dans le deuxième chapitre de ce manuscrit.

3.5.2. Effets des paramètres architecturaux

La variation des paramètres architecturaux du réseau sur puce peut avoir un effet très important sur les performances. Lors des placements-routages nous avons cherché à étudier les effets du **nombre de routeurs**, de la **taille des mémoires tampons**, et de la **synthèse des mémoires RAM dans les blocs macro dédiés ou dans les LUT**.

3.5.3. Graphiques des résultats

Le **Tableau 3.3** présente la nomenclature utilisée dans les **Figures 3.17**, **3.18** et **3.19** des résultats des placement-routages.

Tableau 3.3 : Légende employée dans les graphiques des résultats des placements-routages.

Légende	Signification
NoC32	Topologie maillée de 8 colonnes de 4 routeurs (32 IP).
NoC64	Topologie maillée de 8 colonnes de 8 routeurs (64IP).
NoC100	Topologie maillée de 10 colonnes de 10 routeurs (32 IP).
Buffer4	Taille des mémoires tampons des routeurs de 4 flits de 32 bits.
Buffer8	Taille des mémoires tampons des routeurs de 8 flits de 32 bits.
BRAM	Synthèse automatique des mémoires en BRAM activée.
No BRAM	Synthèse automatique des mémoires en BRAM désactivée (synthèse en LUT).

Figure 3.17 : Utilisation des ressources logiques avec synthèse des mémoires en BRAM.

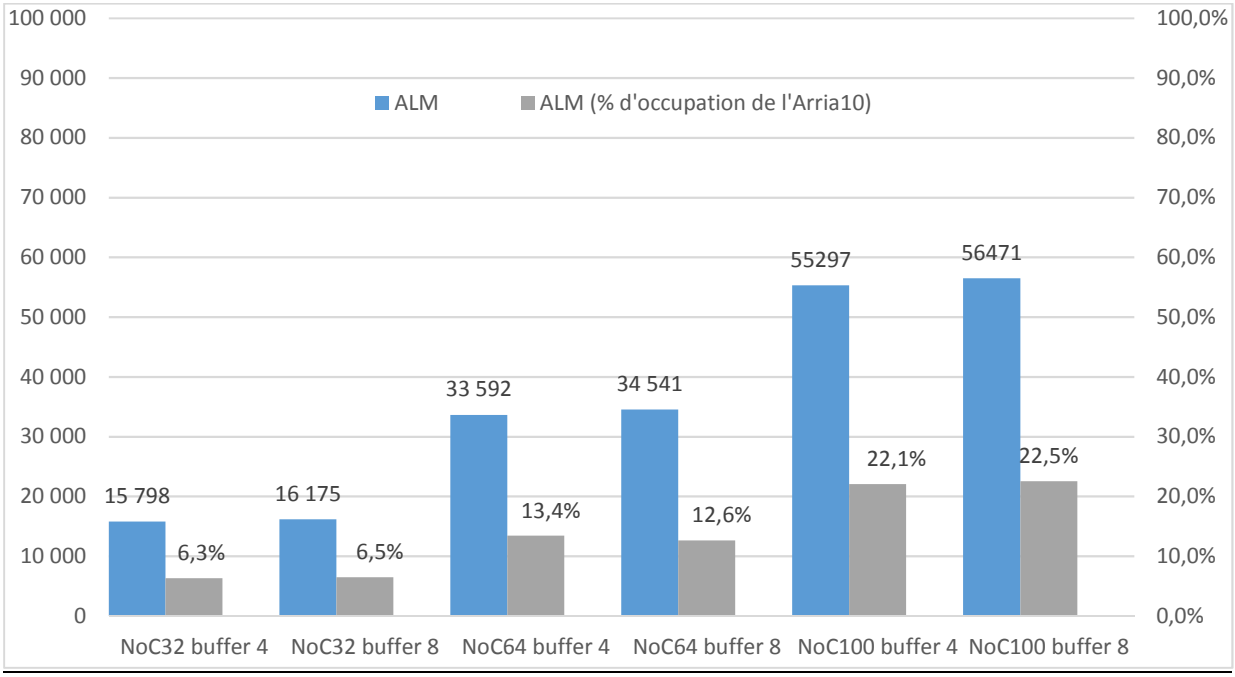


Figure 3.18 : Utilisation des ressources logiques sans synthèse des mémoires en BRAM.

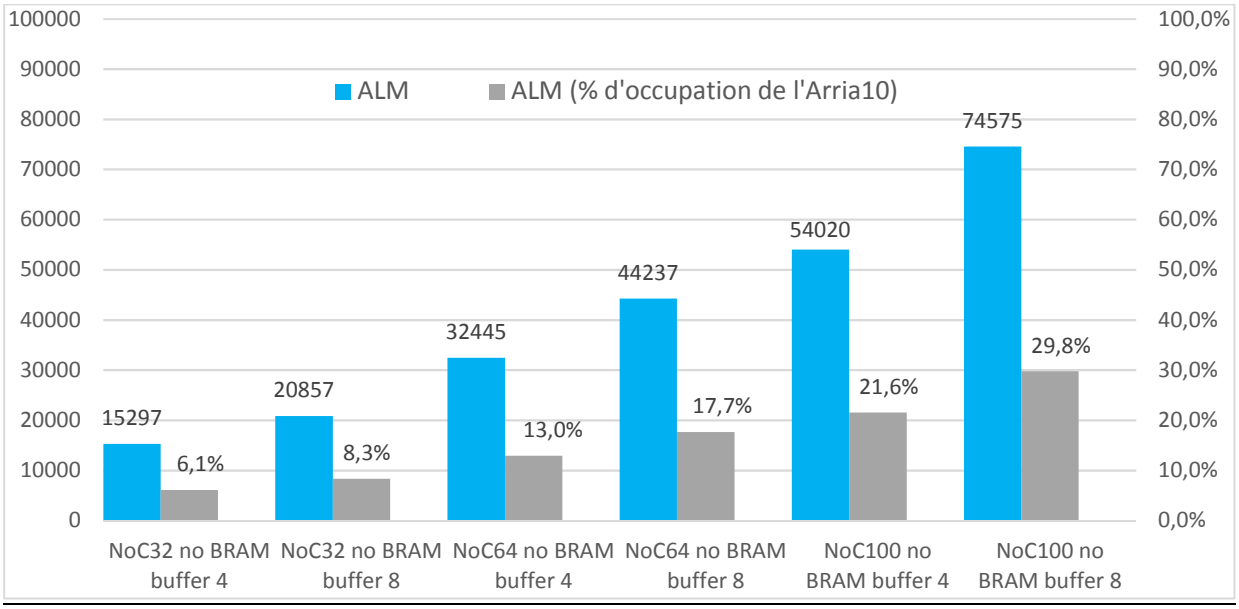
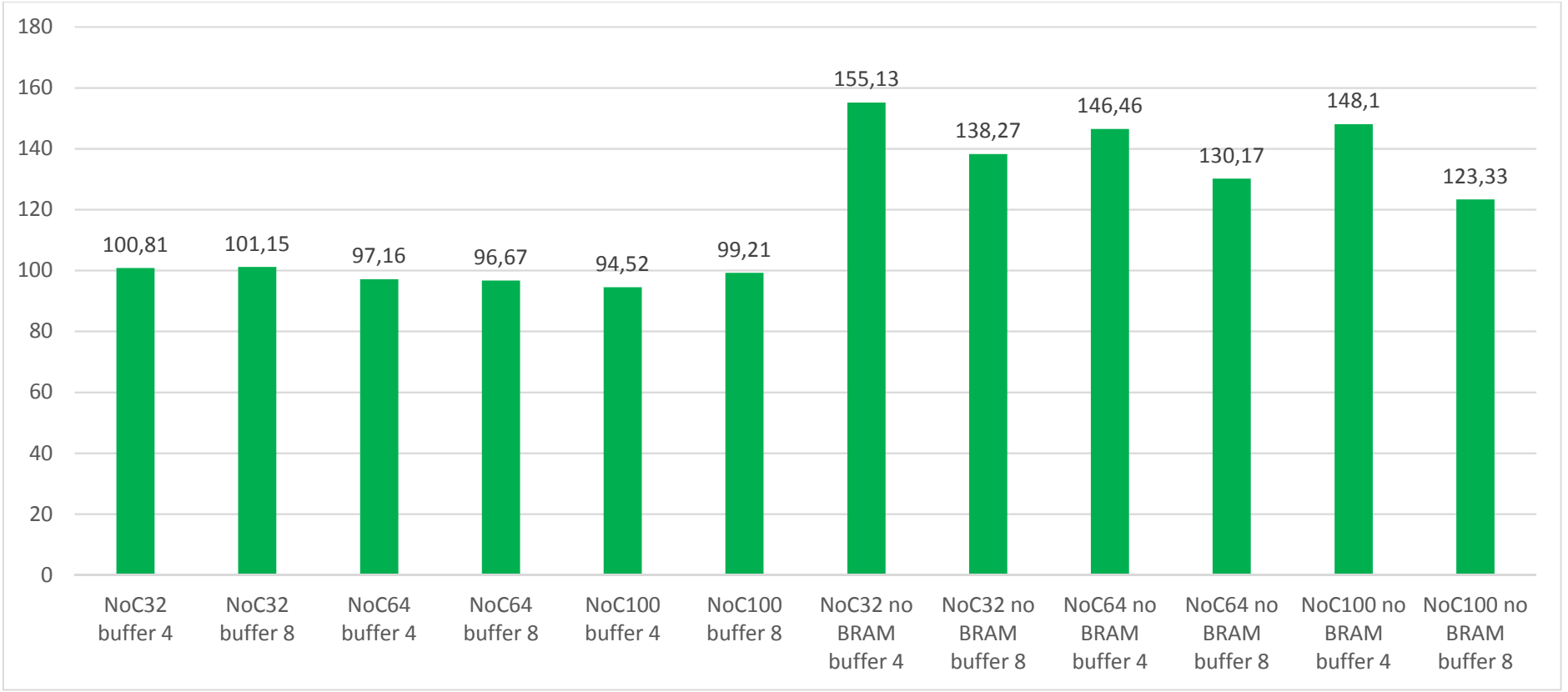


Figure 3.19 : Fréquences de fonctionnement maximales.



3.5.4. Analyses des résultats des placements-routages

Commençons par nous intéresser aux résultats des versions d'Hermes **utilisant la synthèse des mémoires en blocs RAM**, les fréquences de fonctionnement se situent toutes autour des 100 MHz peu importe le nombre de routeurs ou la taille des mémoires tampons utilisés (**Figure 3.19**). Ceci s'explique par le fait que les chemins les plus critiques contiennent tous des registres internes aux blocs RAM, en tant que nœud de départ ou en nœud d'arrivée. Une modification du nombre de routeur ne dépassant pas un certain seuil, ne modifie pas la fréquence de fonctionnement.

Comme les tailles des mémoires tampons sont relativement faibles en comparaison avec celles des blocs RAM, une fois synthétisées, ces dernières n'occupent pas totalement les blocs RAM. Ces blocs peuvent donc supporter des tailles de mémoires tampons supérieures sans que cela n'impacte les résultats des placement-routages. Ceci s'observe par le fait qu'une augmentation de la taille des mémoires tampons pour le même nombre de routeurs, ne modifie que très peu le taux d'occupation des ressources logiques ainsi que la fréquence de fonctionnement.

Toujours en ce qui concerne les résultats des versions utilisant les blocs RAM, une augmentation du nombre de routeurs accroît significativement l'occupation en ressources logiques du circuit (**Figure 3.17**). Premièrement, parce que les modules qui composent les routeurs, autres que les mémoires tampons, occupent des ressources logiques en plus. Et deuxièmement, parce qu'un nombre plus important de blocs RAM est employé. Ceci, car même si les blocs RAM déjà utilisés ne sont pas complètement remplis, un design comptant plus de routeurs s'étend davantage sur la surface du circuit FPGA et donc emploie d'autres blocs RAM, ce qui a pour effet d'augmenter le taux d'occupation en ressources logiques.

Pour l'évaluation des placements-routages des versions d'Hermes utilisant la synthèse des mémoires en blocs RAM (**Figure 3.17**), nous pouvons observer que le taux d'occupation en ressources logiques du circuit répond à nos attentes, puisque les résultats sont d'environs de 22% pour l'interconnexion d'une centaine d'IP et que nous visions un taux de 30% au maximum. Cependant, en ce qui concerne la fréquence de fonctionnement, elle est beaucoup trop faible pour répondre aux contraintes applicatives puisqu'elle ne dépasse que très légèrement les 100MHz et ceci même pour des configurations comptant un nombre de 32 ou 64 IP alors que nous cherchons à atteindre 150 MHz.

Intéressons-nous maintenant aux résultats des placements-routages **sans la synthèse des mémoires en blocs RAM**. En ce qui concerne le taux d'occupation en ressources logiques, la consommation augmente linéairement en fonction du nombre de routeurs et de la taille des mémoires tampons (**Figure 3.18**). Ce résultat est attendu, puisque les mémoires ne sont plus implémentées dans les blocs RAM mais directement dans les ressources logiques. D'où l'importance de les dimensionner correctement, afin de ne pas consommer de ressources inutiles tout en garantissant des performances réseau correctes. En comparaison avec les résultats pour les synthèses des mémoires en blocs RAM, pour des tailles de mémoires tampons de 4 flits la consommation en ressource est équivalente, cependant, avec des tailles de mémoires doublées à 8 flits la consommation en ressources augmente environ de 25%.

Au sujet des fréquences de fonctionnement, (**Figure 3.19**), elles sont bien supérieures à celles des implémentations avec blocs RAM. Pour des mémoires tampons de 4 flits, les fréquences sont accrues de 50% par rapport à la version utilisant les blocs RAM, et pour des tampons de 8 flits, cette augmentation est de 30%. Ceci dit, contrairement aux synthèses utilisant les BRAM, l'accroissement de la taille des mémoires tampons détériore grandement la fréquence de fonctionnement, parce que les chemins critiques sont positionnés au niveau de ces mémoires tampons, et le fait de doubler leur taille, réduit de manière plus importante la fréquence de fonctionnement, que lorsque le nombre de routeurs est multiplié par deux.

Pour conclure sur les résultats de ces placements-routages sans synthèse automatique des mémoires en blocs RAM, l'occupation en ressource est acceptable par rapport aux contraintes applicatives, pour des tailles de tampons faibles, tout en s'approchant de la limite que nous avons fixé pour des tailles de mémoires plus importantes. La fréquence de fonctionnement répond, quant à elle, aux exigences pour des faibles tailles de mémoires tampons, mais pas pour des tailles de mémoire tampons de 8 flits. Le dimensionnement de ces mémoires est d'importance cruciale, puisqu'il a un impact majeur sur les performances.

3.5.5. Chemins critiques

Dans l'environnement de conception Quartus, nous avons utilisé l'outil TimeQuest Timing Analyzer pour faire l'étude des chemins critiques des placements-routages du NoC Hermes [ALT12a]. Cet outil évalue la fréquence maximale que peut atteindre un système donné en fonction des éléments matériels qui le compose. Il donne aussi la possibilité d'éditer de nombreux rapports relatifs au timing et d'identifier entre quels nœuds se trouvent les temps de propagations les plus longs, et violant les contraintes temporelles fixées. Dans le but de désigner un chemin critique, un nœud de départ et un nœud d'arrivée sont toujours utilisés, ainsi que l'horloge correspondant à chacun d'eux.

Pour les placements-routages du NoC dont les mémoires sont synthétisées dans les blocs RAM, les chemins critiques possèdent tous un des sous-registres composant ces blocs, en tant que nœud de départ ou nœud d'arrivée. Les délais importants de ces chemins critiques, peuvent venir d'un nombre trop important d'étages logiques ou de longueurs de lignes trop étendues, ou bien encore de la composition de ces deux facteurs.

Concernant les placements-routages sans synthèse des mémoires en blocs RAM, les mémoires tampons des ports des routeurs sont toujours en cause, sauf que les fréquences de fonctionnement sont plus élevées.

3.6. Conclusions

Dans ce chapitre, nous avons décrit l'architecture du portage du NoC Hermes sur circuit FPGA, que nous avons conçue dans le but d'étudier les performances d'un réseau sur puce générique et polyvalent sur circuit reconfigurable. Pour évaluer les performances de ce portage, nous avons cherché à vérifier si ce dernier pouvait répondre à un certain nombre de contraintes applicatives qui correspondent à des objectifs industriels.

Il s'agit par exemple de s'approcher de faibles latences de communications de l'ordre 1 à 3 cycles d'horloge. Il faut également ne pas dépasser un taux d'occupation de ressources de 30% pour un FPGA ARRIA 10-066 d'Altera, et obtenir des fréquences de fonctionnement d'au moins 150 MHz, pour un réseau supportant une centaine d'IP.

Parmi les contraintes industrielles, il s'agit aussi de pouvoir connecter au réseau des IP communicant dans un protocole propriétaire, et de prendre en compte ces changements dans les résultats des placements-routages. C'est pour cette raison que nous avons modifié l'architecture des routeurs d'Hermes afin qu'ils intègrent un étage d'interface spécifique.

Les résultats obtenus avec les placements-routages pour les implémentations sans synthèse des mémoires en blocs RAM sont viables en termes d'occupation de ressources logiques et de fréquences de fonctionnement, **puisque pour une centaine d'IP et des tailles de mémoires tampons de 4 flits, le taux d'occupation est de 29,8% et la fréquence est 148 MHz.** Le dimensionnement des mémoires tampon se révèle très important, puisqu'il a un impact majeur sur les performances. Pour ce qui est des implémentations utilisant la synthèse des mémoires en blocs RAM, les fréquences de fonctionnement après placements-routages sont trop faibles pour pouvoir nous satisfaire, puisqu'elles n'avoisinent que 100 MHz.

Ensuite, si nous nous intéressons aux **latences de transmission des requêtes, leurs valeurs sont 4 à 5 fois trop élevées par rapport aux contraintes que nous souhaitons satisfaire.** Les valeurs minimales des latences sont de **15 cycles** pour les requêtes d'écriture et de **23 cycles** pour les requêtes de lecture. De plus, il faut ajouter un délai de **5 cycles** d'horloge pour la traversée de chaque routeur qui sépare deux IP. **La latence minimale de transport des messages sur Hermes est donc trop importante pour satisfaire nos contraintes**.

Cette latence initiale relativement importante se retrouve dans la plupart des réseaux sur puce, puisque l'architecture des routeurs en 5 étages de pipeline est très employée. Ces nombreux étages sont plutôt adaptés aux applications sur ASIC qui bénéficient de fréquences de fonctionnement beaucoup plus élevées que sur FPGA, ce qui a pour effet de compenser la latence. Plusieurs améliorations qui concernent la réduction du nombre d'étages de pipeline dans les routeurs ont déjà fait l'objet de publications, notamment dans le cadre de réseaux sur puce pour FPGA. Cependant, il faut veiller à ne pas enlever tous les étages de pipeline, afin de ne pas créer de chemins critiques trop longs, et ainsi faire chuter la fréquence de fonctionnement (consulter l'état de l'art des NoC pour FPGA du deuxième chapitre pour plus de détails).

Le réseau Hermes présente tout de même plusieurs points intéressants, comme celui d'avoir des performances en latence assez élevées une fois que le flit d'entête du paquet a commencé à traverser un routeur ou une interface. En effet, grâce au mécanisme wormhole, lorsque le calcul du chemin emprunté par le paquet a été effectué, chaque flit suivant ne met qu'un cycle d'horloge pour suivre le début du paquet.

Cette première caractéristique rend le réseau Hermes attractif pour le transport d'un faible nombre de paquets de grande taille. Il est possible d'imaginer l'utilisation du NoC Hermes dans une structure hiérarchique de système d'interconnexions similaire à [SEP12], où il aurait pour rôle de relier des clusters qui ont besoin de se transmettre de longs burst de données.

Une deuxième caractéristique intéressante d'Hermes est sa résistance à la charge. Nous n'avons pas encore pu développer la plateforme de simulations adéquate pour pouvoir mesurer cette caractéristique, cependant, des résultats peuvent être trouvés dans la thèse de Séverine Riso qui s'est attachée à étudier l'effet des paramètres architecturaux d'Hermes sur ses performances [RIS06]. Grâce au nombre assez important des mémoires tampons dans ce réseau sur puce, il est capable de résister à des taux d'injection de paquets assez importants.

Le réseau Hermes a l'avantage d'avoir une structure polyvalente, cependant, nous avons opté pour la conception et le développement du réseau sur puce pour circuit FPGA appelé TrustNoC, afin de pouvoir mieux répondre à nos contraintes applicatives. Celles-ci sont le support d'un nombre important de nœuds, tout en garantissant une fréquence de fonctionnement élevée, un taux d'occupation des ressources logiques faibles, et des latences de communications minimales.

Chapitre 4) Conception du réseau sur puce TrustNoC

Le portage du réseau Hermes que nous avons effectué sur FPGA n'a pas satisfait les contraintes applicatives que nous avons fixées. Le NoC Hermes a l'avantage d'être très polyvalent, cependant, il n'est pas particulièrement optimisé pour les circuits FPGA.

Nous avons donc décidé d'appliquer certains principes de conception que nous avons pu trouver dans notre état de l'art des NoC pour FPGA, afin de concevoir et développer TrustNoC, un réseau optimisé pour les circuits reconfigurables.

Ce chapitre présente la philosophie de ce réseau et détaille son architecture. Nous le concluons en comparant les performances de TrustNoC avec celles de notre portage d'Hermes sur circuit reconfigurable.

Sommaire du Chapitre 4

4.1. Introduction

En se basant sur notre état de l'art des NoC pour FPGA, nous avons observé que la quasi-totalité des réseaux proposés nécessitent l'emploi d'un routeur pour chaque IP connectée au medium. Comme la traversée de chaque routeur impose des cycles de latence, pour le stockage des flits, le calcul de l'algorithme de routage et l'application de l'arbitrage, plus un paquet doit traverser de routeurs pour atteindre sa destination, plus sa latence va augmenter.

Afin de réduire la latence moyenne des transmissions sur un réseau sur puce comptant de nombreux routeurs, les deux principaux leviers sont la topologie et la latence de traversée des routeurs.

Plus la topologie est « connectée », c'est-à-dire, plus les routeurs partagent de connexions, plus le nombre moyen de sauts (*hops*) que doivent effectuer les paquets pour se rendre à leur destination va diminuer. Par contre, le nombre d'entrées/sorties des composants internes comme l'arbitre, les modules de routage ou le crossbar, vont s'accroître. Comme l'illustre l'article [YEL11], avec cette augmentation, la surface du crossbar évolue de manière quasiment quadratique, en multipliant le nombre de ressources logiques utilisées et en abaissant la fréquence de fonctionnement.

C'est pour cette raison que la topologie maillée à deux dimensions est l'une des plus utilisées, car elle réalise un compromis intéressant entre le nombre de sauts moyen, et la complexité d'implémentation. De plus, son architecture correspond à celle à deux dimensions des circuits reconfigurables actuels, ce qui est un avantage lors des placements-routages. L'emploi d'une topologie plus interconnectée augmente le risque de voir les performances en consommation de ressources et en fréquence s'effondrer.

Du point de vue de la latence de traversée des routeurs, elle peut être réduite à 1 cycle d'horloge en réalisant toutes les étapes de routage, d'arbitrage, et de transfert durant la même période comme dans le NoC CONNECT [PAP12]. Cependant, le chemin logique qui s'étend sur toute la longueur du routeur a de grandes chances d'être le chemin critique du système sur puce, et de faire chuter drastiquement la fréquence de fonctionnement maximale de ce dernier. Les alternatives [YEL11] et [CHE15] tendent à réduire la latence de traversée à 2, 3 ou 4 cycles grâce à l'emploi de la méthode de routage *lookahead* qui consiste à calculer le routage pour le routeur suivant. Ces solutions atteignent des fréquences 2 à 3 fois plus élevées que CONNECT, cependant leurs latences de transmission croissent rapidement dès lors que les IP communicantes sont séparées par quelques routeurs.

De manière générale, les réseaux sur puce qui tendent à offrir les plus faibles latences sont les réseaux à commutation de circuits et les réseaux à multiplexage temporel. Cependant, ils peuvent manquer de polyvalence et être assez contraignants à mettre en œuvre.

Dans le de but réduire les latences des communications dans TrustNoC, nous avons décidé de réduire le nombre de routeurs employés en portant à 16 le nombre d'IP qui peuvent y être connectées. Au sein de chaque routeur, une architecture de crossbar hybride qui supporte à la fois un **protocole direct** et un **protocole en paquets** est employée. Le **protocole direct** possède des latences de transmission faibles et est utilisé pour supporter les communications au **niveau local** pour les IP connectées au même routeur. Le **protocole en paquets,** est pour sa part employé au **niveau global** pour les IP communicantes connectées à des routeurs différents.

Nous avons vu à travers l'état de l'art qu'il était déconseillé du point de vue des performances d'employer des crossbars, des modules de routage ou des arbitres avec un nombre trop important d'entrées/sorties. Pour cette raison, dans TrustNoC les connexions entre les nœuds sont entièrement configurables. Ce réseau emploie une approche beaucoup plus dédiée que la majorité des NoC pour FPGA proposés dans la littérature.

Pour obtenir les meilleures performances possibles avec TrustNoC, les IP qui communiquent le plus fréquemment ensemble ou qui ont besoin de s'échanger des données avec une latence minimale, sont préférablement connectées au même routeur. Deux IP reliées au même routeur ne partagent pas forcément un canal de communications, en effet, au **niveau local** chaque connexion entre IP doit être explicitement spécifiée. Au **niveau global** pour que deux IP puissent s'envoyer des paquets, il faut que l'architecte du système ait précisé que ces IP possèdent toutes les deux une interface pour les communications en paquets. Dans le but d'augmenter la fréquence de fonctionnement et de réduire le nombre de ressources utilisées par le placement-routage de TrustNoC sur FPGA, il est capital de réduire les connexions entre IP à celles qui sont strictement nécessaires.

Toujours dans l'optique de proposer une solution dédiée et optimisable pour une application précise, TrustNoC supporte n'importe quelle topologie. Ceci est rendu possible par l'emploi de tables de routage et du choix complet des connexions entres les routeurs.

Du point de vue de la sécurité, l'avantage de la configuration complète des connexions locales est de pouvoir spécifier uniquement les chemins entre certaines IP et de pouvoir en isoler certaines. Par exemple, il est possible de seulement connecter une mémoire contenant des clés cryptographiques à un processeur sécurisé. Cela rend toute tentative d'accès de la part d'une autre IP complètement impossible. Cependant, cette solution ne permet pas de prévenir tous les cas d'intrusion, notamment en ce qui concerne les mémoires partagées ou connectées au **niveau global** par une interface paquet. Pour cette raison, nous avons développé des mécanismes de sécurisation, notamment des moniteurs de sécurité, que nous détaillerons dans le cinquième chapitre de ce manuscrit.

4.2. Principes généraux de TrustNoC

4.2.1. Connexions du routeur personnalisables

Afin de réduire le nombre de routeurs que compte le réseau TrustNoC, chacun d'entre eux peut posséder jusqu'à 16 ports pouvant être dédiés à la connexion d'une IP ou d'un autre routeur. Une solution semblable peut être trouvée dans l'article [HIL06], dans laquelle les routeurs comptent 8 ports.

4.2.2. Deux niveaux de communications

Deux niveaux d'interconnexions sont utilisés dans le réseau. Un **niveau local** qui se cantonne à gérer les communications à l'intérieur des routeurs, et un **niveau global** qui gère le transport des messages entre les routeurs.

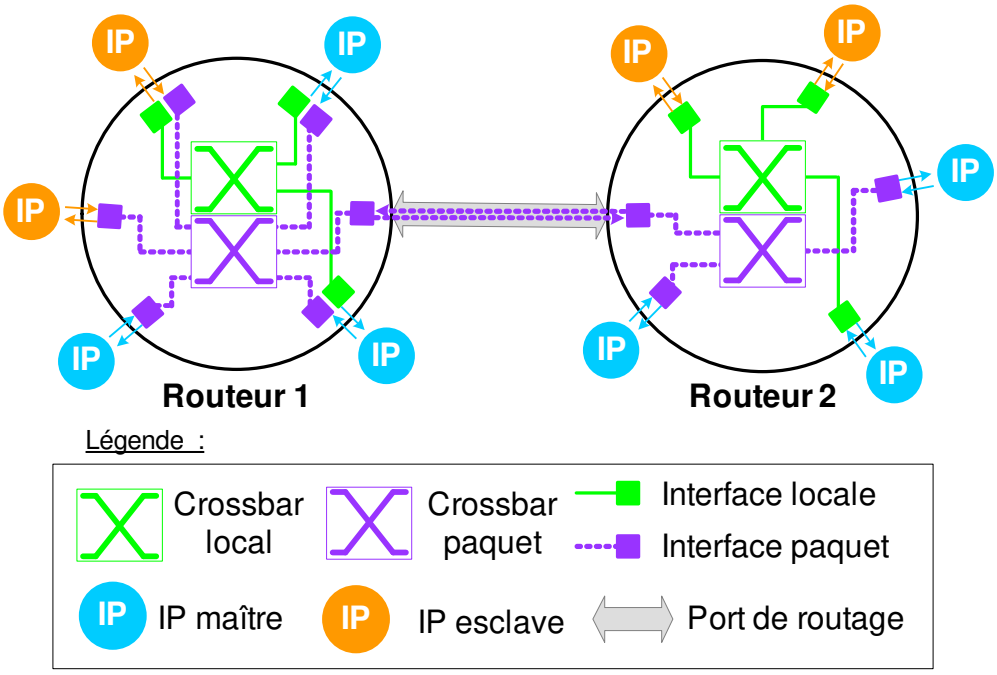


Figure 4.1 : Architecture des connexions aux crossbars de deux routeurs TrustNoC.

Le port de routage est une entité qui supporte les communications entre les routeurs. Comme l'illustre la **Figure 4.1**, chaque niveau de communication possède des interfaces et un crossbar dédié au sein du routeur. Le travail [DXBAR12] évoque la présence de deux crossbars dans le routeur, cependant, pour TrustNoC ceux-ci sont mutuellement isolés. Cela signifie du point de vue de la sécurité, que deux IP placées sur un niveau différent ne peuvent rentrer en communication. Le **Tableau 4.1** explicite les conditions nécessaires pour l'établissement de communications entre deux IP.

Tableau 4.1 : Conditions nécessaires pour la connexion de deux IP dans TrustNoC.

Placement des deux IP	Conditions nécessaires pour l'établissement d'une communication
Sur le même routeur	• L'architecte a précisé une connexion entre les IP. • Les IP possèdent une interface locale. • L'IP initiatrice possède l'IP destinataire dans sa table de décodage d'adresse.
Sur des routeurs différents	• Les IP possèdent une interface paquet • L'IP initiatrice possède l'IP destinataire dans sa table de décodage d'adresse.

Contrairement au réseau Hermes, les arbitres dans TrustNoC sont décentralisés au niveau de chaque interface destinataire. Avec l'utilisation de crossbars, l'établissement de communications parallèles peut s'effectuer sans que celles-ci ne s'interfèrent.

4.2.3. Méthode d'adressage

Il est indispensable que chaque IP ou port de routage possède sa propre adresse dans le réseau afin de pouvoir diriger les messages jusqu'à lui. Donc l'espace d'adressage dans TrustNoC est double. Chaque IP maître, esclave et port de routage possède une **première adresse** qui est **l'adresse réseau**, composée du numéro du routeur auquel l'entité est connectée, et du numéro du port par lequel elle est connectée au routeur.

Le **deuxième espace d'adressage** est celui des **zones mémoires des esclaves**, qui est utilisé par les maîtres pour qu'ils y accèdent. Chaque maître peut posséder sa propre version de l'espace d'adressage mémoire, qu'il utilise pour effectuer des requêtes de lecture ou d'écriture vers les esclaves. Deux IP maîtres peuvent donc atteindre une même IP esclave en utilisant deux adresses mémoire différentes, mais une adresse réseau commune. Les décodeurs d'adresse sont employés pour réaliser la traduction entre adresse mémoire spécifique utilisée par le maître, et le véritable espace d'adressage de l'esclave.

4.2.4. Tables de routage

Les tables de routage sont placées dans trois entités différentes du réseau TrustNoC : au sein des interfaces IP maîtres, des interfaces IP esclaves, et au niveau des ports de routage, c'est-à-dire, dans les jonctions bidirectionnelles qui connectent les routeurs entre eux.

L'avantage des tables de routage est de pouvoir créer des topologies de réseau personnalisées : régulières ou irrégulières. Par contre, plus le réseau est grand, plus les tables de routage risquent d'occuper une surface importante.

Les tables que nous avons utilisées dans TrustNoC sont statiques, elles distribuent les paquets de la même manière pendant toute l'exécution de l'application. Un algorithme d'optimisation peut tout de même être employé en amont, lors du calcul de leur contenu, afin de répartir la charge sur les différents itinéraires possibles.

Il est également envisageable d'utiliser des tables dynamiques afin d'adapter le routage à la congestion, de réduire les blocages, et de diminuer la latence moyenne des échanges.

4.2.5. Format des paquets

Trois types de paquets sont employés dans TrustNoC : le premier pour les **requêtes de lecture**, le second pour les **requêtes d'écriture** et le dernier pour les **acquittements**, autrement dit, pour les réponses aux **requêtes de lecture**. Les flits qui composent ces trois types de paquets sont respectivement représentés sur les **Figure 4.2**, **4.3** et **4.4**.

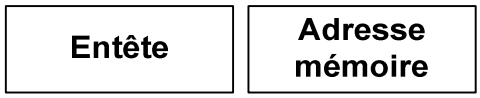


Figure 4.2 : Composition du paquet de requête de lecture.

Figure 4.3 : Composition du paquet de requête d'écriture.

Figure 4.4 : Composition du paquet d'acquittement.

Le contenu du flit d'en-tête illustré sur la **Figure 4.5** varie en fonction du type du paquet, il contient les informations relatives au routage et à la requête transportée dans le paquet. Les bits qui indiquent si le paquet réalise une lecture, une écriture, ou s'il s'agit d'un acquittement sont redondés. Cette redondance permet de détecter une éventuelle erreur de transmission. Elle est vérifiée lors du passage du paquet par un port de routage et lors de sa réception par l'IP destinataire.

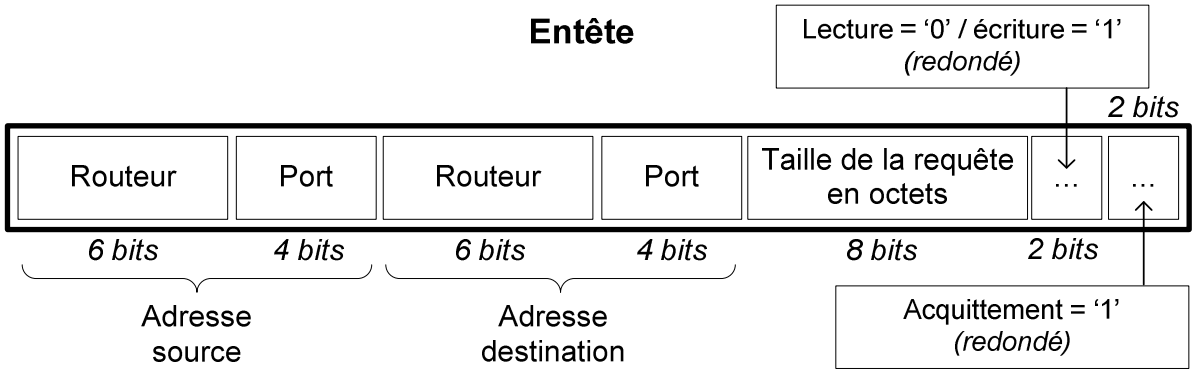


Figure 4.5 : Format du flit d'en-tête

Dans l'optique de mieux partager l'accès des IP au NoC et d'éviter les phénomènes de *starvation*, les paquets sont limités en taille. Lorsqu'une IP souhaite émettre une requête qui dépasse la taille maximale autorisée, celle-ci est scindée en plusieurs paquets envoyés chacun à leur tour. De ce fait, durant l'intervalle de temps qui sépare l'envoi de deux paquets issus de la même requête, un autre paquet peut être autorisé par l'arbitre à être transmis.

4.2.6. Mécanisme de commutation et contrôle de flux

Dans TrustNoC, la commutation de paquets *wormhole* est principalement utilisée, de pair avec le contrôle de flux en *crédit*. L'emploi du mécanisme *wormhole* permet premièrement, d'envoyer des paquets de taille supérieure à celle des mémoires tampons, deuxièmement, de transmettre des données dès que le signal de *crédit* indique que la mémoire tampon contient un espace libre, ce qui n'est pas le cas pour les mécanismes *Store & Forward* et *Virtual-Cut Through (VCT)*. Le mécanisme *Store & Forward* est par contre utilisé par l'interface IP esclave pour transmettre les paquets de réponse aux requêtes de lecture, la raison est détaillée dans la partie qui décrit l'architecture de l'interface IP esclave.

L'article [MAI15] avance que la commutation de paquets *Virtual-Cut Through (VCT)* résisterait mieux à la charge que le mécanisme *wormhole*. Cependant, cette affirmation repose sur les résultats de l'article [REX94] qui ne traite pas des NoC mais des routeurs informatiques. Un futur travail de recherche pourrait avoir pour objet de comparer la résistance à la charge de NoC employant les mécanismes *VCT* et *wormhole,* ceci afin d'évaluer la supériorité d'un mécanisme par rapport à l'autre.

4.2.7. Placement des mémoires tampons

Le nombre très limité de mémoires tampons employées dans TrustNoC contribue à en faire un **réseau léger**. D'autant plus que la synthèse des mémoires sur FPGA consomme une quantité importante de ressources logiques, et se trouve fréquemment dans les chemins critiques.

Dans un NoC classique une mémoire tampon est généralement placée au niveau de chaque port d'entrée, de sortie, ou au niveau des canaux virtuels si le routeur en possède. Au sein de TrustNoC aucune mémoire n'est utilisée au niveau des connexions locales en dehors du décodeur d'adresse de l'interface maître. L'architecture au niveau local se rapproche donc de celles des réseaux sans buffer *(bufferless)* [KAP15].

Pour les communications globales en paquets, une partie des FIFO est positionnée au niveau des ports de routage (**Figure 4.9**), ce qui permet d'absorber le trafic et de limiter la surcharge réseau. Cependant, les ports de routage sont les principaux goulots d'étranglement du réseau, et peuvent être l'objet d'améliorations. Dans l'interface IP maître (**Figure 4.7**), une RAM asynchrone est employée comme table de décodage d'adresses, et un registre sert à stocker un mot de données pour limiter les interruptions lors du transfert des requêtes d'écriture de plusieurs mots, aussi appelées *rafales* ou *burst*. Au sein de l'interface IP esclave (**Figure 4.8**) une FIFO est employée pour stocker et transférer les paquets d'acquittements de plusieurs mots, c'est-à-dire, les paquets de réponse aux requêtes de *burst* de lecture.

Aucune mémoire tampon n'est employée dans le dépaquétiseur de requêtes de l'interface IP esclave ou dans le dépaquétiseur d'acquittement de l'interface IP maître, les paquets sont décomposés à la volée. Grâce au mécanisme *wormhole* qui est utilisé dans TrustNoC sans l'entrelacement des paquets, lorsque la transmission du flit d'en-tête d'un paquet a eu lieu, il est assuré que les flits restants qui composent le paquet suivront sans interruption.

4.2.8. Contrôles d'erreurs

Plusieurs contrôles d'erreurs sont effectués durant le transport des paquets. Ils sont à différencier des mécanismes de sécurité. Ces derniers sont détaillés dans le cinquième chapitre de ce manuscrit.

Les mécanismes de contrôle d'erreurs vérifient principalement la valeur des bits redondés contenus dans l'en-tête, notamment ceux qui signalent le type de la requête : *lecture* ou *écriture*, et ceux qui indiquent si le paquet est un acquittement. Les contrôles d'erreurs sont résumés dans le **Tableau 4.2**.

Tableau 4.2 : Contrôles d'erreurs effectués dans TrustNoC.

Module – sous-module	Contrôle(s) effectué(s)
Interface maître – dépaquétiseur d'acquittement	Flit d'en-tête : valeur redondée des bits d'acquittement.
Interface maître - décodeur d'adresse	Adresse mémoire de la requête non connue, aucune correspondance trouvée.
Interface esclave - dépaquétiseur de requêtes	Flit d'en-tête : valeur redondée des bits de type de requête : *lecture* ou *d'écriture*, et bits d'acquittement désactivés.
Port de routage – réception du paquet / émission du paquet	Flit d'en-tête : valeur redondée des bits de type de requête : *lecture* ou *d'écriture*, et des bits d'acquittement.

La détection de valeurs incohérentes lors d'un des contrôles de redondance mène à l'activation d'un signal d'alarme. Tel que TrustNoC est conçu actuellement, tous les signaux d'alarme issus des différents modules sont rassemblés au niveau de l'entité supérieure en un seul signal. Ce signal peut seulement indiquer la détection d'une erreur dans le réseau sans possibilité de localisation. L'idée étant de limiter le nombre de signaux de sortie dédiés aux alarmes au niveau de l'entité supérieure.

Cependant, il est possible de rassembler les signaux d'alarme pour certains clusters ou pour certaines IP critiques, afin de détecter plus précisément où sont localisées les erreurs. Dans le but de mieux diagnostiquer une erreur, des informations supplémentaires peuvent être envoyées au contrôleur d'alarme, par exemple le contenu de la requête pour laquelle le décodeur d'adresse n'a pu trouver aucune correspondance.

Le rôle de TrustNoC se limite actuellement à la détection de quelques erreurs de transmission qui peuvent toucher les bits qui indiquent le type d'accès réalisé par la requête (*lecture* ou *écriture*) ou les bits d'acquittements. Pour corriger celles-ci, il faudrait employer un code correcteur ou tripliquer l'information et ajouter une logique de vote.

L'alarme déclenchée par le décodeur d'adresse qui signale qu'aucune correspondance n'a pu être trouvée, indique que l'IP tente d'accéder à une région située en dehors de son domaine d'adresse. Cela peut être dû à une erreur de conception se traduisant par une table du décodeur incomplète, mais ça peut aussi signifier que l'IP a été détournée de son comportement habituel par un virus. Afin de pallier ce genre d'évènements, un processeur dédié à la sécurité peut appliquer des contre-mesures en fonction des alarmes qu'il reçoit. Comme dans le travail [COT12], un compteur peut mesurer le nombre d'alarmes déclenchées durant une certaine période de temps, si la valeur du compteur dépasse un certain seuil, alors l'IP en cause peut être complètement déconnectée du système, par conséquent, elle ne pourra même plus effectuer des requêtes non fautives jusqu'à ce qu'elle soit réinitialisée.

Dans TrustNoC, dans le cas où aucune correspondance n'est trouvée dans le décodeur d'adresse, la requête est acquittée sans qu'elle ne soit transmise au reste du réseau. Ceci afin d'éviter le blocage du NoC ou de transmettre une requête vers une mauvaise adresse.

De la même manière, lors des étapes de dépaquétisation, si des valeurs incohérentes sont détectées pour les bits de redondance d'un paquet, celui-ci n'est pas traité et un signal d'alarme est déclenché.

4.3. Architecture détaillée du réseau TrustNoC

Dans cette partie, nous allons nous attacher à décrire les modes de fonctionnement de TrustNoC (**Figure 4.6**). Le principal intérêt de ce nouveau réseau est de diminuer la latence des échanges, d'une part, grâce à l'augmentation du nombre d'IP pouvant être connectées à chaque routeur, et d'autre part, par l'utilisation d'un protocole basse latence pour les communications **locales** au sein du même routeur. Le niveau de communication **global**, quant à lui, répond à la problématique d'évolutivité en permettant une large augmentation du nombre de nœuds communicants.

La prise en compte des contraintes de conception sur FPGA est primordiale lors du développement d'un réseau sur puce sur cette plateforme, ceci dans le but d'atteindre les meilleures performances possibles lors des placements-routages. C'est pour cette raison, que nous avons veillé à réduire au maximum la quantité de mémoires tampons utilisées dans TrustNoC, mais aussi à limiter la taille des chemins critiques, tout en essayant d'avoir un impact minimal sur les latences des échanges, ainsi que sur la résistance du réseau à la charge.

Contrairement à la plupart des réseaux sur puce pour FPGA que nous avons pu étudier dans notre état de l'art, et contrairement au NoC Hermes, TrustNoC a une architecture beaucoup plus personnalisable qui peut être dédiée à une application en particulier. L'idée étant de pouvoir optimiser le réseau assez finement en sélectionnant quelles IP sont interconnectées au niveau local, et quelles IP sont connectées au niveau global.

Cela permet de réduire la quantité de logique employée jusqu'à la valeur minimale nécessaire, donc de réaliser des économies de ressources lors des implémentations sur FPGA, et améliorer les fréquences de fonctionnement lors des placements-routages. Toutefois, le NoC nécessite un temps de paramétrage plus important.

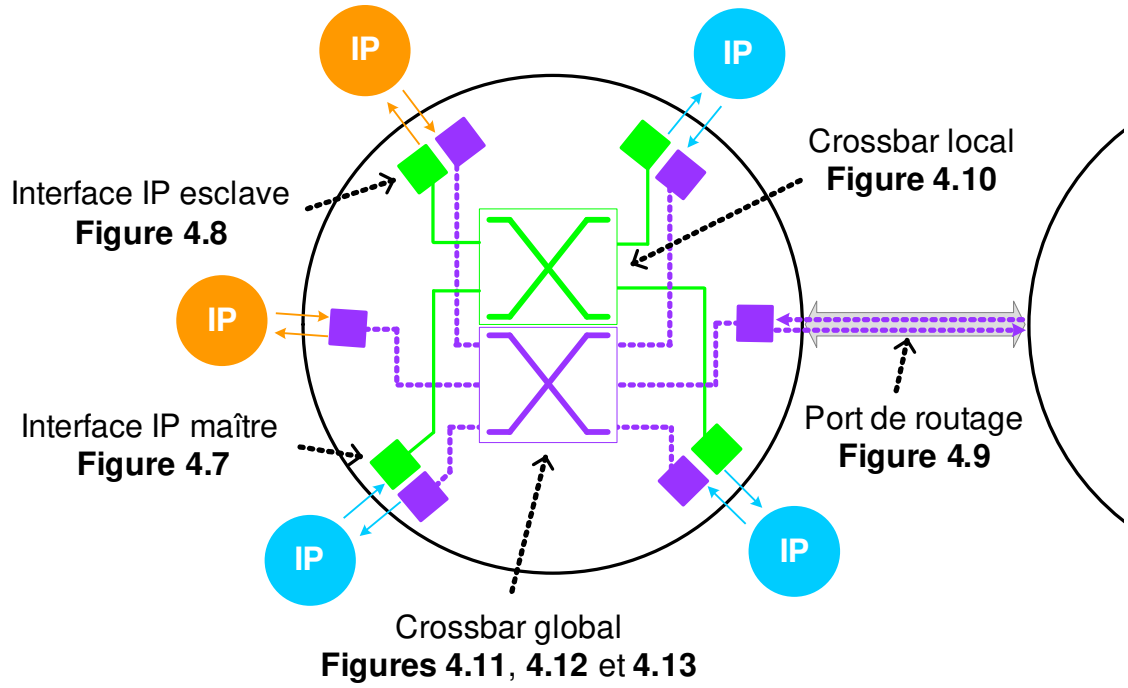


Figure 4.6 : Architecture détaillée de TrustNoC.

Le niveau d'abstraction des **Figures 4.7** à **4.13** est assez élevé. Par exemple, un module de paquétisation qui traduit les requêtes du protocole local au protocole paquet est représenté par une boîte noire. Nous ne décrivons pas la composition complète des différentes machines à états qui le composent. Cela aurait sans doute un intérêt, mais comme chaque protocole de communication est différent, la construction du module de paquétisation pour chacun d'entre eux le serait également. Notre approche est donc plus générale et par conséquent peut potentiellement être adapté à une grande variété de protocoles de communications.

4.3.1. Interface IP maître

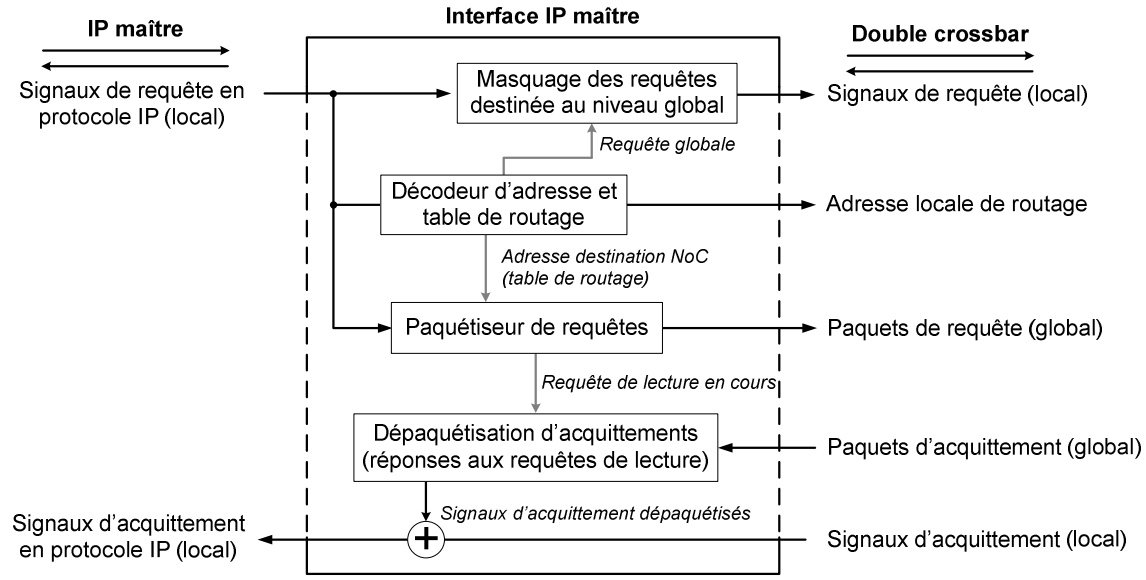


Figure 4.7 : Architecture détaillée de l'interface IP maître.

L'architecture de l'interface IP maître de TrustNoC (**Figure 4.7**) utilise deux niveaux de communications, le premier pour les communications locales (au sein du routeur), et le second pour les communications globales en paquets (entre les routeurs). Lors de la soumission d'une requête par l'IP maître, le décodeur d'adresse indique si la requête est destinée au niveau local ou au niveau global et resynchronise les signaux de transmission.

Dans le premier cas, le masquage des signaux du niveau local n'est pas activé, signifiant que la requête est envoyée à travers le crossbar local vers l'IP destinataire située sur le même routeur. **Dans le second cas**, le masquage est activé, bloquant la transmission des signaux de requête au niveau local. Cette fois-ci, par contre, une commande est envoyée au module de paquétisation afin que celui-ci traduise la requête en format paquet. Cette commande est jointe avec l'adresse NoC de l'IP destinataire afin qu'elle soit ajoutée à l'en-tête du paquet.

Le décodeur d'adresse joue aussi le rôle de table de routage, en indiquant quel port du routeur local doit être emprunté par la requête pour qu'elle se dirige vers sa destination.

Dans la version actuelle du protocole, les paquets d'acquittements arrivant aux IP maîtres sont uniquement employés pour transporter les réponses aux requêtes de lecture. Comme une IP maître peut seulement transmettre une nouvelle requête lorsque la précédente est terminée, le cas où plusieurs acquittements seraient reçus simultanément par celle-ci n'a pas besoin d'être

traité. Le dépaquétiseur d'acquittement filtre l'en-tête des paquets afin de transmettre seulement à l'IP maître les données issues d'une requête de lecture. En plus du filtrage de l'en-tête, le dépaquétiseur vérifie la valeur des bits redondés qui indique qu'il s'agit bien d'un paquet d'acquittement.

Les paquets de requête sont envoyés par l'interface IP maître grâce au mécanisme *wormhole*, et le contrôle de flux *backpressure*. Cela signifie que si le signal d'acquittement est actif durant le cycle de transmission en cours, l'entité émettrice considère sa transmission comme réussie, et émet de nouvelles données au cycle suivant. Grâce à ces deux mécanismes, l'unité de paquétisation de requêtes n'a pas besoin d'intégrer de FIFO pour stocker les paquets en attente de transmission, car elle fonctionne à la volée. C'est-à-dire, qu'elle cesse d'acquitter et met en pause la requête provenant de l'IP maître tant que le réseau n'est pas capable de recevoir de paquets.

4.3.2. Interface IP esclave

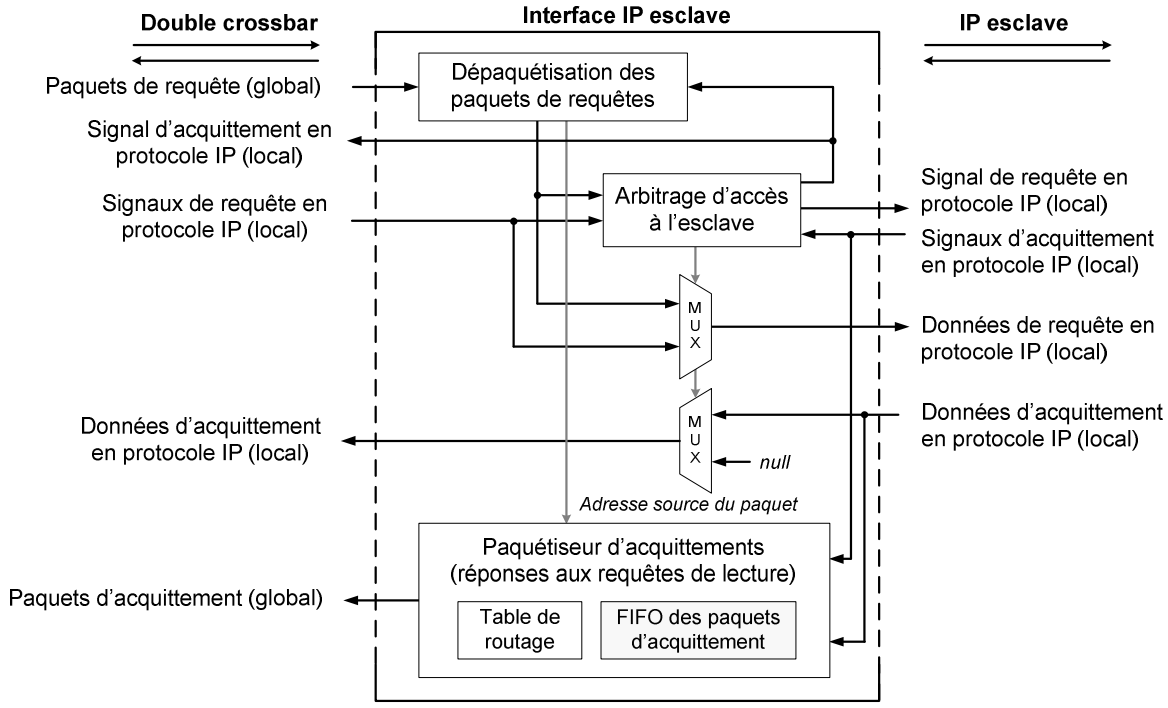


Figure 4.8 : Architecture détaillée de l'interface IP esclave

De la même manière que l'IP maître, l'interface IP esclave (**Figure 4.8**) est connectée aux niveaux de communications : **local** et **global**. Les paquets provenant du niveau **global** sont d'abord désencapsulés par le module de dépaquétisation des paquets de requête avant qu'une demande ne soit envoyée à l'arbitre qui régit l'accès à l'esclave. Cet arbitre autorise chacune à leur tour, les requêtes provenant des niveaux : **local** et **global**. Pour cela, il pilote un multiplexeur qui aiguille les données provenant du crossbar local ou du dépaquétiseur de requêtes, vers le bus de données en entrée de l'esclave.

Un multiplexeur est aussi employé dans le sens opposé, c'est-à-dire, pour choisir si les données d'acquittement de l'IP esclave sont transmises au crossbar local ou non. Ce choix est réalisé en fonction du niveau de communication d'où provient initialement la requête de lecture.

Lorsque le dépaquétiseur réceptionne une requête de lecture du niveau global, il informe le module de paquétisation d'acquittement, afin que celui-ci compose un paquet de réponses à partir des données de lecture provenant de l'IP esclave. Le dépaquétiseur transmet aussi l'adresse source du paquet reçu, pour que celle-ci soit utilisée comme adresse de destination du paquet d'acquittement. Cette adresse source est aussi employée lors de la consultation de la table de routage lors de l'envoi d'un paquet d'acquittement.

Contrairement à la transmission des paquets par l'interface IP maître qui a lieu avec le mécanisme *wormhole*, les paquets d'acquittement sont envoyés par l'interface esclave grâce au mécanisme *store & forward*. Cela signifie que le paquet doit être entièrement stocké dans une FIFO avant d'être envoyé sur le réseau.

Ce choix s'explique par le fait que les IP esclaves peuvent réaliser des acquittements avec des délais non déterministes. Sauf, qu'une des contraintes de TrustNoC, et plus globalement d'une partie des réseaux sur puce, est que lorsque la transmission d'un paquet débute, elle doit se poursuivre sans interruption, tant que l'interface réceptrice est en capacité de recevoir des données.

Dans le cas où le mécanisme *wormhole* est utilisé, si jamais l'interface IP débute la transmission d'un paquet de réponse vers le NoC, alors que l'IP esclave n'a pas terminé de lui transmettre toutes les données issues de la requête de lecture, alors l'interface IP esclave peut se retrouver à devoir transmettre au réseau des données qu'elle ne possède pas encore.

Pour éviter de se retrouver dans ce cas, le mécanisme *store & forward* exige de la part de l'interface IP esclave de devoir stocker le paquet de réponse dans son intégralité, avant de commencer une transmission. L'inconvénient de ce mécanisme par rapport au mécanisme *wormhole* est d'imposer une latence supplémentaire et proportionnelle à la taille du paquet à transmettre. D'autres solutions sont envisageables à la place du *store & forward*, comme celle d'utiliser un signal supplémentaire dans le NoC de mise en pause de la transmission par l'émetteur.

4.3.3. Port de routage

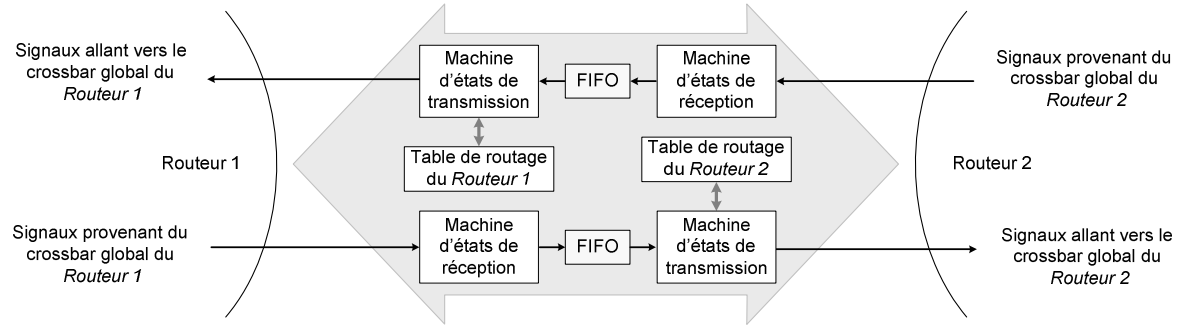


Figure 4.9 : Architecture détaillée du port de routage.

Les ports de routage (**Figure 4.9**) sont employés pour supporter les échanges de paquets entre deux routeurs. Ce sont des connexions bidirectionnelles, composées de deux canaux unidirectionnels de sens contraire ayant une architecture symétrique.

Chaque canal contient une machine à états qui stocke les paquets provenant du routeur émetteur, et qui sont en attente de transmission vers le routeur récepteur. Pour chaque paquet,

la machine à états consulte la table de routage afin de connaître le port du routeur où doit être envoyé le paquet pour qu'il atteigne sa destination.

Les ports de routage sont uniquement connectés aux crossbars globaux des routeurs qui supportent les communications en paquets, ce qui rend impossible les transmissions de paquets vers les IP connectées uniquement par les connexions aux routeurs.

Comme pour les interfaces IP maîtres, le mécanisme *wormhole* et le contrôle de flux *backpressure* sont utilisés pour la transmission des paquets, ceux-ci garantissant apriori les latences de communication les plus minimales.

4.3.4. Crossbar local

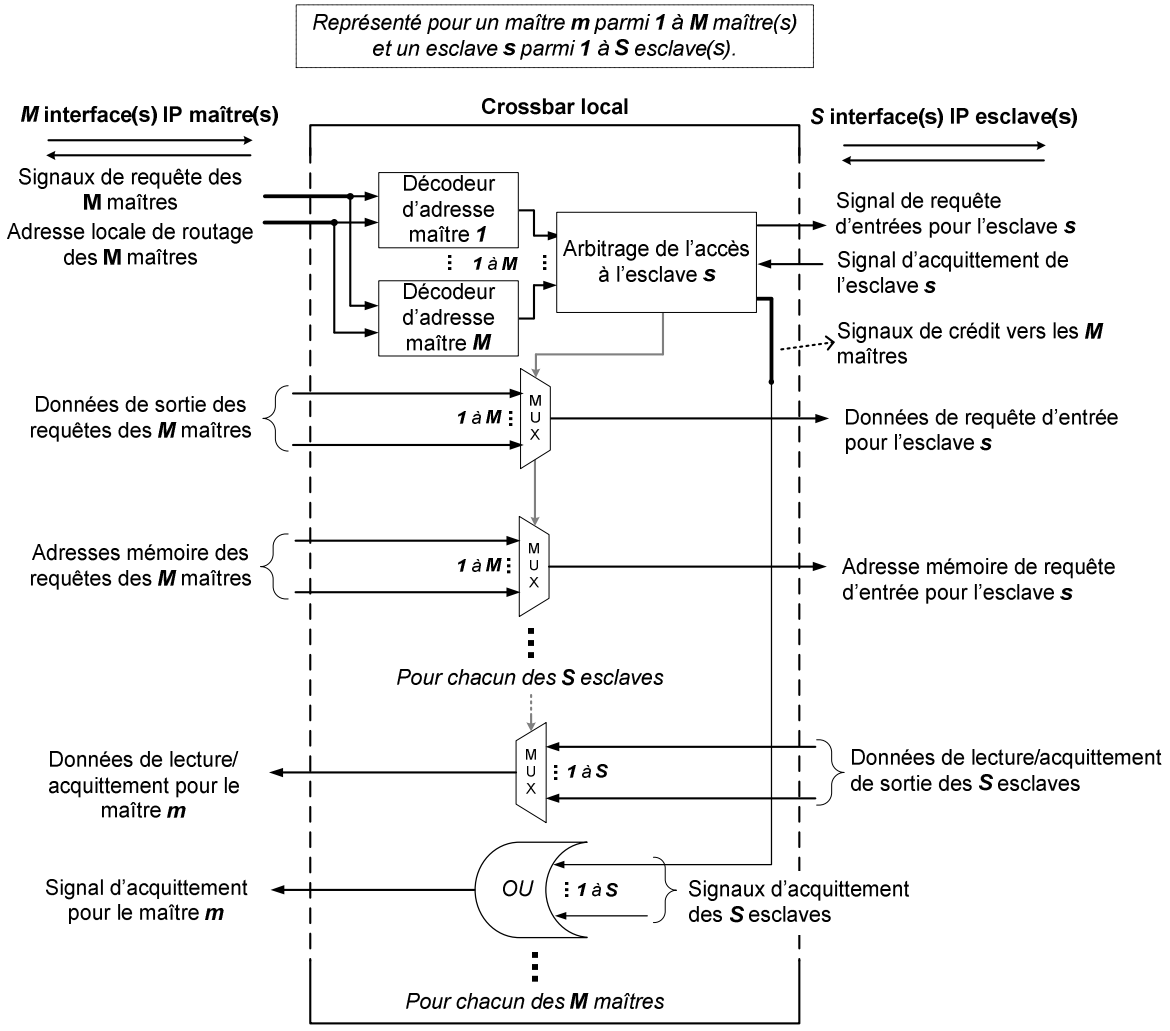


Figure 4.10 : Architecture détaillée du crossbar local.

Un crossbar local (**Figure 4.10**) est employé au sein de chaque routeur, lorsqu'au moins deux IP du même routeur partagent une connexion locale. Les connexions locales sont entièrement configurables, ce qui signifie que seules les IP qui ont besoin de communiquer sont connectées entre elles. D'une part, l'évitement des connexions inutiles permet d'économiser des

ressources, et d'autre part, du point de vue de la sécurité cela permet d'isoler les IP sensibles du reste du réseau.

Le protocole de communication dans le crossbar local est celui qui est nativement utilisé par les IP du système sur puce, ce qui retire la nécessité de réaliser des étapes d'adaptation de protocoles dans les interfaces IP.

Le protocole que nous avons utilisé est le suivant : les IP maîtres transmettent des requêtes de lecture ou d'écriture aux IP esclaves, auxquelles ces dernières répondent avec des signaux d'acquittement. L'acquittement des requêtes d'écriture se limite à la confirmation de la réception des données. Tandis que pour les requêtes de lecture, un bus de données supplémentaire transmet à l'IP maître les données issues des requêtes de lecture.

L'un des principaux avantages du crossbar étant d'autoriser plusieurs paires d'IP différentes à communiquer parallèlement, l'architecture de celui-ci est complètement distribuée, comparable aux routeurs qui utilisent les primitives *split & merge* [HUA12].

Afin d'accéder à l'interface IP esclave, et donc à l'IP esclave en elle-même, les signaux de requête de chaque maître sont d'abord connectés à un étage de décodage d'adresse. Un décodeur d'adresse est employé pour chaque connexion entre un maître et un esclave. Celui-ci a pour rôle de faire suivre le signal de requête à l'arbitre, si l'adresse locale de destination de la requête correspond à celle de l'esclave.

L'arbitre régit ensuite l'accès des différents maîtres souhaitant simultanément transmettre une requête à l'esclave, selon l'algorithme de priorité *round-robin*. Des multiplexeurs commandés par l'arbitre ont ensuite pour rôle d'aiguiller les données qui proviennent du maître qui a gagné l'accès vers l'IP esclave. Des multiplexeurs sont aussi employés dans la direction opposée, en transmettant les données en sortie de l'IP esclave vers l'IP maître responsable de la requête de lecture. L'arbitre a aussi pour rôle de transmettre le signal d'acquittement de l'esclave, seulement vers le maître qui est en train d'y accéder.

Comme les maîtres ne peuvent effectuer localement qu'une seule requête à la fois, ils sont susceptibles de recevoir qu'un seul signal d'acquittement d'un esclave à la fois. Pour cette raison, et pour une question d'économie de ressources, une porte logique « OU » réunit les signaux en direction du maître provenant des esclaves auxquels est connecté ce même maître, en un seul signal d'acquittement.

4.3.5. Crossbar global (première partie)

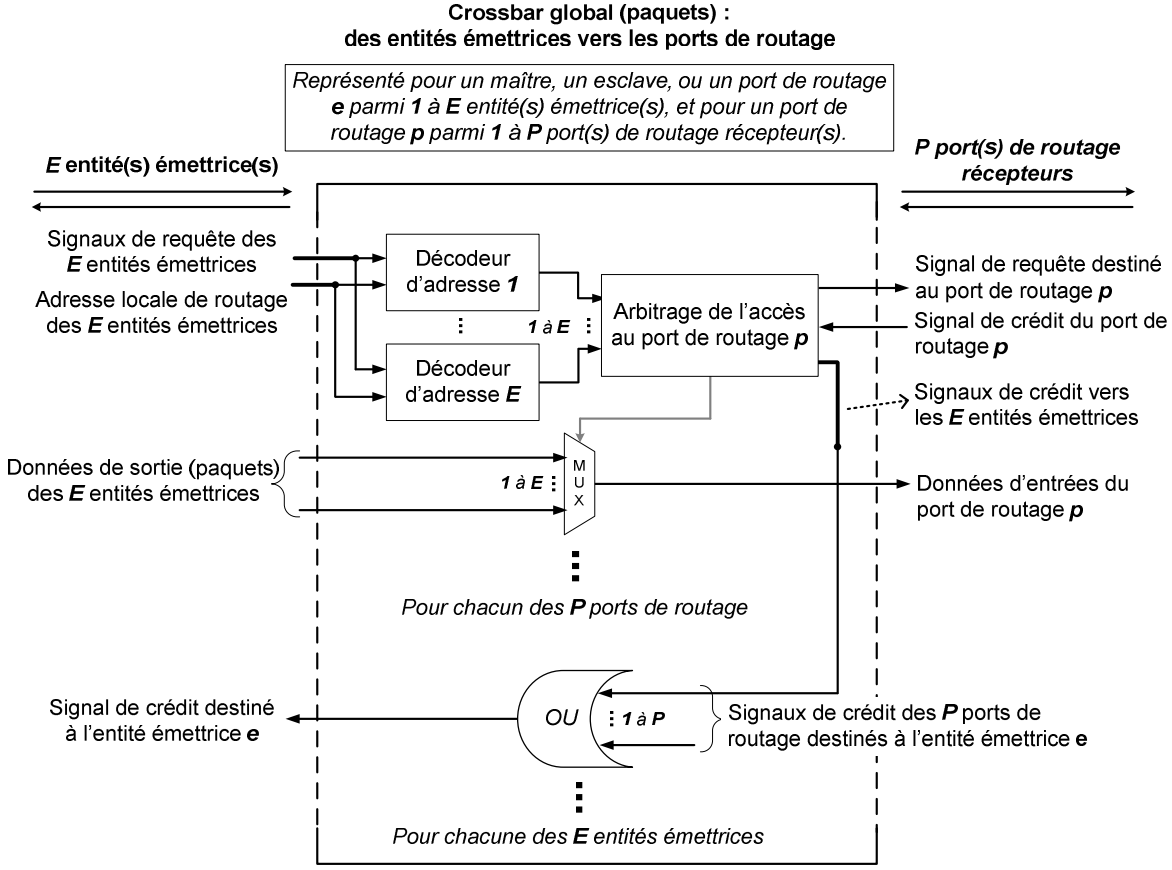


Figure 4.11 : Architecture détaillée du crossbar « paquets » (1ère partie).

Le principe du crossbar global est très similaire à celui du crossbar local, cependant, cette fois-ci, en plus des IP maîtres et des IP esclaves, les ports de routage sont aussi connectés. Afin de détailler l'architecture du crossbar global le plus clairement possible, nous avons divisé la présentation de celle-ci en trois parties.

La première partie (Figure 4.11) s'attache à décrire l'architecture dédiée à l'émission des paquets depuis les nœuds IP maîtres, IP esclaves et des ports de routage, vers les autres ports de routage.

Au sein d'un même routeur, pour qu'une IP maître et une IP esclave puissent communiquer, elles doivent utiliser le crossbar local. De par l'architecture du crossbar global, au sein d'un même routeur il est impossible que l'interface paquet d'une IP maître et d'une IP esclave soient directement connectées. Pour que ces deux types d'IP puissent communiquer par le biais de leur interface paquets, il faut qu'elles soient situées sur des routeurs différents et passent donc d'abord par l'intermédiaire d'un port de routage. C'est pour cette raison que la première partie de l'architecture du crossbar global détaille les éléments employés dans le transport des paquets émis par les IP maîtres et esclaves en direction des ports de routage.

Les ports de routage peuvent aussi se retrouver à échanger des paquets, dans le cas où un message traverse un routeur sans que son IP destinataire ne soit connectée à celui-ci.

A propos de l'architecture de la première partie du crossbar global (**Figure 4.11**), son fonctionnement est très similaire à celui du crossbar local, la principale distinction étant que ce ne sont pas les signaux locaux de requêtes qui sont connectés en entrée et en sortie, mais les signaux dédiés au transport des paquets.

Par rapport au crossbar local, les IP maîtres sont remplacées par les IP « émettrices », c'est-à-dire, à fois les IP maîtres, les IP esclaves et les ports de routage. Le rôle des IP esclaves est quant à lui remplacé par celui des « autres » ports de routage. Le terme « autres » signifie juste que lorsque l'on s'intéresse à l'émission d'un port de routage, il ne peut pas faire partie des entités émettrices, donc il ne peut transmettre des paquets qu'aux « autres » ports de routage.

Afin de régir l'accès des entités émettrices aux ports de routage, une architecture similaire à celle du crossbar local est employée, c'est-à-dire, avec un étage de décodage d'adresse positionné en amont de l'étage d'arbitrage. L'arbitre d'un port de routage donné commande un multiplexeur qui dirige les données provenant de l'entité émettrice ayant remporté l'accès au port de routage, parmi toutes les entités émettrices (de paquets) que compte le routeur.

Le signal de contrôle de flux du port de routage est d'abord envoyé à l'arbitre, qui a pour rôle de le diriger vers l'IP émettrice ayant accès au port de routage

4.3.6. Crossbar global (deuxième partie)

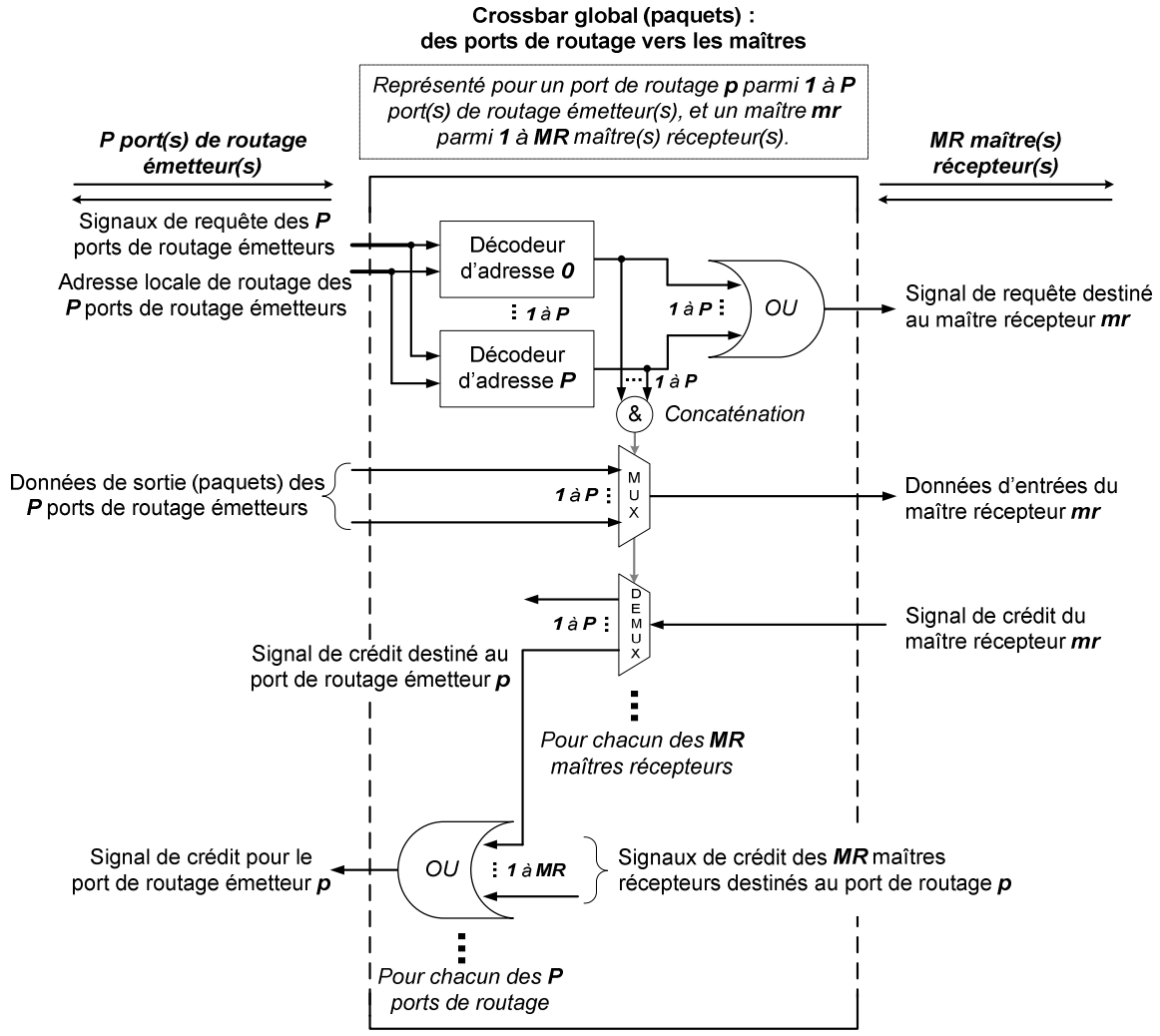


Figure 4.12 : Architecture détaillée du crossbar de « paquets » (2ème partie).

La deuxième partie de notre description du crossbar global (Figure 4.12) s'attache à décrire les communications des ports de routage vers les IP maîtres. Comme pour le crossbar global, les esclaves peuvent envoyer des données d'acquittement, qui prennent cette-fois ci la forme de paquets de réponse aux requêtes de lecture.

Les interfaces paquets des IP maîtres ne pouvant être connectées aux interfaces paquets des IP esclaves que par l'intermédiaire des ports de routage, les paquets de réponse aux requêtes de lecture ne peuvent être reçus par les maîtres que de la part des ports de routage.

Le principe de l'architecture est elle aussi similaire à celle du crossbar local, sauf que la présence d'un arbitre pour gérer l'accès au maître n'est pas requise. Un maître ne peut soumettre qu'une requête à la fois, donc il est seulement susceptible de recevoir un seul paquet à la fois. L'arbitre n'étant pas présent, un démultiplexeur a pour rôle de diriger le signal de contrôle de flux vers le port de routage en train de transmettre des données au maître.

La traversée de la deuxième partie du crossbar global n'impose aucun délai car elle ne contient pas d'arbitre.

4.3.7. Crossbar global (troisième partie)

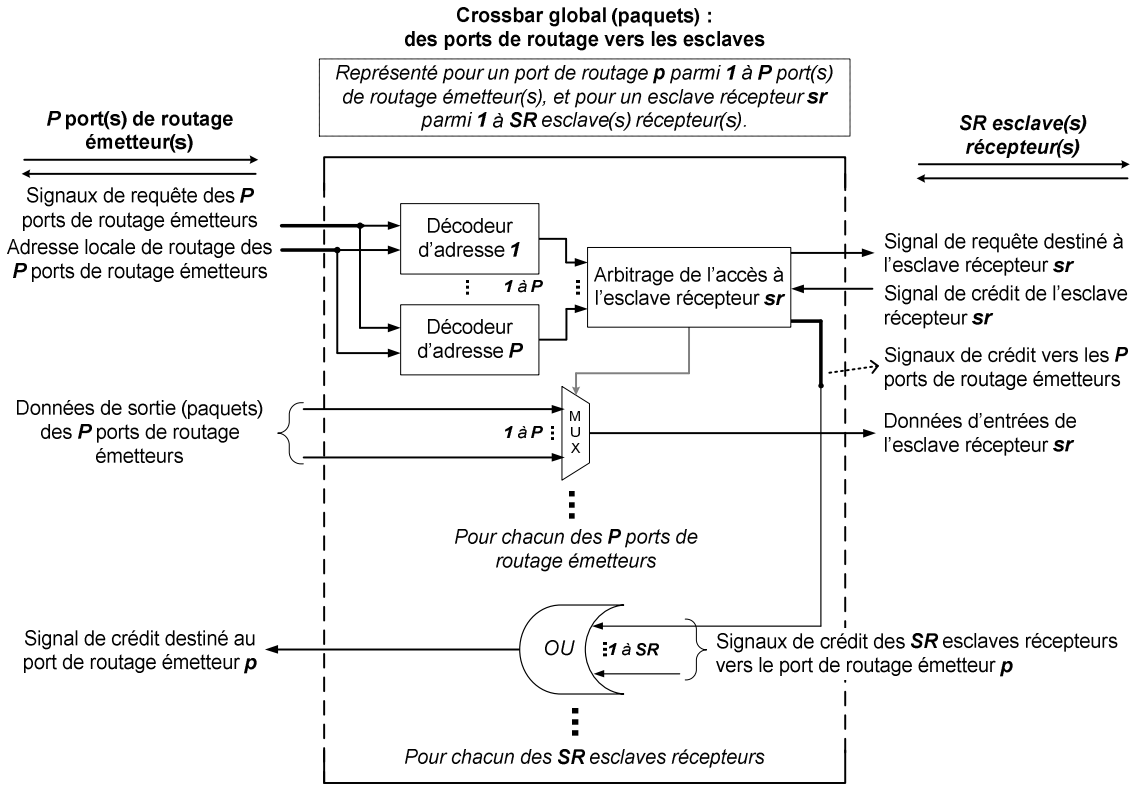


Figure 4.13 : Architecture détaillée du crossbar « paquets » (3ème partie).

La troisième partie du crossbar global (Figure 4.13) est quasiment identique à la première partie, sauf qu'elle décrit l'émission des ports de routage vers les IP esclaves. Comme les esclaves peuvent potentiellement recevoir simultanément des paquets de requête provenant de plusieurs maîtres, un arbitre est indispensable.

4.3.8. Latences de transmission par module

Le **Tableau 4.3** résume pour chaque module qui compose le réseau TrustNoC les latences initiales de traitement d'une requête ou de l'en-tête d'un paquet. Chaque mot de données additionnel qui compose la requête ou le paquet est traité en un cycle d'horloge supplémentaire.

Tableau 4.3 : Latences de fonctionnement des modules qui composent le réseau TrustNoC.

Module	Action	Latence
Interface maître	Transmission d'une requête au niveau **local**	1 cycle
	Réception d'un acquittement au niveau **local**	Aucune
	Transmission d'une requête au niveau **global**	2 cycles
	Réception d'un acquittement au niveau **global**	Aucune
Interface esclave	Réception d'une requête au niveau **local**	1 cycle
	Envoi d'un acquittement au niveau **local**	Aucune
	Réception d'une requête au niveau **global**	- 3 cycles pour une lecture. - 4 cycles pour une écriture.
	Envoi d'un acquittement au niveau **global**	1 cycle de latence pour chaque donnée lue.
Port de routage	Transport d'un paquet	- 4 cycles pour un paquet qui a atteint son routeur destinataire. - 5 cycles si le paquet doit encore traverser au moins un routeur.
Crossbar local	Transport d'une requête	1 cycle
Crossbar global : des IP émettrices vers les ports de routage.	Transport d'un paquet	1 cycle
Crossbar global : des ports de routage vers les IP maîtres.	Transport d'un paquet	Aucune
Crossbar global : des ports de routage vers les IP esclaves.	Transport d'un paquet	1 cycle

4.4. Mise en œuvre pratique

Pour faciliter la mise en œuvre de TrustNoC sur FPGA, nous avons fait le choix de limiter les fichiers de configuration au nombre de deux. Ces deux fichiers sont situés au plus haut niveau de l'arborescence de fichiers VHDL de TrustNoC qui est représentée sur la **Figure 4.14**.

Le premier fichier de « configuration du NoC » est dédié au choix de la topologie, du nombre et du type d'IP connectées à chaque routeur, ainsi que du contenu des décodeurs d'adresses et des tables de routage. La configuration des connexions des niveaux local et global est aussi effectuée dans ce fichier.

Le fichier de « configuration de la sécurité » est seulement utilisé pour le paramétrage des moniteurs de sécurité qui sera détaillé dans le cinquième chapitre du manuscrit.

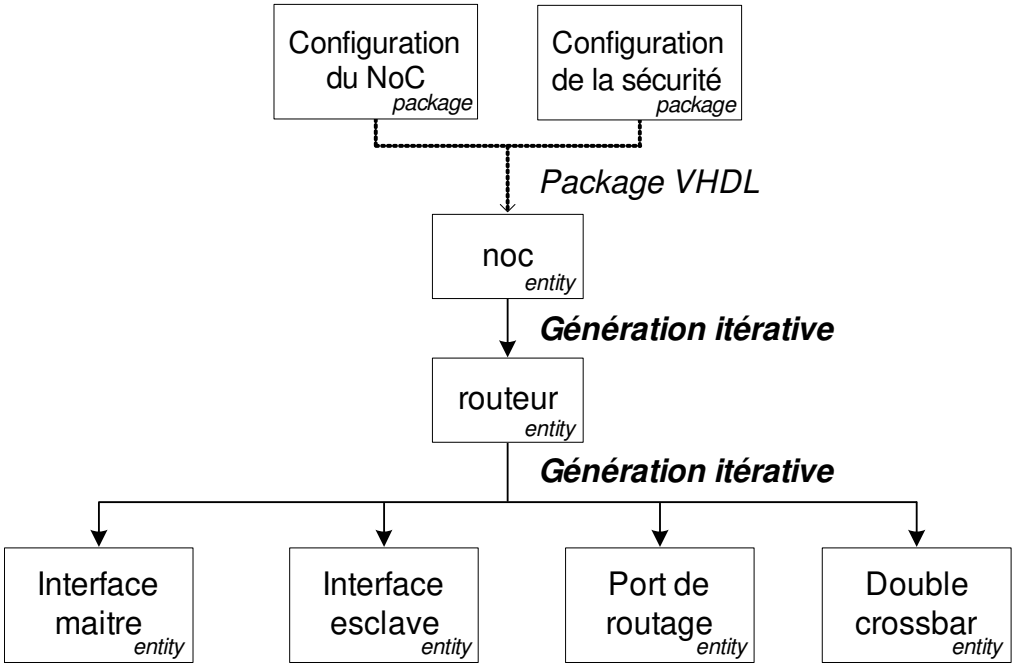


Figure 4.14 : Arborescence des fichiers VHDL de TrustNoC.

Pour n'avoir à modifier que deux fichiers de configuration pour la mise en œuvre de TrustNoC, des boucles de génération conditionnelle sont utilisées dans le code des fichiers VHDL. Celles-ci génèrent le nombre adéquat de routeurs, d'interfaces et de connexions aux niveaux local et global.

https://www.python.org/doc/essays/graphs/La quantité d'informations à saisir lors de la configuration d'un réseau sur puce comportant plusieurs dizaines d'IP peut être très importante.

Afin de faciliter cette étape avec TrustNoC, nous avons développé une application en Python permettant de sélectionner les paramètres du réseau à travers une interface graphique (**Figure 4.15**).

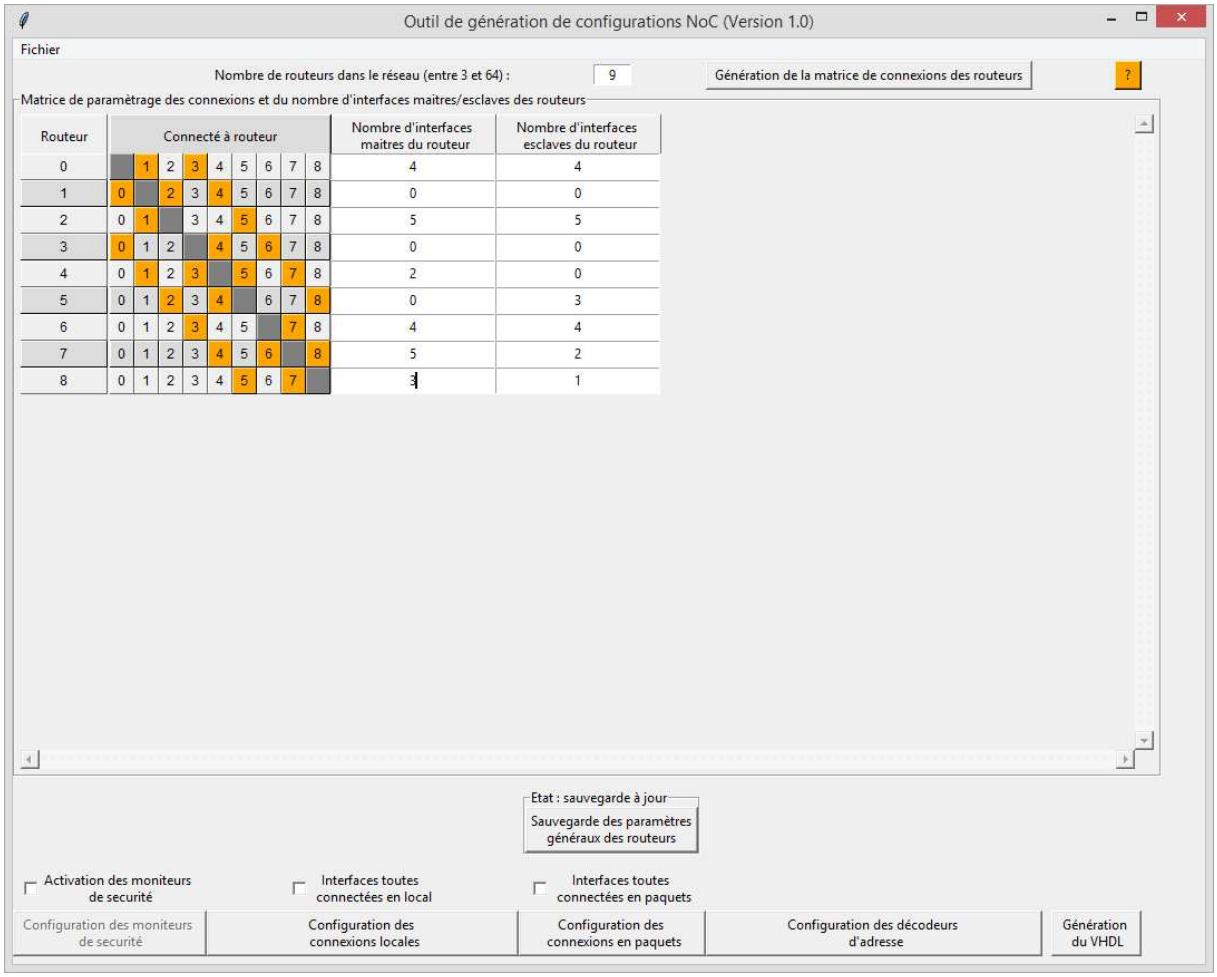


Figure 4.15 : Outil de génération de configuration de TrustNoC codé en langage Python.

A partir des informations renseignées par l'utilisateur, l'outil génère les deux fichiers de configuration en VHDL. En même temps que ces fichiers, un rapport d'erreurs en format texte est aussi généré. Ce dernier signale d'éventuelles incohérences, comme une IP qui ne serait connectée ni au niveau local, ni au niveau global, ou bien encore si les adresses ont été saisies avec un format non reconnu.

Le rapport contient aussi le résultat d'un algorithme de *pathfinding* qui vérifie qu'il existe bien un chemin reliant toutes les IP [PYT].

Nous avons prévu d'intégrer la génération du fichier de configuration de la sécurité, ainsi qu'une représentation graphique du réseau dans une prochaine version de l'outil.

4.5. Résultats des placements-routages

Les placements-routages rendent compte des performances du réseau sur puce une fois programmé dans un FPGA donné. Les résultats qui nous intéressent ici, sont la quantité de ressources logiques occupées et la fréquence de fonctionnement maximale.

L'intérêt est aussi de réaliser des placements-routages pour différentes valeurs de paramètres architecturaux, ceci afin de repérer des tendances et d'identifier les meilleurs choix à faire lors d'une mise en œuvre réelle.

4.5.1. Protocole expérimental

Les placements-routages de TrustNoC ont été réalisés avec les mêmes conditions expérimentales que pour le réseau Hermes, c'est-à-dire, avec la version 15.1.0 du logiciel de conception Quartus, et avec comme cible le FPGA ARRIA 10 10AS066H4F34I3SGES.

Là aussi, pour éviter d'avoir à connecter des broches virtuelles aux entrées-sorties du NoC qui simuleraient la connexion à des broches physiques du FPGA, et qui ne serait pas représentatif d'un cas d'utilisation réel, des IP maîtres et esclaves ont été reliées au NoC pour effectuer les placements-routages. Il est indispensable que ces IP soient de faible complexité pour que le chemin critique soit bien localisé dans le réseau. Lors du calcul des résultats, seule l'occupation en ressource du NoC est comptabilisée.

Nous avons essayé d'étudier les effets de la variation de valeurs de plusieurs paramètres architecturaux, comme le nombre de routeurs, le nombre d'interfaces IP, la topologie, ou bien encore le taux de connexions aux niveaux local et global. Ceci dit, la valeur de certains paramètres architecturaux est identique pour tous les placements-routages (**Tableau 4.4**).

Tableau 4.4 : Paramètres architecturaux invariables lors des placements-routages.

Taille de la mémoire tampon des ports de routage.	**8 mots de 32 bits.**
Taille de la mémoire tampon des paquétiseurs d'acquittements.	**16 mots de 32 bits.**
Plage d'activité des décodeurs d'adresse	**Complète (32 bits)**
Ratio d'IP maîtres/IP esclaves connectées aux routeurs.	**50% (réparti de manière homogène sur tous les routeurs)**

4.5.2. Topologies étudiées et paramètres architecturaux

Nous avons placé-routé trois topologies (**Figure 4.16**) afin d'étudier les effets des paramètres architecturaux sur les performances en occupation de ressources logiques, et en fréquence de fonctionnement.

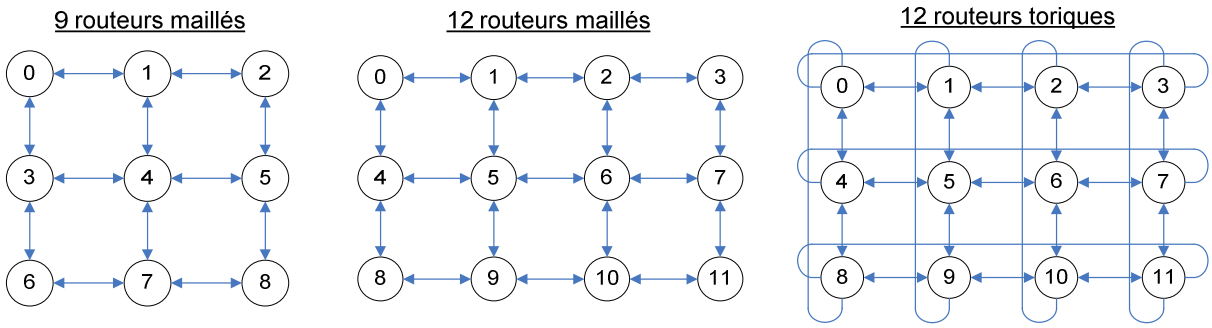


Figure 4.16 : Topologies de TrustNoC utilisées pour les placements-routages.

Afin d'identifier les résultats des placements-routage d'une topologie avec un ensemble de paramètres architecturaux, nous utilisons une nomenclature spécifique qui indique la valeur des différents paramètres (**Figure 4.17**).

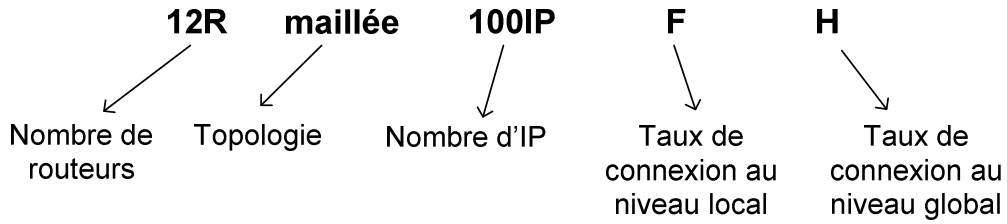


Figure 4.17 : Nomenclature utilisée pour les placements-routages de TrustNoC.

Le taux de connexion des IP aux niveaux **local** et **global** peut avoir deux valeurs, **F** ou **H** :

- **F** *(Full)* lorsque toutes les IP sont connectées aux niveau **local** ou **global.**
- **H** *(Half)* lorsque les IP sont connectées à la moitié des IP au niveau local, et que seule la moitié des IP possède une interface pour le niveau **global**.

L'exploration autour des différents taux de connexions est très importante car ce taux a un impact important sur les résultats des placements-routages.

Le taux moyen de connexion « **H** » est sans doute celui qui se rapproche le plus de la majorité des applications des systèmes sur puce. Car, la plupart du temps il n'est pas nécessaire que l'intégralité des IP partage des connexions entre elles pour qu'une application fonctionne.

4.5.3. Résultats d'occupation de ressources et de fréquence

Les résultats des placements-routages pour les topologies maillées comptant **9 routeurs** sont présentés dans le **Tableau 4.5**, et ceux des topologies à **12 routeurs** et **100 IP** sont présentés dans le **Tableau 4.6**.

Tableau 4.5 : Résultats des p&r pour une topologie maillée comptant 9 routeurs.

	9 routeurs					
	64 IP - maillée		100 IP - maillée			
Taux de connexion aux niveaux : LOCAL – GLOBAL (F = FULL ; H = HALF)	F - F	H - H	F - F	F - H	H - F	H - H
ALM (bloc logique Altera)	31304	19725	48251	32730	44932	29488
% d'occupation du FPGA	12,49%	7,87%	19,26%	13,06%	17,93%	11,77%
Fréquence (MHz)	183	232	191	211	202	216

Dans un premier temps, si on se concentre sur les résultats des topologies **maillées** à **9 routeurs** contenus dans le **Tableau 4.7**, on peut observer que le nombre de ressources consommées est plus important pour toutes les versions du réseau supportant **100 IP** que pour les versions supportant **64 IP**. Il s'agit d'un résultat assez évident, puisque le support d'un plus grand nombre d'IP nécessite l'utilisation d'interfaces supplémentaires, et augmente le nombre d'entrées-sorties des arbitres et des multiplexeurs qui composent les crossbars.

On peut observer que les taux de connexion ont un large impact sur les résultats des placements-routage, puisque pour les topologies **maillées** à **9 routeurs** et **64 IP**, le passage du taux **F–F**, au taux **H–H** réduit les ressources consommées d'environ **37%** et augmente la fréquence de fonctionnement de **26%**.

Concernant les topologies à **9 routeurs maillés** et **100 IP**, nous avons aussi effectué des placements-routages en réduisant seulement le taux de connexion local (**H–F**) ou global (**F–H**). La **réduction du taux de connexion local** engendre moins de gains que **celle du niveau global**, puisque la **consommation de ressources n'est réduite que de 7%** et la **fréquence n'augmente que de 5,7%**. La **diminution du taux de connexion global réduit quant à lui les ressources consommées d'environ 33%** et **augmente la fréquence de 10%**. Il s'agit d'un résultat attendu, puisque les blocs d'interfaces et le crossbar dédiés au niveau global sont plus complexes que ceux dédiés au niveau local. Lorsque les deux taux de connexion sont réduits de moitié **(H-H)**, la diminution de la quantité de **ressources consommées** atteint **38%** et la **fréquence augmente de 11,6%.**

Globalement les mêmes tendances s'observent dans les résultats des placements-routages avec 12 routeurs (Tableau 4.6). Le passage **des taux de connexion maximum (F-F)**, aux **taux de connexion moitié moins importants (H-H)** permet de réaliser une **réduction de la quantité de ressources consommées** de l'ordre de **39%** pour la **topologie maillée** et **33%** pour la **topologie torique**. Le gain est moins important pour la **topologie torique** car les connexions supplémentaires entre les routeurs ajoutent une certaine quantité de ressources non réductible par la diminution des taux de communication aux niveaux **local** et **global**.

Tableau 4.6 : Résultats des p&r pour une topologie comptant 12 routeurs et 100 IP.

	12 routeurs					
	100 IP - maillée				**100 IP - torique**	
Taux de connexion aux niveaux : LOCAL – GLOBAL (F = FULL ; H = HALF)	**F - F**	**F - H**	**H -F**	**H - H**	**F - F**	**H - H**
ALM (bloc logique Altera)	47826	32948	44913	29235	55538	36984
% d'occupation du FPGA	19,09%	13,15%	17,93%	11,67%	22,17%	14,76%
Fréquence (MHz)	216	207	205	217	192	216

Sur le plan de la latence de fonctionnement, nous avons obtenu un résultat plus élevé que celui attendu pour la **topologie maillée à 12 routeurs** et les **taux de connexion maximum (F-F)** puisque sa valeur est de **216 MHz**. Malgré la plus grande complexité de cette version et la consommation accrue en ressources comparé aux versions du réseau avec des taux de connexion inférieurs, grâce à l'hétérogénéité de l'architecture du FPGA, un placement et un routage optimaux et réduisant la taille du chemin critique ont pu être trouvés.

Du fait de la valeur de la fréquence de fonctionnement plus élevée qu'attendue pour les **taux de connexion maximum (F-F)**, le résultat pour le **taux de connexion moyen (H-H)** n'est que très légèrement supérieur, puisqu'il est de **217 MHz**. En ce qui concerne les fréquences des **taux de connexion intermédiaires** (**F-H** et **H-F**), elles sont moins élevées que celle de la version **F-F** de **4,2%** et **4,6%**.

En comparant les résultats de la version à **12 routeurs et 100 IP** avec la **topologie maillée** et la **topologie torique**, on peut observer que le passage d'une topologie à l'autre pour les versions **H-H ne modifie pratiquement pas la fréquence de fonctionnement**, et **augmente la consommation de ressources de 21%**. Un architecte système peut donc faire le choix d'opter pour une **topologie torique** qui **réduit la distance moyenne entre les routeurs**, et donc la **latence moyenne des échanges**, au **détriment d'une augmentation de nombre de ressources consommées**, mais sans **diminuer la fréquence de fonctionnement** du réseau sur puce.

Si on met en parallèle les résultats obtenus pour les versions de TrustNoC comptant **100 IP** pour **9 routeurs** et **12 routeurs**, on peut voir qu'ils sont pratiquement identiques pour les différents taux de connexion. Bien que le nombre de routeurs et donc de ports de routage soit plus important pour les versions avec 12 routeurs, les crossbars comptent moins d'entrées-sorties car le même nombre d'IP est réparti sur un nombre plus important de routeurs. De fait, les arbitres et les multiplexeurs qui les composent sont moins complexes et occupent moins de ressources logiques. Comme on peut le voir dans [YEL11], la surface de ces derniers s'accroit de manière quadratique avec l'augmentation du nombre de leurs entrées-sorties. Du point de vue **de la latence**, il est donc **préférable d'utiliser** une version à **9 routeurs plutôt que 12** puisque les IP sont plus proches les unes des autres, et cela n'entraîne pratiquement aucun changement en ce qui concerne la fréquence de fonctionnement et les ressources occupées.

Tableau 4.7 : Ressources consommées par chaque entité qui compose un routeur.

	Taux de connexion total *(full)*		Taux de connexion moyen *(half)*	
	ALM	% du routeur	ALM	% du routeur
Routeur	4325	100%	2731	100%
Interface IP maître	269	6,2%	30	1%
Interface IP esclave	397	9,2%	55	2%
Port de routage	183	4,2%	180	6,6%
Crossbar local	103	2,4%	42	1,5%
Crossbar global	821	19%	581	21,2%

Le **Tableau 4.7** présente les ressources consommées pour chaque entité qui compose TrustNoC et pour le taux de connexion maximum *(full)* et le taux réduit de moitié *(half)*. Les quantités de ressources occupées par les différentes entités sont données en ALM, autrement dit, en bloc de logiques employés dans les FPGA d'Altera.

Le nombre de ressources consommées par un routeur est tributaire du nombre d'IP et de ports de routage connectés à celui-ci, ainsi que du taux de connexion des IP aux niveaux local et global. Pour illustrer un cas représentatif d'une utilisation réaliste, les données du tableau sont extraites des placements-routages de la topologie maillée à 12 routeurs de TrustNoC (**Figure 4.16**), et pour des taux de connexion maximum et moyen. Les données sont celles du routeur n°5 de cette topologie, qui est connecté à 4 IP maîtres, 4 IP esclaves et 4 ports de routage. Les résultats pour les interfaces et le port de routage ont été obtenus à partir de la moyenne des résultats des 4 instances pour chaque entité. La quantité d'ALM et le pourcentage du routeur occupé est donné pour une seule entité.

Les résultats illustrent bien que le crossbar local occupe peu de ressources par rapport au crossbar global, ceci parce que ce dernier est composé de plus de décodeurs d'adresse, d'arbitres et de multiplexeurs pour l'envoi des paquets de manière bidirectionnelle entre les IP et les ports de routage. Contrairement à l'envoi des requêtes qui a lieu de manière unidirectionnelle entre les IP maîtres et les IP esclaves dans le crossbar local.

Il est également possible d'observer que les interfaces IP maîtres occupent approximativement, entre 33% et 50% de ressources de moins que les interfaces pour les IP esclaves, ceci selon le taux de connexion. Ceci n'est pas un résultat très surprenant, car bien que

les interfaces contiennent toutes deux des blocs logiques de paquétisation et de dépaquétisation, ceux de l'interface pour les IP esclaves sont plus complexes. En effet, la dépaquétisation dans l'interface IP maître consiste juste dans le filtrage du flit d'entête, alors que pour l'interface IP esclave, le paquet est décomposé dans le but de soumettre une requête à l'IP esclave. Le module de paquétisation est lui aussi plus complexe dans l'interface IP esclave, puisque celui-ci contient une mémoire tampon pour le contrôle de flux *store & forward*, alors que ce n'est pas le cas dans l'interface IP maître qui utilise le mécanisme *wormhole*.

En faisant la somme arithmétique des valeurs ou des pourcentages des différentes entités pour le taux de connexion moyen, on n'obtient pas le résultat donné pour le routeur, car pour ce taux de connexion, le routeur contient en réalité une moitié d'interface maître et d'interface esclave, avec un taux de connexion maximum.

4.5.4. Analyse des chemins critiques

Selon les versions des topologies que nous avons explorées durant les placements-routage, les chemins critiques ne sont pas forcément localisés aux mêmes endroits. Cela est dû à l'impact du nombre de routeurs, du nombre d'IP, et des taux de connexion des IP aux niveaux local et global. Le rapport issu de l'outil d'analyse de timing, présente et classe pour chaque placement-routage, les 200 chemins les plus critiques. Dans notre analyse, nous présentons le chemin critique qui revient le plus fréquemment en tête de ces listes.

Ce chemin critique est localisé entre le module d'arbitrage du crossbar global, et la machine à états de réception des paquets du port de routage. Il passe par le multiplexeur et le bus dédiés à l'envoi des données sous forme de paquets (**Figure 4.18**).

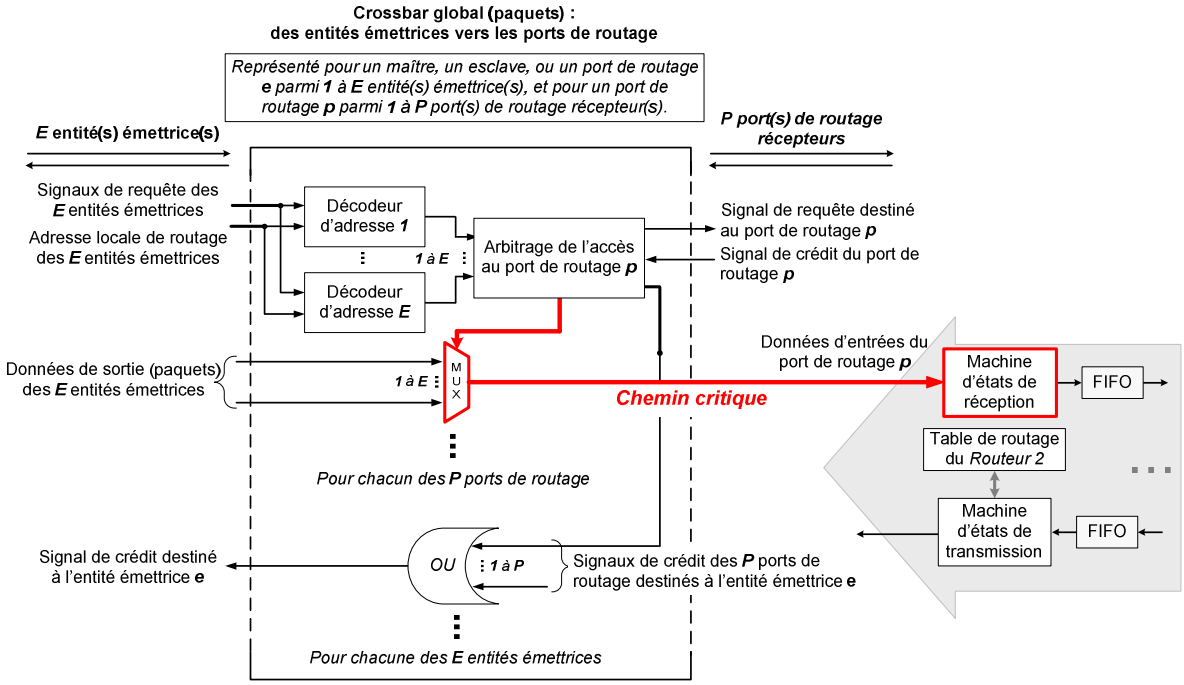


Figure 4.18 : Chemin critique entre le crossbar global et le port de routage.

Bien que le nœud de départ (l'arbitre du crossbar global) soit toujours le même pour ce chemin critique, le nœud d'arrivée est quant à lui, soit le registre qui compte le nombre de flits restant à recevoir pour le paquet en cours de réception, soit le signal qui aiguille les données

vers la bonne case de la mémoire tampon dédiée à la réception les paquets. Ces signaux sont tous deux utilisés dans la machine d'états de réception des flits du port de routage.

Concernant le signal qui compte les flits restant à recevoir pour le paquet en cours de transmission, il est initialisé à partir d'informations contenues dans l'en-tête, plus précisément, le type du paquet et la longueur de la requête qu'il contient. Son rôle est à la fois de fluidifier le contrôle de flux, et de détecter une éventuelle erreur dans la taille du paquet.

Le compteur est initialisé avec une valeur différente selon le type de requête que le paquet transporte. Sa valeur est incrémentée deux fois s'il s'agit d'une écriture, et une seule fois s'il s'agit d'un acquittement (**Figure 4.19**). Le retrait de l'incrémentation enlève la criticité du chemin, cependant, elle est nécessaire pour maintenir le contrôle de la taille du paquet lors de son transport par le port de routage.

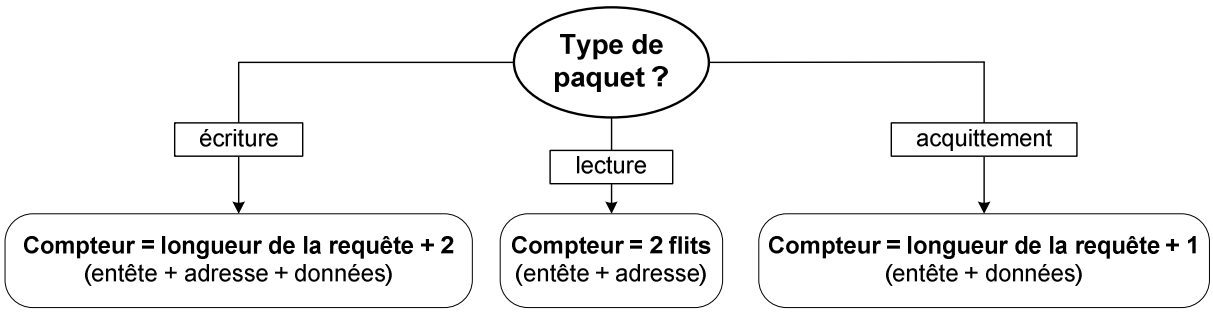


Figure 4.19 : Initialisation du compteur de réception des flits du port de routage.

Le deuxième nœud qui peut terminer le chemin critique est indispensable au fonctionnement du port de routage, et ne peut donc pas être simplifié. Ce signal est incrémenté à chaque fois qu'un flit est reçu par le port de routage, afin d'indiquer à quel endroit de la mémoire tampon doit être stocké le prochain flit reçu.

En vue de raccourcir le chemin critique situé entre l'arbitre du crossbar global et le port de routage, une solution serait d'ajouter un étage de pipeline, et donc un cycle de latence, ceci juste après la sortie du multiplexeur aiguillant le bus de données vers le bon port de routage.

4.6. Comparaison avec les performances d'Hermes

4.6.1. Latence des échanges

Une des raisons qui nous a poussé à développer le réseau TrustNoC pour supporter les applications sur FPGA, est que la latence moyenne de transmission des messages sur Hermes était beaucoup trop importante par rapport à nos contraintes applicatives. Grâce à l'utilisation d'un protocole direct pour les communications locales, ainsi que l'augmentation du nombre d'IP pouvant être connectées à chaque routeur, TrustNoC bénéficie d'une latence moyenne nettement inférieure à celle proposée par notre portage d'Hermes sur FPGA.

Tableau 4.8 : Comparaison des latences entre le portage d'Hermes et TrustNoC.

	Hermes	TrustNoC
Latence minimale de transmission d'une requête d'écriture	15 cycles (pour des routeurs adjacents)	3 cycles (au niveau local, limité à 16 IP au maximum)
Latence minimale de transmission d'une requête de lecture.	14 cycles (pour des routeurs adjacents)	3 cycles (au niveau local, limité à 16 IP au maximum)
Latence minimale de transmission d'une réponse à une requête de lecture (acquittement).	9 cycles (pour des routeurs adjacents)	1 cycle (au niveau local, limité à 16 IP au maximum)
Latence de traversée d'un routeur	5 cycles	1 cycle
Latence de traversée d'un port de routage	Non applicable	- 4 cycles lors de l'arrivée au routeur destination. - 5 cycles sinon.

Le **Tableau 4.8** présente les différences de latences minimales d'échanges entre le portage d'Hermes sur FPGA et le réseau TrustNoC. Les mesures faites pour les deux NoC prennent en compte la traversée des couches d'interface. Comme il s'agit d'interfaces pour le même protocole leur latence est pratiquement identique.

Ce qui explique que la latence du réseau Hermes est beaucoup plus grande, c'est que son routeur emploie une architecture à 5 étages de pipeline. Lors de la transmission d'une requête celle-ci doit traverser au minimum deux routeurs, celui de l'IP expéditrice et celui de l'IP destinataire, ce qui ajoute un délai de 5 cycles de latence pour chacun des deux routeurs. Cela occasionne un délai de 10 cycles d'horloge pour une requête d'écriture, et 20 cycles d'horloge pour une requête de lecture, car on prend en compte le retour du paquet d'acquittement contenant les données lues.

Grâce au niveau de communications local du réseau TrustNoC, les 16 IP au maximum pouvant être connectées au même routeur peuvent se transmettre des requêtes d'écriture ou de lecture en moins de 2 cycles d'horloge. Les données de la requête de lecture issues de l'esclave sont retournées à l'IP initiatrice lors de l'acquittement en un seul cycle d'horloge.

Un avantage du réseau Hermes que l'on peut relever est que le délai de traversée d'un certain nombre de routeurs diminue par rapport à TrustNoC lorsque le nombre de routeurs augmente (**Tableau 4.9**). Cependant, le nombre maximal d'IP pouvant être connectées à un même nombre de routeurs est plus de 10 fois plus important pour TrustNoC (**Tableau 4.10**).

Tableau 4.9 : Latence de traversée de routeurs pour Hermes et TrustNoC.

Nombre de routeurs	2	3	4	5	6	7	...	N
Latence de traversée pour Hermes (cycles)	10	15	20	25	30	35	...	N * 5
Latence de traversée pour TrustNoC (cycles)	6	12	18	24	30	36	...	(N-1) * 6

Tableau 4.10 : Nombre d'IP connectées à une topologie d'Hermes et de TrustNoC.

Topologie	3x3	4x4	5x5	...
Nombre maximal d'IP pouvant être connectées pour Hermes	9	16	25	...
Nombre maximal d'IP pouvant être connectées pour TrustNoC	120	208	308	...

La différence de latence de traversée pour Hermes et TrustNoC (**Tableaux 4.9** et **4.10**) s'explique par le fait que pour TrustNoC, c'est la traversée des ports de routage qui impose de nombreux cycle de latence, alors que pour Hermes il s'agit de la traversée des routeurs.

Une autre manière de mettre en évidence le bénéfice obtenu par la réduction du nombre d'IP pour le réseau TrustNoC par rapport au réseau Hermes est d'étudier le nombre de sauts moyen qui séparent les IP pour des topologies supportant le même nombre d'IP (**Tableau 4.11**). En utilisant des topologies maillées ou toriques pour TrustNoC où le taux d'IP connectées à chaque routeur ne dépasse pas les 60%, ceci afin d'éviter d'agrandir de manière trop importante le nombre d'entrées des crossbars et arbitres internes au routeur, le nombre moyen de sauts séparant toutes les IP est deux fois moins important par rapport à Hermes. De fait, cela se traduit par la réduction de la latence moyenne des échanges.

Tableau 4.11 : Hops moyens pour des topologies de TrustNoC et Hermes (100 IP).

Topologie supportant 100 IP	Nombre moyen de sauts entre les IP (hops)
Hermes - 10x10 maillée	7,7
Hermes - 10x10 torique	6,1
TrustNoC - 9 Routeurs (3x3) maillée	3
TrustNoC – 12 Routeurs (4x3) maillée	3,4
TrustNoC – 12 Routeurs (4x3) torique	2,8

4.6.2. Résultats des placements-routages

La principale motivation qui nous a poussée à développer le réseau TrustNoC au lieu d'utiliser notre portage d'Hermes sur FPGA, était la latence beaucoup trop importante de celui-ci. Ceci-dit, les résultats des placements-routages que nous avons obtenus pour Hermes pouvaient être largement améliorés D'où l'intérêt que nous avons eu à diriger nos recherches bibliographiques vers des implémentations de NoC optimisés pour FPGA, et d'employer les principes de conception que nous avons pu trouver dans le développement de TrustNoC.

Les **Tableaux 4.12** et **4.13** établissent la comparaison des résultats que nous avons présentés pour des topologies maillées d'Hermes et de TrustNoC supportant 64 et 100 IP.

Les résultats d'Hermes sont ceux des placements-routages n'utilisant pas les BRAM et employant des mémoires tampons de taille de 8 flits.

Tableau 4.12 : Résultats des placements-routages d'Hermes et TrustNoC (64 IP).

	Hermes 64 IP	**TrustNoC 64 IP 9 routeurs maillés taux de connexion F – F**	**TrustNoC 64 IP 9 routeurs maillés taux de connexion H – H**
ALM	44237	31304	19725
% d'occupation du FPGA	17,7%	12,49%	7,87%
Fréquence (MHz)	130	183	232
Economie d'ALM par rapport à Hermes		29%	55%
Gain de fréquence par rapport à Hermes		40%	78%

Tableau 4.13 : Résultats des placements-routages d'Hermes et TrustNoC (100 IP).

	Hermes 100 IP	**TrustNoC 100 IP 9 routeurs maillés taux de connexion F – F**	**TrustNoC 100 IP 9 routeurs maillés taux de connexion H – H**
ALM	74575	48251	29488
% d'occupation du FPGA	29,8%	19,26%	11,77%
Fréquence (MHz)	123	191	216
Economie d'ALM par rapport à Hermes		35%	60%
Gain de fréquence par rapport à Hermes		55%	75%

Comme l'illustrent les résultats des comparaisons entre Hermes et TrustNoC, pour un même nombre d'IP et une même topologie, les résultats des placements-routages de TrustNoC sont plus intéressants. **L'économie en ressources logiques** peut aller de **29 à 55% pour 64 IP**, et de **35 à 60% pour 100 IP**. Le **gain de fréquence de fonctionnement** peut s'étendre de **40 à 78% pour 64 IP**, et de **55% à 75% pour 100 IP**.

Là aussi on peut observer que la personnalisation des taux de connexion pour TrustNoC (si l'application le permet) peut apporter un large gain en performances.

4.7. Conclusions et perspectives

Tant sur le plan de la latence que celui des résultats des placements-routages, l'approche beaucoup plus dédiée et optimisée du réseau TrustNoC pour les circuits FPGA, nous a permis d'obtenir de meilleurs résultats que pour notre portage d'Hermes.

A partir de notre état de l'art des NoC pour FPGA, nous avons pu cibler l'importance de la réduction du nombre de mémoires employées dans le réseau, ainsi que de la taille des chemins critiques en ajoutant des étages de pipeline. Nous avons donc mis en pratique ces principes durant la conception de TrustNoC.

Les relatives basses fréquences et les larges consommations de ressources d'Hermes s'expliquent par une architecture beaucoup plus polyvalente et généraliste, ainsi que par la présence d'une large quantité de mémoires tampons.

Pour compléter notre comparaison entre les deux réseaux sur puce, il serait très intéressant de développer une plateforme d'étude de la résistance à la charge. De par l'architecture plus homogène, ainsi que de par le plus grand nombre de mémoires tampons dans Hermes, on peut supposer que ce réseau supporterait des niveaux de charge plus élevés pour des trafics présentant une répartition aléatoire uniforme.

Le nombre d'IP connectées à chaque routeur dans TrustNoC étant beaucoup plus important que celui de la plupart des NoC que nous avons pu trouver dans la littérature, il est aisé de prédire que des phénomènes de goulot d'étranglement apparaissent très rapidement, si toutes les IP tentent d'émettre des paquets à travers la même connexion vers un autre routeur.

TrustNoC est adapté pour supporter des trafics importants entre les IP connectés aux mêmes routeurs, et ceci en offrant de très faibles latences pour les échanges. La personnalisation des connexions dans le réseau est extrêmement importante, et permet de réduire drastiquement le nombre de ressources consommées, ainsi que d'améliorer la fréquence de fonctionnement du réseau.

Plusieurs pistes peuvent être envisagées pour résoudre le problème du goulot d'étranglement rencontré au niveau des ports de routage. Par exemple, en permettant la présence de plusieurs ports de routage entre deux routeurs, ou bien en ajoutant des canaux virtuels à leur architecture.

Le développement d'une plateforme de simulation et d'étude de la résistance à la charge fait partie des travaux que nous pensons réaliser dans le futur. De plus, celle-ci nous permettrait de contrôler la validité d'une configuration de TrustNoC donnée, en simulant l'envoi et la réception de trafic.

Pour l'instant, il n'existe qu'une version de TrustNoC supportant des largeurs de données de 32 bits, cependant nous prévoyons de développer des ports de routage permettant de connecter des routeurs fonctionnant avec des largeurs de bus de données différentes, par exemple 64 ou 128 bits.

Une autre perspective serait d'inclure un système permettant de modifier dynamiquement le contenu des tables de routages en fonction d'informations provenant de l'état du trafic et de la congestion.

Concernant notre outil codé en langage Python permettant de générer des configurations de TrustNoC, afin de faciliter son utilisation, nous prévoyons d'ajouter une représentation graphique du réseau, ainsi que d'ajouter la génération des paramètres de sécurité.

En conclusion, nous pouvons dire qu'en tirant partie des informations trouvées dans la littérature, nous avons réussi à développer le réseau TrustNoC, optimisé pour FPGA et offrant des performances bien plus attractives que le portage d'un réseau sur puce généraliste Hermes, conçu initialement pour circuit ASIC.

Grâce au développement TrustNoC, nous avons atteint notre objectif qui est de pouvoir supporter l'interconnexion d'un nombre important d'IP sur FPGA, ceci tout en garantissant une latence et une occupation de ressources logiques faibles, et une fréquence de fonctionnement supérieure à 150 MHz. Plusieurs pistes vont encore nous permettre d'améliorer ce réseau, et nous comptons les mettre en œuvre prochainement.

Chapitre 5) Moniteurs de sécurité de TrustNoC

Les systèmes sur puce sont soumis à de nombreuses menaces, et l'intégration de mécanismes de sécurité matériels au réseau sur puce est une manière efficace de répondre à certaines d'entre-elles.

Dans l'optique d'ajouter des fonctions de sécurisation au réseau sur puce TrustNoC, nous avons d'abord établi un état de l'art de la sécurité des réseaux sur puce, des SoC, et des mécanismes de sécurité des FPGA.

A partir de cet état de l'art, il nous a été possible de compléter un modèle de menace qui est un prérequis de la conception de mécanismes de sécurité. Enfin, durant le développement nous avons veillé à limiter au maximum l'impact de la sécurité sur les performances du réseau.

Sommaire du Chapitre 5

5.1. Etat de l'art de la sécurité des réseaux sur puce et des FPGA

5.1.1. Propriétés fondamentales de la sécurité

Pour considérer qu'un système est protégé, il faut que les quatre propriétés suivantes soient respectées : la **confidentialité**, **l'intégrité** des données, **l'authentification** et la **non-répudiation** des entités qui participent à l'échange.

5.1.1.1. Confidentialité

Les algorithmes de chiffrement permettent à une entité de protéger la confidentialité des données qu'elle émet ou qu'elle stocke.

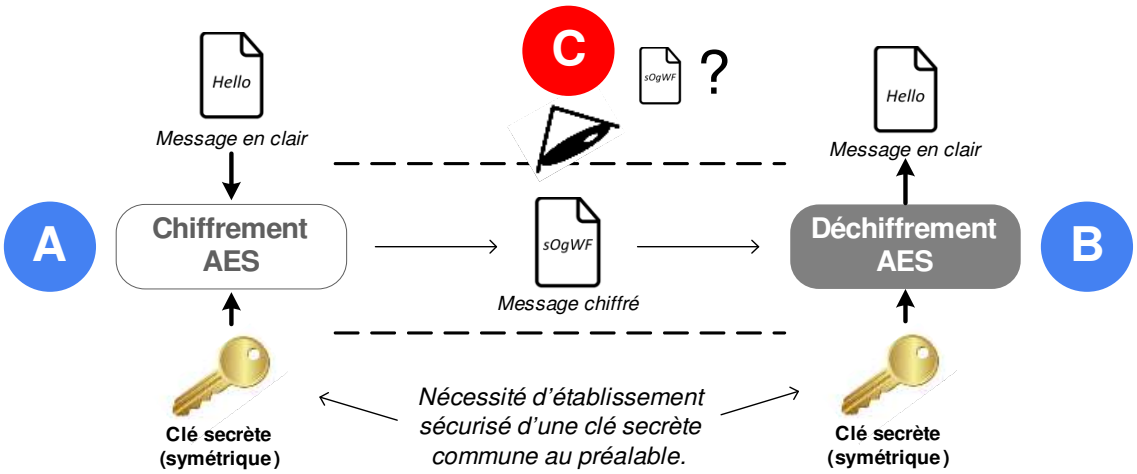


Figure 5.1 : Confidentialité assurée par l'algorithme de chiffrement symétrique AES

L'algorithme est symétrique si la même clé est utilisée pour les opérations de chiffrement et de déchiffrement (**Figure 5.1**).

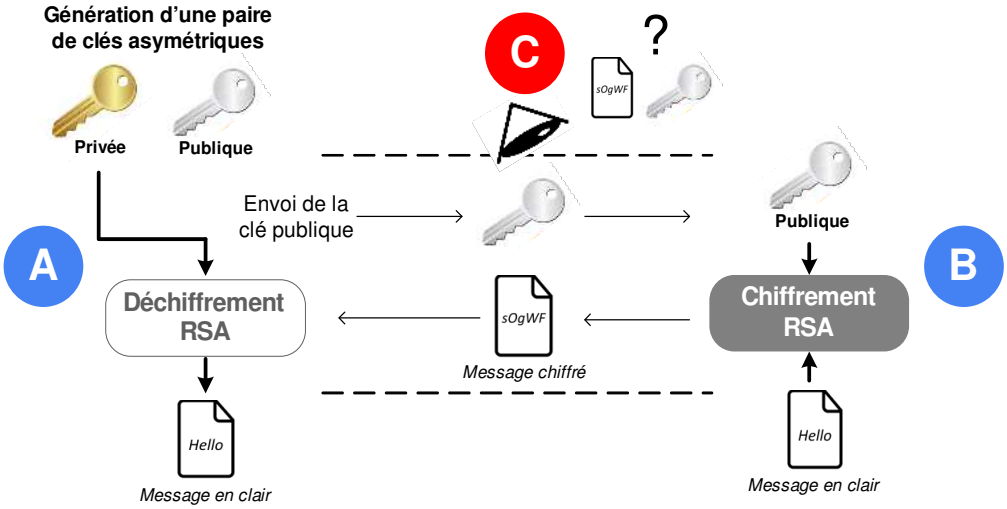


Figure 5.2 : Confidentialité assurée par l'algorithme de chiffrement asymétrique RSA

Un algorithme asymétrique emploie deux clés différentes pour réaliser le chiffrement et le déchiffrement des données (**Figure 5.2**). La clé **privée** doit demeurer secrète contrairement à la clé **publique**. Les informations chiffrées par l'une, peuvent seulement être déchiffrées par l'autre.

5.1.1.2. Intégrité

L'intégrité est la propriété qui indique que la valeur des données n'a pas été modifiée. Pour contrôler l'intégrité des données, les algorithmes de hachage sont communément employés. Ils permettent de générer une empreinte qui est théoriquement non réversible, c'est-à-dire, qu'elle ne permet pas de retrouver la valeur des données hachées. Un des intérêts de cette empreinte est qu'elle est de taille très inférieure à celle des données, ce qui facilite son transport, ainsi que le processus de vérification de l'intégrité. Lors de cette vérification, les données sont de nouveau hachées, puis l'empreinte qui en résulte est comparée au résultat du hachage précédemment réalisé (**Figure 5.3**). Si les deux empreintes ont une valeur différente, cela signifie que les données initiales ont été modifiées.

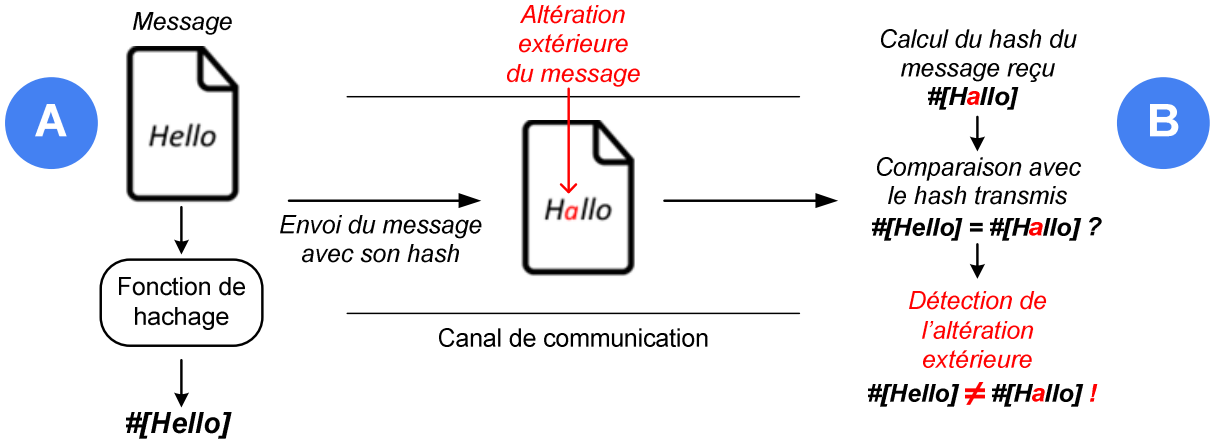


Figure 5.3 : Utilisation d'une fonction de hachage.

Grâce à l'effet d'avalanche qui est notamment présent dans les algorithmes de hachage SHA-256 ou SHA-512, la moindre modification des données en entrée provoque un grand changement de valeur de l'empreinte en sortie. Cela facilite la détection et permet de différencier une modification qui se serait produite dans l'empreinte des données hachées plutôt que dans les données en elles-mêmes.

5.1.1.3. Authentification

L'authentification est une méthode qui permet de s'assurer qu'une entité est bien celle qu'elle prétend être (**Figure 5.4**). L'authentification est indispensable pour empêcher les tentatives d'usurpation d'identité (**Figure 5.5**).

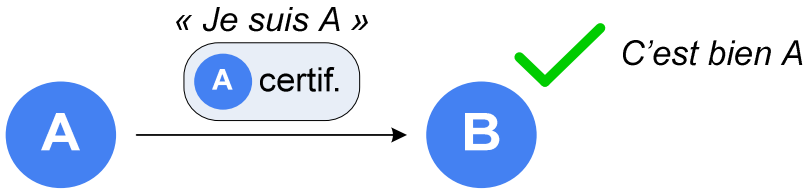


Figure 5.4 : Authentification conforme.

La **Figure 5.5** illustre un cas de tentative d'usurpation de l'identité de l'entité **A** par l'entité **C**, auprès de l'entité **B**. Dans cet exemple, un certificat validé par une autorité par une autorité de certification permet de réaliser l'authentification de l'échange.

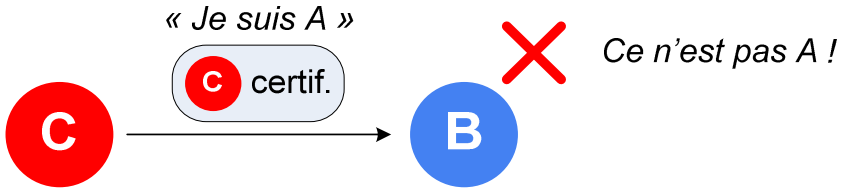


Figure 5.5 : Détection d'une usurpation d'identité par le mécanisme d'authentification.

Dans le domaine de la sécurité numérique, les certificats sont largement utilisés pour jouer le rôle de pièce d'authentification. Il s'agit d'informations ayant été chiffrées par une clé privée et un algorithme asymétrique, et ne pouvant être déchiffrées que par la clé publique. L'efficacité des certificats requiert néanmoins, qu'une autorité de certification ait préalablement validé et enregistré la clé publique de l'entité devant être authentifiée. Un autre exemple de mécanisme d'authentification très utilisé, c'est le mot de passe personnel. Dans un cas idéal, celui-ci n'est connu que de son utilisateur et permet de l'authentifier.

5.1.1.4. Non-répudiation

Cette dernière propriété signifie que l'émetteur et le récepteur d'une communication ne peuvent nier avoir respectivement participé à l'envoi et à la réception de cette communication. La non-répudiation est généralement assurée par l'utilisation des certificats.

5.1.2. Etat de l'art de la sécurité des NoC et des MPSoC

L'état de l'art de la sécurisation des réseaux sur puce et des systèmes multiprocesseurs répond à divers modèles de menace qui concernent les systèmes sur puce. Une partie des travaux sélectionnée traite des réseaux sur puce malveillants qui peuvent être la cause de fuites de données ou d'injections de paquets, et qui visent à réduire les performances du système. Une autre partie des articles s'intéresse aux attaques par canaux cachés en timing qui utilisent le réseau sur puce comme vecteur d'attaques. Enfin, le reste des publications propose des moyens de sécuriser le système sur puce contre les effets du détournement d'IP, soit par une approche issue de la couche logicielle, soit en reposant sur le niveau matériel.

5.1.2.1. Réseaux sur puce malveillants : fuite de données et injection de paquets

Parmi les articles qui traitent de la sécurité des systèmes sur puce, il est déjà possible de s'intéresser à ceux qui considèrent le NoC comme pouvant être potentiellement malveillant. Cette problématique se pose surtout lorsque dans le but de réduire le temps de mise sur le marché d'un produit, les concepteurs font le choix d'avoir recours à des vendeurs tiers comme [ART] et [SON], pour obtenir des outils de configuration et de génération de NoC. A notre connaissance, aucune de ces deux sociétés n'est connue pour avoir déjà fourni une solution vérolée. Cependant, d'autres sociétés fournissent ce genre de service, et comme à chaque fois qu'un acteur extérieur intervient dans le flot de conception et de fabrication d'un produit, les questions de la confiance et de la sécurité rentrent en jeu.

Que ce soit délibéré ou non, une IP achetée à un vendeur tiers peut contenir des parties malveillantes. Le diagnostic du code source dans le but de détecter un éventuel cheval de Troie est une solution envisageable. Cependant, il peut être plus simple et moins coûteux d'ajouter en prévention, un étage de mécanismes dédiés à la sécurisation d'un NoC potentiellement malveillant. En effet, le code source d'une IP peut être chiffré et non observable, dans le but de protéger la propriété intellectuelle du vendeur. De plus, lors des tests, il peut être pratiquement impossible de trouver la séquence qui a pour effet de déclencher l'attaque d'un cheval de Troie.

Les travaux [ANC15] et [RAJ15] ont justement pour objet les NoC potentiellement malveillants. Dans [ANC15], plusieurs solutions sont proposées pour réduire l'efficacité d'un NoC créant des **fuites de données**. **Première solution**, un mécanisme léger de brouillage des données (ou *Data Scrambling*), basé sur l'opérateur *ou exclusif* XOR rend les données transportées sur le réseau plus difficilement compréhensibles et limite le déclenchement du processus créant des fuites. **Deuxième solution**, lors de l'émission, la couche firmware placée entre le routeur et l'IP ajoute en queue du paquet un tag d'identification de la destination. Lors de la réception, si le tag ne correspond pas à celui de l'IP réceptrice, cela signifie que le paquet n'est pas arrivé à la bonne destination, et donc qu'il s'agit soit d'une erreur, soit d'une fuite. Dans ce cas, le paquet est détruit et une interruption est déclenchée pour alerter le superviseur du système. **La troisième et dernière solution**, est essentiellement réalisable pour les systèmes composés de processeurs. Elle consiste à migrer et à interchanger périodiquement la place des processeurs dans le NoC (*Node Obfuscation*). Le but est de mélanger et de rendre incohérentes du point de vue du réseau, les communications entre deux nœuds. L'opération de migration d'un processeur, nécessite l'enregistrement et le déplacement de tous ses registres d'exécution. Pour réaliser la troisième solution sur FPGA, il serait envisageable d'utiliser la reconfiguration dynamique partielle.

La solution partagée dans la publication [SEP15b] répond également à la menace d'un NoC malicieux ayant la capacité de créer des **fuites de données**. Ce travail propose de créer des zones sécurisées grâce à une implémentation légère de l'algorithme Diffie-Hellman qui permet la création d'un secret partagé en utilisant un medium de communication ouvert et observable. Un des avantages de l'utilisation de cet algorithme, est notamment le fait que les zones sécurisées n'ont pas la nécessité d'être continues, c'est-à-dire, que toutes IP appartenant à la même zone sécurisée n'ont pas besoin d'être côte à côte.

Dans [SEP15b] c'est une tâche logicielle exécutée sur processeur sécurisé qui a pour rôle de gérer la mise en place des secrets partagés. Pour éviter qu'une IP malveillante puisse demander de faire partie d'une zone sécurisée, on peut imaginer que seul un ensemble d'IP de confiance ait l'autorisation de demander d'établir des secrets partagés, puis durant l'exécution, certaines de ces IP peuvent demander de former des groupes selon les besoins. Un mécanisme d'authentification est nécessaire dans le but de s'assurer qu'aucune usurpation d'identité ne se produise. Il est également important de préciser que cette proposition demeure vulnérable aux attaques par déni de service.

L'algorithme Diffie-Hellman répond parfaitement à la problématique des fuites de données produites par un NoC malveillant, cependant, comme tous les algorithmes de cryptographies asymétriques, il est vulnérable aux attaques de l'homme du milieu *(Man-in-the-middle)*. Cette attaque peut être complexe à mettre en place, sachant que les paquets contenant les informations nécessaires à l'établissement du secret commun, peuvent prendre de multiples chemins pour se rendre à leur destination. Cependant, si une entité est capable d'intercepter ces différents paquets, de les remplacer par ses propres paquets, et qu'elle possède « la couleur commune » à tous les mélanges, elle est théoriquement capable de déchiffrer tous les échanges, sans qu'aucune des IP de la zone sécurisée ne puisse s'en rendre compte. Pour contrer les attaques de l'homme du milieu, l'utilisation des certificats est une solution envisageable.

Le risque avec l'utilisation d'un NoC malveillant ne se limite pas seulement à la fuite de données sensibles, il peut aussi délibérément ralentir du réseau en injectant des paquets parasites ou en ajoutant des latences inutiles. Le travail [RAJ15], propose un protocole de détection de routeurs ajoutant des cycles de latence inutiles dans les communications, grâce à l'utilisation des paquets « sondes » horodatés, et capables de détecter des comportements irréguliers.

5.1.2.1. Protection contre les attaques en timing (canaux cachés)

Un réseau sur puce peut être le vecteur d'attaques par canaux cachés. Le but de celles-ci est de réduire le nombre d'hypothèses de la valeur d'une clé cryptographique, jusqu'à pouvoir trouver sa véritable valeur. Les termes « canaux cachés » fait référence aux informations qui fuitent d'un système malgré lui, et qui peuvent révéler des informations censées demeurer secrètes. Pour un circuit électronique, ces canaux cachés peuvent se trouver dans la consommation électrique, ou bien dans le rayonnement électromagnétique ou thermique.

Pour un NoC, un de ces canaux cachés réside dans la comparaison des instants où un processeur qui exécute un algorithme AES, consulte sa mémoire cache (**Figure 5.6**). En effet, ces accès au cache dépendent de la valeur de la clé. L'article [REI16] a d'ailleurs prouvé la faisabilité de cette attaque. Selon la latence que met le processus malveillant à accéder au cache, il peut en déduire le moment où l'IP attaquée réalise ses accès. La solution proposée dans [REI16] consiste en un mécanisme capable de détecter les perturbations de bande-passante causées par l'attaque, et de modifier l'algorithme de routage pour fausser ses résultats. Dans [SEP15a], de l'aléa est ajouté à l'arbitrage des paquets dans le même but.

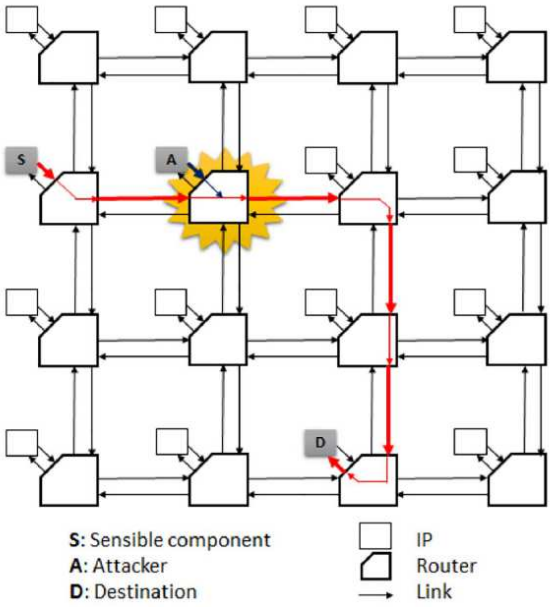


Figure 5.6 : Représentation de l'attaque en timing (extrait de [SEP15a]).

L'article [FER16] propose de lutter contre les attaques en timing, en utilisant un algorithme de routage favorisant le passage des paquets par des routeurs où sont connectées des IP considérées comme sûres ou appartenant à des zones sécurisées. Cela empêche qu'une IP souhaitant réaliser une attaque en timing se situe sur le chemin du paquet, ce qui est un prérequis indispensable pour la réussite de cette attaque.

Enfin, [SURFNOC13] utilise le multiplexage temporel pour chaque message ait une fenêtre réservée et ne puisse interférer avec celles des autres.

5.1.2.2. Détournement d'IP : sécurisation par le niveau logiciel

Les malwares (virus, vers, cheval de Troie, etc.) peuvent être considérés comme l'origine la plus commune du détournement des IP. De plus, il est admis que la majorité des attaques qui ciblent les systèmes sur puce sont d'origines logicielles. En effet, les attaques visant le matériel nécessitent la plupart du temps un accès physique au circuit, ce qui est le cas par exemple, des attaques par analyses différentielles de la consommation *(DPA)* ou les attaques en fautes. L'avantage des attaques par la voie logicielle, c'est qu'elles peuvent être effectuées à distance, et qu'elles peuvent exploiter une faille d'une des nombreuses couches logicielles qui composent les systèmes actuels. Elles peuvent aussi exploiter la naïveté d'un utilisateur, qui va négligemment ouvrir les portes de son système à un cheval de Troie mal intentionné.

En partant de ce constat, il peut être pertinent d'étudier la manière dont est traitée la question de la sécurité d'un système sur puce multiprocesseurs du point de vue logiciel. C'est notamment l'objet des travaux des chercheurs Marcello Coppola, Miltos Grammatikakis & Al., dans les publications [COP13] et [GRA14]. L'article [COP13] décrit la manière dont la sécurité est appréhendée pour une plateforme de *cloud computing*. Dans des systèmes multi-applications, pour éviter qu'un processus ou une application ne puisse gêner l'exécution d'une autre, ou accéder à ses données, des domaines d'exécution et une unité de protection de la mémoire sont utilisés. Un superviseur a pour objectif de gérer la virtualisation des plusieurs domaines.

Grâce à cette virtualisation, l'adressage logiciel des mémoires est totalement décorrélé de l'adressage physique du circuit. Cela permet de créer des sortes de « bulles » d'exécution, où chacune des applications n'a aucun moyen d'interagir avec les données d'une autre, si elle n'y a pas été au préalablement autorisée. Par exemple, lors de l'installation d'une application sur smartphone, l'utilisateur doit explicitement lui autoriser l'accès aux données de différentes natures : contacts, photos, vidéos, informations bancaires, etc... Toute la sécurité du système repose en réalité sur la protection du superviseur qui gère les différents domaines d'exécution.

Cependant, à l'intérieur des systèmes sur puce multiprocesseurs, la virtualisation n'est pas supportée par tous les composants. Par exemple, la plupart des unités d'accès à la mémoire (DMA) ne sont pas capables de différencier les domaines d'exécution. Afin d'étendre la sécurité de la virtualisation, les travaux [POR11], [COP13] et [GRA14] proposent l'utilisation d'une couche de sécurisation matérielle composée de pare-feux et associée à un NoC. Chaque paquet émit sur le réseau est marqué grâce à un *tag* qui précise à quel domaine d'exécution, celui-ci appartient. Les pare-feux contrôlent ensuite lors d'un accès mémoire, si le tag correspond à un domaine d'exécution dont l'accès est autorisé. Comme nous allons le voir dans la prochaine partie, il ne s'agit que d'un exemple de mécanismes de sécurité ajoutés au niveau matériel.

5.1.2.3. Détournement d'IP : sécurisation par le niveau matériel

Dans la précédente partie, nous avons brièvement introduit la notion de sécurisation des systèmes sur puce multiprocesseurs du point de vue logiciel, grâce à la virtualisation. Cependant, cette méthode n'est pas suffisante pour sécuriser tous les composants d'un système sur puce, car certains d'entre eux sont pas capables de la gérer. C'est la raison pour laquelle les travaux [POR11], [COP13] et [GRA14] se sont portés sur l'utilisation d'un réseau sur puce associé à des pare-feux.

Il est important de noter que la majorité des attaques logicielles, prenant la forme de virus, de vers et de chevaux de Troie, se résume à des accès non autorisés à des espaces mémoires, et qu'ils peuvent être contrés grâce à la délimitation de zones spécifiques d'exécution et à des pare-feux.

Les contrôles effectués par les mécanismes de sécurité des réseaux sur puce sont en réalité relativement simples, ils servent à s'assurer qu'une entité puisse lire et écrire seulement les données dont elle a besoin pour fonctionner. L'enjeu de la conception de ces mécanismes repose sur le fait de les rendre suffisamment flexibles, avec des surcoûts en ressources et en performances minimaux, et d'une granularité assez faible pour, par exemple, faire une différenciation entre les tâches d'une même IP. Il est également capital que le composant en charge de la supervision de la sécurité, celui qui reçoit les alarmes et dirige les contre-mesures, soit isolé et protégé des attaques.

a. SECA : Security Enhancement Communication Architecture (2005)

[COB05] est l'un des premiers travaux majeurs qui traite des pare-feux pour les systèmes d'interconnexions des systèmes sur puce. Le système d'interconnexions employé dans l'article est le bus AMBA-AHB du vendeur ARM.

La plupart des contrôles effectués par les différents mécanismes de sécurité présentés dans cette publication sont identiques à ceux proposés à l'heure actuelle. Trois genres de pare-feux sont proposés, se basant chacun sur un type d'informations différent : APU *(Address Protection Unit)* pour les adresses mémoire, DPU *(Data Protection Unit)* pour les données, et SPU *(Sequence Protection Unit)* pour les séquences de données. Seules les unités DPU sont distribuées au niveau des interfaces entre le système d'interconnexions et les IP. Le reste des informations est envoyé depuis les interfaces réseau vers un composant central de sécurité SEM *(Security Enforcement Module)* qui réalise les contrôles des unités APU et SPU. Ce composant central est lui-même programmé par un processus exécuté sur une plateforme de confiance appelée TCB *(Trusted Base Computing)*.

L'unité DPU vérifie que les valeurs écrites dans un registre donné sont comprises dans une certaine plage autorisée. Pour chaque requête soumise, le contexte d'exécution du maître doit correspondre à celui de la DPU. Lors d'une requête, ce contexte est transmis depuis le maître par des signaux dédiés du bus AMBA-AHB.

L'APU réalise les contrôles que l'on retrouve dans la très grande majorité des pare-feux proposés dans la littérature. C'est-à-dire, que le maître qui a soumis la requête a bien accès en lecture ou en écriture à la plage d'adresses mémoire de l'esclave. De plus, pour éviter les attaques par surcharge de buffer, la largeur de la requête est vérifiée afin qu'elle n'empiète pas sur une plage d'adresses mémoire à laquelle le maître n'aurait pas accès.

La dernière unité de contrôle SPU a pour rôle de détecter d'éventuelles séquences de requêtes correspondant à des signatures d'applications malveillantes.

Suite à la détection d'un comportement irrégulier et à la soumission d'une alerte au SEM, les auteurs avaient pour idée d'investiguer comme contre-mesure la possibilité de blocage d'accès d'une IP malveillante au bus AMBA.

Un des principaux désavantages de la solution [COB05] est d'employer une solution partiellement centralisée pour les contrôles des APU et SPU, qui est moins demandeuse en ressources que la solution distribuée, mais peut engendrer de larges latences pour un réseau comptant trop d'IP.

b. Zones sécurisées graduelles (2005)

Dans la publication [EVA05], les auteurs proposent de disposer les pare-feux de façon à créer des zones sécurisées dans le NoC. Ces pare-feux sont uniquement positionnés au niveau des liens qui séparent les IP considérées comme de confiance, et les IP non-sûres. Grâce à cette approche, les requêtes sont seulement contrôlées lorsqu'elles tentent d'accéder à une zone ayant un niveau de sécurité plus élevé, ce qui limite les contrôles inutiles ainsi que le nombre de pare-feux. Plusieurs niveaux de zones sécurisées sont proposés, avec des contrôles plus ou moins poussés.

La gradualité croissante des niveaux de sécurité proposée pour les zones sécurisées est la suivante : 1) Le maître est autorisé à accéder à la plage mémoire de l'esclave. 2) Le chemin du paquet entrant dans la zone ne mène pas à un deadlock ou un livelock. 3) Le maître émetteur de la requête est authentifié grâce à l'utilisation d'un algorithme de routage spécifique permettant le *backtracking* du paquet, c'est-à-dire, de retracer son parcours jusqu'ici.

En plus des accès non autorisés, le modèle de menace de cette publication comprend les attaques par dénis de service utilisant l'émission de paquets menant à des deadlocks ou des livelocks. Il prend aussi en compte l'émission de paquets intempestifs qui réduisent la bande-passante et augmentent la consommation de puissance. La contre-mesure proposée est de placer des compteurs mesurant le taux d'émission des IP et de le limiter en cas de comportements abusifs.

c. Mécanismes de sécurité distribués (2007)

La publication [DIG07] fait en quelque sorte la synthèse des deux articles précédents. Il n'est plus question des zones sécurisées comme dans [EVA05], cependant il utilise le même algorithme de *backtracking* des paquets dans le but d'authentifier le maître responsable de l'émission.

Les pare-feux ou SNI *(Secure Network Interface)* sont placées aux interfaces des IP comme dans [COB05], sauf que cette-fois, tous les contrôles sont distribués aux interfaces. Le composant central de sécurité *(SCM ou Security Configuration Manager)*, est seulement sollicité par les SNI lors des alertes de sécurité afin qu'il déclenche les contre-mesures. Afin de configurer les SNI, le SCM envoie les polices de sécurité par un canal virtuel dédié.

d. Contrôles parallèles des pare-feux (2008)

L'article [FIO08] reprend globalement le même modèle de menace et la même architecture pour les mécanismes de sécurité que [DIG07], cependant il n'utilise pas l'algorithme de *backtracking* des paquets.

Une des principales différences avec le travail précédent est que les pare-feux peuvent être placés soit à l'émission, soit à la réception. A l'émission, cela permet de bloquer un paquet malveillant avant qu'il n'entre sur le réseau et consomme inutilement de la bande-passante. Dans ce cas, par contre, les contrôles sont effectués même si la requête est destinée à une IP pour laquelle ce n'est pas nécessaire.

Au lieu uniquement de se baser sur le type de la requête : **lecture** ou **écriture**, les droits d'accès prennent aussi en compte le fait que la requête cible, soit des **données**, soit une **instruction**, ainsi que le rôle de l'entité responsable de la requête : **utilisateur** ou **superviseur**.

Un mécanisme sécurisé de modification dynamique des droits d'accès est proposé dans cet article. Seule l'IP à laquelle correspondent les droits d'accès peut demander au composant de sécurité centralisé NSM *(Network Security Manager)* l'ajout ou la suppression d'une police de sécurité. Pour éviter qu'une IP malveillante usurpe l'identité d'une autre pour modifier les droits d'accès de cette dernière, lorsque le NSM reçoit une requête reconfiguration, il transmet un paquet de demande de confirmation qui ne peut arriver qu'à l'IP concernée, si le NoC est de confiance. Cette demande de confirmation contient un numéro de transaction qui doit lui être renvoyé, ce qui réduit drastiquement les chances de réussite d'une usurpation d'identité, dans le cas où l'IP malveillante anticiperait l'envoi du paquet de confirmation.

e. NoC Hiérarchique sécurisé (2012)

L'article [SEP12] présente un NoC hiérarchique sécurisé. Différents groupes d'IP appelés *clusters* sont dans un premier temps interconnectés à travers des *Low-NoC* de faible hiérarchie. Puis, tous les *Low-NoC* sont connectés entre eux par l'intermédiaire d'un NoC de plus haute hiérarchie : le *High-NoC*. L'intérêt de cette solution est de pouvoir utiliser des polices de sécurité différentes plus ou moins strictes selon les *clusters*. L'accès au *High-NoC* et aux autres *Low-NoC* requiert le passage de contrôles de sécurité supplémentaires. Une des faiblesses de cette solution réside dans les goulots d'étranglement des performances formés par les connexions entre les *Low-NoC* et le *High-NoC*. Ce type d'architecture favorise les communications au sein des *Low-NoC*.

Les polices de sécurité sont organisées en deux types : « droits d'accès » et « authentification », et en des niveaux pouvant se cumuler. A chaque niveau correspond une certaine police de sécurité. Si le plus haut niveau de sécurité est « actif » pour un pare-feu donné, alors les polices de sécurité de chaque niveau seront appliquées lors du contrôle d'un paquet.

Voici les niveaux qui sont proposés dans l'article. Droit d'accès : 1erniveau) L'accès complet d'une IP à une autre est autorisé ou refusé. 2ème niveau) La taille de la requête et de la zone mémoire ciblée sont vérifiées. 3ème niveau) Le rôle de l'IP initiatrice est pris en compte. Authentification : 1er niveau) L'adresse de l'IP source est utilisée pour le contrôle. 2ème niveau) L'algorithme de *backtracking* qui retrace le parcours du paquet à travers les routeurs est employé, afin d'authentifier l'IP source. 3ème niveau) Le N° de transaction entre chaque paire d'IP est ajouté dans le contrôle.

Pour gérer la reconfiguration des polices de sécurité, le composant *Policy Keeper* qui contient toutes les règles de chaque tâche, est associé avec le *Configuration Control* qui a pour rôle de distribuer les règles aux pare-feux. Les auteurs préconisent l'utilisation d'un programme exécuté par un processeur sécurisé, dans le but d'envoyer les demandes de reconfiguration au composant *Configuration Control*. Afin de faciliter et d'optimiser le processus de modification des polices de sécurité d'un pare-feu, une commande d'interruption des communications destinées à l'IP à laquelle est connecté le pare-feu est transmis à tout le réseau. De plus, la priorité des paquets destinés à cette IP est augmentée, s'ils ne l'ont pas encore atteinte. Une fois que tous les paquets ont été reçus, et que la reconfiguration a eu lieu, le trafic reprend normalement.

f. Protection du système d'interconnexion et de la mémoire externe (2012)

En plus de porter sur la sécurité des systèmes sur puce multiprocesseurs, la thèse de Pascal Cotret [COT12] porte sur la sécurisation du canal de communication avec la mémoire externe. En effet, il constate que ce canal est vulnérable à différents types d'attaque, et qu'il pouvait potentiellement donner un accès au réseau interne du système sur puce. Dans le but de protéger la mémoire externe, plusieurs mécanismes sont ajoutés au pare-feu du canal, comme un horodateur qui date les requêtes pour éviter les attaques de type *rejeu*, consistant en la copie d'un message transmis sur le canal, et en sa réémission plus tard. Des algorithmes de chiffrement, d'authentification et d'intégrité protègent les accès aux zones critiques de la mémoire. Il important de noter que l'utilisation de ces algorithmes sur la totalité de la mémoire, risque d'engendrer un surcoût important.

Une des originalités de cette thèse à propos des pare-feux, réside dans le traitement des contre-mesures. Les pare-feux placés au niveau des IP émettrices, possèdent des compteurs qui enregistrent le nombre de violations de sécurité durant une certaine période de temps. Si ce nombre dépasse un premier seuil, l'IP passe en mode « lecture seule » afin que ce son contenu puisse être enregistré. Ensuite, si le nombre de violations continue d'augmenter et dépasse un deuxième seuil, l'IP est totalement isolée du réseau. Le niveau d'isolement peut être dégressif, et les restrictions d'accès peuvent être levées si l'IP cesse de transmettre des requêtes malveillantes. L'utilisation de cette fonctionnalité est adaptée à un processeur généraliste après qu'il ait stoppé une attaque logicielle.

Le composant qui configure les règles de sécurité et qui définit la mise en quarantaine des IP, est isolé du reste du système et communique avec les pare-feux par un canal dédié.

g. Sécurisation d'un réseau sur puce tridimensionnelle (2014)

Ce travail [SEP14] porte sur les puces 3D composées de multiples couches de circuits 2D, connectés entres eux par des *Through-Silicon-Vias* ou *TSV*, c'est-à-dire, des fils traversants verticaux. Chaque circuit possède un NoC à deux dimensions et les TSV servent à les interconnecter. Cet article propose principalement des méthodes de sécurisation des TSV, qui avec l'effet de la diaphonie *(crosstalk)* peuvent être détournés pour espionner ou modifier les données transportées sur un TSV voisin. Le multiplexage temporel est utilisé pour contrer cette menace et empêcher les communications simultanées sur des TSV adjacents.

Cependant, puisque les travaux de notre thèse ne ciblent pas les puces à trois dimensions, l'intérêt de cet article pour nous réside seulement dans la sécurisation des NoC utilisés dans les circuits à deux dimensions. Les contrôles des pare-feux sont très similaires à ceux de [SEP12] avec plusieurs niveaux de sécurité, ils sont par ailleurs désactivables et ré-activables.

Une des originalités est le mécanisme de cache employé dans le but de limiter la taille des tables dans les pare-feux. Chaque pare-feu contient un nombre limité de règles de sécurité dans sa table, si aucune ne correspond lors du contrôle d'une requête, il envoie une demande à la table centralisée qui contient toutes les règles. Dans le cas où la table centralisée contient une règle autorisant l'accès à la requête, cette règle remplace dans le pare-feu celle qui a été consultée depuis le plus longtemps. Dans le cas contraire, une alerte de violation d'accès est déclenchée. Un canal dédié est utilisé pour les échanges entre les pare-feux et la table centrale.

h. Reconfiguration des pare-feux via un canal hamiltonien (2015)

Dans [FER15], le modèle de menace est similaire aux travaux précédents, avec le détournement d'une IP par une attaque logicielle, qui peut effectuer des accès non autorisés dans le but d'extraire des informations confidentielles, modifier des données, ou bloquer le système. Cependant, le canal de reconfiguration des pare-feux de cet article, utilise une architecture légère permettant de limiter la surface du circuit ou la consommation de ressources logiques.

Le canal de reconfiguration dans [FER15] relie les pare-feux en série et fonctionne avec un protocole très simple. Le composant dédié qui gère la reconfiguration des règles de sécurité des pare-feux, est connecté tout en amont du canal. Lorsqu'il envoie une nouvelle règle de sécurité, il précise l'adresse du pare-feu ciblé. La règle est transmise d'un pare-feu à un autre en un cycle de latence. Si l'adresse de destination du paquet contenant la règle de sécurité correspond à celle du pare-feu courant, ce dernier interrompt la transmission. Dans le cas contraire, il transmet le paquet de reconfiguration au routeur suivant.

L'avantage de cette méthode de reconfiguration est d'utiliser peu de ressources, que ce soit pour le canal de configuration ou pour le protocole. Le surcoût en latence peut être important selon le nombre d'IP. C'est la raison pour laquelle pour TrustNoC, certains pare-feux peuvent être statiques et ne font pas partie du canal de reconfiguration. De plus, des règles de sécurité sont définies par défaut dans chaque pare-feu, donc le canal n'est pas utilisé pour initialiser tous les pare-feux au démarrage de l'application.

5.1.3. Etude des mécanismes de sécurité des FPGA modernes

Dans le but de mieux comprendre les menaces qui ciblent les circuits reconfigurables, et pouvoir comparer les forces et les faiblesses des FPGA de chaque vendeur, nous avons porté notre attention sur les mécanismes de sécurité disponibles dans des circuits FPGA des vendeurs Altera (Intel FPGA), Microsemi et Xilinx. Cette étude porte sur les modèles Cyclone III LS d'Altera [ALT12b], IGLOO2 de Microsemi [MIC16] et la 7ème série de FPGA de Xilinx [XIL13], car ils s'agissaient pour chaque vendeur, des FPGA les plus sécurisés au moment où nous avons réalisé cette étude.

Altera et Xilinx se partagent la majorité du marché des ventes des circuits FPGA. Microsemi est quant à elle la société qui semble proposer les FPGA contenant le plus de mécanismes de sécurité. Une partie des mécanismes de sécurité des FPGA des différents vendeurs, est présenté dans le **Tableau 5.1**.

Lorsqu'on recense tous les mécanismes de sécurité intégrés dans les FPGA actuels, on se rend compte du très grand nombre de menaces auxquelles sont confrontés ces circuits. De plus, ce nombre peut être augmenté avec la prise en compte des chevaux de Troie, ou des portes dérobées qui peuvent être ajoutées durant la fabrication du circuit ou la conception de l'application programmée dans le FPGA.

Cette partie ne dresse pas une liste exhaustive de toutes les menaces et mécanismes de sécurité des circuits reconfigurables. Les articles [TRI14] et [DRU15] donnent plus d'informations à ce sujet.

Les faiblesses les plus évidentes des FPGA touchent au bitstream, autrement dit, le fichier de configuration. La majorité des FPGA sont de technologie SRAM, ce qui signifie que le bitstream doit être rechargé lors de chaque mise sous tension du circuit reconfigurable. Si ce mécanisme n'est pas sécurisé, un adversaire ayant un accès physique au circuit pourrait exécuter de nombreuses attaques. Un des avantages de Microsemi est d'ailleurs de proposer des FPGA Flash et Anti-fusible qui sont non-volatiles, et donc non reconfigurables.

Sans mécanisme de chiffrement du bitstream, l'attaquant peut récupérer ce fichier durant son chargement. S'il possède suffisamment de connaissances sur la structure du fichier de configuration et sur l'architecture du FPGA cible, il peut être capable de reformer l'application programmée en langage RTL par rétro-ingénierie. Une fois cette étape accomplie, l'attaquant peut réutiliser la propriété intellectuelle de cette application pour son propre bénéfice. Afin de protéger la confidentialité du fichier de configuration, les trois vendeurs proposent d'utiliser l'algorithme de chiffrement symétrique AES-256 qui est à l'heure actuelle considéré comme résistant aux attaques par force brute. Microsemi précise utiliser une implémentation de l'AES qui serait protégée contre l'attaque par canaux cachés par analyses différentielles de la consommation (DPA). En ce qui concerne le placement des clés racines, il doit avoir lieu dans un lieu protégé, durant la fabrication. Le bitstream est souvent stocké sous forme chiffrée dans une mémoire Flash non-volatile, afin d'empêcher un adversaire de pouvoir l'exploiter. Enfin, le format des fichiers bitstreams est maintenu confidentiel par les vendeurs afin d'éviter la rétro-ingénierie.

	Altera (Intel-FPGA) - Cyclone III LS	**Microsemi -** IGLOO2	**Xilinx -** 7-series
.bit encryption/decryption	AES-256	AES-256 (anti-DPA)	AES-256
.bit authentication/integrity	Authentication using an external memory device.	Tag based on SHA-256	HMAC-SHA-256
.bit backtracking prevention	No	Yes (with versioning)	No
.bit readback deactivation	Yes (readback not supported).	Yes	Yes (hardened triple-redundancy).
Programming interface monitoring and disabling	JTAG command restriction (only with MAX-II CPLD)	JTAG monitoring and disabling. Disabling only for SPI external flash interface.	JTAG monitoring and disabling.
Configuration memory integrity checking	Continuous CRC-32.	Exportable keyed digest	Continuous CRC.
.bit key encrypted loading	No	Yes	No
.bit key storage memory type	Battery-backed RAM (BBRAM) or eFUSE (key stored in a scrambled form).	Flash	BBRAM or eFUSE.
.bit key zeroization	Yes	Yes	Yes (BBRAM)
.bit secret factory decryption key	None	Yes	None
FPGA memory type	SRAM (60nm)	Flash (65nm)	SRAM (28nm)
Supply chain assurance	None	Device ID and X.509 device certificate.	57-bit device ID set in fuses. 32-bit eFUSE user dedicated.
Internal integrity tests	Continuous CRC-32 on the embedded RAM (only with MAX-II CPLD).	SHA-256 on the ROM, SECDED on eSRAM, eNVM and DDR controller.	Only on configuration memory.
External condition monitoring	Internal oscillator watchdog (only with MAX-II CPLD). Temperature sensor (Arria V & Stratix IV-V).	Dual internal clocks monitoring, active metal mesh.	Temperature, voltage and internal clock monitors.
Active countermeasures	Key zeroization. Registers zeroization (configuration memory, user memory).	Device lockdown, complete device zeroization (with post verification), I/Os set to 'Z' state	Configuration memory zeroization, user flip-flop reset and outputs set to 'Z' state.
Hardware accelerators and security services	None	AES-128/256 (ECB, OFB, CRT, CBC), SHA-256, HMAC, ECC (anti-DPA), Keytree derivation algorithm (anti-DPA), Quiddikey™ PUF key storage, AMBA bus hardware firewalls.	None
Isolation/partitioning tool	Design Separation Flow	No	Isolation Design Flow

Tableau 5.1 : Mécanismes de sécurité des FPGA modernes (source : [DRU15]).

Si aucun mécanisme d'authentification n'est utilisé lors du chargement du bitstream, un attaquant peut être capable de le remplacer par un bitstream malveillant et de compromettre ainsi tout le système dans lequel est placé le FPGA. Dans le cas où l'adversaire malveillant aurait pu extraire et déchiffrer le bitstream initialement prévu pour programmer le FPGA, il pourrait y insérer des processus malveillants. **Les trois vendeurs proposent un mécanisme d'authentification durant le chargement du bitstream.**

Lors des phases de tests, des interfaces spécialisées comme le JTAG, permettent d'accéder facilement à l'intérieur du circuit afin de faciliter le débogage. Les objectifs de tests et les objectifs de sécurité sont souvent contradictoires. Tandis que l'ouverture du circuit facilitent les opérations de test, cela peut aussi provoque certaines failles de sécurité. **Les trois vendeurs permettent la désactivation l'interface de test JTAG une fois que le circuit est déployé**.

Afin de détecter et corriger les fautes (qu'elles soient originaires de l'environnement ou d'un attaquant) qui provoquent des changements de valeurs de données dans le circuit, **des mécanismes de vérification d'intégrité sont proposés par Altera, Microsemi et Xilinx.**

La température et la tension d'un circuit peuvent être modifiées dans le but de créer des glitchs et des comportements inattendus, menant à des failles de sécurité. **Des capteurs de température sont disponibles chez Altera et Xilinx. Seul Xilinx propose un capteur de tension et Microsemi propose une grille de métal active, capable de détecter des intrusions physiques.** Ces détecteurs peuvent déclencher des contre-mesures intégrées au circuit, comme le blocage des broches d'entrées/sorties, l'effacement des mémoires ou du circuit complet.

Enfin, comme la société Microsemi s'est spécialisée dans les FPGA destinés aux applications de défense et de sécurité, des accélérateurs cryptographiques matériels, AES, SHA et ECC sont disponibles.

Pour conclure, les trois vendeurs proposent les mécanismes les plus essentiels, qui protègent le chargement du bitstream. Le niveau de sécurité des Microsemi semble plus important, et les accélérateurs matériels cryptographiques peuvent apporter un réel gain en performances, et contrebalancer la technologie Flash moins performante que la technologie SRAM, qui compose les circuits d'Altera et de Xilinx. Il faut noter que notre étude ne prend pas en compte les familles de FPGA Stratix-10 et Arria-10 d'Altera, car les informations les concernant n'étaient pas disponibles lorsque nous l'avons réalisée.

Malgré tous ces mécanismes, il peut être nécessaire d'ajouter une couche de sécurité supplémentaire pour protéger efficacement un système. En effet, la publication [MOR11] rapporte l'extraction de la clé de chiffrement de bitstream du FPGA Virtex-II Pro de Xilinx, [PAW15] prétend avoir cassé les clés des FPGA Stratix-II et Stratix-III d'Altera. Et enfin, Sergei Skorobogatov et Christopher Woods semblent avoir trouvé une porte dérobée et une manière d'extraire la clé de chiffrement du FPGA ProASIC3 dans [SKO12a], [SKO12b]. A notre connaissance, aucune attaque n'a été publiée pour les familles de FPGA que nous avons étudiées.

5.2. Modèles de menace

5.2.1. Modèles de menace issus de notre état de l'art de la sécurité

Comme le suggère notre taxonomie non exhaustive des attaques visant les systèmes sur puce présentée durant le premier chapitre (**Figure 1.16**), les menaces concernent aussi bien les aspects matériels que logiciels, et interviennent durant toute la durée de vie d'un circuit, que ce soit durant sa conception, sa fabrication ou son déploiement.

Dans l'état de l'art de la sécurité des NoC et des MPSoC, plusieurs modèles de menace sont utilisés comme support d'étude, de conception et de développement des mécanismes de sécurité. Nous avons vu qu'une partie des travaux pouvaient considérer le réseau sur puce comme une entité n'étant pas de confiance et pouvant exécuter des processus malveillants. Cette possibilité est tout à fait envisageable, du fait qu'en ayant recours à l'achat d'IP tierce telle qu'un NoC, on ne peut affirmer que ce n'est pas le cas sans avoir effectué des vérifications pouvant s'avérer très onéreuses et complexes.

Une autre partie des travaux s'intéresse aux attaques par canaux cachés et études de timing visant les mémoires caches partagées.

Et enfin, la majorité des travaux intègre principalement les menaces logicielles dans leur modèle, ce qui se traduit par des violations de confidentialité, d'intégrité, d'authentification et de non-répudiation du point de vue des IP.

5.2.2. Modèle de menace de TrustNoC

Pour établir le modèle de menace servant de support au développement des mécanismes de sécurité pour TrustNoC, on peut se baser sur les modèles de menace issus de l'état de l'art.

Premièrement, nous ne considérons pas notre réseau sur puce comme malveillant car nous avons entièrement développé TrustNoC sans employer d'IP provenant d'un tiers. De ce fait, nous écartons la menace d'un NoC pouvant créer délibérément des fuites de données ou des attaques par déni de service.

Deuxièmement, les attaques en timing sont théoriquement réalisables, sauf qu'elles semblent un peu trop complexe à effectuer pour que l'on s'y intéresse en priorité. Ceci-dit, il est quand même possible de lutter contre ce type d'attaques avec des mécanismes de sécurité développés pour lutter contre le détournement d'IP. Par exemple, en restreignant l'accès aux mémoires caches qui stockent le résultat d'opérations cryptographiques.

En réalité, notre modèle de menace rejoint celui de la majorité des travaux en considérant les attaques logicielles comme étant critiques. Celles-ci proviennent du détournement d'un processeur ou d'une IP, et conduisent à des violations de confidentialité, d'intégrité, d'authentification, et de non-répudiation dans le système sur puce.

Le détournement d'un processeur peut avoir lieu suite à une contamination par un virus, un ver ou un cheval de Troie. La mémoire contenant le code source du processeur peut aussi être la cible d'une attaque visant à modifier son contenu, dans le but d'y injecter des processus malveillants. En dehors des processeurs, les autres IP composant les systèmes sur puce

modernes peuvent contenir des chevaux de Troie, ou être détournées de manière similaire à celle d'un processeur.

Que ce soit pour un processeur ou d'autres IP, le résultat est le même du point de vue du système sur puce, et plus particulièrement, du réseau. Le risque est qu'une entité soit détournée de son comportement nominal et puisse accéder à des zones critiques dans le but de réaliser des fuites d'informations confidentielles, ou de les modifier, et ceci sans être détectée.

La position centrale du NoC dans le système sur puce est une aubaine pour la conception de mécanismes de sécurité. Etant parfois le support de la totalité des communications du système, le NoC permet d'identifier quelles sont les entités impliquées dans chaque échange, et peut même vérifier le contenu des informations transmises. Le réseau sur puce est donc un site très intéressant d'intégration de mécanisme de sécurité visant à contrer certaines attaques logicielles.

Notre modèle de menace ne couvre pas toutes les menaces qui concernent les systèmes sur puce, parce que certaines d'entre elles sont, ou paraissent complètement hors de portée de mécanismes de sécurité placés au niveau du réseau sur puce. Par exemple, il semble difficile de sécuriser le chargement du bitstream à partir du NoC.

Nous avons aussi fait le choix de nous intéresser en priorité aux menaces qui nous semblent les plus critiques, les plus probables, et pouvant avoir les conséquences les plus graves. La sécurité de TrustNoC peut encore évoluer, et plusieurs points pourront faire l'objet de futures améliorations. Notamment en ce qui concernent les attaques de type « *zero day* », c'est-à-dire, n'ayant pas encore été découvertes publiquement.

Le modèle que nous avons fixé considère le détournement d'une IP comme une menace, ce qui aurait comme conséquence la transmission de n'importe quel type de requête vers n'importe quel nœud du réseau. **Nous allons tenter de proposer des mécanismes de sécurité garantissant la protection du système sur puce programmable contre des détournement d'IP, tout en veillant à limiter au maximum l'impact sur les performances.**

5.3. Conception des moniteurs de sécurité

5.3.1. Protections inhérentes à TrustNoC

Avec notre modèle de menace, nous considérons le détournement d'une IP comme possible, ce qui peut mener à l'émission de requêtes malveillantes vers n'importe quel autre nœud du réseau.

Des spécificités de l'architecture de TrustNoC protègent le système sur puce contre certaines attaques. En premier lieu, l'adresse réseau est gérée de manière automatique et transparente pour les IP. C'est-à-dire, que l'adresse source est ajoutée à l'entête des paquets sans intervention des IP, ce qui empêche ces dernières de pouvoir usurper l'identité d'une autre.

En deuxième lieu, les connexions des IP au réseau sont totalement personnalisables, donc en précisant qu'une mémoire est seulement connectée à un processeur au niveau local, sans être connectée au niveau global, il est impossible qu'une autre entité que le processeur puisse accéder à son contenu. C'est le cas par exemple, de l'IP *esclave 2* représentée sur la **Figure 5.7**.

Il s'agit d'une différence essentielle par rapport aux réseaux sur puce classiques, puisque pour ces derniers, toutes les IP se retrouvent connectées au même medium, et peuvent donc potentiellement se transmettre des requêtes.

Pour finir, les décodeurs d'adresse présents dans les interfaces maîtres agissent comme une couche de protection supplémentaire, puisque les requêtes sont générées et dirigées en fonction des informations qu'ils contiennent. En d'autres termes, si le décodeur d'adresse d'une IP maître n'a pas été initialisé avec les informations lui permettant d'atteindre une IP esclave, le détournement de cette IP maître ne permettra pas d'attaquer l'IP esclave. De plus, les décodeurs d'adresse dans la version actuelle de TrustNoC sont statiques, il n'y a aucun moyen logiciel de les modifier durant l'exécution de l'application.

Ceci-dit, les décodeurs d'adresses ne sont pas munis de mécanismes de protection contre les fautes d'origines environnementales (SEU) ou d'attaques par injection.

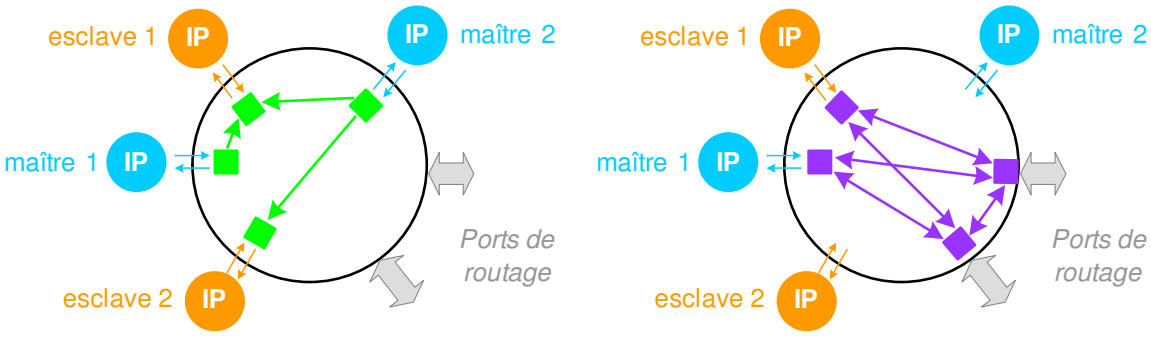


Figure 5.7 : Exemples de connexions des IP dans un routeur TrustNoC.

5.3.2. Principes des moniteurs

- Pare-feu ou moniteur ?

Le détournement d'une IP ou d'un processeur peut potentiellement permettre à un attaquant d'envoyer des requêtes de lecture ou d'écriture à n'importe quelle autre IP sur le réseau, et donc d'accéder à des données confidentielles, voire de modifier la valeur d'informations critiques.

Le placement d'unité de contrôle pouvant aussi réaliser des contre-mesures, comme des pare-feux, est la solution majoritairement utilisée dans les articles traitant de la sécurité des systèmes sur puce et des NoC. Lors de l'émission ou de la réception d'une requête, l'authentification des entités participantes peut être réalisée afin de vérifier des droits d'accès. Dans le cas où une entité viole un droit d'accès, par exemple, en essayant d'écrire des données dans une région mémoire à laquelle elle ne doit pas parvenir, cela peut déclencher des contre-mesures. Pour un pare-feu, celles-ci peuvent se traduire par le blocage de la requête en cours de soumission, et par l'activation d'un signal d'alarme.

Durant la conception de TrustNoC, une de nos principales préoccupations a été de proposer un réseau pour circuit FPGA avec les performances les plus élevées possibles. **C'est pour cette raison que nous avons fait le choix d'opter pour l'utilisation de moniteurs de sécurité, plutôt que pour celle des pare-feux**. Pour que les pare-feux puissent bloquer une requête en cours d'émission, ils doivent être positionnés sur le chemin de données, et engendrer un cycle de latence. Les moniteurs sont quant à eux positionnés en parallèle du chemin de données, ils ne peuvent donc pas bloquer une requête qui viole des polices de sécurité, mais ont pour bénéfice de ne pas nuire aux performances en latence du réseau (**Figure 5.8**).

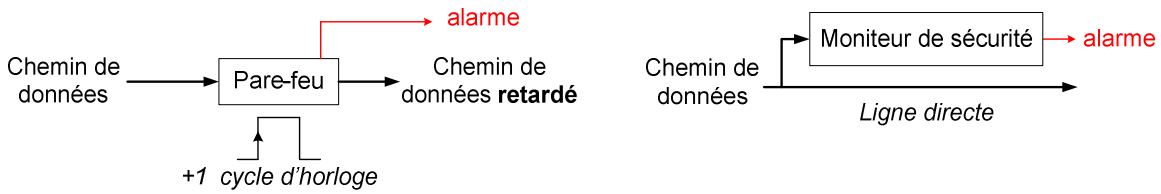


Figure 5.8 : Pare-feu et moniteur de sécurité.

- Contre-mesures

La contre-mesure des moniteurs se limite donc au déclenchement d'un signal d'alarme qui se propage jusqu'à un processeur ou à un composant dédié à la sécurité du système sur puce. Ce composant peut ensuite prendre la décision d'appliquer des contre-mesures à un niveau global, par exemple, en déclenchant des mesures spécifiques aux circuits FPGA, que nous avons détaillées dans notre état de l'art en début de ce chapitre. Ces contre-mesures peuvent consister à bloquer toutes les entrées-sorties du circuit, et à effacer le contenu des mémoires. De fait, bien que le moniteur ne bloque pas une requête malveillante, si le déclenchement des contre-mesures du circuit FPGA est suffisamment rapide, aucun préjudice n'est possible.

Une perspective d'évolution de TrustNoC est de créer une version des moniteurs de sécurité transformés en pare-feux, afin de laisser le choix à l'architecte du système d'opter pour l'un ou l'autre.

- La nécessité de l'emploi de mécanismes de sécurité dans TrustNoC

Bien que TrustNoC possède certaines protections qui lui sont propres face aux attaques, elles ne sont pas suffisantes pour apporter une réelle réponse à notre modèle de menace, d'où notre motivation à concevoir et à développer les moniteurs de sécurité. Une de ces protections propres à TrustNoC est de pouvoir connecter une IP critique à une seule autre IP au niveau local, et de pouvoir isoler cette dernière du reste du réseau. **Les moniteurs de sécurité sont tout de même indispensables pour pouvoir protéger une IP qui a besoin d'être partagée, et dont certaines zones doivent être réservées à certaines IP.**

Une deuxième protection liée à TrustNoC réside dans le fait qu'une IP n'ayant pas les informations nécessaires dans ses décodeurs d'adresse pour accéder à un nœud du réseau, ne puisse pas non plus y accéder une fois qu'elle est détournée, puisque le contenu des décodeurs d'adresse est statique et ne peut être modifié. C'est le cas pour les IP qui n'ont aucune raison de communiquer avec certaines autres durant le fonctionnement nominal de l'application. **Cependant, les moniteurs ont toute leur utilité pour une IP qui est en relation avec une autre, seulement durant certaine partie de l'exécution d'une application.**

- Flexibilité, gradualité et granularité des mécanismes de contrôle de l'état de l'art

Le principe des droits d'accès est assez simple et quasiment identique dans tous les articles scientifiques qui proposent l'utilisation de pare-feux pour protéger les systèmes sur puce multiprocesseur. Les différences et les évolutions des différents travaux se trouvent en réalité dans la **flexibilité**, la **gradualité** et la **granularité** de ces droits d'accès.

La **flexibilité** détermine la capacité des polices de sécurité à pouvoir évoluer durant l'exécution de l'application. Dans [FER15] cela se traduit par la reconfiguration possible des polices de sécurité présentes dans les pare-feux. Il serait même possible d'étendre cette notion de flexibilité aux pare-feux présentés dans la thèse [COT12], dont les droits d'accès se durcissent pour une IP suspectée d'avoir effectué une attaque.

La capacité à pouvoir imposer des contrôles plus ou moins poussés selon les IP ou les zones du réseau [EVA05] est définie par la **gradualité**. L'intérêt est de pouvoir élargir le champ des vérifications pour des IP étant plus vulnérables aux attaques que d'autres. Cela peut être le cas d'un processeur généraliste connecté à un réseau externe, ou bien, ayant son code source stocké dans une mémoire hors du FPGA.

Enfin, la **granularité** définit le niveau de finesse pour lequel les vérifications des polices de sécurité sont effectuées. On peut considérer la granularité comme *grosse* lorsque les droits d'accès sont basés au niveau des IP, et *petite* lorsque les polices de sécurité font la distinction entre les différentes tâches d'un même processeur. Dans [FIO08] le rôle de l'entité qui émet la requête rentre en compte dans le contrôle, puisque la distinction est faite entre un utilisateur et un superviseur.

La proposition de [SEPULV15] combine **gradualité** et **granularité**, car plus le nombre de vérifications est élevé, plus la granularité des contrôles est fine.

- **Flexibilité** des mécanismes de sécurité de TrustNoC

Les mécanismes de sécurité conçus pour **TrustNoC** ont un certain niveau de **flexibilité**, puisqu'il est possible de modifier les autorisations de lecture et d'écriture des polices de sécurité des moniteurs reconfigurables.

Tous les moniteurs sont reliés par un canal de reconfiguration série, similaire à celui employé dans [FER15]. Un composant de sécurité peut être connecté en entrée de ce canal afin de modifier les règles des moniteurs. Nous revenons plus en détail sur le fonctionnement de la reconfiguration des moniteurs plus loin dans ce chapitre.

- **Gradualité** des mécanismes de sécurité de TrustNoC

Les contrôles dans TrustNoC ne sont pas graduels puisque les mêmes sont réalisés par tous les moniteurs. Il s'agit de la vérification du **droit d'accès en lecture et/ou en écriture** d'une IP maître aux plages mémoire de l'IP esclave.

La **longueur des requêtes** est aussi contrôlée, pour éviter qu'une attaque par surcharge de buffer ait lieu. C'est-à-dire, qu'une requête qui débute dans une zone mémoire autorisée, puisse se poursuivre dans une zone où l'accès lui est interdit.

- **Granularité** des mécanismes de sécurité de TrustNoC

La granularité des contrôles se limite au niveau des IP pour les IP maîtres. Cela signifie que tous les processus issus d'une même IP maître possèdent les mêmes droits d'accès aux esclaves. Ce niveau de granularité assez élevé correspond en réalité à nos besoins applicatifs.

La granularité du côté des esclaves est beaucoup plus fine, puisqu'un droit d'accès différent peut être spécifié pour chaque mot de la plage mémoire complète de l'esclave.

Pour résumer, chaque IP maître possède des droits d'accès propres pour chaque zone mémoire de chaque esclave.

Nous avons prévu d'améliorer la granularité des contrôles de TrustNoC en ajoutant une vérification dont nous n'avons trouvé aucune référence dans la littérature scientifique. Il s'agit de la prise en compte des phases d'exécution des processeurs cryptographiques. Le fonctionnement d'un processeur cryptographique peut être partagé en différentes étapes selon son architecture. Ces étapes peuvent être par exemple : **la lecture de la clé**, **la lecture du vecteur d'initialisation**, **la lecture des données en clair**, et enfin **l'écriture des données chiffrées**. Il est possible d'extraire l'information qui indique quelle étape le processeur est en train d'effectuer, afin d'améliorer la finesse des contrôles des droits d'accès. Lors de la phase de **la lecture de clé**, seul l'accès à la **mémoire de clé** doit être autorisé au processeur. S'il accède à une autre mémoire à la place, il s'agit d'un comportement anormal, pouvant être le résultat d'une erreur ou d'une attaque. Le même genre de contrôle peut être réalisé pour chaque étape du fonctionnement du processeur. Cela permet d'éviter par exemple, qu'un processeur accède à la mémoire de clé durant le cycle de lecture de données, et puisse faire fuiter une ou plusieurs clés cryptographiques.

- Placement des moniteurs de sécurité

Le placement des moniteurs à la source est préconisé par [FIO08] afin de bloquer les paquets ne respectant pas les règles de sécurité avant qu'ils ne soient émis sur le réseau, et consomment de la bande-passante inutilement. Le désavantage du contrôle à la source est d'imposer un cycle de latence supplémentaire à chaque paquet émis par une entité, même si ce paquet est destiné à un esclave pour lequel aucun contrôle n'est nécessaire.

Puisque ce sont des moniteurs de sécurité et non des pare-feux qui sont utilisés dans TrustNoC, la question de la consommation inutile de bande-passante ne se pose pas, puisque dès qu'un comportement anormal est détecté, des contre-mesures de verrouillage et d'effacement du FPGA sont préconisées.

Cependant, le contrôle des paquets à l'émission peut s'avérer obligatoire pour les communications du niveau local. En effet, le protocole employé au niveau local est très simple, et dans sa version actuelle, aucun signal ne transmet à l'esclave l'adresse source du maître responsable de la requête. Pour cette raison, les moniteurs ne peuvent pas être employés pour contrôler les communications locales au niveau des IP esclaves.

Concernant les communications globales sous format paquets, les contrôles de sécurité peuvent tout à fait être effectués au niveau de l'IP esclave car l'adresse de l'IP source est transportée dans le flit d'entête du paquet.

Pour ces raisons, dans TrustNoC, l'architecte du système aura le choix à terme de placer des moniteurs de sécurité au niveau des IP maîtres et/ou des IP esclaves. De cette manière, l'architecte peut choisir de ne pas placer de moniteurs de sécurité au niveau d'une IP maître qu'il considère comme parfaitement sûre. Et il peut aussi choisir de renforcer les contrôles pour une IP esclave considérée comme critique, tout en sachant que les contrôles ne seront effectués que pour les requêtes provenant du niveau global. Dans la version actuelle de TrustNoC, les moniteurs sont générés pour chaque IP maître et IP esclave présentes dans le système.

Le **Tableau 5.2** résume les possibilités de contrôle des requêtes selon leur provenance et le positionnement des moniteurs.

Tableau 5.2 : Possibilité du contrôle d'une requête.

Positionnement du moniteur	IP maître (émission)		IP esclave (réception)	
Niveau de communications	Local	Global	Local	Global
Possibilité de contrôle	Oui	Oui	Non	Oui

5.3.3. Fonctionnement détaillé des moniteurs

Pour les IP maîtres, les signaux de requête en sortie de l'IP sont directement connectés au moniteur de sécurité sans passer par un étage d'adaptation (**Figure 5.9**).

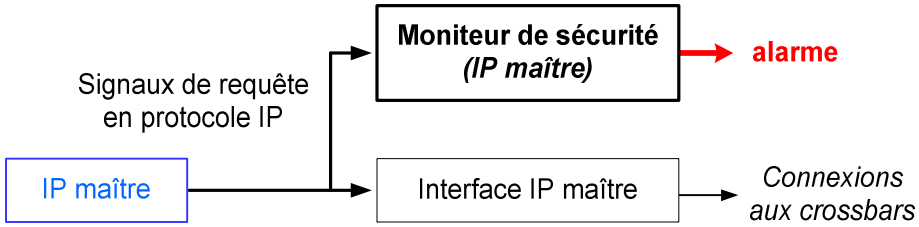


Figure 5.9 : Placement du moniteur IP maître.

Par contre, pour les IP esclaves, seuls les signaux en sortie du dépaquétiseur de requêtes provenant du niveau global sont contrôlés (**Figure 5.10**). Les requêtes du niveau local ne peuvent être contrôlées par le moniteur de sécurité de l'IP esclave, car dans le protocole que nous utilisons il n'y a pas de signal qui indique l'adresse de l'IP maître responsable de la requête.

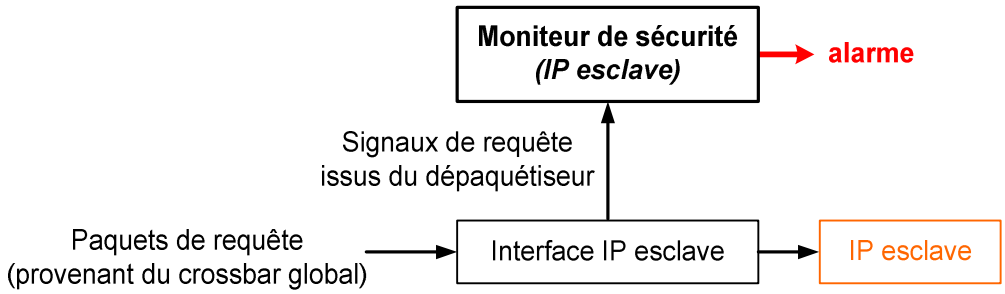


Figure 5.10 : Placement du moniteur IP esclave.

Un organigramme détaille le fonctionnement des moniteurs de sécurité sur la **Figure 5.11**. Au sein des moniteurs de sécurité, un registre resynchronise le signal de requête, puis durant le cycle d'horloge suivant, le contenu de la requête est comparé aux règles de sécurité présentes dans le moniteur.

Les règles du moniteur forment une **liste blanche**, c'est-à-dire, qu'une des polices doit autoriser la requête pour que celle-ci soit permise. En principe, une liste blanche est moins permissive et plus sécurisé qu'une **liste noire**, puisque cette dernière ne spécifie que les accès non permis, et autorise tous les autres.

Dans les moniteurs de TrustNoC, dès qu'une règle qui autorise la requête est trouvée, le contrôle s'interrompt et le moniteur revient dans l'état d'attente d'une nouvelle requête. Dans le cas contraire, le contrôle se poursuit jusqu'à ce que toutes les règles aient été examinées. Si aucune des règles n'autorise la requête, alors l'état d'alerte est déclenché car il s'agit d'une **violation des paramètres de sécurité**.

Un contrôle supplémentaire est effectué par le moniteur de l'IP esclave par rapport à celui de l'IP maître, il s'agit de la vérification de l'adresse du maître qui a soumis la requête.

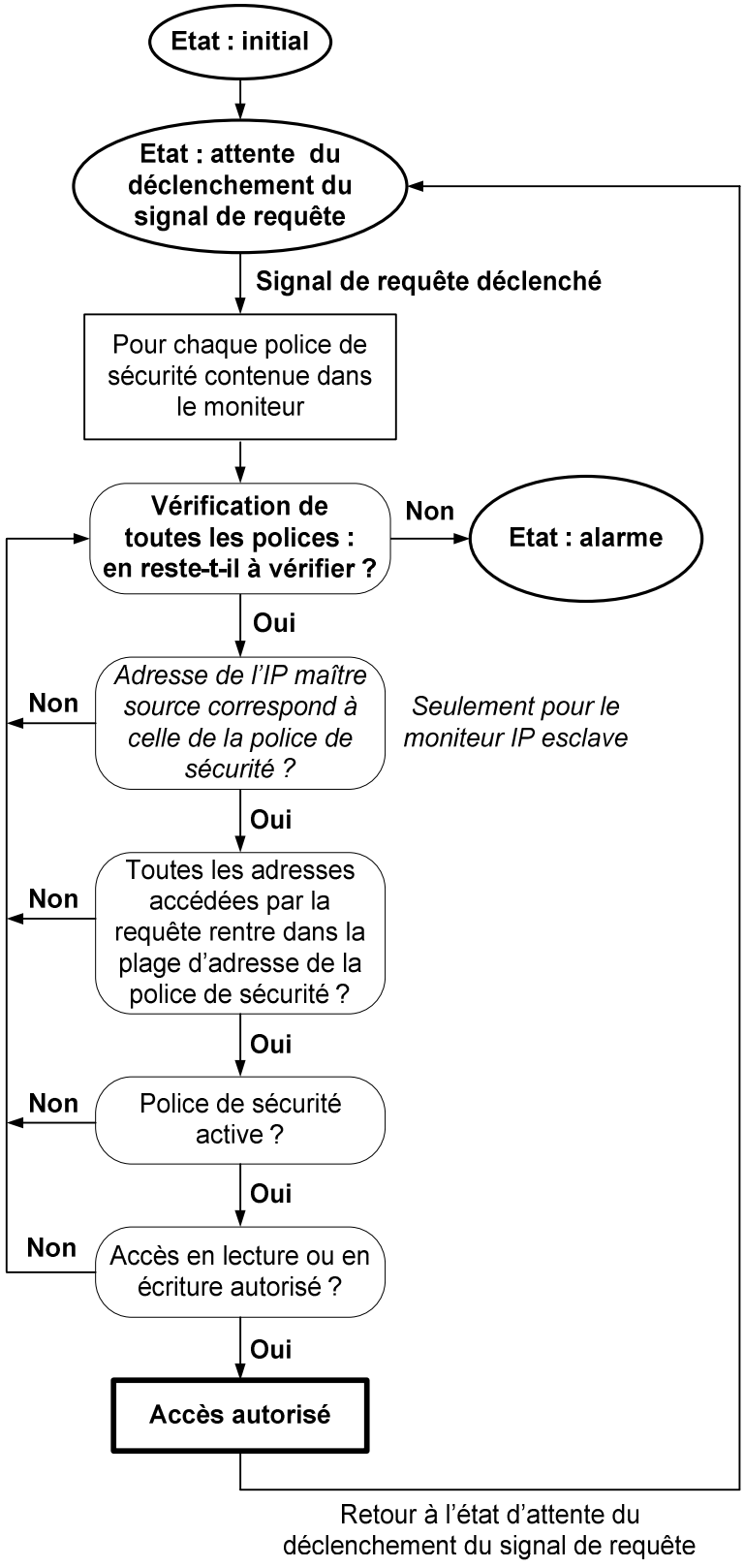


Figure 5.11 : Fonctionnement des moniteurs de sécurité.

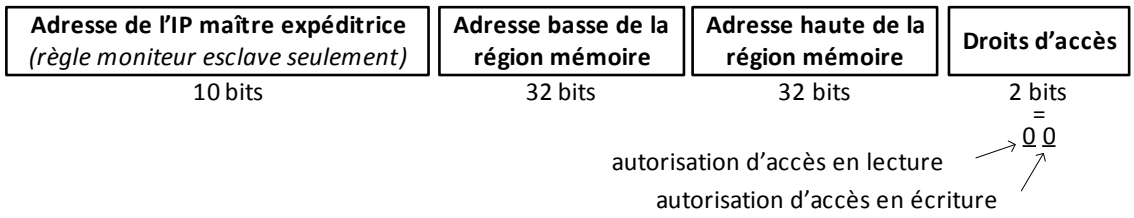
5.3.4. Composition des règles de sécurité

La taille des tables de sécurité est totalement libre, l'architecte du système peut choisir un nombre spécifique de règles pour chaque moniteur.

Comme les moniteurs fonctionnent par liste blanche, plutôt que de spécifier une règle où les accès en lecture et en écriture sont désactivés pour une certaine zone de la mémoire, il est plus intéressant de ne déclarer aucune règle, puisque l'accès à cette zone sera interdit par défaut. La limitation du nombre de règles de sécurité dans les moniteurs permet d'économiser des ressources logiques utilisées lors de l'implémentation.

La composition des règles de sécurité est détaillée sur la **Figure 5.12**. Les règles employées dans les moniteurs esclaves contiennent un champ supplémentaire par rapport à celles des interfaces maîtres. Ce champ supplémentaire permet d'identifier le maître responsable de la requête. Que ce soit pour les moniteurs des IP maîtres ou des IP esclaves, seul le champ de **droits d'accès** est modifiable via le mécanisme de reconfiguration.

Figure 5.12 : Composition des polices de sécurité.



5.4. Mécanisme de reconfiguration des règles de sécurité

5.4.1. Cas d'utilisation

Durant le fonctionnement du système sur puce, il peut s'avérer utile que les paramètres de sécurité évoluent. Par exemple, dans le cas où une IP doit écrire dans la mémoire de clé durant la période d'initialisation du système, puis, n'a plus de raison d'y accéder pendant le reste de l'exécution. Dans ce cas, les règles de sécurité doivent pouvoir suivre l'évolution de l'exécution de l'application pour protéger au mieux le système, sans déclencher de fausses alarmes.

5.4.2. Architecture du canal de reconfiguration

Le canal de reconfiguration des moniteurs de sécurité de TrustNoC reprend certains principes de celui présenté dans le travail [FER15]. Les informations y circulent de manière unidirectionnelle, et sont émises au départ par un composant de sécurité. Les moniteurs sont ensuite reliés en série par le canal de reconfiguration (**Figure 5.13**). L'utilisation de connexions en série permet de limiter la quantité de fils nécessaire pour l'implémentation.

Le canal est indépendant du réseau sur puce. Dans le cas contraire, un mécanisme d'authentification serait nécessaire pour éviter que n'importe quelle entité puisse modifier arbitrairement les règles de sécurité.

L'architecte du système peut déterminer quels moniteurs sont dotés de règles de sécurité statiques et reconfigurables. Les moniteurs statiques consomment moins de ressources que ceux modifiables dynamiquement, puisque les signaux d'entrées/sorties ainsi que la machine d'état dédiés à la reconfiguration des règles ne sont pas synthétisés.

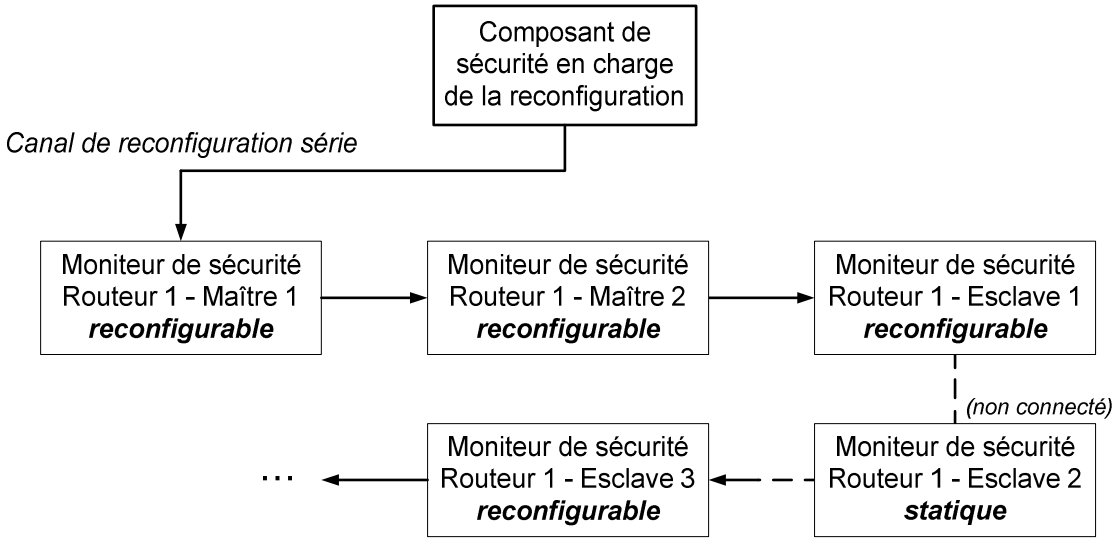


Figure 5.13 : Canal de reconfiguration des moniteurs de sécurité.

5.4.3. Protocole de reconfiguration

Les moniteurs sont initialisés avec un certain nombre de règles de sécurité, puis grâce au mécanisme de reconfiguration, le champ de droit d'accès d'une règle peut être modifié. Par exemple, en autorisant ou en révoquant l'accès en lecture ou en écriture d'une IP à une certaine zone mémoire.

Le canal transporte des paquets de reconfiguration composés de deux flits d'une largeur de 10 bits (**Figure 5.14**). Le premier indique l'adresse de l'IP à laquelle est connecté le moniteur. Et le second contient le numéro de la règle reconfigurée ainsi que les nouveaux droits d'accès qui lui seront associés.

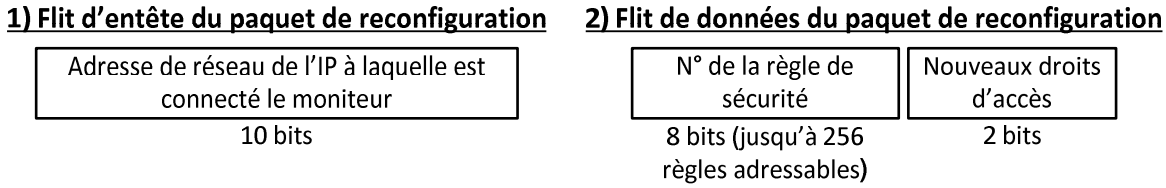


Figure 5.14 : Paquet de reconfiguration des moniteurs.

Chaque flit de reconfiguration passe d'un moniteur au suivant en 1 cycle de latence. Lorsque l'adresse du moniteur correspond avec celle du flit d'entête, ce dernier récupère les données du paquet pour modifier la règle de sécurité.

5.5. Résultats des placements-routages

Les placements-routages que nous avons effectués pour évaluer les performances des moniteurs de sécurité ont été faits dans les mêmes conditions expérimentales que pour le portage d'Hermes et le réseau TrustNoC, à savoir, avec le FPGA Altera ARRIA 10 AS066H4F34I3SGES et la version 15.1 de Quartus.

Le **Tableau 5.3** présente les ressources consommées par les moniteurs destinés aux IP maîtres et aux IP esclaves. Les placements-routages ont été effectués pour des moniteurs statiques et reconfigurables contenant 8 à 64 règles de sécurité.

Tableau 5.3 : Ressources logiques consommées par les moniteurs de sécurité.

Nombre de règles	Moniteur IP maître		Moniteur IP esclave	
	Statique	Reconfigurable	Statique	Reconfigurable
8	35	55	70	80
16	55	80	110	130
32	75	145	145	190
64	100	250	210	320

Les versions statiques des moniteurs des IP esclaves consomment environ deux fois plus de ressources que les versions statiques des moniteurs des IP maîtres. Cela s'explique par le champ supplémentaire présent dans les règles de sécurité des moniteurs IP esclaves, et le contrôle additionnel associé à ce champ.

Les versions reconfigurables des moniteurs consomment plus de ressources que les versions statiques, car elles intègrent une machine d'états et des signaux d'entrées-sorties supplémentaires pour la reconfiguration.

Pour le réseau TrustNoC, la comparaison de la taille des moniteurs à celle des routeurs est délicate parce que beaucoup de paramètres peuvent les faire varier. La complexité des routeurs dépend du nombre et du type d'IP qui y sont connectées, du nombre de connexions avec les autres routeurs, ainsi que du taux de connexions des IP aux niveaux local et global. Par exemple, le routeur n°5 de la topologie à 12 routeurs maillés de TrustNoC (**Figure 4.16**) consomme 4325 ALMs pour des taux de connexions maximum, et 2731 ALMs pour des taux de connexions réduits de moitié. Ce routeur est connecté à 4 IP maîtres, 4 IP esclaves et à 4 autres routeurs, ce qui signifie que les ports du routeur sont occupés à 75%.

Le **Tableau 5.4** compare le nombre de ressources occupées par les moniteurs contenant 32 règles à celles occupées par le routeur n°5 de la topologie à 12 routeurs maillés de TrustNoC. Le nombre de connexions au routeur ainsi que le nombre de règles présentes dans les moniteurs correspondent à des cas d'utilisations assez avancées. On peut observer que les moniteurs représentent entre 1,7% et 7% de la taille du routeur, selon qu'ils soient statiques, reconfigurables, et destinés pour les IP maîtres ou esclaves.

Tableau 5.4 : Tailles des moniteurs comparées à celle d'un routeur de TrustNoC.

	Moniteur IP maître (32 règles)		Moniteur IP esclave (32 règles)	
	Statique 75 ALMs	**Reconfigurable** 145 ALMs	**Statique** 145 ALMs	**Reconfigurable** 190 ALMs
% du routeur avec taux de connexions maximum 4325 ALMs	1,7%	3,4%	3,4%	4,4%
% du routeur avec taux de connexions moyen 2731 ALMs	2,7%	5,3%	5,3%	7%

Les placements-routages réalisés pour l'évaluation de la taille des moniteurs ont été faits avec la topologie à 9 routeurs maillés, 64 IP, et un taux de connexions moyen (**Figure 4.16**). La fréquence de fonctionnement obtenue pour cette topologie sans présence de moniteur est de **223,88 MHz.**

Lorsque tous les moniteurs sont statiques, les chemins critiques ne sont pas impactés puisque la fréquence obtenue de **223,71 MHz** est pratiquement la même.

Par contre, lorsque les moniteurs sont tous reconfigurables, la fréquence de fonctionnement descend à une valeur de **208,51 MHz**, ce qui représente une baisse d'environ **6,9%**. Le chemin critique dans ce cas se situe entre l'IP maître et la machine à états qui réalise la comparaison entre les paramètres de la requête et les polices de sécurité.

On pourrait s'attendre à ce que le chemin critique implique la partie relative à la reconfiguration du moniteur, puisque c'est son rajout qui ralentit la fréquence de fonctionnement. Cependant, il semble plutôt que cela ait augmenté la complexité globale du moniteur, et ait engendré le chemin critique au niveau de la machine à états qui effectue le contrôle de la requête entrante.

Il est sans doute possible de diminuer ce chemin critique en rajoutant des étages de pipeline entre les différentes vérifications, ce qui aura pour effet d'augmenter le délai de déclenchement de l'alarme dans le cas d'une violation des paramètres de sécurité.

5.6. Conclusions et perspectives

Dans ce chapitre, nous avons présenté les mécanismes de sécurité matériels que nous avons conçus et intégrés à TrustNoC, et qui permettent de protéger le système sur puce contre les conséquences du détournement d'IP ou de processeurs.

Durant la conception de ces mécanismes, nous avons suivi la même logique qu'avec TrustNoC en cherchant à développer des protections qui puissent garantir les meilleures performances possibles sur FPGA. C'est pour cette raison que nous avons opté pour des moniteurs de sécurité positionnés en parallèle du chemin de données, plutôt que des pare-feux qui auraient rajoutés des cycles de latence aux échanges. A partir des signaux d'alarme générés par les moniteurs, nous suggérons d'utiliser comme contre-mesures les fonctions de blocage des entrées-sorties, et d'effacement du contenu des mémoires proposées par la plupart des FPGA.

Le niveau de flexibilité, de gradualité et de granularité des contrôles de sécurité sont adaptés à nos contraintes applicatives.

Une des principales contributions de cette thèse est la proposition de la prise en compte du cycle d'exécution des processeurs cryptographiques dans les polices de sécurité. Cette granularité supplémentaire dont nous n'avons pas trouvé d'équivalent dans la littérature scientifique permet de différencier un accès normal ou anormal d'une IP selon le contexte.

Concernant la flexibilité des polices de sécurité, les moniteurs peuvent être dotés d'un canal de reconfiguration série, qui a un impact limité sur les performances des placements-routages et qui ajoute la capacité à un composant de sécurité de modifier les droits d'accès d'une règle.

Les performances obtenues durant les placements-routages de différentes versions de TrustNoC dotées des moniteurs de sécurité sont très satisfaisantes, puisqu'un moniteur possédant 32 règles ne représente qu'environ 2% à 7% de la surface totale d'un routeur de TrustNoC de taille moyenne, voire assez élevée. La fréquence de fonctionnement n'est quant à elle pas, ou peu impactée par l'ajout des moniteurs.

Comme pour le réseau TrustNoC, la personnalisation des moniteurs est assez avancée car l'architecte du système peut choisir un nombre spécifique de règles pour chaque moniteur, et il peut aussi déterminer quels moniteurs sont statiques ou reconfigurables.

Plusieurs pistes d'améliorations sont envisageables pour TrustNoC, comme la création d'une plateforme de simulations capable de supporter des échanges de messages et des scénarios d'attaques, ceci afin d'évaluer l'efficacité des mécanismes de sécurité.

Il sera également intéressant de développer une version des moniteurs transformée en pare-feu afin d'offrir le choix d'utiliser l'une ou l'autre lors d'une implémentation réelle.

Enfin, un enjeu majeur sera d'intégrer le contrôle du cycle d'exécution des processeurs cryptographiques, cependant cela nécessitera aussi la modification d'IP que nous utilisons.

Chapitre 6) Conclusions et perspectives

6.1. Conclusions

Aujourd'hui, grâce à l'amélioration des techniques de conception et de fabrication, les performances des systèmes sur puce qui nous entourent dans la vie quotidienne, ne cessent de se développer. Ils sont capables d'effectuer de plus en plus de tâches, et ceci de manière de plus en plus performante. En lien avec cette évolution, la quantité et la diversité des données traitées par les systèmes numériques sont sans précédents. C'est pour ces raisons, que la recherche autour des architectures qui soutiennent les communications à l'intérieur des systèmes sur puce, visent constamment à améliorer leurs performances, ainsi que leur sécurité.

Dans notre travail, nous nous sommes concentrés sur l'étude des réseaux sur puce sécurisés pour les circuits FPGA. Ces plateformes matérielles reconfigurables sont un des choix privilégiés pour la conception de produit de sécurité destinés aux administrations ou à la défense, notamment car leur coût est plus adapté aux faibles volumes de production. Ils atteignent également des performances assez élevées lors de l'exécution d'algorithmes cryptographiques.

Cependant, le choix du circuit (ASIC ou FPGA) pour l'implémentation d'un système sur puce, et plus particulièrement, du réseau sur puce, a une grande importance, et peut avoir un large impact sur les performances. Dans cette optique, nous avions pour objectif de réaliser des recherches dans le but d'identifier l'architecture de réseau pour FPGA qui pourrait répondre au mieux à nos contraintes applicatives. Notre deuxième objectif était de réaliser une étude de la sécurité de ces réseaux et des systèmes qui les supportent, pour proposer des mécanismes de sécurisation ayant un faible surcoût.

L'état de l'art a été principalement basé sur les réseaux sur puce destinés aux circuits FPGA. Nous avons tenté de mettre en évidence la difficulté de comparer les performances de plusieurs réseaux. Ceci parce que les circuits utilisés pour les implémentations peuvent avoir des architectures matérielles sensiblement différentes, mais aussi, car les motifs des trafic utilisés lors de l'étude de la résistance à la charge d'un réseau sur puce ne sont pas forcément représentatifs d'une application finale. De plus, certains réseaux peuvent se révéler largement supérieurs, ou largement inférieurs à d'autres, selon l'application à l'origine du trafic. La solution idéale pour comparer deux architectures de NoC, serait de pouvoir réaliser leur placement-routage sur le même circuit, et d'effectuer des mesures de résistance à la charge, en utilisant les motifs du trafic de l'application finale.

Malgré ces difficultés, et le fait que le choix d'un NoC dépend fortement de l'application qu'il est sensé soutenir, il est quand même possible d'identifier certains choix d'architecture donnant de meilleurs résultats que d'autres, sur FPGA. Par exemple, le fait de décentraliser les unités de routages et d'arbitrages dans le routeur, et de créer entre 2 et 3 étages de logiques, semblent maximiser la fréquence de fonctionnement, et la bande-passante des NoC à commutation de paquets. Ceci, tout en minimisant la latence et le coût en ressources logiques du réseau.

Concernant l'approche expérimentale, nous avons réalisé le portage sur FPGA du réseau sur puce Hermes, qui a une architecture générique et polyvalente, initialement conçue pour une utilisation sur ASIC. Afin d'évaluer la viabilité de cette solution, nous avons fixé à partir des connaissances que nous avions des produits de sécurité du partenaire industriel, l'objectif d'une consommation d'environ 30% d'un FPGA de dernière génération (de taille moyenne), et une fréquence de fonctionnement minimale de 150 MHz. Ces contraintes applicatives, étaient fixées pour un réseau pouvant supporter une centaine d'IP, et nous n'avions pas précisé des niveaux de résistance à la charge spécifique. En effet, les applications visées ont des temps de calculs, plus longs que les temps de communications, donc il s'agit d'un objectif moins critique, que celui de ne pas faire chuter la fréquence globale du système, et de laisser suffisamment de ressources logiques pour le reste des fonctions.

Les résultats des placements-routages du portage d'Hermes atteignent la limite suffisante, cependant, les latences de ses communications se sont révélées beaucoup trop importantes pour l'application. Ce défaut provient d'ailleurs de sa structure polyvalente.

Après avoir réalisé ce portage, nous avons décidé de développer TrustNoC, un réseau sur puce pour FPGA, avec une architecture adaptée à l'application ciblée. A partir de l'état de l'art, et de l'étude du portage d'Hermes, les idées étaient principalement de limiter le nombre de routeurs, et de créer deux niveaux de communication dans le réseau. Un niveau à faible latence pour chaque *cluster* (autour de chaque routeur), et un niveau de communication global, reliant certaines IP des différents clusters. Les espaces mémoires ont aussi été fortement réduits par rapport à Hermes, car il s'agit de ressources assez rares et lourdes sur FPGA. TrustNoC est également configurable de manière assez précise, ceci afin d'optimiser au maximum les ressources consommées. Les résultats des placements-routages ont pu répondre aux contraintes applicatives, et la latence moyenne des échanges a pu être très grandement réduite par rapport à Hermes.

Afin de répondre aux problématiques qui gravitent autour de la sécurité du réseau et du système sur puce dans lequel il est intégré, nous avons au préalable établi un état de l'art. C'est un sujet qui a déjà été l'objet de nombreux travaux, et qui évoluent en fonction du développement des systèmes sur puce. Lorsque l'on considère un réseau sur puce comme sûr, l'intégration de contrôles de droits d'accès est déjà un moyen de contrer un large panel d'attaques issues du détournement d'une IP. L'enjeu est essentiellement d'adapter la finesse, et la flexibilité des contrôles à l'application finale. Pour TrustNoC, nous avons fait le choix d'utiliser des moniteurs qui peuvent, seulement détecter des attaques, plutôt que des pare-feux qui peuvent aussi les bloquer ; ceci afin de limiter le surcoût en latence. Un canal de reconfiguration léger peut être utilisé pour modifier le contenu des règles de sécurité des moniteurs voulus.

6.2. Perspectives

Certains d'objectifs n'ont pu être menés à bien durant le temps imparti pour la thèse, cependant, ils pourront faire l'objet de futurs travaux.

En ce qui concerne TrustNoC, une plateforme d'injection de trafic, et de mesures des performances serait un excellent moyen de parfaire sa validation. Cela permettrait également d'obtenir des mesures de résistance à la charge servant de support de publications, et de comparaison avec d'autres NoC. Grâce à la capacité de modéliser le trafic d'une application donnée, il serait aussi possible d'identifier la capacité du réseau à fournir suffisamment de performances. Si ce n'est pas le cas, il serait possible d'identifier ses faiblesses et de pouvoir l'améliorer. La plateforme d'injection est aussi une bonne approche pour simuler des attaques ou des fautes, et d'évaluer la résistance du réseau face à celles-ci.

Une autre perspective d'évolution pour TrustNoC, est la possibilité de pouvoir supporter simultanément des IP 32 et 64 bits. Une approche possible est de dédier entièrement certains routeurs aux IP 64 bits, afin de ne pas à avoir à diviser la bande-passante de celles-ci de moitié pour les communications locales.

Afin d'améliorer la résistance à la charge du réseau, il est envisageable de rendre les tables de routage dynamiques et adaptives à la charge. L'outil en Python de configurations de TrustNoC, qui génère actuellement le contenu des tables de routage statiques à partir d'un algorithme de recherche du plus court chemin, peut être amélioré en intégrant une notion de « poids », répartissant mieux la charge sur les différents chemins possibles.

Concernant les moniteurs de sécurité de TrustNoC, il est envisagé d'intégrer la reconnaissance du cycle d'exécution de l'IP dans les vérifications. Cela n'a pas encore été le cas, car toutes les IP du système ne possèdent pas les signaux nécessaires à la mise en place du contrôle.

Nous prévoyons de développer une version « pare-feu » des moniteurs de sécurité, qui auraient la possibilité de bloquer les accès illégaux. Ceci pour laisser le choix à l'architecture d'utiliser l'une ou l'autre version, selon l'IP en question.

L'outil de génération de configuration doit être également complété pour générer le contenu des moniteurs de sécurité.

D'autres pistes d'amélioration des moniteurs, consisteraient à augmenter la flexibilité des polices de sécurité avec la possibilité d'ajouter des règles complètes, contrairement à la version actuelle où il est seulement possible de modifier la valeur des droits d'accès pour une zone mémoire. La finesse des polices de sécurité peut être aussi optimisée, en permettant l'identification de la tâche, et de l'utilisateur responsable de l'émission de la requête.

Publications réalisées dans le cadre de cette thèse

[DRU15] Rémy Druyer, Pascal Benoit, Lionel Torres, Patrick Le-Quéré, « *A Survey on Security Features in Modern FPGAs* », 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015.

[DRUYER14] Rémy Druyer, Pascal Benoit, Lionel Torres, Patrick Le-Quéré, Anne-Marie Guilloux, « *Distributed Security Based on Hardware Firewall for NoC on FPGA* », MEDIAN-TRUDEVICE'14, Amsterdam, 2014.

Bibliographie et références

[ALT10] Altera, "*Arria 10 Device Overview*", Fiche Technique, 2015. https://www.altera.com/en_US/pdfs/literature/hb/arria-10/a10_overview.pdf

[ALT12a] Altera. "*Using TimeQuest Timing Analyzer*", Fiche Technique, octobre 2012.

[ALT12b] Altera, "*AN 589: Using the Design Security Feature in Cyclone III LS Devices*" , Fiche Technique, 2012.

[ANC15] Dean Michael B. Ancajas, "*Design of Reliable and Secure Network-on-Chip Architectures*", Manuscrit de thèse, Paper 4150, Université d'Utah, 2015.

[ANS] ANSSI, « *Certification Critères Communs* », https://www.ssi.gouv.fr/administration/produits-certifies/cc/, Lien web.

[ART] http://www.arteris.com/, Lien web.

[BAR03] T.A. Bartic, J-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, R. Lauwereins, *"Highly Scalable Network on Chip for Reconfigurable Systems"*, In: Proceedings. International Symposium on System-on-Chip, 2003.

[BEN02] Luca Benini, Giovanni De Micheli, « *Networks on chips: a new SoC paradigm* », Journal : IEEE Computer Society, Volume: 35, Issue: 1, p70-78, Jan 2002.

[BOS08] Lilian Bossuet, Guy Gogniat, *« La sécurité des systèmes embarqués »*, Article, 2008.

[CHE15] Qi Chen, Qiand Liu, "*Pipelined NoC Router Architecture Design with Buffer Configuration Exploration on FPGA*", In: International Conference on FPL, 2015.

[COB05] Joel Coburn, Srivaths Ravi, Anand Raghunathan, Srimat Chakradhar, "*SECA: Security-Enhanced Communication Architecture*", Proceedings of the International Conference on Compilers, Architectures and Synthesis and Embedded Systems (CASES), 2005.

[COP13] Marcello Coppola, Miltos D. Grammatikakis, Alexander Spyridakis, « *Trusted Computing on Heterogeneous Embedded Systems-on-Chip with Virtualization and Memory Protection*", In: The 4th International Conference on Cloud Computing GRIDS and Virtualization, p225-229, 2013.

[COT12] Pascal Cotret, *« Protection des architectures hétérogènes multiprocesseurs dans les systèmes embarqués : Une approche décentralisée basée sur des pare-feux matériels*", Manuscrit de Thèse, Université de Bretagne-SUD, Lab-STICC, HAL : tel-00789541, version 2**,** 2012.

[DIG07] Jean-Philippe Diguet, Samuel Evain, Romain Vaslin, Guy Gogniat, Emmanuel Juin, "*NoC-centric security of reconfigurable SoC* », In : First International Symposium on Networks-on-Chip (NOCS), 2007.

[DRU15] Rémy Druyer, Pascal Benoit, Lionel Torres, Paul-Vincent Bonzom, Patrick Le-Quéré, « *A Survey on Security Features in Modern FPGAs »*, In: 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015.

[DXBAR12] Yixuan Zhang, Randy Morriz, Dominic DoTomaso & Avinash Kodi, « *Energy-Efficient and Fault-Tolerant Unified Buffer and Bufferless Crossbar Architecture for NoCs* », In: IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012.

[EHL07] Andreas Ehliar, Dake Liu, "*An FPGA Based Open Source Network-on-Chip Architecture*", In: International Conference on Field Programmable Logic and Applications (FPL), 2007.

[EVA05] Samuel Evain, Jean-Philippe Diguet, "*From NoC Security Analysis to Designs Solutions* », In: IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS), 2005.

[FER15] Ramon Fernandes, Bruno Oliveira, Johanna Sepulveda, Cesar Marcon, Fernando G. Moraes, "*A Non-intrusive and Reconfigurable Access Control to Secure NoCs*", In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2015.

[FER16] Ramon Fernandes, Cesar Marcon, Rodrigo Cataldo, Jarbas Silveira, George Sigl, Johanna Sepulveda, "*A Security Aware Routing Approach for NoC-based MPSoCs*", In: 29th Symposium on Integrated Circuits and Systems Design (SBCCI), 2016.

[FIO08] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, Valerio Catalano, Cristina Silvano, "*Secure Memory Accesses on Networks-on-Chip*", Journal: IEEE Transactions on Computers, Vol.57, Issue: 9, 2008.

[FIS11] David A. Fisher, Jonathan M. McCune, Archie D. Andrews, "*Trust and Trusted Computing Platforms*", Fiche Technique, CMU/SEI-2011-TN-005, CERT Program, Janvier 2011.

[GRA14] Miltos D. Grammatikakis, Kyprianos Papadimitriou, Polydoros Petrakis, Antonis Papagrigoriou, George Kornaros, Ioannis Christoforakis, Marcello Coppola, "*Security Effectiveness and a Hardware Firewall for MPSoCs*", In: IEEE International Conference on High Performance Computing and Communications (HPSS), 6th International Symposium on Cyberspace Safety and Security (CSS) and IEEE 11th International Conference on Embedded Software and Systems (ICESS), 2014.

[HIL06] C. Hilton, B. Nelson, "*PNoC: a Flexible Circuit-Switched NoC for FPGA-based Systems*", In: IEEE Proceedings – Computers and Digital Techniques, Volume: 152, Issue: 3, 2006.

[HUA12] Yutian Huan, André DeHon, "*FPGA Optimized Packet-Switched NoC using Split and Merge Primitives*", In: International Conference on Field-Programmable Technology (FPT), 2012.

[IBM99] IBM. The Coreconnect™ Bus Architecture, Fiche Technique, 1999.

[KAP06] Nachiket Kapre, Nikil Mehta, Michael deLorimier, Raphael Rubin, "*Packet Switched vs. Time Multiplexed FPGA Overlay Networks*", In: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2006.

[KAP15] Nachiket Kapre, Jan Gray, "*Hoplite: Building Austere Overlay NoCs for FPGAs*", In: 25th International Conference on Field Programmable Logic and Applications (FPL), 2015.

[KHA16] Mohamed, Khaled Salah, "*SoC Buses and Peripherals: Features and Architectures*", Livre : IP Cores Design from Specifications to Production, pp 77-96, Springer International Publisher, 2016.

[KWA12] Jimmy Kwa, Tor M. Aamodt, "*Small Virtual Channel Routers on FPGAs Through Block RAM Sharing*", In: International Conference on Field-Programmable Technology (FPT), 2012.

[LEE09] Jason Lee, Lesley Shannon, « *The Effect of Node Size, Heterogeneity, and Network Size on FPGA based NoCs* », In: International Conference on Field-Programmable Technology (FPT), 2009.

[LEE10] Jason Lee, Lesley Shannon, "*Predicting the Performance of Application-Specific NoCs implemented on FPGAs*", In: Proceedings of the Annual International Symposium on Field Programmable Array (FPGA'10), 2010.

[MAI15] Pongstorn Maidee, Alireza Kaviani "*Improving FPGA NoC Performance using Virtual Cut-Through Switching Technique*", In: International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2015.

[MAR02] Théodore Marescaux, Andrei Bartic, Dideriek Verkest, Serge Vernalde, Rudy Lauwereins, "*Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs*", In: International Conference on Field Programmable Logic and Applications (FPL), pp795-805, 2002.

[MIC16] Microsemi, "*SmartFusion 2 SoC FPGA and IGLOO2 FPGA Security Best Practices*" Guide d'utilisation, UG0443, 2016.

[MOL10] Leandro Möller, Peter Fischer, Fernando Moraes, Leandro Soares Indrusiak, Manfred Glesner, "*Improving QoS of Multi-Layer Networks-on-Chip with Partial and Dynamic Reconfiguration of Routers*", In: International Conference on Field Programmable Logic and Applications (FPL), 2010.

[MOR04] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, Luciano Ost, "*HERMES: an infrastructure for low area overhead packet-switching networks on chip*", Journal: The VLSI journal Integration 38, p69-93, 2004.

[MOR11] Amir Moradi, Alessandro Barenghi, Timo Kasper, Christof Paar, "*On the Vulnerability of FPGA Bitstream Encryption against Power Analysis Attacks – Extracting Keys from Xilinx Virtex-II FPGAs*", In: ACM Conference on Computer and Communications Security, 2011.

[NIS] NIST - "*FIPS Publications*", http://csrc.nist.gov/publications/PubsFIPS.html, Lien web.

[PAP12] Michael K. Papamichael, James C. Hoes, "*CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs*", In: FPGA'12. http://www.ece.cmu.edu/calcm/connect/

[PAW15] Pawel Swierczynski, Amir Moradi, David Oswald, Christof Paar, "*Physical Security Evaluation of the Bitstream Encryption Mechanism of Altera Stratix II and Stratix III FPGAs*", Journal: ACM Transactions on Reconfigurable Technology and Systems (TRETS), Volume 7, Issue 4, 2015.

[POR11] Joël Porquet, Alain Grenier, Christian Schwarz, « *NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs* », DATE 2011.

[PYT] https://www.python.org/doc/essays/graphs/, Lien web.

[QSY] https://www.altera.com/products/design-software/fpga-design/quartus-prime/features/qts-qsys.html, Lien web.

[RAJ15] Rajesh J.S., Dean Michael Ancajas, Koushik Chakraborty, Sanghamitra Roy, « *Runtime Detection of a Bandwidth Denial Attack from a Rogue Network-on-Chip*", In: Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS), Article n°8, 2015.

[RAN06] Ville Rantala, Teijo Lehtonen, Juha Plosila, "*Network on Chip Routing Algorithms*", Rapport Technique, TUCS No 779, Août 2006.

[REX94] Jennifer Rexford, Kang G. Shin, « *Support for Multiple Class of Traffic in Multicomputer Router*", In: Parallel Computer Routing and Communication Workshop, p116-p130, 1994.

[REI16] Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, Georg Sigl, Johanna Sepulveda, "*Side Channel Attack on NoC-based MPSoCs are practical: NoC Prime+Probe Attack*", In: 29th Symposium on Integrated Circuits and Systems Design (SBCCI), 2016.

[RIS06] Séverine Riso, Michel Robert, « *Évaluation des paramètres architecturaux des réseaux sur puce* », Thèse, Université des Sciences de Montpellier, France, 2006.

[SEP12] Sepulveda, J., Ricardo, P., Guy, G., Wang, J. C., & Marius, S. (2012). "*QoSS hierarchical NoC-based architecture for MPSoC dynamic protection*", International Journal of Reconfigurable Computing, p1-10, 2012.

[SCH08] Graham Schelle, Dirk Grunwald, "*Exploring FPGA Network on Chip Implementations Across Various Application and Network Loads*", In: International Conference on Field Programmable Logic and Applications (FPL), 2008.

[SET05] Balasubramanian Sethuraman, Prasun Bhattacharya, Jawad Khan, Ranga Vemuri, "*LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip"*, In: Proceedings of the 15th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2005.

[SON] https://SONinc.com/, Lien web.

[SEP12] Johanna Sepulveda, Ricardo Pires, Guy Gogniat, Wang Jian Chau, Marius Strum, "*QoSS Hierarchical NoC-Based Architecture for MPSoC Dynamic Protection*", In: International Journal of Reconfigurable Computing, Article ID 578363, 2012

[SEP14] Johanna Sepulveda, Guy Gogniat, Daniel Florez, Jean-Philippe Diguet, Cesar Pedraza, Marius Strum, *"3D-LeukoNoC: A Dynamic NoC Protection*", In: International Conference on ReconFigurable Computing and FPGAs (ReConFig), 2014.

[SEP15a] Martha Johanna Sepulveda, Jean-Philippe Diguet, Marius Strum, Guy Gogniat, "*NoC-Based Protection for SoC Time-Driven Attacks*", Journal: IEEE Embedded Sytems Letters, Vol.7, No.1, March, 2015.

[SEP15b] Johanna Sepulveda, Daniel Florez, Guy Gogniat, "*Reconfigurable Security Architecture for Disrupted Protection Zones in NoC-Based MPSoCs*", In: 10th Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015.

[SURFNOC13] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, Timothy Sherwood, "*SurfNoC: A Low Latency and Provably Non-Interfering Approach to Secure Networks-On-Chip*", In: 40th International Symposium on Computer Architecture (ISCA), 2013.

[SICa] Silicon.fr - "*ARM vise maintenant... les supercalculateurs* », article du 22 août 2016, http://www.silicon.fr/arm-vise-supercalculateurs-155549.html, Lien web.

[SICb] Silicon.fr - "*Bull et Cray vont concevoir les premiers supercalculateurs ARM*», article du 20 janvier 2017, http://www.silicon.fr/bull-cray-concevoir-premiers-supercalculateurs-arm-167602.html, Lien web.

[SKO12] Sergei Skorobogatov, Christopher Woods, "*Breakthrough Silicon Scanning Discovers Backdoor in Military Chip*", In: Proceedings of the 14th international conference on Cryptographic Hardware and Embedded Systems, septembre 2012.

[SKO12a] Sergei Skorobogatov, Christopher Woods, "*Breakthrough silicon scanning discovers backdoor in military chip (DRAFT of 05 March 2012)*", 2012.

[SKO12b] Sergei Skorobogatov, Christopher Woods, "*In the blink of an eye: There goes your AES key (DRAFT of May 2012)*", 2012.

[TEH10] Mohammad Tehranipoor, Farinaz Koushanfar, "*A Survey of Hardware Trojan Taxonomy and Detection*", Journal: IEEE Design & Test of Computers, Volume: 27, Issue: 1, Jan.-Feb. 2010.

[TCG11] Trusted Computing Group, "*TPM Main Part 1 Design Principles*", Document de specification, Version 1.2, Revision 116, Mars 2011. https://trustedcomputinggroup.org//wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf

[TPM] Wikipedia – "*Trusted Platform Module*" https://en.wikipedia.org/wiki/Trusted_Platform_Module, Lien web.

[TRI14] Stephen M. Trimberger, Jason J. Moore, "*FPGA Security: Motivations, Features, and Applications*", Invite Paper in In: Proceedings of the IEEE Vol. 102, No. 8, 2014.

[WIS10] OpenCores, "*Wishbone B4 - WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*", Document de specifications, 2010.

[XIL13] Xilinx, "*Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs*", Document technique, XAPP1084 (V1.3), 2013.

[XIL15] Xilinx, « *Vivado Design Suite* », Document technique, AXI Reference Guide, UG1037 V3.0, 2015.

[XIL08] Xilinx, Ken Chapman, "*Saving Costs with the SRL16E*", Papier blanc, WP271 V.10, 2008.

[YEL11] Ye Lu, John McCanny, Sakir Sezer, "*Generic Low-Latency NoC Router Architecture for FPGA Computing Systems*", In: International Conference on Field Programmable and Logics (FPL), 2011.