Sommaire

Table des Figures

Table des Tableaux

Introduction générale

Depuis les années 70, les techniques d'intégration de transistors dans les systèmes électroniques ne cessent de s'améliorer. Ainsi, les systèmes à base de puces électroniques font de plus en plus partie de notre quotidien. Ceux-ci intègrent maintenant plusieurs millions de transistors. Cette tendance semble non seulement se confirmer mais se renforcer : les SOC (*System on chip*) contiendront plusieurs processeurs. De ce fait, les systèmes embarqués (SE) à base de microprocesseurs ont été introduits dans de nombreux domaines d'application tels que les appareils électroménagers, l'automobile, l'avionique, les équipements réseaux, les terminaux de communication sans fils, les systèmes multimédias, les systèmes de contrôle industriels. Il est donc crucial de maîtriser la conception de tels systèmes tout en respectant les contraintes de mise sur le marché et les objectifs de qualité. En effet, l'augmentation de la capacité et de la complexité des systèmes monopuces a stimulé les chercheurs pour concevoir de nouvelles plateformes d'interconnexion fiable, à énergie réduite et à rendement élevé, baptisées réseaux sur puce (NOC : *Network On Chip*), afin de remédier aux problèmes de communication générés par les anciennes architectures d'interconnexion (les bus).
Les réseaux sur puces semblent être une solution appropriée pour gérer la communication entre les ressources (Processeur, DSP, IP, ASIP, etc.…). La difficulté de la conception d'un NOC réside dans un compromis entre une Qualité de Service optimale, une bande passante élevée, une latence faible, une flexibilité, une extensibilité d'utilisation importantes, et une possibilité de réutilisation de la conception, tout en limitant la consommation d'énergie et de surface dans la puce.

C'est dans ce cadre que s'inscrit notre projet qui consiste à l'étude et prototypage des réseaux d'interconnexions multi-étages (MINs) de type Delta dans une architecture multiprocesseur (MPSOC), et à trouver ensuite des techniques pour améliorer leurs performances.

Ce travail couvre les aspects suivants:

- Etude et conception des réseaux multi-étages sur des plateformes reconfigurable FPGA de type Delta dédiés aux MPSOC.
- Améliorer les performances du NOC conçu suivant le besoin et le contexte de travail (Synchrone ou Asynchrone).
- Implémenter les réseaux multi-étages dans une architecture multiprocesseur.
- Validation du fonctionnement de l'architecture par une application.

L'organisation du mastère est comme suit :

Le chapitre premier étudie l'architecture multiprocesseur sur puce et ses propriétés en exposant les principales étapes de flot de conception. Ensuite le deuxième chapitre présente les réseaux sur puces ainsi que leurs caractéristiques. Le chapitre 3 développe en détail la conception des réseaux multi-étage de type Delta, en proposant 2 approches d'amélioration des performances du notre NOC selon le besoin.

Quant au dernier chapitre, nous décrivons l'implantation du notre réseau Delta MIN dans une architecture MPSOC. La validation du fonctionnement de l'architecture a été faite dans un contexte applicatif propre au domaine télécommunication.

Chapitre1 : Les architectures Multiprocesseur sur puce

1. Introduction

Les applications embarquées appartiennent à un domaine en évolution phénoménale. Néanmoins elles sont de plus en plus soumises à des fortes contraintes fonctionnelles (Puissance de calcul, consommation, miniaturisation...) et non fonctionnelles (tels que, temps de mise sur le marché, forte croissance de la quantité de la production). D'un autre coté, le progrès technologique permettra une grande capacité d'intégration sur une seule puce, l'ITRS [8] prévoit en 2012 l'intégration des systèmes électroniques de 4 milliards de transistors pour des fréquences proches de 10 GHz . L'architecture associée à un flot de conception efficace qui traduit le potentiel de la technologie en performance et capacité. De point de vue architectural, les trois manières exploitant un grand volume de ressources afin d'améliorer la performance sont le parallélisme, la localité, et la spécialisation. Ces trois critères impliquent respectivement le caractère multiprocesseur, monopuce et la focalisation sur une application spécifique. Quant au flot de conception, manipuler un grand volume de ressources revient à remonter le niveau d'abstraction et à proposer une méthodologie systématique pour le passage de ce niveau à une implémentation optimale.
Ce chapitre présente les systèmes multiprocesseurs monopuces en indiquant leurs propriétés. Ensuite, il expose les principales étapes de conception, la dernière partie illustre la communication entre les processeurs.

2. Du monoprocesseur au Multiprocesseur

Avec le progrès technologique il y a croissance de la capacité d'intégration de transistors sur une seule puce (centaines de millions), deux tendances architecturales ont émergé pour relever ce défi. La première tendance s'est restreinte à l'utilisation d'architectures monoprocesseurs tout en améliorant considérablement les performances du CPU utilisé et l'utilisation de coprocesseurs. Un tel CPU se distingue par : une fréquence de fonctionnement très élevée, des structures matérielles spécialisées, un ensemble d'instructions sophistiquées, plusieurs niveaux de hiérarchie mémoire (plusieurs niveaux de caches), et des techniques spécialisées d'optimisation logicielle (nombre d'accès mémoire, taille, etc).

La deuxième tendance s'est tournée vers les architectures multiprocesseurs (multimaîtres). Ces architectures qui étaient réservées aux machines de calculs scientifiques (tel que CM [1]) ne le sont plus avec le progrès de la technologie.
Le domaine d'application couvert par des systèmes multiprocesseurs sur puce est très vaste, en effet les applications télécommunications et multimédias sont les plus représentatives de ce domaine. Elles constituent l'un des marchés le plus en expansion actuellement (modems, téléphones mobiles, les décodeurs audio et vidéo, etc.), notamment par le développement des télécommunications sans fil et d'Internet.

3. Système multiprocesseur monopuce MPSoC

Un système multiprocesseur sur puce MPSoC dénoté *multiprocessor System-on-Chip* est un système complet intégrant sur une seule puce de silicium plusieurs composants complexes et hétérogènes tels que des unités de calcul spécifiques programmables et/ou non programmable (CPU, DSP, ASIC-IP, FPGA) des réseaux de communication complexes (Bus hiérarchiques sur puce, réseau sur puce) des composants de mémorisation variés, périphériques E/S, etc. La Figure1 représente un modèle générique d'un système MPSoC hétérogène avec des parties matérielles et logicielles structurées en couches pour maîtriser la complexité. Le matériel se divise en deux couches :

- La couche basse contient les composants de calcul et de mémorisation utilisés par le système (µP, µC, DSP, matériel dédié IPs, mémoires).
- La couche matérielle de communication embarquée sur la puce composée de deux sous-couches : média de communication (liens point-à-point, bus hiérarchique, réseau sur puce) et adaptateurs de communication entre le réseau et les composants de la première couche.

Le logiciel embarqué est aussi découpé en couches :

- La couche la plus basse est l'abstraction du matériel (HAL, pour Hardware Abstraction Layer en anglais) permet de faire le lien avec le matériel en implémentant les pilotes E/S des périphériques, des contrôleurs de composants, les routines d'interruption (ISR, pour Interrupt Service Routine).
- La couche système d'exploitation qui permet de porter l'application sur l'architecture (gestion de ressources, communication et synchronisation, ordonnancement).
- La couche application.

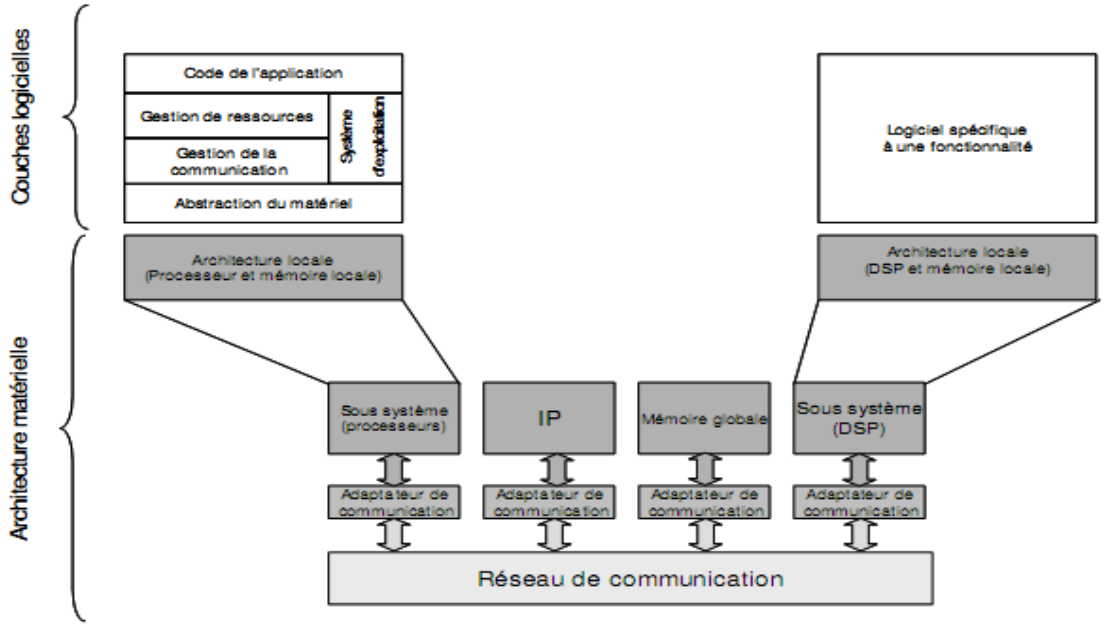

Figure 1. Architecture typique d'un système multiprocesseur hétérogène

4. Principales étapes de la conception des MPSoCs

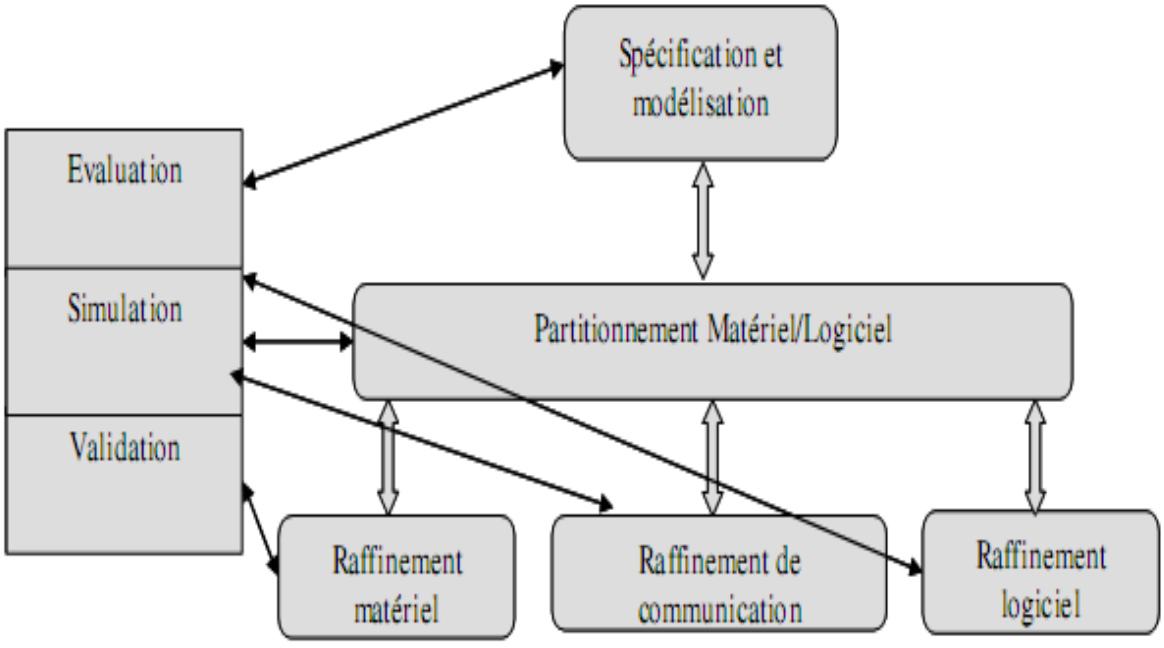

Figure 2. Flot de conception des MPSOC

Il s'agit essentiellement de la spécification, la modélisation, le partitionnement, le raffinement du logiciel, le raffinement du matériel, la génération d'interface entre le logiciel et le matériel (i.e raffinement des communications) et la génération de code ou le prototypage. La succession de ces étapes forme le flot typique d'une approche de conception des systèmes multiprocesseurs schématisée dans la Figure2.

Généralement la conception des systèmes sur puce part d'une spécification unique décrivant l'architecture et/ou le comportement du système à concevoir [6]. Après cette phase de spécification survient l'étape de partitionnement du système ayant pour but de décomposer ce dernier en trois parties :

- Une partie matérielle implémentée sous forme de circuits (FPGA, ASIC, ...).
- Une partie logicielle implémentée sous forme d'un programme exécutable sur processeur, par exemple à usage général.
- Une interface de communication entre les parties matérielles et logicielles.

Les trois parties obtenues doivent ensuite être vérifiées et validées avant de passer à la phase du raffinement et d'implémentation. Il est nécessaire de faire des retours aux étapes précédentes (feed-back), plus précisément à l'étape de partitionnement, tant que l'architecture obtenue ne répond pas aux contraintes auparavant fixées.

4.1.Spécification et modélisation

La spécification est le point de départ du processus de conception des systèmes sur puce. Cette étape consiste, en général, à décrire les fonctionnalités du système à concevoir ainsi que toutes les contraintes qu'il doit satisfaire sans se soucier du découpage matériel/logiciel qui en suit.

4.2.Vérification

La vérification de la conception est un processus de test des fonctionnalités et des performances de l'architecture à implémenter. Les outils de conceptions offrent plusieurs méthodes pour ce but :

· La simulation fonctionnelle et temporelle.
· L'analyse temporelle statique.
· La vérification sur le circuit.

Pour vérifier le design, une simulation fonctionnelle ou temporelle peut être exécutée.

Un processus de *Back-Annotation* doit avoir lieu avant la simulation temporelle. Avant cette phase, la description physique du design doit être traduite en design logique compréhensible par le simulateur. Cette tâche est appelée *Back-Annotation.*

4.2.1. Simulation fonctionnelle

La simulation fonctionnelle ou comportementale détermine si la logique du design est correcte avant la phase d'implémentation. Ce type de simulation peut avoir lieu tôt dans le processus de conception du système. Et puisque les informations temporelles ne sont pas disponibles à ce moment, le simulateur teste la logique en utilisant des délais élémentaires comme unités. Le simulateur utilisé (ModelSim) est un simulateur intégré dans l'environnement de développement de Xilinx : le passage entre les outils de conceptions (éditeurs HDL ou schématiques) et le simulateur se fait automatiquement sans besoin d'utilisation intermédiaire d'outils de translation.

4.2.2. Simulation temporelle

La simulation temporelle examine le temps d'exécution du design dans les pires conditions. Ce processus peut avoir lieu après le mapping, le placement et le routage du design. A ce moment là, tous les délais du design sont bien connus. La simulation temporelle est très importante parce qu'elle peut vérifier les relations temporelles et détermine les chemins critiques du design dans les pires conditions. Avant la simulation temporelle, il faut passer par le *Back-Annotation process* déjà mentionné.

4.3.Implémentation

L'implémentation consiste à la réalisation physique du matériel (par la synthèse) et du logiciel exécutable (par la compilation).

Les étapes de cette phase d'implémentation sont principalement :

- Le partitionnement matériel/logiciel
- Le raffinement et la synthèse du code pour la partie matérielle et la partie logicielle

4.3.1. Partitionnement logiciel/matériel

La phase de partitionnement a pour but d'assurer la transformation des spécifications du système en une architecture composée d'une partie matérielle et d'une partie logicielle. Cette phase de partitionnement consiste donc à déterminer les parties du système qui seront réalisées en matériel et celles qui seront réalisées en logiciel ainsi que l'interface entre ces différentes parties.

4.3.2. Raffinement

L'étape de raffinement consiste à transformer les spécifications fonctionnelles en descriptions directement implantables sur les composants matériels et logiciels de l'architecture cible.

Son rôle est d'assurer la conception physique de la partie matérielle (par synthèse de haut niveau), la génération du code exécutable correspondant à l'implémentation de la partie logicielle (par la compilation) ainsi que le raffinement des interfaces matériel/logiciel.

4.3.2.1.Le raffinement logiciel

Le raffinement logiciel consiste à la synthèse et la génération d'un code exécutable correct et efficace correspondant à l'implémentation de la partie logicielle à partir d'une spécification de haut niveau. Les intérêts de ce raffinement logiciel résident dans la diminution du coût de développement et surtout l'augmentation de la fiabilité du code généré. La complexité de cette étape dépend du type de processeur utilisé.

4.3.2.2. Le Raffinement du matériel

Le raffinement matériel consiste à transformer la spécification de haut niveau de l'application vers un circuit électrique. Cette opération est appelée la synthèse du matériel. On distingue généralement deux niveaux de synthèse matérielle : synthèse logique et synthèse comportementale. La synthèse logique consiste en la transformation d'une description de niveau RTL en un réseau de portes logiques interconnectées qui réalise les fonctionnalités souhaitées. La synthèse comportementale consiste quant à elle en la transformation d'une description comportementale faite en langage de haut niveau (un algorithme) vers une architecture décrite au niveau RTL, composée d'une partie chemin de données et d'une partie chemin de contrôle.

4.3.2.3. Le Raffinement d'interface ou raffinement des communications

Le raffinement des communications est la réalisation des interfaces de communications entre les différentes ressources.

5. La communication dans les MPSOC

La multitude des composants qui intègrent l'architecture multiprocesseur exige une communication entre les processeurs et des moyens pour la gestion de leurs accès concurrents

aux mémoires d'où la nécessité d'un réseau d'interconnexion sur puce appelé NOC (*Network On Chip*).

Les réseaux sur puce ou NOC sont susceptibles de proposer des solutions efficaces aux problèmes d'intégrations complexes des systèmes sur puce [18].

La difficulté de la conception d'un NOC réside dans un compromis entre une Qualité de Service optimale, une bande passante élevée, une latence faible, une flexibilité, une extensibilité d'utilisation importante, et une possibilité de réutilisation de la conception, tout en limitant la consommation d'énergie et de surface dans la puce. Le coût et les caractéristiques des réseaux sur puce dépendent des applications considérées [19].

6. Conclusion

Dans ce chapitre, nous avons présenté les systèmes multiprocesseurs sur puce ainsi que les principales étapes de leur conception. Enfin, nous avons évoqué la communication dans les MPSOCs qui va être détaillée dans le chapitre qui suit.

Chapitre2 : Etude des architectures de Réseaux sur Puce

1. Introduction

Dans ce chapitre, nous présentons tout d'abord les propriétés des réseaux d'interconnexion, leurs performances. Puis, nous classifions et détaillons quelques topologies de ces réseaux.

2. Propriétés des réseaux d'interconnexions

Cette partie présente une description détaillée des caractéristiques des NOCs. Pour cela, nous définissons quelques paramètres tels que le coût d'interconnexion, la latence, le débit, la fiabilité…

2.1.Le coût d'interconnexion

Ce coût se mesure en surface d'occupation des ressources (blocs logiques, mémoires…) sur l'FPGA, et l'énergie consommée d'un circuit.

2.1.1. Le coût en surface sur FPGA

La surface est une préoccupation essentielle pour tous les concepteurs de systèmes intégrés sur FPGA. Malgré leur complexité croissante, ils doivent rester très compacts, pour des raisons de rendement de fabrication. Un réseau sur puce utilise un grand nombre de routeurs qu'il faut interconnecter par plusieurs milliers de fils. Le problème topologique n'est pas simple à résoudre. De plus, tous les composants du NOC sont cadencés par la même horloge. La faisabilité topologique d'une telle implémentation centralisée mérite d'être étudiée de façon approfondie. Il faudra en particulier déterminer si la capacité d'FPGA disponible permet effectivement de router le *design* sans agrandir de façon significative la surface sur FPGA utilisée par le réseau sur puce. D'autre part, le coût de l'extensibilité en terme de surface doit être quantifié. C'est-à-dire que lorsque le nombre de composants interconnectés augmente, la surface occupée par le réseau sur puce doit être évaluée.

2.1.2. Energie consommée

L'énergie est facteur aussi important que la surface pour tous concepteurs de systèmes intégrés sur FPGA. Elle comporte une partie dynamique et une autre statique qui est négligeable devant la première. L'énergie dépend essentiellement de le complexité de circuit, de la fréquence, de la tension et de l'application

2.2.Les performances d'un NOC

Les performances d'un réseau sur puce représentent l'efficacité du réseau à acheminer les données d'un composant vers un autre. Deux métriques évaluent les performances d'une interconnexion à savoir : la latence moyenne mesurée en cycles d'horloge et le débit mesuré en mots par unité de temps.

2.2.1. La latence

La latence est le temps écoulé entre le moment où le message transmis est initialisé jusqu'au moment où il est acquitté. Deux hypothèses sont possibles :

- Il s'agit d'une opération d'écriture, la latence correspond alors au temps entre l'émission du paquet du processeur et sa réception par la mémoire.
- Il s'agit d'une opération de lecture, la latence correspond alors au temps entre l'émission du paquet et sa réception par le processeur en passant par la mémoire comme intermédiaire.

2.2.2. Le débit

Le débit est la quantité maximale d'informations transitant dans une interconnexion par unité de temps (cycle d'horloge). Il mesure la capacité d'un canal à transmettre des données sous forme numérique, c'est à dire la vitesse de transfert des données. Le débit peut être mesuré en mots par seconde ou en mots par cycle d'horloge selon que l'on considère un temps absolu (en seconde) ou relatif (en fréquence).

2.2.3. La fiabilité d'interconnexion

2.2.3.1.La flexibilité

La flexibilité exprime la capacité d'intégration d'une architecture d'interconnexion à un système. Cette flexibilité est composée de deux paramètres : la portabilité du l'architecture et son extensibilité décrite ci-après.

Le temps de conception d'un élément est un paramètre important dans l'élaboration d'un composant. En effet, la durée de vie des composants étant de plus en plus faible, leur

conception doit se faire en un temps réduit. Les concepteurs utilisent donc de plus en plus de librairies afin de les aider à gagner du temps. L'interconnexion d'un système sur puce doit être conçue dans la même optique : elle doit être facilement réutilisable et donc reconfigurable.

2.2.3.2.L'extensibilité

L'extensibilité correspond à sa capacité à évoluer en fonction du nombre de composants.
Si l'ajout de blocs fait augmenter les performances et le coût de l'interconnexion de façon proportionnelle, l'architecture est alors extensible. En revanche, si l'ajout de cœurs supplémentaires conduit à un goulot d'étranglement du système, le composant ne pourra pas être utilisé. Il faut mesurer l'évolution de son coût et de ces performances en fonction du nombre de composants connectés.

3. Topologies des réseaux sur puce

Le critère incontournable de classification des réseaux sur puce (NoCs) est la topologie [16]. Cette caractéristique spécifie l'organisation physique du réseau. Elle définie donc comment les nœuds et les liens sont connectés entre eux. De nombreuses topologies sont envisageables. La Figure 3 montre les plus couramment utilisées.
Ces topologies sont dîtes régulières compte tenu de leur loi de construction géométrique.
Les topologies présentent plusieurs paramètres [23]. Les plus courants sont :
– Le diamètre : C'est le nombre maximal de liens qui séparent deux ressources quelconques du réseau (en considérant les plus courts chemins).
– La distance moyenne : C'est le nombre moyen de liens entre deux ressources.
– La connectivité : C'est le nombre de voisins directs des nœuds dans le réseau.
– La largeur de bissection. C'est le nombre minimal de liens qu'il faut couper pour séparer le réseau en deux parties égales (plus ou moins un nœud). Cela permet d'évaluer le coût de transférer les données d'une moitié du réseau à l'autre.

3.1.Communications point-à-point

Ces communications sont les plus simples à réaliser. Lorsque l'application est structurée on pourra souvent combiner plusieurs communications point-à-point en une seule communication collective. Figure3(e)

3.2.Réseau en étoile

Les réseaux en étoile (“star-tree” et non “star-graph” qui est un autre type de réseau) ont, entre chaque source et destination, un chemin unique de longueur maximale 2. Le routage est très simple et consiste à passer par un seul nœud intermédiaire soit le nœud central (Figure3(c)).

3.3.Mesh et tore

Parmi les topologies maillées à deux dimensions, le choix entre le type maillage simple (mesh), tore est souvent discuté [3] [25]. En effet, les interconnexions en tore offrent une meilleure utilisation des ressources réseau, puisque pour un même nombre de nœuds, il y a plus de liens ce qui a pour effet de diminuer le diamètre et d’augmenter la bande-passante (largeur de bissection double). En contre partie, la structure est moins régulière que pour un réseau maillé et les fils utilisés pour boucler le réseau sont plus longs et pénalisent donc les performances des liens (voir Figure 3(a, b)).

3.4.Arbre élargi

Les réseaux de types arbre élargi ou anneau à corde ont un diamètre plus petit que les réseaux maillés. Ils permettent donc de réduire la latence mais leurs structures sont moins régulières. Une étude comparative a été effectuée [4] et on observe qu’avec un trafic uniformément réparti, le débit atteignable avant saturation est plus élevé avec ces topologies par rapport aux topologies maillées. Cependant, l’étude montre que les topologies maillées exploitent avantageusement le fait que dans les SoC le trafic sera majoritairement local (Figure3(d)).

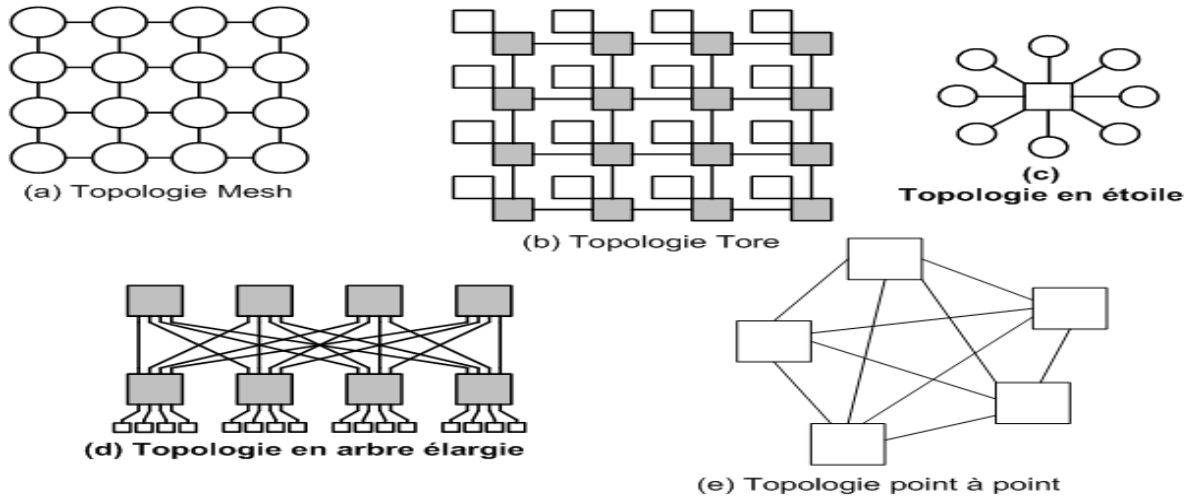

Figure 3. Quelques topologies des réseaux sur puce

4. Classification des réseaux pour MPSoC

Cette partie introduit les différents types de réseaux d'interconnexion en montrant les avantages et inconvénients de chacun de ceux-ci afin de justifier le choix de celui de notre architecture.

Cette classification comporte 4 classes majeures: les réseaux à ressources partagées, les réseaux directs, les réseaux indirects et les réseaux hybrides. [17]

4.1.Réseaux à ressources partagées-bus

La structure d'interconnexion la moins complexe est celle dans laquelle le moyen de transmission est partagé par tous les éléments. Dans ce genre de réseau, un seul élément peut transmettre à la fois. Chaque élément se trouvant sur le réseau dispose de circuits de requête, d'envoi et de réception afin de pouvoir manipuler les données et les adresses.

Un bus partagé se compose notamment :

- D'un bus de données partagé (unique et connecté à tous les éléments du système).
- D'un bus d'adresses ayant les mêmes caractéristiques.
- D'un élément particulier du système, appelé arbitre du bus, connecté par des liaisons point à point à tous les autres éléments.

Un bus partagé, en particulier sa topologie, possède plusieurs avantages. En effet, il peut supporter directement le modèle de communication par adressage mémoires des CPUs.

De plus, le mode de fonctionnement de l'arbitre est simple et on peut facilement l'adapter aux différentes applications requises. Ainsi que, le concept de bus est parfaitement maîtrisé par les concepteurs de matériel, ce qui favorise son usage et sa large diffusion.

En revanche un bus partagé implique de nombreux problèmes de mise à l'échelle. Ceci s'avère être inacceptable, surtout pour le débit global. Un arbitrage plus sophistiqué et des mémoires caches ne peuvent qu'alléger cette contrainte assez forte.

Ce problème peut être résolu de deux façons : soit en augmentant la largeur du bus, soit en augmentant la fréquence d'horloge. Les deux solutions ne sont pas satisfaisantes car elles impliquent des problèmes électriques auxquels il est coûteux de remédier convenablement.

4.2.Réseaux directs

Les réseaux directs (aussi appelés réseaux point à point) s'adaptent très bien à un très grand nombre d'éléments. Ils consistent en un ensemble de points qui peuvent chacun être connecté vers un petit sous ensemble de nœuds du réseau.

Le contrôleur de transfert est un composant important du nœud; il gère tous les messages échangés entre différents nœuds. Si on augmente le nombre de nœuds dans le système, la bande passante totale de communication et les capacités de calcul augmentent aussi. Ceci permet une extensibilité très élevée.

Un réseau direct est principalement caractérisé par trois facteurs: sa topologie, ses transferts et son multiplexage. La topologie définit comment les nœuds sont interconnectés. Pour des réseaux directs, la topologie idéale connecte chaque nœud vers tous les autres nœuds. Un message ne doit donc passer par aucun nœud intermédiaire

Quand un message arrive dans un nœud intermédiaire, un mécanisme de multiplexage détermine comment et quand les multiplexeurs internes sont activés, par exemple pour connecter une entrée à une sortie.

4.3.Réseaux indirects

Dans ce type de réseau, plutôt que de fournir des liaisons directes entre nœuds, la communication entre deux nœuds se fait à travers de commutateurs. Chaque nœud est connecté à un multiplexeur disposant d'un certain nombre de ports composé d'un lien d'entrée et d'un lien de sortie

Un réseau indirect est caractérisé par trois facteurs: la topologie, le routage et l'arbitrage.

Un réseau indirect ou dynamique est un réseau dont la topologie peut varier au cours de l'exécution d'un programme parallèle ou entre deux exécutions de programmes ; il s'agit du comportement du réseau.

4.3.1. Les réseaux à zéro étage (réseaux connectés en bus)

Le réseau en bus permet d'établir dynamiquement un et un seul lien direct à la fois entre n'importe quelle paire de noeuds source et destination (Figure4).

En fait, lorsque la complexité du traitement à réaliser est limitée i.e. lorsque le nombre d'éléments du système est moyen et que le taux d'utilisation du bus par ces éléments n'est pas trop élevé, un réseau en bus ne pose pas de problèmes particuliers tout en étant simple et relativement peu coûteux.

Par contre, lorsque les contraintes sont plus exigeantes et donc que le nombre d'éléments (Processeurs) augmente ainsi que leur taux d'utilisation du bus, il se produit un problème de blocage (*bottleneck*) **[7]**. Celui-ci résulte du fait que seulement deux éléments peuvent communiquer entre eux à chaque instant.

De plus, une défaillance du bus ("failure") est toujours catastrophique car cela signifie le blocage de la communication entre les éléments du système.

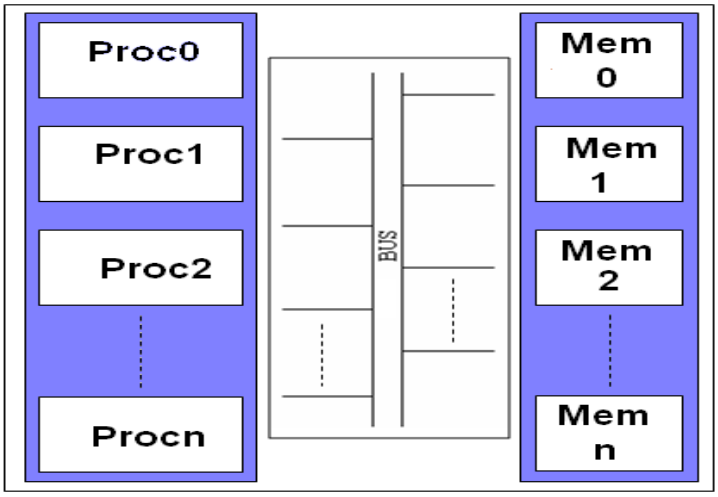

Figure 4. Réseau bus partagé

4.3.2. Les réseaux crossbar

Dans un réseau à crossbar (Figure5), n'importe quel élément peut être connecté à un autre élément de sorte que plusieurs communications peuvent se faire simultanément. Une nouvelle connexion peut être réalisée tant que les ports d'entrée et de sortie soient libres. Les crossbars sont surtout utilisés dans les petits systèmes multiprocesseurs (dans les routeurs des réseaux directs et dans les réseaux indirects). Un crossbar comporte N entrées et M sorties, permettant jusqu'à min{N,M} connexions point à point sans collision.

Quand deux ou plusieurs éléments essayent d'accéder au même élément, un arbitre laisse l'accès à un seul élément tandis que les autres doivent attendre. Dans un crossbar, l'arbitre est distribué au niveau de tous les commutateurs ayant la même sortie. Cependant, cet arbitre est nettement moins complexe que dans le cas des bus.

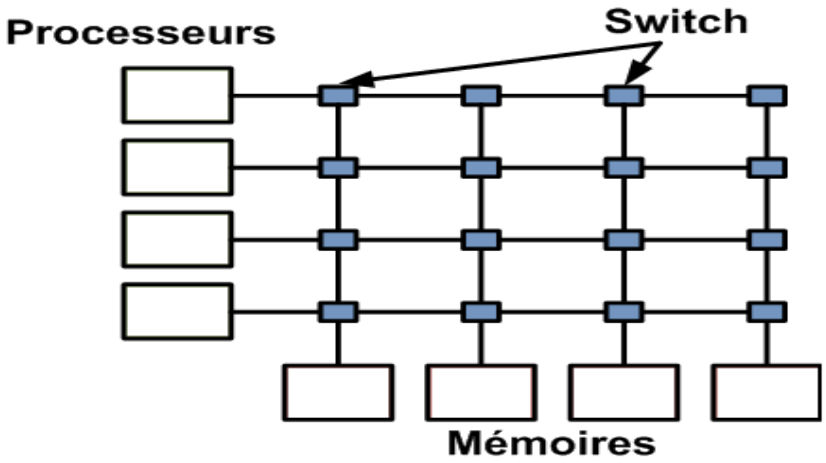

Figure 5. Réseau Crossbar

4.3.3. Les réseaux d'interconnexion multi-étages

4.3.3.1.Classification des MINs

Il existe divers critères de classification des réseaux multi-étages [10]. En effet, on peut distinguer plusieurs classes de MINs suivant les types de commutateurs utilisés et / ou les types de permutation. Avant de présenter la classification choisie pour les MINs, il est important de commencer par explorer quelques définitions. Figure7

- *banyan :* il offre un chemin unique entre n'importe quelle entrée et n'importe quelle sortie.

- Un réseau d'interconnexion est dit uniforme : "uniform MIN" lorsque tous les éléments de commutation (SE) d'un étage sont de même degré.

-Un réseau est dit rectangulaire si le nombre d'entrées est égal au nombre de sorties.

-Un réseau est dit carré "Square MIN", lorsqu'il est de degré r et il est construit à partir des SEs de taille r.

-Réseau avec Blocage (Blocking) :

La connexion entre les entrées libres et les sorties n'est pas toujours possible à cause des conflits avec les connexions existantes. Typiquement, il y a un chemin unique entre chaque paire d'entrée/sortie, de ce fait le nombre de *switch* et d'étages sera réduit au minimum.

- *non bloquant :* Si de toute entrée inactive il existe toujours un chemin vers toute sortie inactive. On peut donc effectuer n'importe quelle permutation en cours d'exécution. Un exemple populaire de réseau sans blocage est le réseau de Clos [11].Figure6

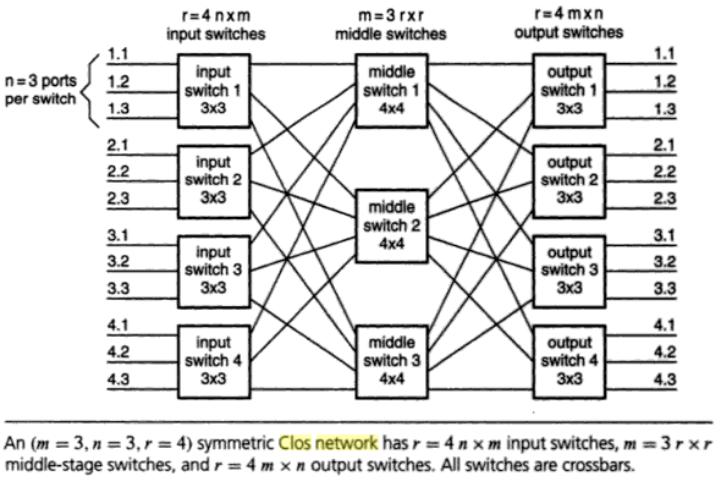

Figure 6. Réseau clos

Le réseau Banyan est une structure de commutation spatiale définissant un schéma d'interconnexion avec une seule voie d'accès entre les entrées et les sorties. Cette topologie est réalisée à partir d'éléments de commutation (crossbar) a x b.

Une autre caractéristique qui donne à ce réseau un grand intérêt, est sa capacité de routage automatique (*self Routing*) qui consiste à déterminer la décision de routage en utilisant l'adresse destination.

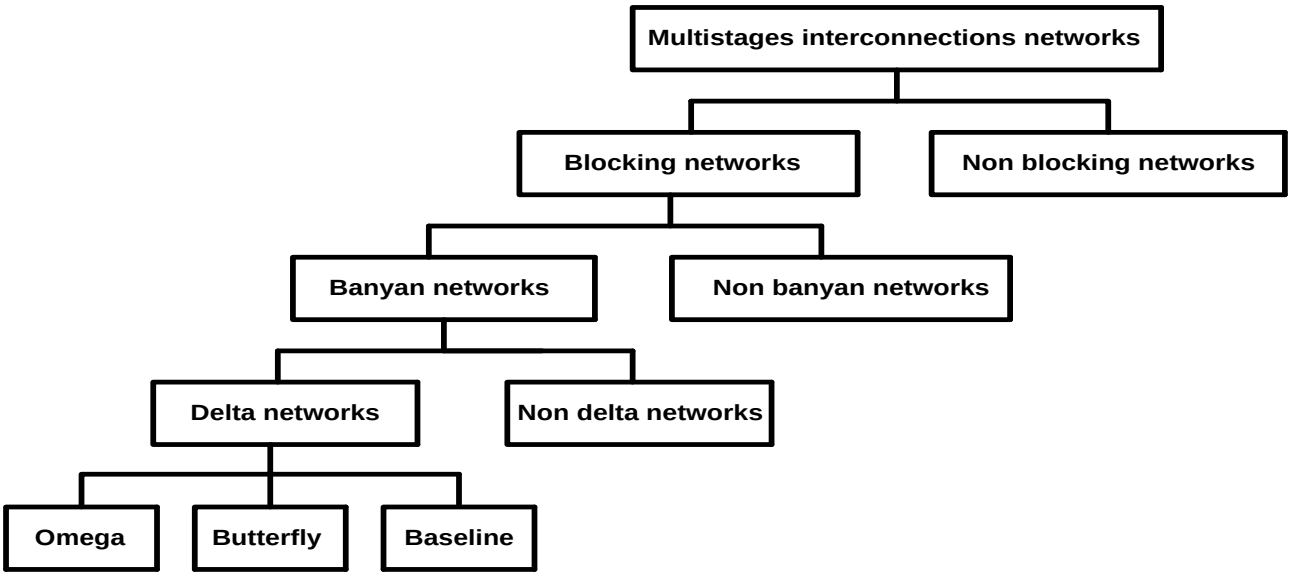

Figure 7. Classification des MIN

4.3.3.2.Topologie des réseaux MINs

Un grand nombre de réseaux MIN NXN (de N processeurs d'entrée vers N processeurs ou vers N mémoires de sortie) sont formés de n = $\log_2 N$ colonnes ayant chacune N/2 commutateurs 2x2 i.e. avec 2 entrées et 2 sorties [5] [26].

Le nombre total de commutateurs est donc d'ordre $\log_2 N$, comparativement à N^2 pour les commutateurs matriciels.

On représente normalement un tel réseau MIN par une matrice de n colonnes de N/2 commutateurs (rangées) reliant les N nœuds d'entrée (source) placés à gauche de la dernière colonne aux N nœuds de sortie (destination) placés à droite de la première colonne.

On numérote normalement les colonnes de commutateurs de 0 à n-1 en allant de la première (à droite) à la dernière (à gauche) (figure8) [9].

Un étage d'un réseau MIN comprend une colonne de commutateurs et les liens d'entrée de ces commutateurs. Le numéro d'un étage est le même que celui de la colonne de commutateurs.

Un réseau MIN comprend donc n étages suivis des liens de sortie de la dernière colonne de commutateurs.

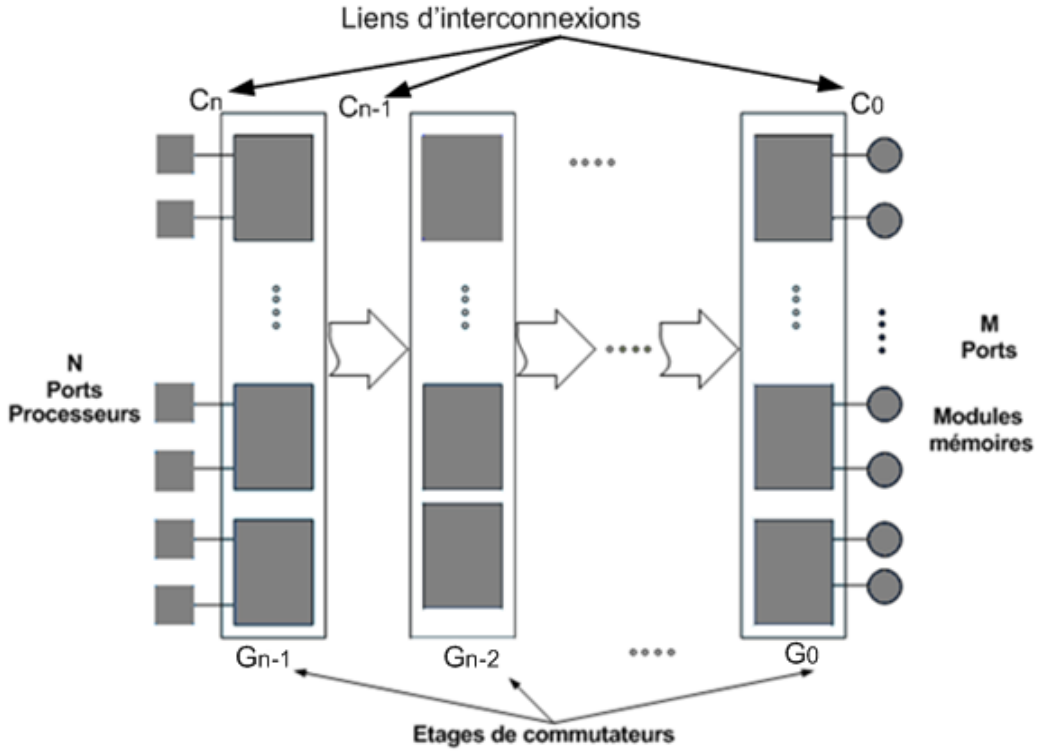

Figure 8. Architecture générique du réseau MIN

4.3.3.3.Les réseaux Delta

D'après la définition formelle de Patel [12], les réseaux Delta sont des réseaux d'interconnexion multi étages de type Banyan, qui comprennent axb crossbars, dont tous les ports d'entrées et de sorties sont connectés. Le nombre total de crossbar pour construire un réseau Delta est :

$$\sum a^{n-i} n^{i-1} = (a^n - b^n) / (a-b) \quad a \neq b$$
$$= nb^{n-1} \quad a=b$$

Les réseaux Delta sont des réseaux d'interconnexion multi étages basés sur des crossbar axb. Ils sont dépourvus de contrôle, et peuvent donc engendrer des pertes de données. L'avantage principal des réseaux Delta est qu'ils sont moins complexes que les fulls crossbar. Dans le réseau Delta pour N entrées et N sorties avec $N=k^n$, on aura n étages dont chacun contient N/k commutateurs.

Le réseau delta est caractérisé par un accès total : les types de permutations utilisées pour construire les étages de connexion, doivent garantir l'accès total au réseau. Ainsi, par une configuration correcte des commutateurs à chaque étage, n'importe quelle entrée doit être capable d'atteindre n'importe quelle sortie.

Aussi, le réseau delta se caractérise par sa capacité de routage automatique des messages depuis la source vers la destination. En ce sens le canal de sortie choisi à chaque commutateur ne dépend pas de la source mais seulement de la destination.

Une autre propriété pour les réseaux delta est l'équivalence topologique : il a été prouvé dans [13], [14] et [15] que tous les MINs Delta sont équivalents du point de vue topologique. Il suffit de réordonner les positions des commutateurs sans rompre les connexions pour passer d'un réseau à un autre.

Il existe plusieurs formes (types) de réseaux Delta, dépendamment de leurs connexions. Les réseaux les plus utilisés sont:

- Omega networks
- Butterfly Networks
- Baseline Networks
- (Generalized) Cube Networks
- Flip Networks
- Reverse Butterfly Networks
- Reverse Baseline Networks
- Indirect Binary N-Cube Networks

4.3.3.4.Réseau Oméga

Le réseau Oméga est constitué de 2^k entrées et 2^k sorties, k représente le nombre d'étages.

Chaque étage contient 2^{k-1} *switch* 2x2 (Figure 9).

Le réseau oméga est un exemple important de réseau multi-étage de type "*shuffle/exchange*".

Il est formé d'une suite d'étages de type *shuffle/exchange* terminée par une permutation identité.

Dans les réseaux oméga la connexion C_i $(0 < i \leq n)$ entre les étages est décrite par la formule de permutation circulaire suivante: $\sigma^k (x_{n-1}\ x_{n-2} \ldots x_1\ x_0) = x_{n-2} \ldots x_1\ x_0\ x_{n-1}$

Signification : décalage cyclique de tous les bits de l'index d'une position vers la gauche

La connexion C_0 est définie par I

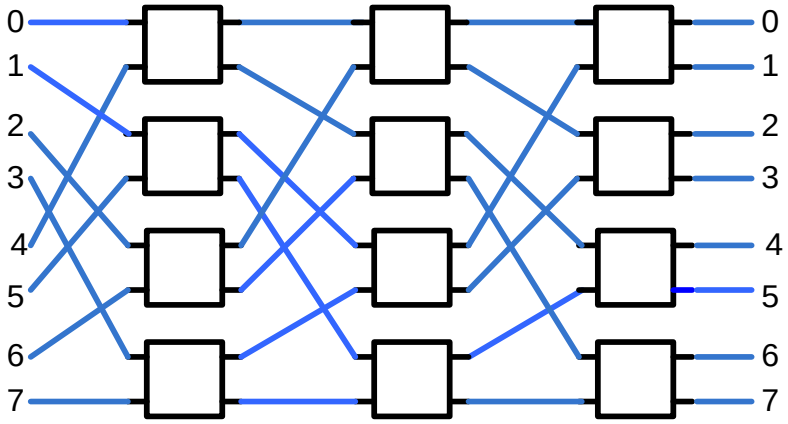

Figure 9. Réseaux Omega 8x8

- **Perfect-shuffle et shuffle-exchange *:***

Dans une permutation *perfect-shuffle* de N = 2^n noeuds, chaque noeud est donc relié au noeud obtenu par un décalage cyclique de 1 bit vers la gauche, de son adresse binaire.

Un étage de type mélange et échange ("*shuffle/exchange*") consiste en une permutation *perfect-shuffle* suivie d'une colonne de commutateurs 2x2 [7].

Cette colonne de commutateurs permet de réaliser la permutation d'échange (Figure10).

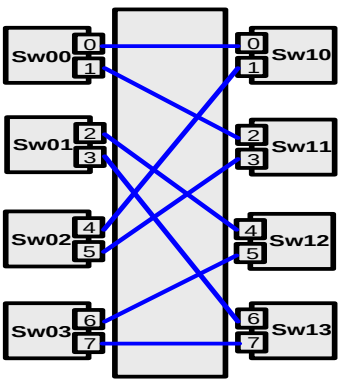

Figure 10. Perfect Shuffle (N est égal à 8 dans cet exemple)

4.3.3.5.Réseau Butterfly

Un réseau butterfly (réseau papillon) de dimension r, est un réseau composé de $(r + 1)2^r$ noeuds organisés en 2^r lignes de r +1 niveaux (Figure11).

Dans les réseaux butterfly la connexion C_i entre les étages est décrite par la formule suivante

$\sigma^k \quad C_n$

$\beta_i^k \quad C_i \ (1 \leq i \leq n-1)$

$I \quad C_0$

- $ß_i^k (x_{n-1}\ x_{n-2} \ldots x_{i+1}\ x_i\ x_{i-1} \ \ldots \ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots x_{i+1}\ x_0\ x_{i-1} \ \ldots \ x_1\ x_i$
- β_i^k : échange entre le i^{ème} bit et le bit 0

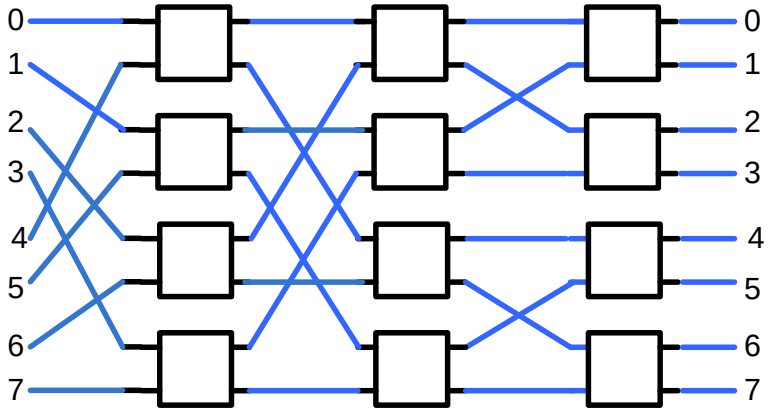

Figure 11. Réseau Butterfly 8x8

4.3.3.6.Réseau Baseline

Un réseau MIN Baseline [7] de n étages utilise une permutation identité à l'entrée de l'étage n, une permutation δ_i^k à l'entrée des étages i=1 à n-1 et une permutation identité à l'étage0 (Figure12).

La *ith baseline* permutation dans un réseau Baseline, est définie comme suit:

$\delta_i^k \ (x_{n-1}\ x_{n-2} \ldots x_{i+1}\ x_i\ x_{i-1} \ \ldots \ x_1\ x_0) = x_{n-1}\ x_{n-2} \ldots x_{i+1}\ x_0\ x_i\ x_{i-1} \ \ldots \ x_1$

avec $1 \leq i \leq n-1$

Signification : décalage cyclique d'une position vers la droite, des (i+1) bits les moins significatifs de l'index.

On aura alors cette formule :

$I \quad C_n$

$\delta_i^k \quad C_i \ (1 \leq i \leq n-1)$

$I \quad C_0$

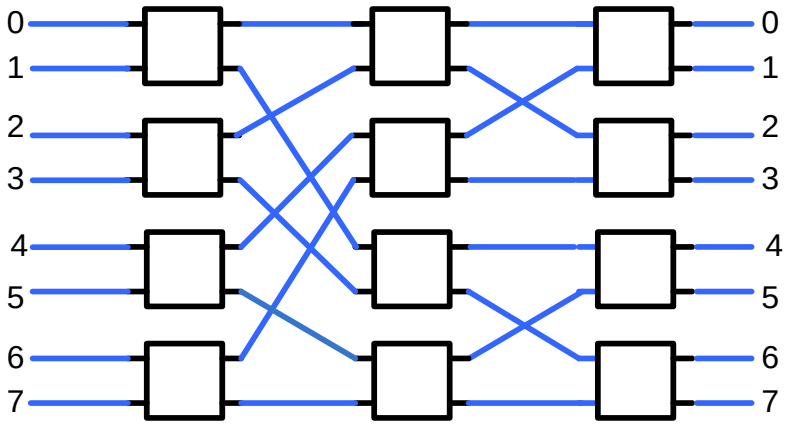

Figure 12. Réseau Baseline 8x8

4.3.3.7.Commutateurs 2x2

Les réseaux Delta utilisent des commutateurs 2x2. Pour un commutateur 2x2, 16 états ou configurations sont possibles en considérant que le message sur chaque canal d'entrée peut être transmis ou non vers un des canaux de sortie.

Dans les réseaux MIN, seulement quatre de ces états sont significatifs. Chaque commutateur se place donc dans une des quatre configurations de base qui permettent de transmettre sans ambiguïté (Figure13).

Les messages arrivant sur les canaux aux entrées :

• Directe : L'entrée IN0 passe par la sortie OUT0 et IN1 passe par OUT1

• En croisé : L'entrée IN0 passe par la sortie OUT1 et IN1 passe par OUT0

• Vers le haut : Les entrées IN0 et IN1 passent par la sortie OUT0.

• Vers le bas : Les entrées IN0 et IN1 passent par la sortie OUT1.

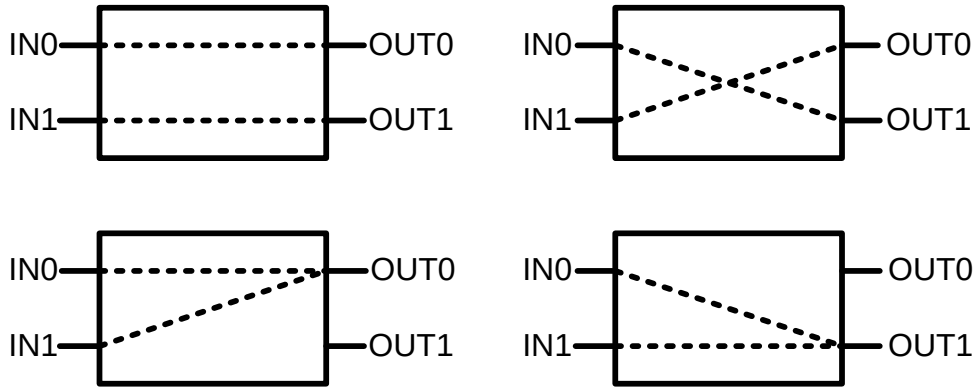

Figure 13. Les différents états du *switch* 2x2

5. Conclusion

Dans ce chapitre, nous avons passé en revue les différentes notions relatives aux réseaux multi-étages (MINs). Nous avons insisté en particulier sur les réseaux de la famille Delta. Vu les propriétés intéressantes que possède cette dernière classe de réseaux, notre travail de conception et modélisation se focalisera sur cette famille de réseaux.

Chapitre3 : Conception de réseaux multi-étage reconfigurable sur puce

1. Introduction

Dans ce chapitre, on va présenter l'environnement de travail : le langage de spécification et les outils de conceptions utilisés. Par la suite, on passe à la conception des réseaux multi-étages configurable sur puce. Enfin, on donne une estimation des performances en terme de surface sur FPGA et de latence.

2. Plateforme choisie

La carte de développement VIRTEX4 comporte un FPGA (**XC4VLX200-FF1513**) avec plusieurs périphériques [20] **(**Figure14).

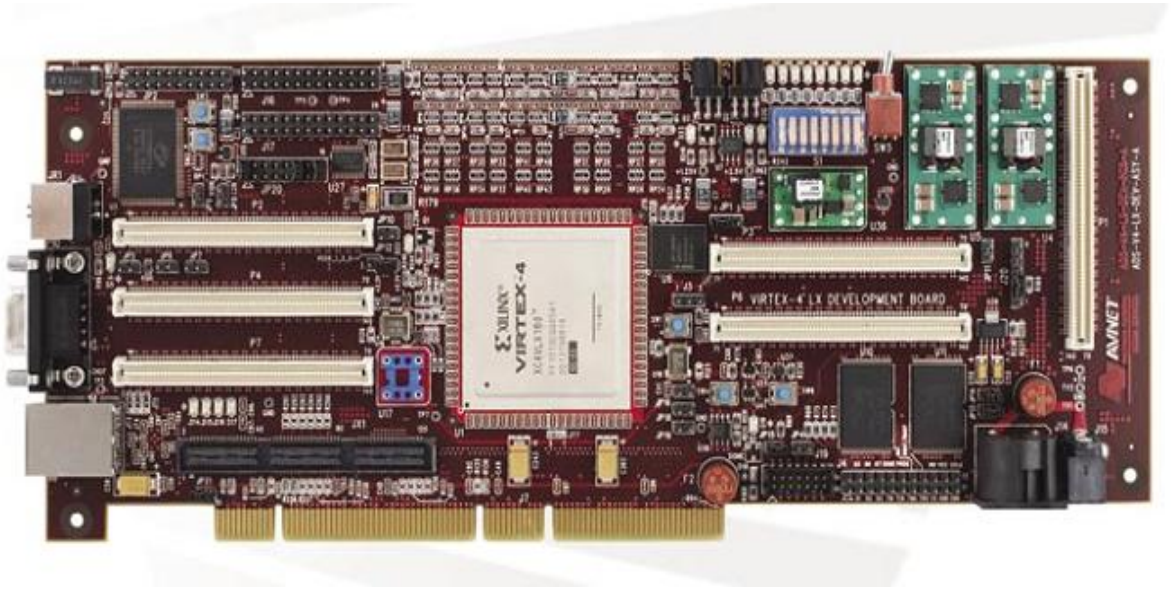

Figure 14. Carte de développement VIRTEX4

Nous citons ci-dessous les spécifications technologiques de la plateforme choisie.
Ci-dessous une liste des principaux périphériques de la carte choisie est présentée:

- Un circuit FPGA VIRTEX4 - LX200-FF1513

- Une mémoire flash de 16 MB et une mémoire DDR SDRAM 128 Mo.
- Deux blocs SRAM.
- Un port série RS-232
- Un port USB 2.0
- Interface Ethernet
- 2 PROM de 32 MB
- Un oscillateur générateur de signal d'horloge (500 MHz).

3. Langage de conception

Les langages de description matérielle supportent les concepts spécifiques aux systèmes matériels tels que les concepts de temps, de parallélisme, de réactivité et de communication interprocessus (signaux et protocoles). Les deux langages de description matérielle les plus connus sont VHDL et Verilog.

Le langage VHDL permet la description de tous les aspects d'un système matériel (hardware system): son comportement, sa structure et ses caractéristiques temporelles.

Le langage VHDL est aussi utilisé pour la synthèse [2], par exemple pour dériver automatiquement un circuit à base de portes logique optimisé à partir d'une description au niveau RTL ou algorithmique.

La description d'un système matériel en VHDL est apte à être simulé. Il est possible de lui appliquer des vecteurs de test, également décrits en VHDL et d'observer l'évolution des signaux du modèle dans le temps.

4. Outils d'analyse et de simulation

Pour notre travail nous avons utilisé l'environnement ISE 9.1i [21] (Figure15) et Modelsim de la société de XILINX.

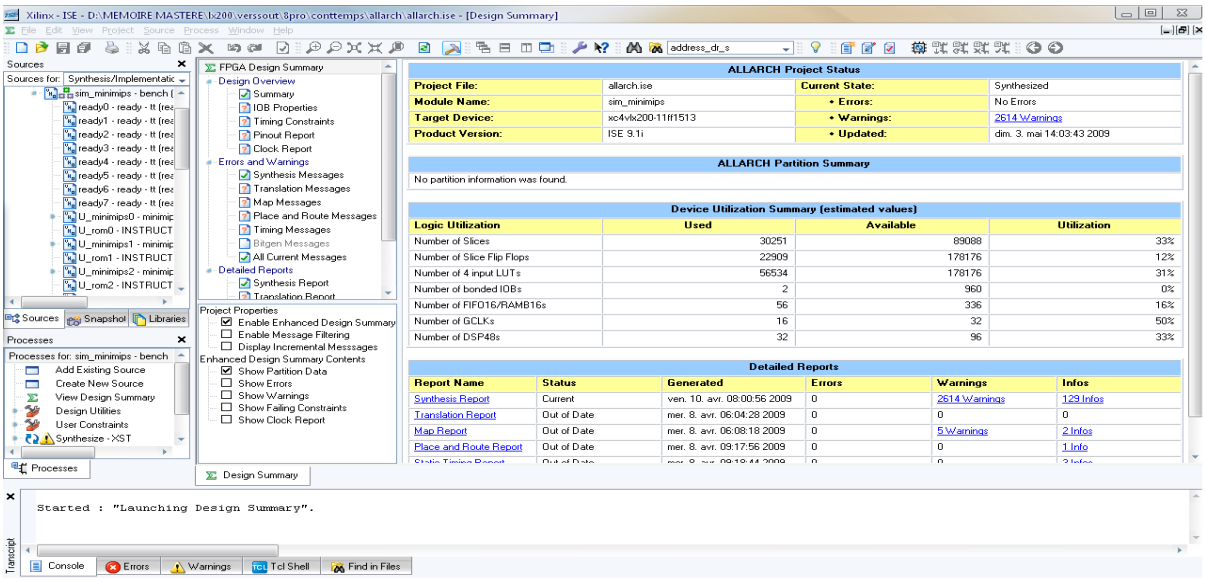

Figure 15. Xilinx ISE Foundation software

Ces outils nous permettent de faire :

- Synthèse logique d'une description en VHDL
- Placement et routage
- Génération des fichiers de programmation du circuit FPGA
- Suivi des comportements des signaux au cours du temps
- Estimation de ressources utilisées sur FPGA

5. Modélisation de réseaux multi-étage reconfigurable sur puce

5.1.Introduction

Dans ce chapitre nous détaillons la conception des réseaux multi-étages de type DELTA dédiés aux architectures multiprocesseurs sur des plateformes reconfigurables FPGA ainsi que leur fonctionnement.

Notre architecture est composée essentiellement des modules suivants:

– Un module de routage qui permet de diriger les informations vers leurs destinations.

– Un module d'arbitrage qui garanti l'exécution des demandes provenant de chaque port d'entrée/sortie.

– Les modules de mémorisation, dans notre cas nous utilisons les FIFO (premier arrivé, premier servi), ils permettent de stoker des informations afin de gérer les conflits dans les réseaux.

– Les modules de connexion qui assurent la connexion entre les étages suivant une topologie bien définie.

Ensuite, nous détaillons l'architecture interne de chaque composant en expliquant son mode de fonctionnement.

5.2. Paquetage des données

Les données échangées entre les nœuds de MIN sont fragmentées en paquets. Ces derniers dépendent des protocoles adoptés dans la conception des NOCs. Le paquet est composé de trois parties (Figure16) :

- Une en-tête : codification de l'opération (lecture, écriture, commande)

Bit r/w : bit qui définit l'action : s'il s'agit d'une écriture (r/w=1) ou d'une lecture (r/w=0).

Bit enable : l'activation ou la commande.

- Un message : la donnée à échanger.
- Une queue : contient l'adresse de la source, l'adresse de la destination et l'adresse dans la mémoire.

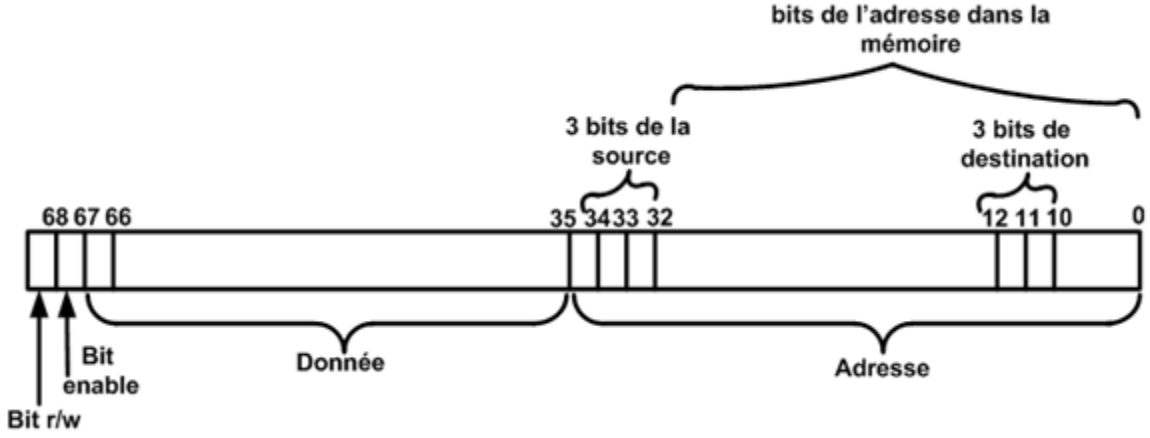

Figure 16. Paquet processeur_mémoire

- Les sources de données

Les paquets circulant à travers le réseau proviennent de plusieurs sources. Ces paquets peuvent être classés en quatre catégories :

- Paquet_processeur_mémoire: demande d'accès à la mémoire (opération de lecture, opération d'écriture).

- Paquet_ mémoire_réponse : ce paquet n'a pas la même forme que le premier paquet envoyé à travers le NOC. Il contient seulement 32 bits de donnée et les bits d'adresse (2,3 ou 4 bits : suivant le nombre de processeurs).

5.3.Composants de MIN

5.3.1. Module *Switch*

C'est le composant qui présente le cœur du réseau multi-étage (Figure17).

Le *switch* est formé d'un couple de FiFOs connectés à l'ordonnanceur.

Son fonctionnement, sa fiabilité influent sur les performances de réseau en terme de consommation de surface sur la puce ou en terme de latence.

Le *switch* possède 2 entrées qui reçoivent les paquets de type processeur_mémoire et les stocke dans les files d'attentes pour être ensuite acheminée par l'ordonnanceur suivant leur destination vers la sortie désirée.

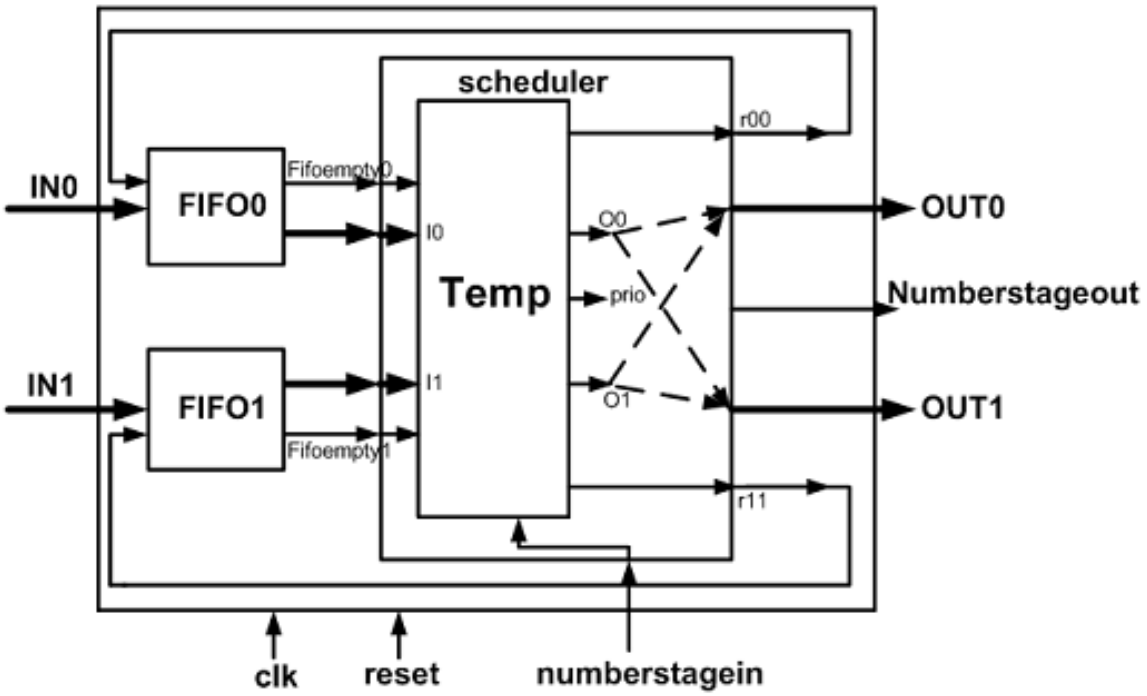

Figure 17. Architecture de Switch

Le *switch* a comme signaux (Figure17) :

- Deux vecteurs « IN0 » et « IN1 » de taille 69 bits et qui ont la forme de paquet décrit précédemment. Ces deux vecteurs vont être stockés dans les FIFO.
- Un vecteur « numberstagein » de taille 4 bits qui est lié au numéro de l'étage pour être utilisé dans l'algorithme de routage.

- Un signal d'horloge « clk».
- Un signal d'initialisation « reset».
- Deux vecteurs « OUT0 » et « OUT1 » qui représentent les paquets après l'arbitrage et le routage.
- Un vecteur « numberstageout » de taille 4 bits qui est égale à « numberstagein-1 »

5.3.2. Module Ordonnanceur : routage, arbitrage et mémorisation temporaire

Il s'agit de l'acheminement des paquets dans le réseau via des routeurs, qui sont chargés de véhiculer des paquets en vue de rejoindre la destination dans un délai minimum.

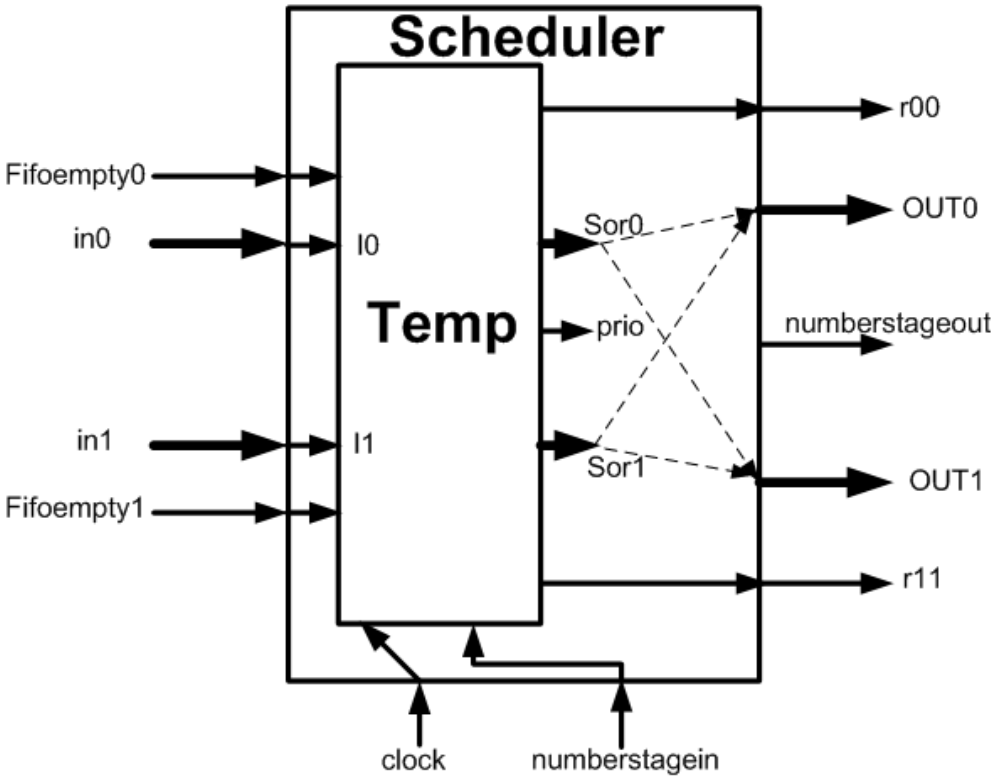

Figure 18. Architecture de l'ordonnanceur

L'Ordonnanceur a comme signaux (Figure18) :

- Deux vecteurs « in0 » et « in1 » de taille 69 bits et qui ont la forme de paquet décrit précédemment. Ces deux vecteurs vont être stockés temporairement dans TEMP pour subir l'arbitrage puis le routage.
- Deux signaux « fifoempty0 » et « fifoempty1 », de taille d'un bit, qui indiquent que les Fifos sont vides.
- Un vecteur « numberstagein » de taille 4 bits qui est lié au numéro de l'étage pour être utilisé dans l'algorithme de routage.
- Un signal d'horloge « clock».

- Deux vecteurs « OUT0 » et « OUT1 » qui représentent les paquets après l'arbitrage et le routage.
- Deux signaux r00 et r11 qui gèrent la lecture à partir des Fifo.
- Un vecteur « numberstageout » de taille 4 bits qui est égale à « numberstagein-1 »

On passe à la description du composant Ordonnanceur et son fonctionnement. En effet ce dernier joue le rôle d'un routeur, arbitre et une mémorisation temporaire.

5.3.2.1.Routage et mémorisation

Le routage est fait en fonction de l'adresse destination. En fait, le routeur redirige le message sur l'étage suivant, afin de s'approcher de la destination finale. Il doit être capable de gérer les conflits de chemin (deux messages désirant emprunter simultanément le même canal) c'est à dire, la mémorisation temporaire d'un message pour l'envoyer plus loin une fois la voie sera libre. L'algorithme de routage, en raison des conflits potentiels peut devenir assez complexe. Il faut être sûr qu'aucun message ne puisse être bloqué dans le réseau.

Pour réaliser le routage dans un réseau Delta, le i ème bit le plus significatif de l'adresse de destination détermine l'activation du commutateur de l'étage i. Ainsi, si ce bit est à 0, le chemin passe par la sortie du haut sur le commutateur. Inversement, si le bit est à 1, la sortie du bas du commutateur est utilisée [22].
Des algorithmes de "self-routing" existent pour la plupart des réseaux MIN.
Pour un réseau oméga par exemple, le routage se fait comme suit : supposons que nous partons du nœud 1 au nœud 5, selon cet algorithme, 5 signifie 101 en binaire, d'ou bas (1), haut (0), bas(1) tel que montré par la Figure19.

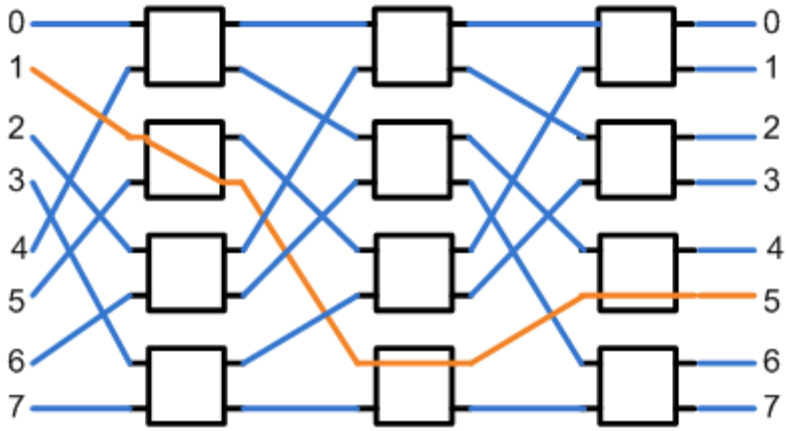

Figure 19. Routage dans le réseau oméga. Le trajet du message de 1 à 5

5.3.2.2.Arbitrage

Le *switch* possède deux ports d'entrées où chacun de ces ports peut demander l'accès à un même port de sortie. Pour cela un arbitrage doit avoir lieu, pour qu'on puisse d'une part, garantir le service des demandes provenant de chaque port d'entrée, et d'autre part éviter les problèmes de type famine (Risque d'affamer les flux les moins prioritaires) [24].

La technique d'arbitrage Round Robin :

Cette technique est basée sur le principe suivant : le port d'entrée ayant la priorité la plus forte aura à la prochaine itération la priorité la plus faible. Pour cela nous avons utilisé une variable statique qui prend les valeurs 1 ou 0 selon le cas où nous avons deux requêtes qui se présentent et qui demandent accès au même port de sortie.

5.3.2.3.Ordonnancement des paquets

-Initialement la mémoire Temp est vide, les signaux r00 et r11 vont être alors mis à 1 pour la remplir.

-Les entrées « entrée0 » et « entrée1 » venant des FIFOs vont être stockées suivant :

- Leurs bits de commande (bits d'activation)
- Si les FIFOs ne sont pas vides (fifoempty=0).

-Une fois que TEMP est pleine, on va affecter ces paquets suivant leurs bits de destination. Plusieurs cas seront présentés : accès par la même sortie ou non, passage par la sortie haut ou par la sortie bas, les deux entrées sont actives ou non.

-Avec cette variété de cas, l'arbitre doit gérer l'accès de chaque entrée à la sortie demandée en utilisant sa technique d'arbitrage et enfin vient le routage de ces entées vers les sorties voulues.

-Dés que TEMP sera vide elle envoie une requête aux FIFOs : r00 et r11 s'activent pour lire une autre fois auprès des Fifo et remplir les deux cases temporaires.

-Ce processus se répète jusqu'à ce que les Fifo soient vides.

5.3.3. Module de mémorisation : FIFO

Afin de mémoriser les données et gérer le conflit dans les réseaux, nous avons ajouté à chaque entrée de chaque module Ordonnanceur un bloc de mémoire de type FIFO (Figure20).

Une FIFO est très utile dans notre architecture permettant de lire des données qui viennent successivement d'un autre module et de les stocker dans un ordre bien défini. Elle permet aussi de vider les données de telle façon que la première donnée qui a été enregistrée sera la première à sortir.

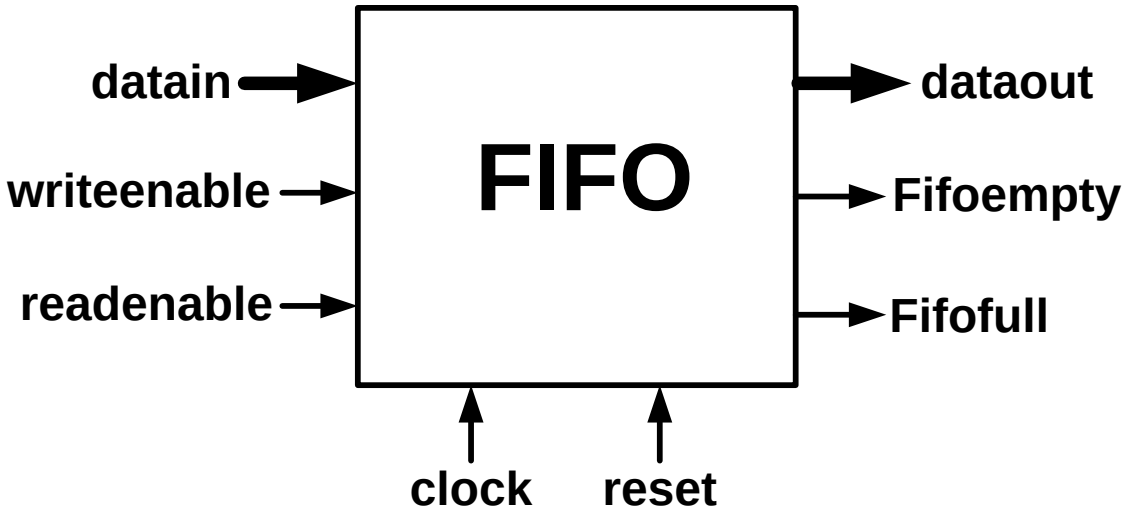

Figure 20. Schéma d'une FIFO

Une FIFO comporte essentiellement :

- Un signal « clock » d'horloge de type entrée, qui assure la synchronisation des signaux.
- Un signal « readenable » de type entrée, qui active la lecture à partir de la FIFO.
- Un signal « writeenable » de type entrée, qui active l'écriture dans la FIFO.
- Un vecteur « datain » de type entré de taille 69, qui représente l'information à écrire dans la FIFO.
- Un vecteur «dataout » de type sortie de taille 69, qui représente l'information à lire à partir de la FIFO.
- Deux signaux "Fifofull" et "Fifoempty" de type sortie, indiquant l'état de la mémoire (pleine ou vide).
- La taille de FIFO : c'est un paramètre critique pour les NOCs. Il influe directement sur les performances des routeurs implémentés dans le modèle.

  Plusieurs variantes de FIFOs sont implémentées et simulées en vu d'évaluer les performances de notre réseau sur puce (2^N) avec N ∈ [1,3]).

5.3.4. Module de connexion

Le bloc de connexion est un module qui assure la connexion entre n signaux d'entrées et n signaux de sorties (Figure21). Si on veut changer la topologie de notre réseau multi-étages (par exemple changer un réseau oméga par un réseau Baseline), il suffit d'arranger le bloc de connexion et décrire les types de connexions correspondantes à la topologie. Pour simplifier la tâche on a décrit la manière de connexion des topologies dont on a besoin et il suffit de changer la position de commutateur topologie pour passer d'une topologie à une autre.

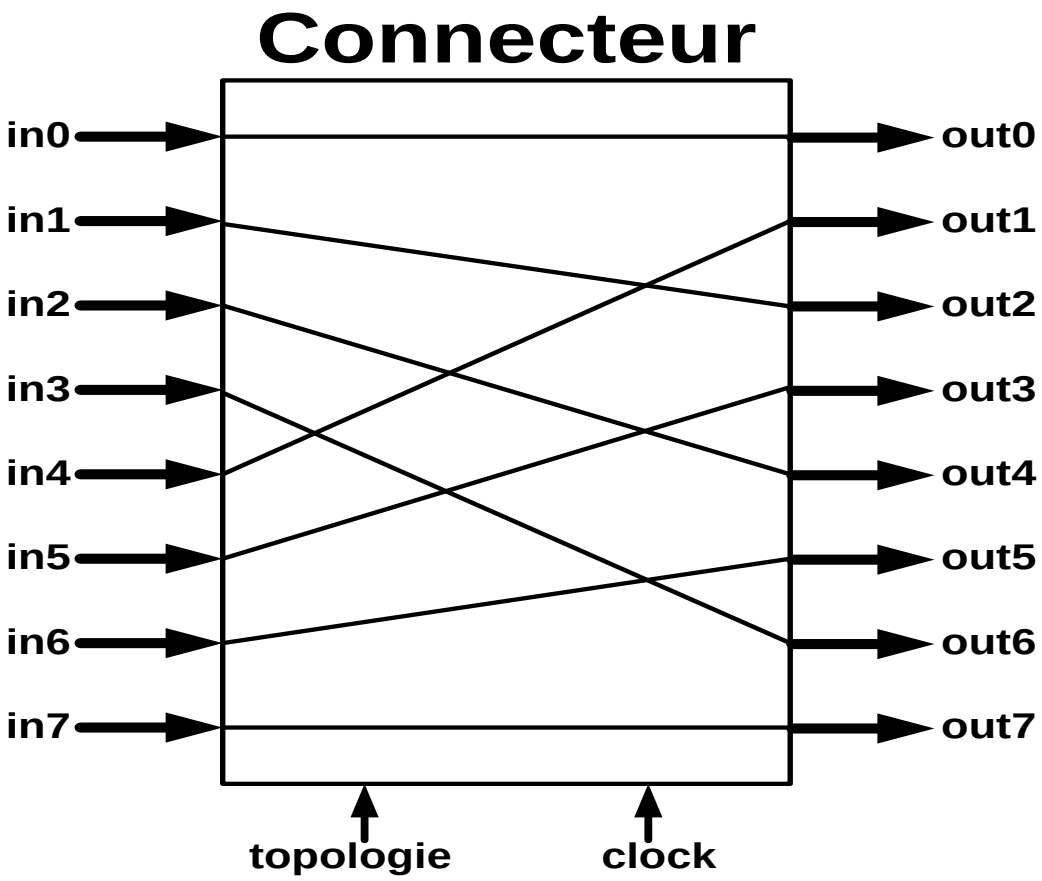

Figure 21. Schéma du module connecteur (réseau Omega)

6. Implémentation des réseaux multi-étage

Un réseau multi-étage est composé essentiellement des étages de commutateurs connectés entre eux par les blocs des connexions suivant une topologie bien définie (Figure22).

Le réseau multi-étage permet de connecter N processeur à n modules mémoire.

La figure22 présente un exemple d'implémentation d'un réseau MIN de type OMEGA de taille 8x8, c'est-à-dire qui connecte 8 processeurs à 8 mémoires. Le réseau est composé de 3 étages de *switch* et de 4 étages de connecteurs. Le schéma a été obtenu en utilisant l'outil RTL VIEWER de XILINX. Cet outil permet de transformer une description en langage VHDL en une représentation schématique comportant les blocs utilisés ainsi que les connexions entre ces blocs.

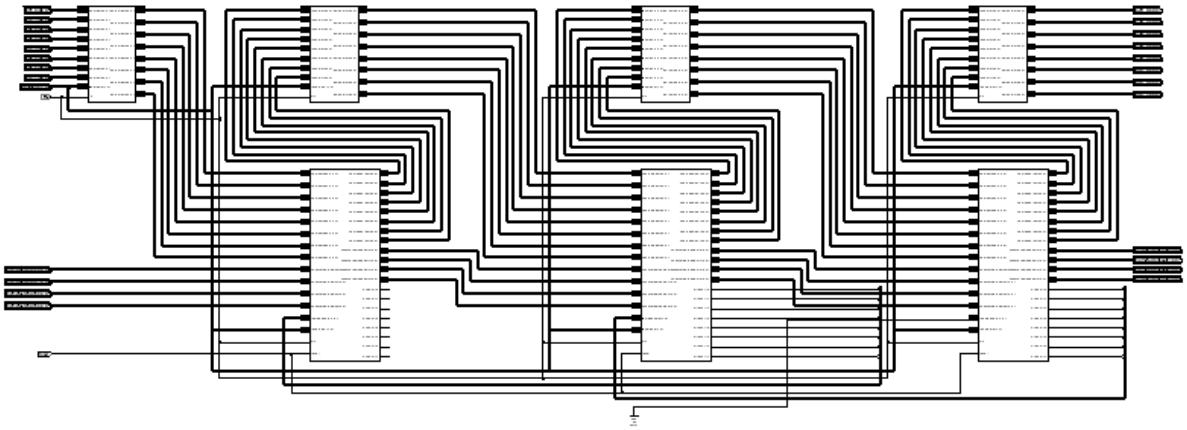

Figure 22. Composant réseau MIN

7. Estimation des performances

Après avoir implémenté le réseau multi-étage, nous avons dégagé les performances. Ces performances se focalisent essentiellement sur la Surface sur FPGA et la latence. On commence par le *switch* qui présente le cœur du réseau puis on passe à l'estimation des performances du MIN. Enfin, on commente les résultats obtenus.

7.1.Switch

Deux types de *switch* ont été conçus. Le premier qui fonctionne en mode Asynchrone et qui se caractérise par une latence minimale mais une consommation importante de blocs logiques sur FPGA (Tableau1) et un deuxième qui fonctionne en mode synchrone et qui est caractérisé par rapport au premier par moins de surface sur FPGA et plus de latence (Tableau2).

De plus on a varié la profondeur de la FIFO utilisée pour les deux modes de fonctionnement (2,4 et 8 places par FIFO) pour évaluer la surface de *switch*.

On a constaté que la solution la plus adéquate pour notre *switch* est d'utiliser une FIFO de profondeur 8 car cette dernière présente moins de surface sur FPGA.

Ceci s'explique par l'usage des blocs FIFO préfabriqués par le concepteur XILINX et qui se trouve dans l'FPGA pour un *switch* qui utilise une FIFO de profondeur 8 tandis qu'un *switch* qui utilise une FIFO de profondeurs 4 ou 2. Cet usage sera compensé par la combinaison des blocs logiques élémentaires (Tableau1, Tableau2).

Mode Asynchrone

cible xc4vlx200 -11ff1513	**Utilisation des ressources logiques**				**Latence**
Prof de FIFO	Nombre de Slices	Nombre de Slice Flip Flops	Nombre de 4 input LUTs	Nombre de FIFO16/RAMB16s	
2	**1073**	**719**	**1590**	**0**	**2 cycle**
4	**1081**	**731**	**1606**	**0**	**2 cycle**
8	**756**	**587**	**1349**	**4**	**2 cycle**

Tableau 1.Synthèse et latence de *Switch* en mode Asynchrone

Mode Synchrone

cible xc4vlx200 -11ff1513	**Utilisation des ressources logiques**				**Latence**
Prof de FIFO	Nombre de Slices	Nombre de Slice Flip Flops	Nombre de 4 input LUTs	Nombre de FIFO16/RAMB16s	
2	**812**	**713**	**1182**	**0**	**3 cycle**
4	**826**	**719**	**1208**	**0**	**3 cycle**
8	**568**	**587**	**929**	**4**	**3 cycle**

Tableau 2. Synthèse et latence de *Switch* en mode Synchrone

7.2.MIN

L'estimation des performances du NOC est basée sur les deux types de *switch* dont on a déjà parlé. Il existe alors deux types de NOC : un qui fonctionne en mode Asynchrone et un autre qui fonctionne en mode Synchrone (Tableau3, Tableau4).

Les résultats ont été pris en variant le nombre des processeurs (4, 8, 16 et 32).

Mode Asynchrone

cible xc4vlx200 -11ff1513	**Utilisation des ressources logiques**							
Nombre de processeurs	Nombre de Slices	%	Nombre de Slice Flip Flops	%	Nombre de 4 input LUTs	%	Nombre de FIFO16/RAMB16s	%
4	3538	3	2344	1	6521	3	16	4
8	10718	12	7036	3	19673	11	48	14
16	33199	37	19024	10	61337	34	128	38

Tableau 3. Synthèse de MIN en mode Asynchrone

Mode Synchrone

cible: xc4vlx200 -11ff1513	**Utilisation des ressources logiques**							
Nombre de processeurs	Nombre de Slices	%	Nombre de Slice Flip Flops	%	Nombre de 4 input LUTs	%	Nombre de FIFO16/RAM B16s	%
4	2281	2	2364	1	3993	2	16	4
8	6839	7	7092	3	12257	6	48	14
16	18220	20	18912	10	33057	18	128	38

Tableau 4. Synthèse de MIN en mode Synchrone

7.3.Commentaires

Après avoir dégagé les estimations des performances de *switch* et de MIN, on remarque bien que l'évolution de l'utilisation des ressources logiques sur FPGA en fonction du nombre de processeur est linéaire pour le mode synchrone tandis qu'elle ne l'est pas pour le mode asynchrone (Nombre de Slices, Nombre de 4 input LUTs).

8. Conclusion

Dans ce chapitre, nous avons présenté l'environnement de travail à savoir le langage VHDL et les outils de conception de XILINX. Puis nous avons modélisé les réseaux multiétages de type DELTA. Ensuite nous avons fait l'implantation de notre réseau. Enfin nous avons terminé par une estimation des performances.

Chapitre4 : L'intégration du réseau multi-étage dans une architecture multiprocesseur

1. Introduction

Notre objectif est d'adopter le réseau d'interconnexions multi-étage en tant qu'une architecture de communications dédiées aux MPSOC.

Après avoir élaboré la spécification des réseaux DELTA MIN qui présente une bonne solution pour connecter N processeurs à N mémoires, ce chapitre s'intéressera à l'implémentation notre réseau dans une architecture multiprocesseur en faisant varier les topologies afin de tester la fiabilité et l'efficacité du NOC et comparer les performances des différents types de réseaux.

On va s'intéresser tout d'abord à spécifier l'architecture multiprocesseur.

Par la suite, on va valider le fonctionnement de notre architecture par une application.

2. Architecture MPSOC reconfigurable

L'architecture est composée de N processeurs, N mémoires de données, N mémoires d'instructions, un réseau de requête DELTA MIN et un réseau de réponse (voir Figure23).

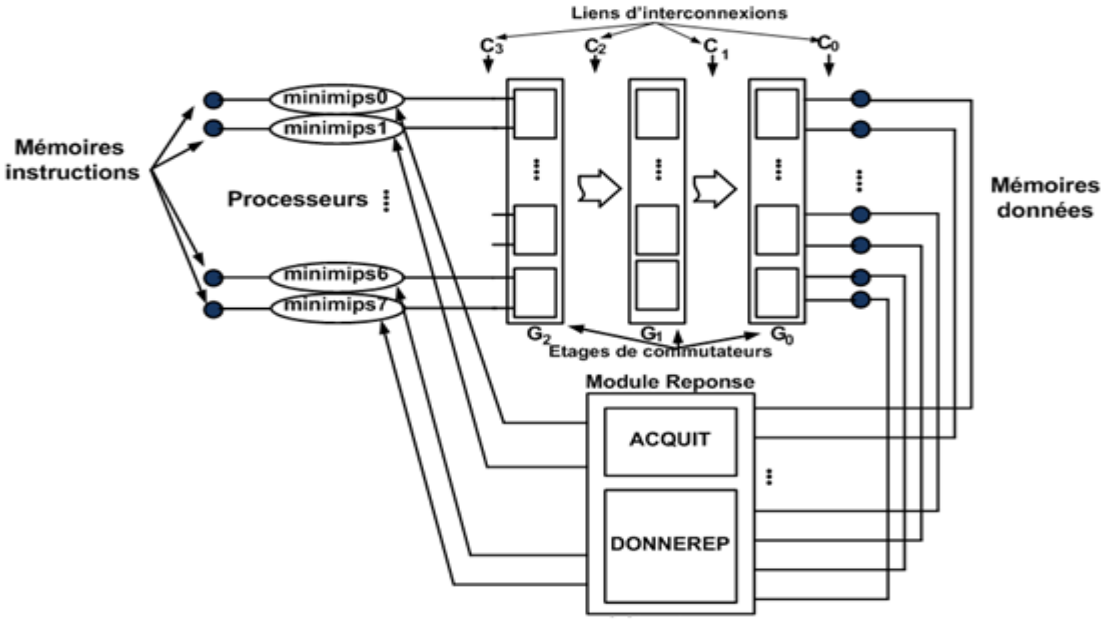

Figure 23. Architecture proposée

Chaque processeur est connecté à sa propre mémoire d'instruction d'une part et au réseau DELTA MIN d'autre part qui est à son tour connecté aux mémoires des données. Chaque processeur peut accéder à n'importe quelle mémoire de données via le réseau pour faire une opération de lecture ou d'écriture de données.

Initialement chaque processeur accède à sa propre mémoire d'instructions, ensuite suivant cette instruction il peut faire une opération de calcul ou un accès à la mémoire de données. S'il s'agit d'une opération d'écriture la mémoire envoie un acquittement au processeur qui a fait la requête indiquant la fin de l'opération sinon (en cas d'une opération de lecture) la mémoire doit envoyer la donnée désirée avec l'acquittement.

Par la suite le processeur passe à l'instruction suivante. De cette façon si un processeur accède à une mémoire de données il restera en pause jusqu'à ce qu'il reçoit un acquittement de cette mémoire.

3. Paramétrage du NOC pour une plateforme MPSOC

3.1.Le Composant Processeur

Il existe une grande variété de processeurs commercialisés et autres *opencores*. La spécification de ces processeurs en langage matériel (VHDL ou VERILOG) peut être portable (comme le processeur MIPS, LEON, etc.) ou non portable et spécifique pour une plateforme bien déterminée (tel que le processeur Microblaze de XILINX, etc.).

Le processeur miniMIPS est classé dans la catégorie des processeurs *opencore* et compatibles pour plusieurs plateformes de prototypages. La spécification en langage VHDL est disponible sur Internet. C'est un processeur efficace pour notre application et répond bien à nos besoins.

Le processeur miniMIPS est une version simplifiée du processeur MIPS R3000. C'est un RISC 32 bits auquel ils manquent quelques instructions, et le mécanisme d'interruption externe. Il comporte 32 registres et inclut deux mémoires cache de 4 Koctets. Il comporte 5 étages pipeline.

Le processeur miniMIPS possède une interface avec 7 ports (Figure24).

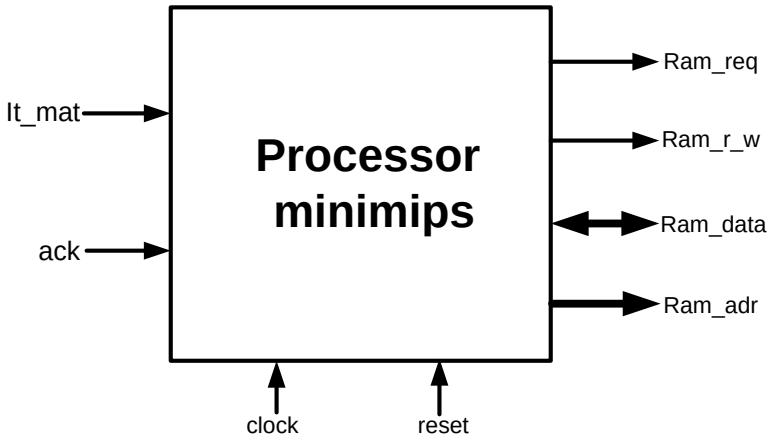

Figure 24. Composant Processeur

– Clock : est l'entrée d'horloge. Les registres internes de miniMIPS sensitives au front montant de CLK.
– Ram_ack : est le signal qui permet l'exécution d'une nouvelle instruction à l'état '1', par suite il indique la fin d'exécution de la dernière instruction.
– Reset : est le signal d'initialisation du processeur. C'est une entrée synchrone active à l'état bas.
– Ram_r_w : est une commande sur un bit définissant le type d'accès à une mémoire externe. Elle peut prendre deux valeurs : lecture d'un mot de 32 bits ou écriture d'un mot de 32 bits.
– Ram_adr : est un vecteur de 32 bits. Dans notre cas, nous avons choisie que les 12 premiers bits de poids fort représentent l'adresse (le numéro) de la source (le processeur), les 8 bits suivants représentent l'adresse de la destination (la mémoire) et le reste des bits représentent l'adresse dans la mémoire (Figure 16).
– Ram_data : est le mot de 32 bits de type entrée/sortie. Ce denier contient l'instruction lue à partir de la mémoire d'instruction ou la donnée à lire ou à écrire dans la mémoire de données.
– it_mat : interruption du processeur.

Implémentation du composant processeur

Afin d'avoir une idée sur les ressources utilisées par un composant processeur, nous avons effectué une analyse et synthèse pour le code de ce composant (décrit en VHDL) en utilisant l'outil ISE 9.1i de XILINX. Par suite, nous avons obtenu les résultats suivants (Tableau5) :

cible: xc4vlx100-12ff1148			
Utilisation de ressources	**utilisé**	**disponible**	**Utilisation**
Nombre de Slices	3344	89088	3%
Nombre de Slice Flip Flops	1883	178176	1%
Nombre de 4 input LUTs	6375	178176	3%
Nombre de bonded IOBs	74	960	7%
Nombre de GCLKs	1	32	
Nombre de DSP48s	4	96	4%

Tableau 5. Synthèse du Processeur miniMIPS

3.2.Le composant mémoire

Une mémoire est tout dispositif capable de stocker des informations (instructions et données) de telle sorte que l'organe qui les utilise (processeur) puisse à n'importe quel moment accéder à l'information qu'il demande.

Les informations peuvent être écrites ou lues. Il y a écriture lorsque nous enregistrons des données en mémoire, lecture lorsque nous appelons des informations précédemment enregistrées. Le temps d'accès est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture en mémoire et l'instant où la première information est disponible. Le temps de cycle représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.

Une mémoire est formée d'un certain nombre de cellules, ou cases, contenant chacune une information. Chaque cellule a un numéro qui permet de la référencer et de la localiser.

Ce numéro est son adresse. Avec une adresse de n bits il est possible de référencer directement au plus 2 cellules. La capacité d'une mémoire est le nombre total de cellules qu'elle contient. Elle s'exprime en nombre de bits, d'octets (bytes) ou de mots (words).

Dans une mémoire à semi-conducteur, nous accédons directement à n'importe quelle information dont nous connaissons l'adresse. Le temps pour obtenir l'information ne dépend pas de l'adresse. On dira que l'accès à une telle mémoire est aléatoire, direct ou encore sélectif.

Les mémoires utilisées dans notre architecture sont des mémoires de type RAM (*Read Access Memory*) et ROM (*Read Only Memory*). Les RAM représentent les mémoires de données, par suite il est possible d'effectuer des lectures et des écritures, alors que les ROM représentent les mémoires d'instructions afin de faire des lectures seulement. Les mémoires utilisées possèdent une interface comportant sept ports : Figure25, Figure26

– Clk : est l'entrée d'horloge. Les lectures et les écritures dans les mémoires sont synchronisées par les fronts montants de CLK.

– Adresse : est le mot en entrée de 8 bits qui indique l'adresse dans la mémoire dans laquelle une lecture ou une écriture sera effectuée.

– Data_out : est un mot en sortie sur 32 bits. Il représente la donnée à lire à partir de la mémoire.

– Ack : est un signal en sortie qui indique la fin d'une manipulation à savoir une écriture ou une lecture.

Les signaux qui suivent sont propres à la mémoire de donnée

– req : est l'entrée qui active ou désactive toute manipulation dans la mémoire. Ce signal est actif à l'état '0'.

– WR: est un signal d'entrée qui indique le type de manipulation dans la mémoire. Il s'agit d'une lecture si le signal est à l'état '0' et une écriture si le signal est à l'état '1'.

– Data_in : est un mot en entrée sur 32 bits. Il représente la donnée à écrire dans la mémoire.

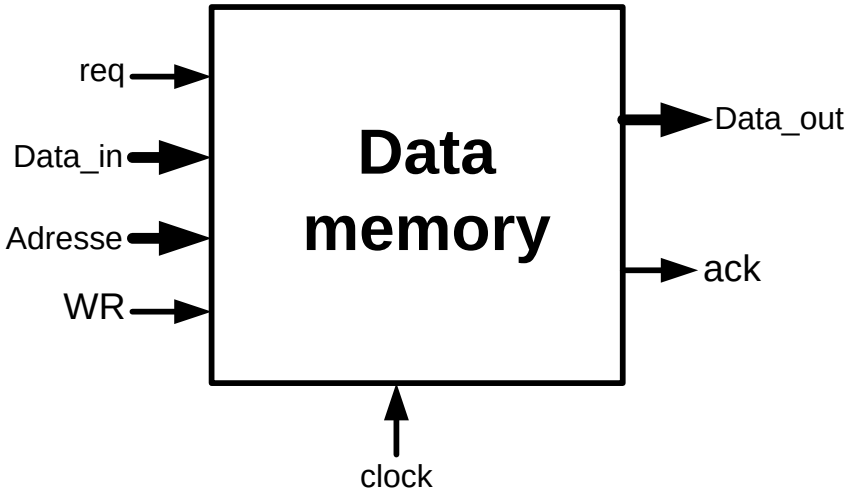

Figure 25. Mémoire de données

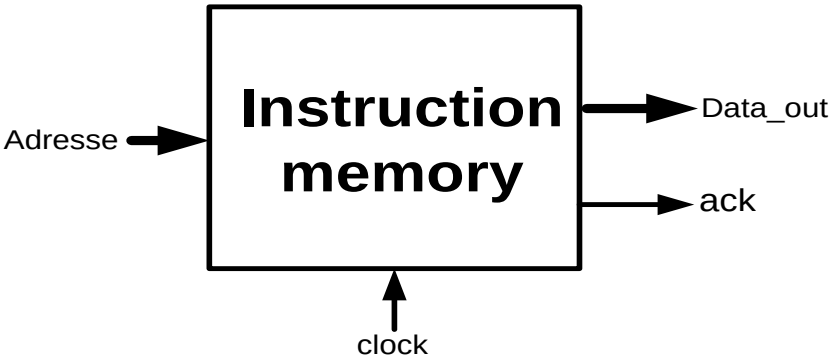

Figure 26. Mémoire d'instructions

Après une analyse et synthèse de ces composants, nous avons obtenu les résultats présentés dans les Tableau6 et Tableau7.

cible: xc4vlx200-11ff1513			
Utilisation de ressources	**utilisé**	**disponible**	**Utilisation**
Nombre de Slices	11	89088	0%
Nombre de 4 input LUTs	22	178176	0%
Nombre de FIFO16/RAMB16s	1	336	0%

Tableau 6. Synthèse du module Mémoire de donnée

cible: xc4vlx200-11ff1513			
Utilisation de ressources	**utilisé**	**disponible**	**Utilisation**
Nombre de Slices	7	89088	0%
Nombre de 4 input LUTs	12	178176	0%

Tableau 7. Synthèse du module Mémoire d'instruction

On remarque que les deux types de mémoires consomment peu de ressources sur FPGA

3.3. Composant NOC

3.3.1. Réseau de requête

Dans notre architecture proposée, le réseau de requête est présenté par le réseau Multi-étage de type Delta dont on a déjà parlé dans le chapitre précédent. Ce réseau permet aux processeurs d'accéder aux mémoires de données (écriture ou lecture). De plus il est possible d'avoir plusieurs types de ce réseau en changeant seulement la topologie des modules de connexion entre les étages de *Switch*. La figure27 présente un réseau de type Butterfly utilisé dans l'architecture adoptée.

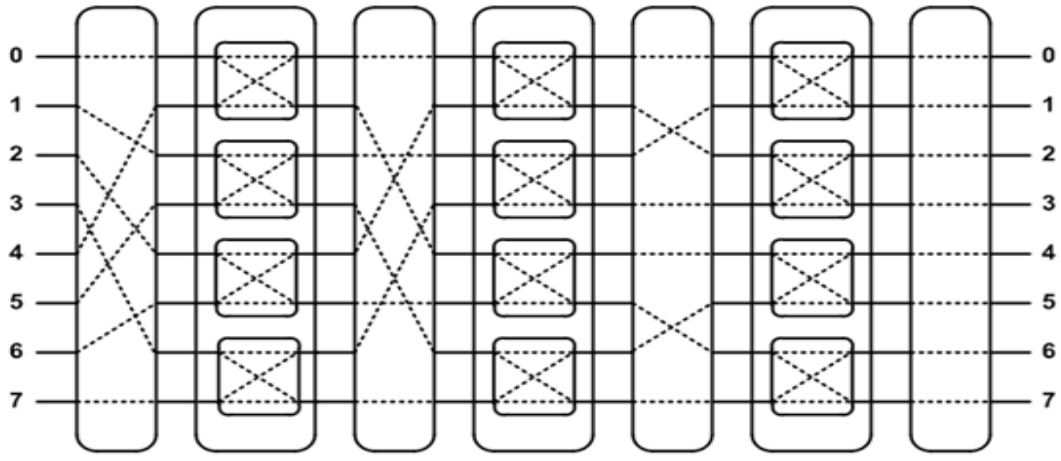

Figure 27. Réseau multi-étage-Topologie Butterfly

Notre réseau est unidirectionnel c'est-à-dire qu'il ne peut transférer que des paquets circulant des processeurs vers les mémoires. Pour cela il nous faut un autre réseau pour le transfert des paquets mémoire_réponse.

3.3.2. Réseau de réponse

Il s'agit de l'acheminement des paquets mémoire_réponse via ce module de réponse en vue de rejoindre la destination (processeurs) dans un délai minimum, sans blocage d'information et sans perte d'information (Figure28).

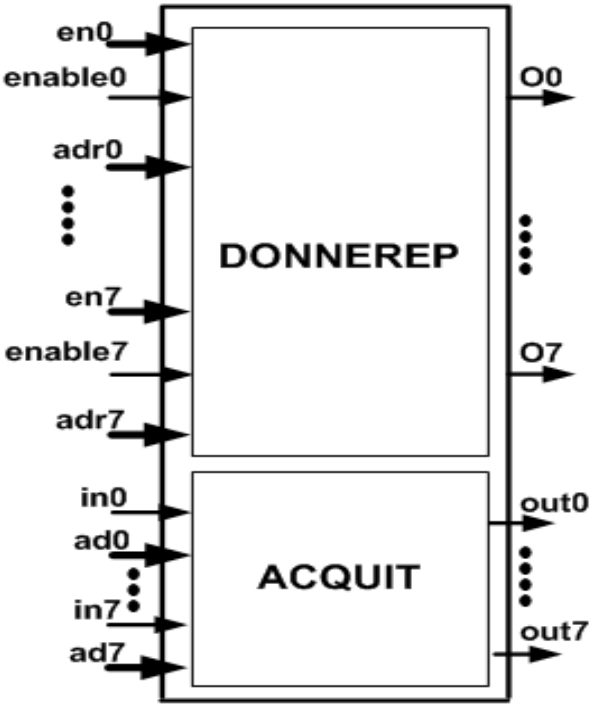

Figure 28. Réseau de réponse

Ce module comporte donc 2 composants : ACQUIT et DONNEREP.

ACQUIT : pour les signaux d'acquittement (Figure29)

DONNEREP : pour les données (Figure30)

ACQUIT

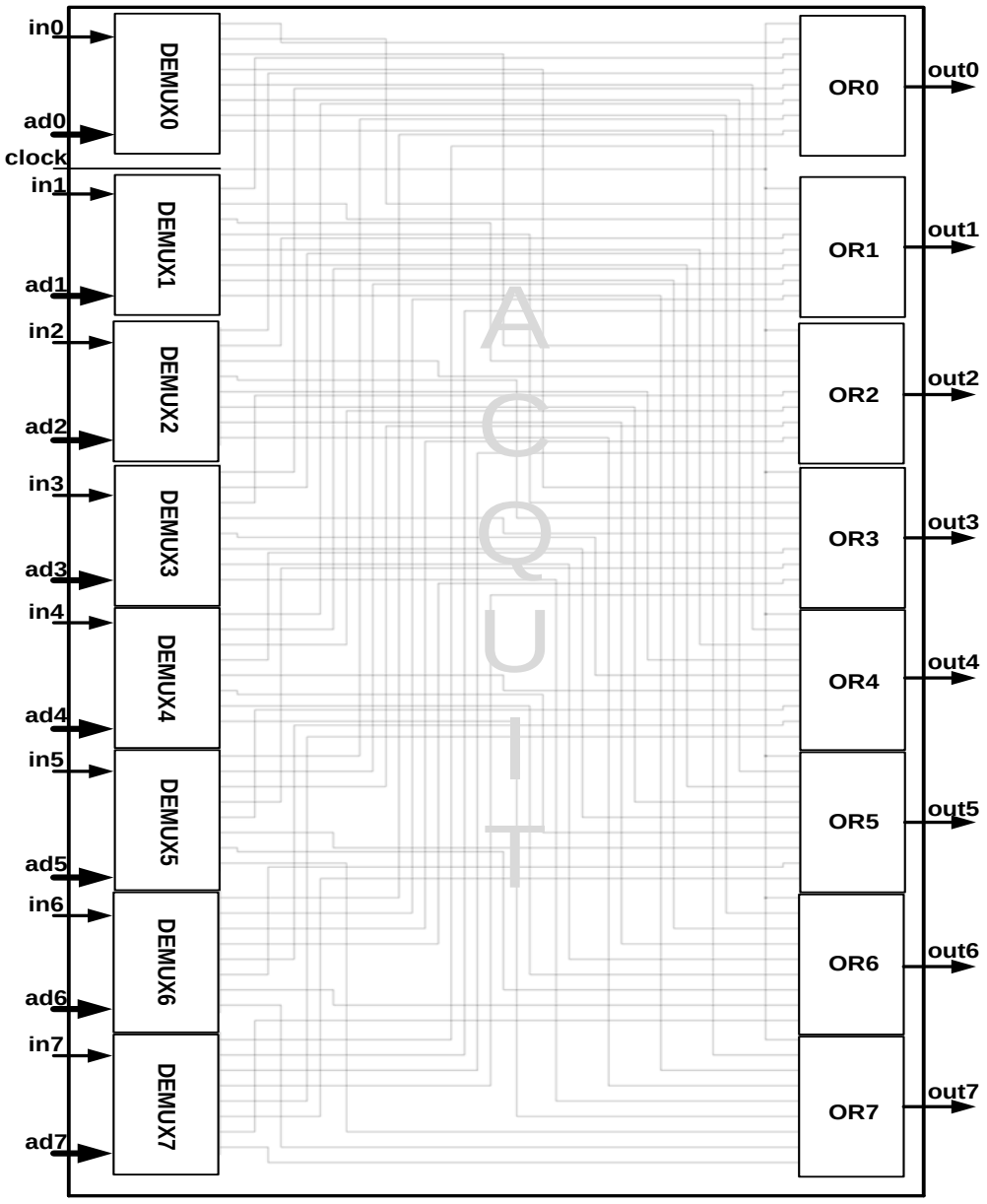

Figure 29. Module ACQUIT

Le composant ACQUIT a comme signaux :

8 vecteurs (de in0 à in7) de taille d'un bit qui représentent les acquittements sortants des mémoires.

8 vecteurs (de ad0 à ad7) de taille 3 bits qui représentent les adresses des destinations de chaque acquittement sortant de la mémoire.

8 vecteurs (de out0 à out7) de taille d'un bit qui représentent les acquittements vers les processeurs.

DONNEREP

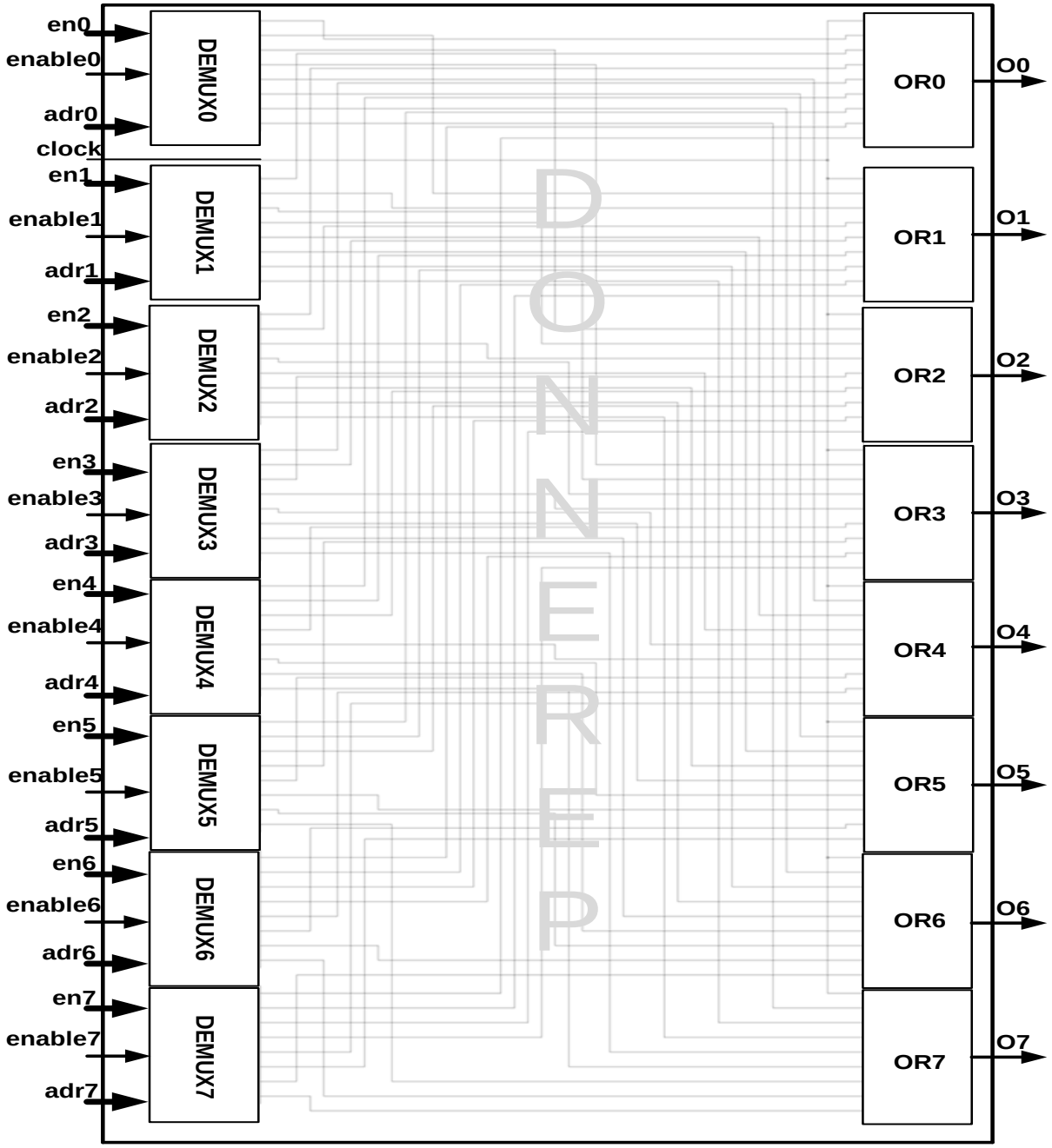

Figure 30. Module DONNEREP

Le composant DONNEREP a comme signaux :

- 8 vecteurs (de en0 à en7) de taille 32 bits qui représentent les données sortantes des mémoires.
- 8 vecteurs (de adr0 à adr7) de taille 3 bits qui représentent les adresses des destinations de chaque donnée sortant de la mémoire.

- 8 vecteurs (de out0 à out7) de taille 32 bits qui représentent les données vers les processeurs.

- 8 vecteurs (de enable0 à enable7) de taille d'un bit qui représentent les signaux d'activation car le transfert de données ne devra fonctionner qu'en cas de lecture.

Les performances de module réponse se focalisent en terme de surface (Tableau8). D'après les résultats estimés, on peut conclure que ce composant présente un gain important comparé à la solution qui utilise un réseau NOC comme un réseau de réponse.

cible: xc4vlx200 -11ff1513	**Utilisation de ressources**					
Nombre de processeurs	Nombre de Slices	%	Nombre de Slice Flip Flops	%	Nombre de 4 input LUTs	%
4	761	0	-	-	1340	0
8	2824	3	256	0	5173	2
16	10369	11	-	-	18889	10

Tableau 8. Synthèse du module Réseau de réponse

4. Implémentation de l'architecture sur FPGA

Après l'implémentation du NoC dans une architecture multiprocesseur, l'estimation de performance se focalise sur la surface de l'architecture sur FPGA.

Les mesures ont été prises en variant le nombre de processeur (4, 8,16) et en utilisant les deux modes de fonctionnement : Asynchrone et Synchrone (Tableau9, Tableau10).

Mode Asynchrone

cible: xc4vlx200 -11ff1513	**Utilisation de ressources**							
Nombre de processeurs	Nombre de Slices	%	Nombre de Slice Flip Flops	%	Nombre de 4 input LUTs	%	Nombre de FIFO16/RAMB16s	%
4	14390	16	10048	5	26551	14	20	5
8	35321	39	22896	12	65540	36	56	16
16	77563	87	49459	27	142761	80	144	42

Tableau 9. Synthèse de l'architecture multiprocesseur en mode Asynchrone

Mode Synchrone

cible: xc4vlx200 -11ff1513	**Utilisation de ressources**							
Nombre de processeurs	Nombre de Slices	%	Nombre de Slice Flip Flops	%	Nombre de 4 input LUTs	%	Nombre de FIFO16/RAMB16s	%
4	12697	14	10057	5	23667	13	20	5
8	30251	33	22909	12	56534	31	56	16
16	68432	76	49466	27	127564	71	144	42

Tableau 10. Synthèse de l'architecture multiprocesseur en mode Synchrone

Après avoir dégagé les estimations des performances de l'architecture en mode synchrone et asynchrone, on remarque bien que l'évolution de l'utilisation des ressources logiques sur FPGA en fonction du nombre de processeur n'est pas linéaire pour les modes synchrone et asynchrone.

5. Validation du réseau multi-étage dans l'architecture MPSOC. Etude de cas : FILTRE FIR et FILTRE IIR

Dans cette partie, nous allons mettre en application notre architecture multiprocesseur. Pour cela, nous allons procéder à l'implémentation d'un Filtre FIR puis un Filtre IIR. Nous allons alors présenter l'application en premier lieu ensuite on va donner une estimation des performances.

5.1.Présentation du Filtre FIR et Filtre IIR

L'application choisie comme démonstration est une application typique du domaine des télécommunications mobiles.

Un filtre numérique peut-être défini par une équation aux différences, c'est-à-dire l'opération mathématique du filtre dans le domaine temporel (discret).

La forme générale du filtre d'ordre M est la suivante:

$$\mathbf{y[n]} = \sum_{k=0}^{n} \mathbf{b_k} * \mathbf{C[n-k]} - \sum_{k=1}^{m} \mathbf{a_k} * \mathbf{y[n-k]}$$

Sa fonction de transfert dans le domaine fréquentiel (Transformée en Z) est :

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}$$

Il y a deux grandes familles de filtres numériques : la première, les filtres FIR, de l'anglais "*Finite Impulse Response*" (Filtre à réponse impulsionnelle finie). Ce type de filtre est dit fini, car sa réponse impulsionnelle se stabilisera ultimement à zéro. Un filtre FIR est non récursif, c'est-à-dire que la sortie dépend uniquement de l'entrée du signal, il n'y a pas de contre-réaction.

Ainsi, les coefficients « a » de la forme générale des filtres numériques sont tous égaux à zéro.

Une propriété importante des filtres FIR est que les coefficients du filtre « b » sont égaux à la réponse impulsionnelle « h » du filtre.

D'autre part, la forme temporelle du filtre est tout simplement la convolution du signal d'entrée x avec les coefficients (ou réponse impulsionnelle) b (ou h).

En opposition, les filtres de la seconde famille, les IIRs, de l'anglais "*Infinite Impulse Response*" (Filtre à réponse impulsionnelle infinie) possèdent une réponse impulsionnelle qui ne se stabilisera jamais, et ce, même à l'infini. Ce type de filtre est récursif, c'est-à-dire que la sortie du filtre dépend à la fois du signal d'entrée et du signal de sortie. Il possède ainsi une boucle de contre-réaction (*feedback*). Les filtres IIR sont principalement la version numérique des filtres analogiques traditionnels: Butterworth, Tchebychev, Bessel, Elliptique.

5.2.Modélisation du Filtre FIR

Pour tester la fonctionnalité de l'architecture multiprocesseur, une implantation de filtre FIR a été faite. Dans ce qui suit la façon dont un filtre FIR peut être mis en œuvre est décrite. Huit processeurs ont été utilisés. La réponse impulsionnelle utilisée est :

h(n) = { h(0), h(1), h(2), h(3), h(4), h(5), h(6), h(7) }

Le vecteur d'entrée est :

x(n) = {C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16}

Le vecteur de sortie est :

y(n) = { S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S23 }

Chaque processeur envoie des requêtes demandant l'accès aux mémoires de données pour apporter les entrées Ci, ensuite il effectue les opérations de calculs nécessaires et enfin il stocke les résultats obtenus Si dans la mémoire de donnée Si.

L'envoi et la réception des requêtes sont cadencés par le signal d'horloge globale du système. L'allocation des données dans les mémoires est un choix effectué, par suite il est possible d'utiliser de différentes distributions et comparer à chaque fois les performances du réseau. L'application sera repartie sur 3 itérations présentées dans les figures 31, 32 et 33.

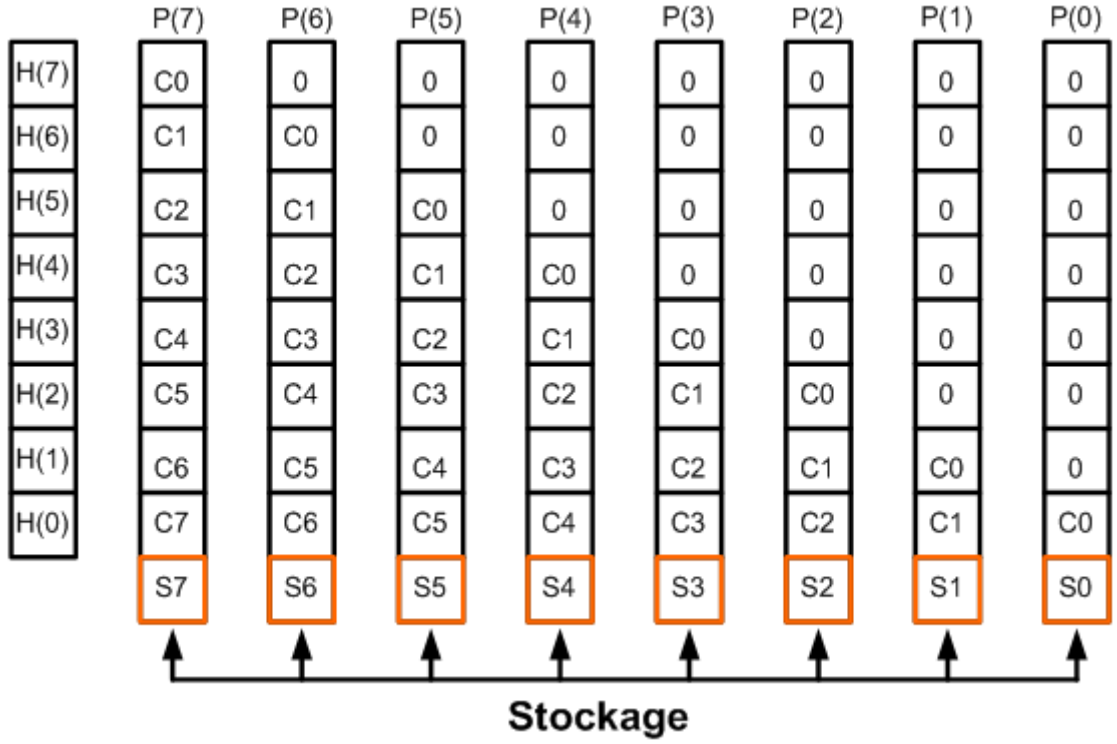

Figure 31. 1ère itération de l'application Filtre FIR

Y(n) est calculé en utilisant des opérations de multiplication et d'addition comme suit :

$S0 = h(0) * C0 + h(1) * 0 + h(2) * 0 + h(3) * 0 + h(4) * 0 + h(5) * 0 + h(6) * 0 + h(7) * 0$

$S1 = h(0) * C1 + h(1) * C0 + h(2) * 0 + h(3) * 0 + h(4) * 0 + h(5) * 0 + h(6) * 0 + h(7) * 0$

$S2 = h(0) * C2 + h(1) * C2 + h(2) * C0 + h(3) * 0 + h(4) * 0 + h(5) * 0 + h(6) * 0 + h(7) * 0$

$S3 = h(0) * C3 + h(1) * C2 + h(2) * C1 + h(3) * C0 + h(4) * 0 + h(5) * 0 + h(6) * 0 + h(7) * 0$

$S4 = h(0) * C4 + h(1) * C3 + h(2) * C2 + h(3) * C1 + h(4) * C0 + h(5) * 0 + h(6) * 0 + h(7) * 0$

$S5 = h(0) * C5 + h(1) * C4 + h(2) * C3 + h(3) * C2 + h(4) * C1 + h(5) * C0 + h(6) * 0 + h(7) * 0$

$S6 = h(0) * C6 + h(1) * C5 + h(2) * C4 + h(3) * C3 + h(4) * C2 + h(5) * C1 + h(6) * C0 + h(7) * 0$

$S7 = h(0) * C7 + h(1) * C6 + h(2) * C5 + h(3) * C4 + h(4) * C3 + h(5) * C2 + h(6) * C1 + h(7) * C0$

La même procédure se répète jusqu'à ce que les entrées soient finies (Figure32, Figure33).

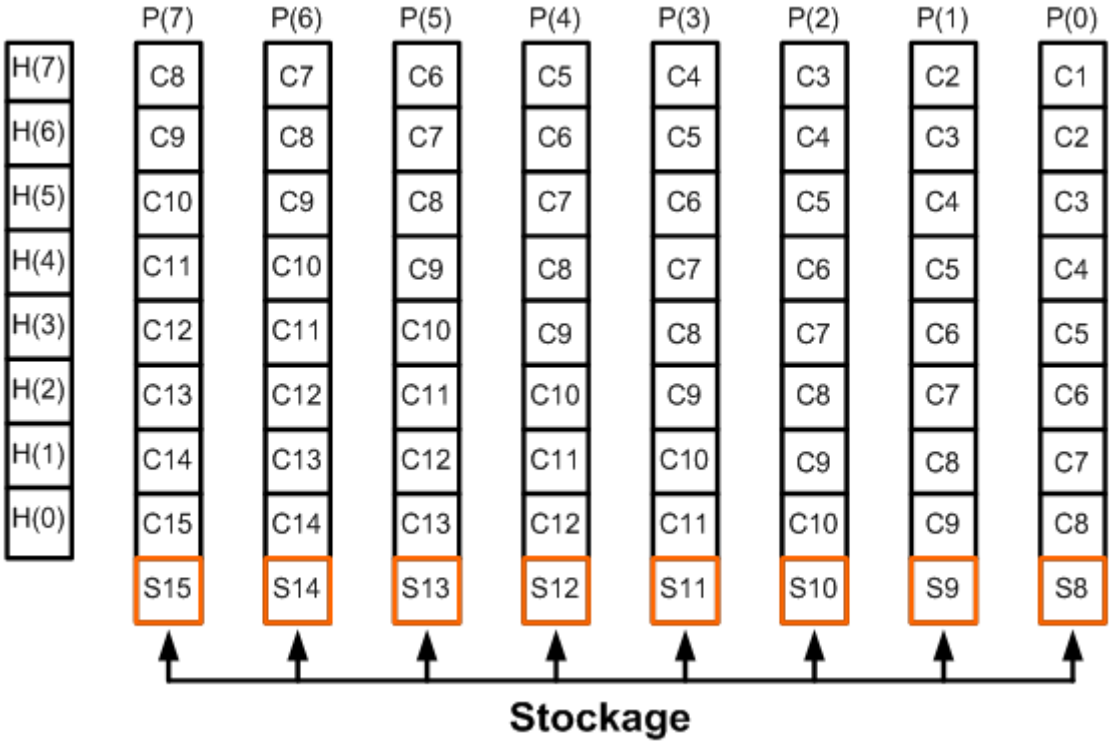

Figure 32. 2ème itération de l'application Filtre FIR

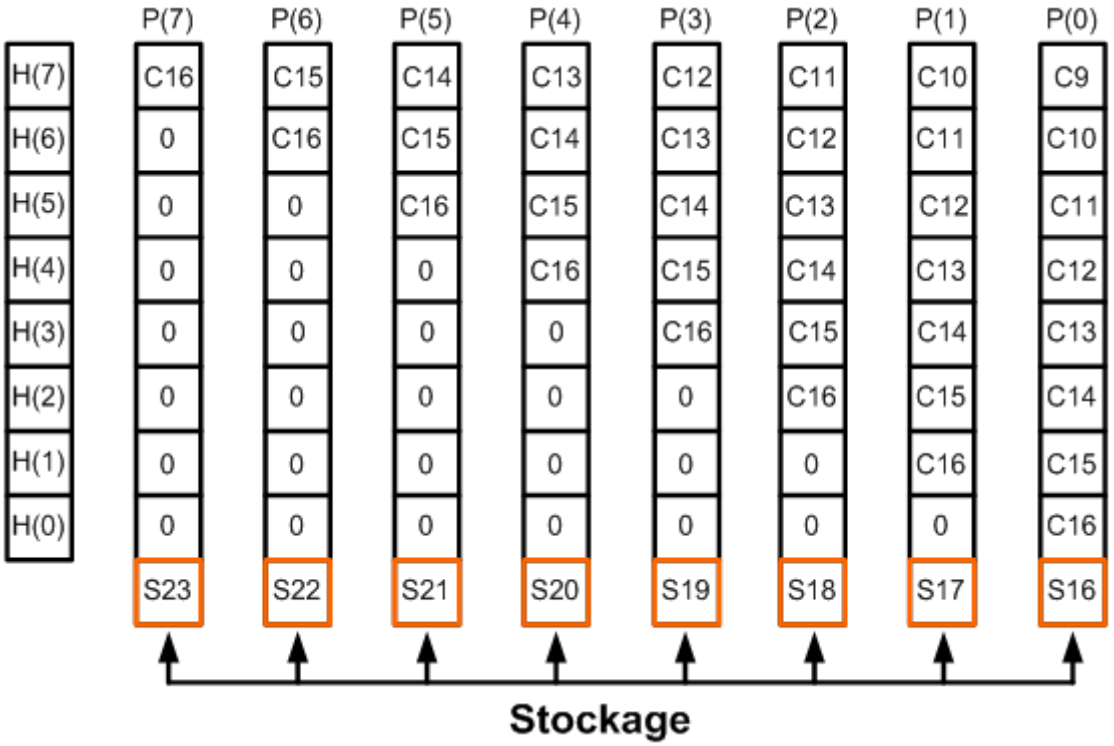

Figure 33. 3ème itération de l'application Filtre FIR

5.3.Modélisation du Filtre IIR

Une 2ème application a été choisie pour mieux évaluer la fonctionnalité de notre architecture multiprocesseur. L'application consiste à l'implantation de filtre IIR.

Dans ce qui suit, la mise en œuvre d'un filtre FIR est décrite. Huit processeurs ont été utilisés. Sachant que ce type de filtre est récursif, on trouve que chaque sortie dépend des sorties qui la précédent.

Chaque processeur Pi envoie des requêtes demandant l'accès aux mémoires de données pour apporter les entrées Ci, ensuite il effectue les opérations de calculs nécessaires mais cette fois, contrairement au filtre FIR, il continue à demander l'accès aux mémoires de données pour apporter les Si s'ils sont disponibles et enfin il y revient pour stocker les résultats obtenus Si. Dans le cas de dépendance de données le processeur refait le processus (demande de lecture de Si) jusqu'à ce qu'il trouve sa valeur désirée.

L'application sera repartie sur 3 itérations présentées dans les Figures 34, 35 et 36.

$$\begin{bmatrix} S0 \\ S1 \\ S2 \\ S3 \\ S4 \\ S5 \\ S6 \\ S7 \end{bmatrix} = \begin{bmatrix} C0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C1 & C0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C2 & C1 & C0 & 0 & 0 & 0 & 0 & 0 \\ C3 & C2 & C1 & C0 & 0 & 0 & 0 & 0 \\ C4 & C3 & C2 & C1 & C0 & 0 & 0 & 0 \\ C5 & C4 & C3 & C2 & C1 & C0 & 0 & 0 \\ C6 & C5 & C4 & C3 & C2 & C1 & C0 & 0 \\ C7 & C6 & C5 & C4 & C3 & C2 & C1 & C0 \end{bmatrix} \times \begin{bmatrix} h0 \\ h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & S0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & S1 & S0 & 0 & 0 & 0 & 0 & 0 \\ 0 & S2 & S1 & S0 & 0 & 0 & 0 & 0 \\ 0 & S3 & S2 & S1 & S0 & 0 & 0 & 0 \\ 0 & S4 & S3 & S2 & S1 & S0 & 0 & 0 \\ 0 & S5 & S4 & S3 & S2 & S1 & S0 & 0 \\ 0 & S6 & S5 & S4 & S3 & S2 & S1 & S0 \end{bmatrix} \times \begin{bmatrix} 0 \\ a1 \\ a2 \\ a3 \\ a4 \\ a5 \\ a6 \\ a7 \end{bmatrix}$$

Figure 34. 1ère itération de l'application Filtre IIR

La même procédure se répète jusqu'à ce que les entrées soient finies (Figure35, Figure36).

$$\begin{bmatrix} S8 \\ S9 \\ S10 \\ S11 \\ S12 \\ S13 \\ S14 \\ S15 \end{bmatrix} = \begin{bmatrix} C8 & C7 & C6 & C5 & C4 & C3 & C2 & C1 \\ C9 & C8 & C7 & C6 & C5 & C4 & C3 & C2 \\ C10 & C9 & C8 & C7 & C6 & C5 & C4 & C3 \\ C11 & C10 & C9 & C8 & C7 & C6 & C5 & C4 \\ C12 & C11 & C10 & C9 & C8 & C7 & C6 & C5 \\ C13 & C12 & C11 & C10 & C9 & C8 & C7 & C6 \\ C14 & C13 & C12 & C11 & C10 & C9 & C8 & C7 \\ C15 & C14 & C13 & C12 & C11 & C10 & C9 & C8 \end{bmatrix} \times \begin{bmatrix} h0 \\ h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \end{bmatrix} - \begin{bmatrix} 0 & S7 & S6 & S5 & S4 & S3 & S2 & S1 \\ 0 & S8 & S7 & S6 & S5 & S4 & S3 & S2 \\ 0 & S9 & S8 & S7 & S6 & S5 & S4 & S3 \\ 0 & S10 & S9 & S8 & S7 & S6 & S5 & S4 \\ 0 & S11 & S10 & S9 & S8 & S7 & S6 & S5 \\ 0 & S12 & S11 & S10 & S9 & S8 & S7 & S6 \\ 0 & S13 & S12 & S11 & S10 & S9 & S8 & S7 \\ 0 & S14 & S13 & S12 & S11 & S10 & S9 & S8 \end{bmatrix} \times \begin{bmatrix} 0 \\ a1 \\ a2 \\ a3 \\ a4 \\ a5 \\ a6 \\ a7 \end{bmatrix}$$

Figure 35. 2ème itération de l'application Filtre IIR

$$\begin{bmatrix} S16 \\ S17 \\ S18 \\ S19 \\ S20 \\ S21 \\ S22 \\ S23 \end{bmatrix} = \begin{bmatrix} C16 & C15 & C14 & C13 & C12 & C11 & C10 & C9 \\ 0 & C16 & C15 & C14 & C13 & C12 & C11 & C10 \\ 0 & 0 & C16 & C15 & C14 & C13 & C12 & C11 \\ 0 & 0 & 0 & C16 & C15 & C14 & C13 & C12 \\ 0 & 0 & 0 & 0 & C16 & C15 & C14 & C13 \\ 0 & 0 & 0 & 0 & 0 & C16 & C15 & C14 \\ 0 & 0 & 0 & 0 & 0 & 0 & C16 & C15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C16 \end{bmatrix} X \begin{bmatrix} h0 \\ h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \end{bmatrix} - \begin{bmatrix} 0 & S15 & S14 & S13 & S12 & S11 & S10 & S9 \\ 0 & S16 & S15 & S14 & S13 & S12 & S11 & S10 \\ 0 & S17 & S16 & S15 & S14 & S13 & S12 & S11 \\ 0 & S18 & S17 & S16 & S15 & S14 & S13 & S12 \\ 0 & S19 & S18 & S17 & S16 & S15 & S14 & S13 \\ 0 & S20 & S19 & S18 & S17 & S16 & S15 & S14 \\ 0 & S21 & S20 & S19 & S18 & S17 & S16 & S15 \\ 0 & S22 & S21 & S20 & S19 & S18 & S17 & S16 \end{bmatrix} X \begin{bmatrix} 0 \\ a1 \\ a2 \\ a3 \\ a4 \\ a5 \\ a6 \\ a7 \end{bmatrix}$$

Figure 36. 3ème itération de l'application Filtre IIR

5.4.Estimation des performances

L'estimation des performances se focalise sur la mesure des latences et les temps de simulation.

La simulation est faite pour une architecture qui comporte (4,8) processeurs et (4,8) mémoires et un réseau (4x4, 8x8) en utilisant l'outil Modelsim de XILINX.

Après avoir lancé la simulation de notre architecture, nous avons obtenu le chronogramme représenté par la Figure37.

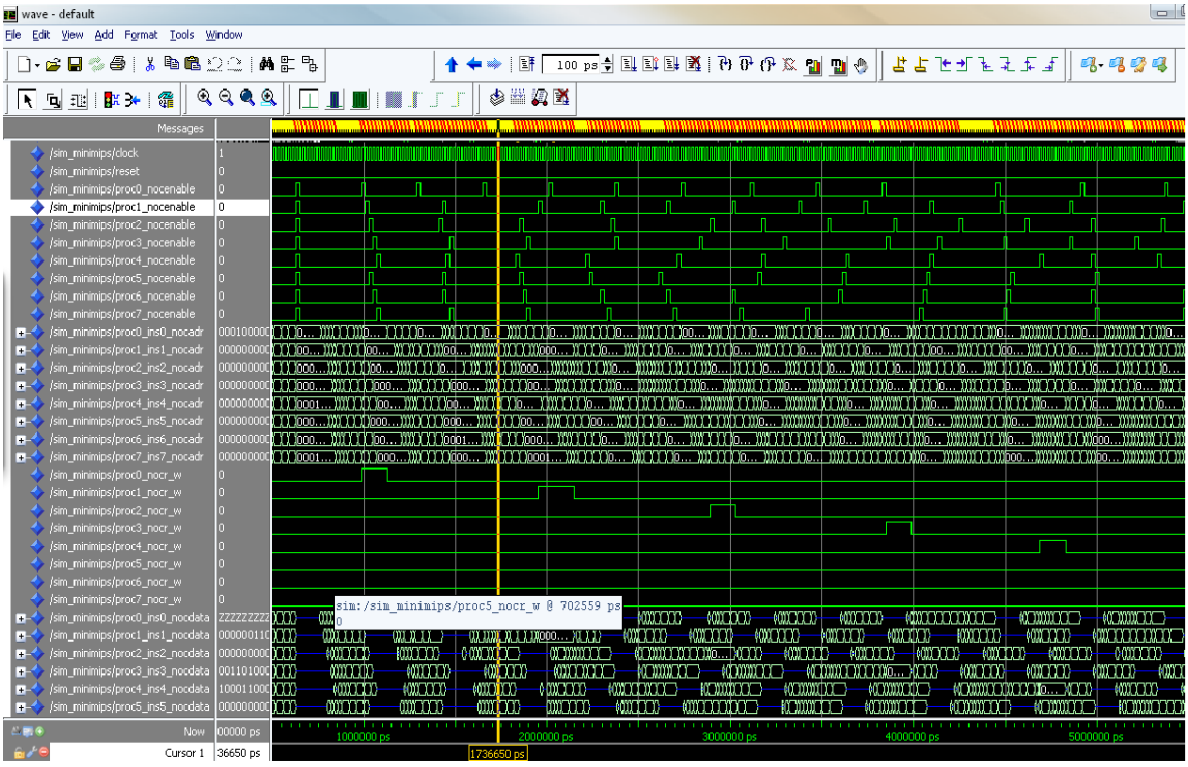

Figure 37. Chronogramme de simulation de l'architecture

Les résultats de simulation ont été pris en faisant varier la topologie (OMEGA, BASLINE, BUTTERFLY) et en augmentant le nombre de processeurs de 4 à 8 processeurs.
La fréquence utilisée est 50 MHZ.

5.4.1. Implémentation du Filtre FIR

4 processeurs

	Mode Asynchrone			**Mode synchrone**		
	topologie			**topologie**		
	Baseline	**Omega**	**Butterfly**	**Baseline**	**Omega**	**Butterfly**
Latence moyenne	**105 ns**	**105,8 ns**	**105,8 ns**	**144,2ns**	**145,82ns**	**145,82ns**
Temps de simulation	**4050ns**	**4070 ns**	**4070 ns**	**4530ns**	**4550ns**	**4550ns**

Tableau 11. Latence et Temps de simulation de l'application Filtre FIR pour 4 processeurs

8 processeurs

	Mode Asynchrone			**Mode synchrone**		
	topologie			**topologie**		
	Baseline	**Omega**	**Butterfly**	**Baseline**	**Omega**	**Butterfly**
Latence moyenne	**144 ns**	**146 ns**	**146 ns**	**203,2 ns**	**206 ns**	**206 ns**
Temps de simulation	**7930 ns**	**8070 ns**	**8070 ns**	**9270 ns**	**9270 ns**	**9130 ns**

Tableau 12. Latence et Temps de simulation de l'application Filtre FIR pour 8 processeurs

Les résultats obtenus montre qu'en passant du mode Asynchrone vers le mode Synchrone le temps de simulation et la latence moyenne augmentent (voir Tableau11, Tableau12).
Aussi on peut conclure que le réseau Baseline est le réseau le plus performant dans la plupart des cas (mode Synchrone, Asynchrone).

5.4.2. Implémentation du Filtre IIR

4 processeurs

	Mode Asynchrone			Mode synchrone		
	topologie			topologie		
	Baseline	Omega	Butterfly	Baseline	Omega	Butterfly
Latence moyenne	103,8 ns	103,2 ns	103,2 ns	144,4ns	143,9ns	143,9ns
Temps de simulation	9450ns	9830 ns	9830 ns	10850ns	10510ns	10510ns

Tableau 13. Latence et Temps de simulation de l'application Filtre IIR pour 4 processeurs

8 processeurs

	Mode Asynchrone			Mode synchrone		
	topologie			topologie		
	Baseline	Omega	Butterfly	Baseline	Omega	Butterfly
Latence moyenne	146 ns	146,6 ns	146,6 ns	202,8 ns	205,4 ns	205,6 ns
Temps de simulation	23050ns	23210 ns	23210 ns	25110 ns	25330 ns	25330 ns

Tableau 14. Latence et Temps de simulation de l'application Filtre IIR pour 8 processeurs

Les résultats obtenus dans les tableaux pour l'implantation de Filtre IIR montrent une augmentation importante au niveau du temps de simulation par rapport à la première application (Filtre FIR). Ceci s'explique par la complexité de l'application vue la dépendance de données (voir Tableau13, Tableau14).

6. Conclusion

Dans ce chapitre nous avons achevé l'implémentation des MINs dans une architecture multiprocesseur. Nous avons présenté tout d'abord notre architecture adoptée puis nous avons validé son fonctionnement dans un contexte applicatif en variant la topologie utilisée. Enfin, nous avons donné une estimation des performances en terme de surface, latence et temps de simulation.

Conclusion générale et perspectives

Nous récapitulons ici les travaux de mastère et ses apports. Nous avons passé en revue les conceptions des réseaux et leurs implémentations dans des systèmes sur puce. Nous avons décelé plusieurs besoins auxquels doit répondre une architecture d'interconnexion à haute performance d'une part, et des directives générales indispensables pour la conception d'un réseau sur puce fiable et flexible.

Notre architecture d'interconnexion a été modélisée en composants contenant des descriptions en langage VHDL, puis elle a été évaluée par simulation avec Modelsim et synthétisé avec ISE de XILINX.

La conception du modèle à été réalisée en trois phases :

Nous avons commencé par détailler les composants du Delta MIN (*Switch*, ordonnanceur, blocs de connexions) puis leurs implémentations.

Nous avons proposé deux modèles de réseaux Delta MIN, un modèle Asynchrone et un autre Synchrone et nous avons détaillé les caractéristiques et les avantages de chacun en termes de surface et de latence.

Dans la deuxième phase nous avons intégré le réseau Delta MIN dans une architecture multiprocesseur, ce qui a nécessité la conception d'un réseau de réponse acheminant les paquets des mémoires vers les processeurs. Ce réseau de réponse s'avère performant pour notre architecture MPSOC puisqu'il présente une optimisation en termes de surface, de latence ainsi que sa fiabilité de fonctionnement.

La dernière phase renferme l'application pour évaluer et tester la fiabilité de notre architecture et estimer ses performances. Nous avons fait l'implémentation de deux types de Filtres : un Filtre FIR et Filtre IIR. L'estimation des performances s'est focalisée sur la surface sur FPGA, la latence et le temps de simulation.

Un des travaux futurs est de prototyper notre architecture sur une plateforme FPGA afin d'estimer la consommation d'énergie.

Une autre perspective envisageable comme extension à ce mastère consiste à concevoir un environnement qui nous permet de spécifier notre application et architecture a haut niveau et qui utilise le réseau Delta MIN et tous les autres composants de l'architecture MPSOC comme des IP.

Un des objectifs futurs aussi est d'introduire la puce conçu dans une chaine industrielle complète.

Bibliographie

[1] CM, “The CM-5 Connection Machine: A Scalable Supercomputer”, W. Daniel Hillis and Lewis W. Tucker, Communications of the ACM, November 1993, Vol. 36, No. 11.

[2] Modélisation de Systèmes Intégrés Numériques Introduction à VHDL Alain Vachoux, Laboratoire de Systèmes Microélectroniques STI-LSM, alain.vachoux@epfl.ch

[3]C. A. Zeferino et A. A. Susin – « SoCIN : a parametric and scalable network-on-chip », Proc. 16th Symposium on Integrated Circuits and Systems Design, Septembre 2003, p. 169–174.

[4] P. P. Pande, C. Grecu, M. Jones, A. Ivanov et R. Saleh – « Performance evaluation and design trade-offs for network-on-chip interconnect architectures », IEEE Transactions on Computers 54 (2005), p. 1025–1040.

[5] Les réseaux dynamiques ou configurables ,Robert Bergevin

[6] A.A. Jerraya and W.Wolf, editors. "Multiprocessor System-on-Chips". Morgan Kaufmann Publishers Inc., Octobre 2004

[7] Architectures et traitement parallèles 19268.gel.ulaval.ca/notes/pdf_A05/CH_2.pdf

[8] P.M. Zeitzoff and J. E. Chung, A perspective from the 2003 ITRS, IEEE circuits and devices magazine, février 2005.

[9] Collier, M. « A systematic analysis of equivalence in multistage networks»,Lightwave Technology, Journal of Volume 20, Issue 9, Sep 2002 Page(s): 1664 – 1672

[10] Szymanski T, Hamacher V« On the universality of multistage interconnection networks», *IEEE Computer Society Press*, 1994, p. 73-101.

[11] C. Clos, A study of Non-Blocking Switching Networks, Bell Systems Technical Journal, March 1953

[12] J. H. Patel, Performance of processor-memory interconnections for multiprocessors. IEEE Trans. Comput.Octobre 1981.

[13] Argawal D.P., « Graph theoretical analysis and design of multistage interconnection networks », *IEEE Transactions on Computer*, vol. 32, n° 7, July 1983.

[14] Wu C.L., Feng T.Y., « On a class of multistage interconnection networks », *IEEE Transactions on Computers*, August 1980.

[15] Collier M., « A Systematic Analysis of Equivalence in Multi-Stage Networks », *JLT*, 2002.

[16] Bjerregaard T, Mahadevan S, « A survey of research and practices of Network-on-chip », *ACM Computing Surveys*, vol. 38, n° 1, 2006.

[17] L. Benini, G. De Micheli, Powering Network on Chip, ISSS'2001, pp. 33-38.

[18] L. Benini, G. De Micheli; "Networks on chips: a new SoC paradigm", IEEE Computer 35 (1) (2002), pp70–78.

[19] Ateris, A comparison of Network-on-Chip and Busses ,www.ateris.com, 2005

[20] http://em.avnet.com/evk/home

[21] Xilinx ISE9.1i Foundation software.Technical Document.

[22] Daoud.I, Routage dans les NoCs : Etude de cas, rapport de Master, ENIS 2005.

[23] Guerrier. P., Un Réseau d'Interconnexion pour systèmes intégrés, Thèse de doctorat, Université Paris VI, 2000.

[24] B.Ch, Parallélisme, cui.unige.ch/aei/r/Para/CoursParallelisme.pdf

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

[26] Dietmar Tutsch, Matthias Hendler and Günter Hommel.« Multicast Performance of Multistage Interconnection Networks with Shared Buffering »