الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR ANNABA UNIVERSITY

UNIVERSITE BADJI MOKHTAR - ANNABA

جامعة باجي مختار- عنابة

Faculté des sciences de l'ingéniorat

Département d'Électronique

Année : 2017

THÈSE

Présentée en vue de l'obtention du diplôme de

Doctorat de Troisième Cycle en Électronique

Thème

''Génération d'un réseau sur puce au format VHDL RTL à partir d'une modélisation de haut niveau UML par raffinement''

Option :
Télécommunications

Par

BOUGUETTAYA Abdelmalek

Directeur de thèse : TOUMI Salah **Professeur** **Université de Annaba**

Co-Directeur de thèse : KIMOUR Mohamed Tahar **Professeur** **Université de Annaba**

DEVANT LE JURY

PRESIDENT : LARBI Allal **Professeur** **Université de Annaba**

EXAMINATEURS :

FEZARI Mohamed **Professeur** **Université de Annaba**

CHEMALI Hamimi **M. C. A.** **Université de Sétif**

MESSAOUDI Kamel **M. C. A.** **Université de Souk Ahras**

REMERCIEMENTS

Louange à Allah, Seigneur de l'univers, le tout Puissant, le Miséricordieux, qui me protège, m'inspire et me comble de ses bienfaits. Je lui en rends toujours grâce.

Je tiens à exprimer mes remerciements et ma reconnaissance :

- A mes Directeurs de thèse, S. TOUMI et M. T. KIMOUR pour les conseils judicieux et constants qu'ils m'ont prodigués tout au long de ce travail, leur grande disponibilité ainsi que la confiance qu'ils m'ont toujours inspirée. Ils m'ont été d'un soutien scientifique et moral tout simplement inestimable. Qu'ils soient assurés de ma reconnaissance, mon respect et mon amitié indéfectibles.
- A l'ensemble des membres de mon jury d'avoir accepté de juger et d'évaluer ces travaux de thèse. Je remercie sincèrement A. LARBI, Professeur à l'université d'Annaba, de m'avoir fait l'honneur de présider mon jury ; qu'il trouve ici l'expression de mon profond respect. Je remercie également M. FEZARI, Professeur à l'université d'Annaba, H. CHEMALI, Maître de Conférences à l'université de Sétif ainsi que K. MESSAOUDI, Maître de Conférences à l'université de Souk Ahras d'avoir accepté de me donner un peu de leurs précieux temps pour faire part de mon jury en tant qu'examinateurs.
- A Messieurs A. BOUKABOUB, A. BOUDJEDRA et H. ATOUI pour leur aide et leurs encouragements.
- A mes amis et ma famille, et plus particulièrement mes parents pour leur soutien constant durant mes longues années d'études.
- A tous mes enseignants de l'Université Badji Mokhtar d'Annaba, depuis le tronc commun en Sciences et Techniques au département d'Electronique. Je vous porte tous dans mon cœur.
- A tous les membres du laboratoire LERICA que j'ai eu le plaisir de côtoyer durant mes longues années de thèse.
- A tous ceux qui ont participé de près ou de loin à l'élaboration de ce mémoire.

DÉDICACE

Je dédie ce travail :

- A mes chers parents.
- A notre très cher Monsieur Amar BOUKABOUB.
- A mes frères Abdellah, Nadir et Moncef.
- A mes amis Fateh, Seyf, Yaâkoub, Adel, Fayçal (Kabran), Heythem, Abderrezzak (Chafik) et Mahdi (Pehni).
- A mon entraineur de Kung-fu, Badreddine (Lehwawi).
- A tous les membres du laboratoire LERICA.
- A tous mes enseignants, depuis l'école primaire.

RÉSUMÉ

Dans le passé, les systèmes embarqués et numériques ont été confinés surtout aux systèmes informatiques. Aujourd'hui, ces systèmes sont appliqués dans un grand nombre de domaines et d'appareils tels que les télévisions numériques, les systèmes de communication, les radars, les systèmes militaires et les instrumentations médicales. L'un des plus grands challenges au niveau de la conception de ces systèmes est l'interconnexion entre ses différents modules. Les réseaux sur puce (NoC) constituent un nouveau paradigme d'interconnexion pour les systèmes sur puce (SoC). Ils ont été proposés comme une solution prometteuse pour résoudre les problèmes rencontrés au niveau des interconnexions classiques.

L'augmentation de la taille du réseau provoque plusieurs inconvénients, comme la réduction au niveau de la bande passante et la fréquence de fonctionnement et une augmentation au niveau de la latence et la consommation de l'énergie. Dans le présent document, nous présenterons une nouvelle approche appliquée pour les réseaux sur puce (topologie Mesh 2D) afin de résoudre les problèmes rencontrés dans les architectures classiques. Cette approche est basée sur une combinaison entre une stratégie de placement des modules, un routage XY à deux niveaux et une technique de clustering basée sur la charge de communication entre les modules.

Afin d'accélérer la conception de cette structure, nous avons utilisé une approche de spécification orientées modèles à base de l'Ingénierie Dirigée par les Modèles (IDM). Nous avons utilisé le paquetage RSM pour modéliser la topologie Mesh 2D à base de cluster et le paquetage à machine d'état ou encore le paquetage d'activité pour la modélisation de l'algorithme de routage XY à deux niveau (intra-cluster et inter-cluster).

Mots Clés : Réseaux sur puce, Algorithme de routage dynamique, Clustering, Topologie Mesh, Systèmes sur puce.

ABSTRACT

Traditionally, embedded systems and digital electronics technology were confined to computer systems. Today, embedded systems and systems on chip are applied in a wide range of areas such as television, communication systems, radar, military systems, medical instrumentation, and consumer electronics use digital techniques. The interconnection between these systems blocks is one of the biggest development challenges. Network on Chip (NoC) is a new interconnection structure that is used for Systems on Chip (SoCs). It is come to replace classic interconnections and to solve its problems. NoC structure provides a high performance, scalable and power efficient communication infrastructure for connecting SoCs modules.

The increase in the network size causes several disadvantages, such as reducing the bandwidth and the operational frequency and an increase in latency and energy consumption. In the present document, we present a new approach applied for networks on a chip (2D Mesh topology) to solve the problems encountered in conventional architectures. This approach is based on a combination between a module placement strategy, a two-level XY routing algorithm and a clustering based on the communication load between modules.

In order to accelerate the design of this structure, we used a Model Driven Engineering (MDE) approach. We used the RSM package to model the cluster-based 2D Mesh topology and the state machine package or the activity package for modeling the two level XY routing algorithm (intra-cluster and inter -cluster).

Keywords: Networks on chip, Dynamic routing algorithm, Clustering, Mesh topology, Systems on chip.

الملخص

الأنظمة الرقمية و المضمنة كانت في القديم مستعملة في أجهزة الكمبيوتر خاصة. في الوقت الحالي، هذه الأنظمة مستعملة في عدد كبير من المجالات و الأجهزة كالتلفاز الرقمي، أنظمة الاتصالات، الرادارات، الأجهزة العسكرية و الطبية. من أكبر التحديات في تصميم هذه الأنظمة هو الربط بين الوحدات المختلفة. الشبكة على الرقاقة هي نموذج تواصل جديد للأنظمة على الرقاقة. وقد اقترحت كحل واعد للمشاكل التي واجهتها هذه الأنظمة في الوصلات التقليدية.

الزيادة في حجم الشبكة تسبب عيوب عديدة، مثل خفض مستوى عرض الحزمة وتردد التشغيل و زيادة في استهلاك الطاقة وزمن نقل الحزمة. في هذا البحث، سوف نقدم طريقة جديدة تطبق على الشبكات ذات طوبولوجيا 2D لحل المشاكل التي واجهتها في البنى التقليدية. تستند هذه الطريقة على مزيج بين استراتيجية موضعة الوحدات، توجيه من نوع XY ذو مستويين و تقنية تجميع على أساس حمولة الاتصالات بين الوحدات.

من أجل الإسراع في تصميم هذه البنية، استخدمنا نموذج هندسة توجيه (MDE). استعملنا الحزمة RSM لنمذجة الطوبولوجيا و حزمة الحالات لنمذجة خوارزمية التوجيه XY ذات مستويين (داخل المجموعة و خارجها).

الكلمات المفتاحية **:** الشبكة المضمنة على الرقاقة، خوارزمية توجيه ديناميكية، التجميع، طوبولوجيا Mesh 2D، الأنظمة المضمنة على الرقاقة.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Introduction Générale

Les systèmes sur puce (SoC) deviennent de plus en plus un aspect essentiel de nos vies professionnelle et personnelle. L'avionique, les transports, la défense, les systèmes médicaux et les télécommunications et des appareils commerciaux à usage général tels que les smartphones, les téléviseurs à haute définition, les consoles de jeux. Aujourd'hui, les SoCs sont omniprésents, et il est difficile de trouver un domaine où ces systèmes miniaturisés n'ont pas fait leur marque. Ces systèmes sont largement reconnus pour représenter le prochain marché majeur pour la microélectronique. Il y a intérêt considérable dans le monde entier à développer des méthodes et des outils efficaces pour soutenir le paradigme SoC. C'est qu'il offre de nombreux avantages par rapport aux solutions multi-puces traditionnelles.

Selon la loi de Moore qui sera probablement vraie, pour la plupart des experts pour les deux prochaines décennies, le nombre des transistors intégrés dans une seule puce doublaient presque tous les deux ans. Cette densité d'intégration nous a menés à pouvoir intégrer des systèmes hétérogènes intégrant différentes unités fonctionnelles sur une seule et même puce. Avec le nombre croissant des noyaux intégrés sur une seule puce, l'interconnexion entre les différents modules est devenue l'un des problèmes majeurs dans la conception des systèmes sur puce. Ainsi, les performances du système dépendent en grande partie de la structure de communication, notamment en ce qui concerne le débit, la surface et la puissance consommées. Aujourd'hui, la conception des SoCs est de plus en plus utilisée dans les systèmes à haute performance. L'architecture d'interconnexion de type bus est la structure la plus utilisée dans les SoCs actuels. Cette architecture ne semble cependant pas fonctionnelle avec les futurs systèmes qui demandent l'intégration de plusieurs éléments pouvant communiquer entre eux en parallèles.

Avec le nombre croissant des noyaux intégrés sur une seule puce, la recherche en réseau sur puce (NoC = Network on Chip) devient plus importante. Le NoC a été proposé comme solution pour remplacer les interconnexions classiques dans les systèmes sur puce (SoC). Dans ces systèmes sur puce à coût optimisé, l'existence d'une architecture NoC efficace est un facteur crucial. Par rapport aux autres architectures, cette structure offre une très grande bande passante, une diversité des chemins, le parallélisme des communications, la flexibilité, l'extensibilité et la réutilisation facile des IPs.

Les flots de développements et les outils de conception des SoCs, utilisés aujourd'hui, n'ont pas été aussi prompts à suivre les grandes évolutions matérielles ainsi que logicielles. En raison de la croissance exponentielle continue de la complexité de la conception des SoCs, la

grande densité d'intégration des modules et le temps de fabrication, il est devenu très difficile de réaliser ces systèmes en bas niveau (RTL) où il nous faut décrire le moindre détail du comportement des composants. Afin d'accélérer la conception des SoCs, des approches de spécification orientées modèles sont alors apparues. Parmi ces approches, on trouve l'Ingénierie Dirigée par les Modèles (IDM). Cette complexité a encouragé les concepteurs à développer des environnements censés faciliter leur conception. L'objectif majeur de ces environnements est d'automatiser les phases de conception et donc la génération automatique d'un code HDL à partir d'une description de haut niveau unifiée abstraite, en utilisant UML/MARTE, donc sans détail d'implémentation.

L'environnement Gaspard2, développé au sein de l'équipe DaRT, est un genre d'outil de CAD basé sur l'approche IDM, qui permet de modéliser une architecture SoC entière en utilisant le langage UML (plus précisément le profil MARTE). Le but de l'environnement GASPARD est de générer le code de simulation dans divers langages, tels que le VHDL qui permet la synthèse des accélérateurs matériels et le SystemC permettant la simulation, à partir d'une modélisation en MARTE. L'avantage majeur de modéliser en suivant le flot GASPARD est de faire la modélisation indépendamment du code à produire, ce qui veut dire qu'il permet la génération du code à partir de l'emplacement d'une application sur une architecture.

Comme on le verra dans le chapitre à venir, les structures d'interconnexion classique ne répondent pas aux exigences des futurs systèmes qui intègrent un très grand nombre d'IPs. Le NoC est un nouveau paradigme d'interconnexion pour les systèmes sur puce (SoC). Nous avons proposé une nouvelle approche de routage pour les réseaux sur puce d'une topologie Mesh 2D modifié. Cette approche est basée sur une combinaison entre une stratégie de placement des modules, un routage XY modifié et une technique de clustering basée sur le taux de communication entre les modules.

Le reste de notre document est organisé selon le plan suivant :

Chapitre 1 : Dans ce premier chapitre, nous introduirons le concept des systèmes sur puce et leurs évolutions depuis l'invention du transistor. Les différentes structures d'interconnexion classiques et actuelles pour les SoCs seront présentées aussi, ainsi que leurs limites et les solutions proposées pour résoudre les problèmes rencontrés. L'architecture des circuits FPGAs sera décrite avec suffisamment de détails afin que le lecteur puisse avoir suffisamment d'éléments

pour comprendre la complexité du sujet. Ensuite, les exigences pour choisir les circuits FPGAs et non d'autres composants programmables, comme les ASICs, seront présentées.

Chapitre 2 : Dans ce chapitre, nous introduirons le paradigme des réseaux sur puce ainsi que les notions qui lui sont associées. A cet effet, nous détaillerons les différents éléments de base, qui constitues cette structure, puis nous décrirons les points de caractérisation des réseaux sur puce, à savoir les métriques de performances, les différentes topologies, les algorithmes de routage, les protocoles de communication, le format de paquet, la qualité de service, le contrôle de flux et de congestion. Nous présenterons aussi quelques NoCs académiques et industriels.

Chapitre 3 : Dans ce chapitre, nous présenterons les bases de l'ingénierie dirigée par des modèles et comment elle peut faciliter le développement des SoCs, le langage de modélisation UML et les différents profiles utilisés pour la modélisation des SoCs. Nous présenterons aussi le profile UML/MARTE, l'environnement de conception des systèmes embarqués GASPARD2 et les transformations des modèles.

Chapitre 4 : Dans la première partie de ce chapitre, nous détaillerons l'architecture de l'approche proposée afin de résoudre les limites des structure Mesh 2D, 3D et 3D modifié. Les communications dans un NoC d'une topologie Mesh 2D à base de cluster sont gérer grâce à la combinaison entre les techniques de routage, de placement et de clustering.

Dans la deuxième partie, nous présenterons la modélisation des concepts de l'approche proposée comme la topologie proposée, l'algorithme de routage utilisé, le contrôle de flux, la structure du réseau appliqué et son comportement.

Dans la troisième partie, nous présenterons les résultats expérimentaux de l'approche proposée et les comparés avec les structure 2D, 3D et 3D modifié.

Dans la dernière partie de notre document, nous conclurons par la présentation du bilan de notre contribution, et l'évocation de quelques perspectives envisageables.

CHAPITRE 1

Systèmes sur Puce

Depuis l'invention du transistor en 1947 puis le développement du premier circuit intégré (IC) environ dix ans plus tard, une évolution rapide de la technologie des semi-conducteurs a eu lieu. En 1965, Gordon Moore, co-fondateur de la compagnie INTEL, a prédit que le nombre de transistors intégrés sur une puce d'un pouce carré doublerait tous les 18 à 24 mois [1]. Ceci est devenu la définition actuelle de la loi de Moore. La plupart des experts prévoient que la loi de Moore tiendra pendant au moins deux autres décennies. Cette densité d'intégration nous a menés à pouvoir intégrer des systèmes hétérogènes intégrant différentes unités fonctionnelles sur une seule et même puce VLSI (Very Large Scale Integration).

Les dispositifs électroniques numériques tels que les téléphones mobiles, les consoles de jeux vidéo, et les routeurs de réseaux contiennent typiquement une ou plusieurs puces électroniques (circuits intégrés), qui sont composées de plusieurs dispositifs tels que les processeurs et la mémoire, qui sont appelés systèmes sur puce ou Systems on Chip en anglais (SoCs). Aujourd'hui, un seul circuit intégré peut contenir plus de 100 millions de transistors, et l'ITRS (International Technology Roadmap for Semiconductors) prédit que le nombre de transistors atteignables, dans le plus proche futur, sera d'un milliard de transistors [2]. Une conséquence des progrès technologiques est que les circuits FPGAs (Field Programmable Gate Array) ont également évolué et portent suffisamment de ressources permettant la mise en œuvre de systèmes embarqués complexes sur une seule puce [3]. Une grande partie de la logique personnalisée dans les produits d'aujourd'hui sont conçus dans des ASICs et FPGAs.

Dans ce chapitre, nous présenterons :

- La définition des systèmes embarqués et leurs évolutions depuis l'invention du transistor et les circuits intégrés jusqu'à la conception des SoCs ;
- les différentes structures d'interconnexions classiques et actuelles pour les SoCs, ainsi que leurs limites et les solutions proposées pour résoudre les problèmes rencontrés ;
- les différentes catégories de circuits reconfigurables, leurs architectures internes et leurs avantages et inconvénients ;
- l'architecture des FPGAs, les différentes technologies de configuration ainsi que les étapes de programmation de ces types de circuits.

1. Les systèmes sur puce.

Chaque fois que le mot microprocesseur est mentionné, nous pensons directement à un ordinateur exécutant une application. Il est vrai que l'ordinateur est la machine la plus populaire et la plus connue dans l'utilisation des microprocesseurs. L'intégration de microprocesseurs dans les équipements et les appareils aux consommateurs a commencé avant même l'apparition des ordinateurs [4] ; ces appareils consomment en effet la majorité des microprocesseurs actuellement produits. Ceci fait que les microprocesseurs embarqués soient ancrés dans la vie quotidienne plus profondément que tout autre circuit électronique. On les trouve partout : à la maison, dans la voiture, dans l'industrie, dans les hôpitaux ou même … dans la poche ! Le tableau 1 présente quelques exemples de produits susceptibles d'être équipés de systèmes embarqués.

Tableau 1 : Quelques exemples familiers d'utilisation des systèmes embarqués.

Maison	Voiture	Industrie	Office et commerce
Machine à laver	Systèmes d'alarme	APIs	Imprimantes
Micro-onde	Contrôle de climatisation	Robots	Scanneurs
Smart TV	Systèmes de freinage	Contrôle de processus	Serveurs
Smart phones	Systèmes de verrouillage	Systèmes de supervision	Photocopieurs
Consoles de jeux			

1.1. Evolution des systèmes sur puce.

Depuis les années 1950, Jack Kilby et Robert Noyce, de Texas Instrument et de Fairchild Semiconductors respectivement, ont eu l'idée de regrouper quelques transistors dans un seul circuit ; ce fut le début des circuits intégrés. Dans le passé, un circuit contenait seulement un petit nombre de transistors ce qui implique un système électronique composé d'un grand nombre de circuits logés dans de nombreux circuits imprimés qui étaient à leur tour mis dans une armoire. Par conséquent, nous pouvons les appeler systèmes sur carte. Actuellement, environ un milliard de transistors peuvent être mis en œuvre sur une seule puce (fig.1) offrant un énorme potentiel pour la fonctionnalité électronique. Cette fonctionnalité est de plus en plus exigée par les marchés de consommation. Les applications multimédias, les dispositifs de réseaux et le domaine de la communication sans fil, par exemple, connaissent une augmentation rapide des bandes passantes ainsi que des fonctionnalités. Une conséquence des progrès technologiques est que les circuits FPGAs (Field Programmable Gate Array) ont également évolué et portent suffisamment de ressources permettant de mettre en œuvre des systèmes embarqués complexes sur une seule puce.

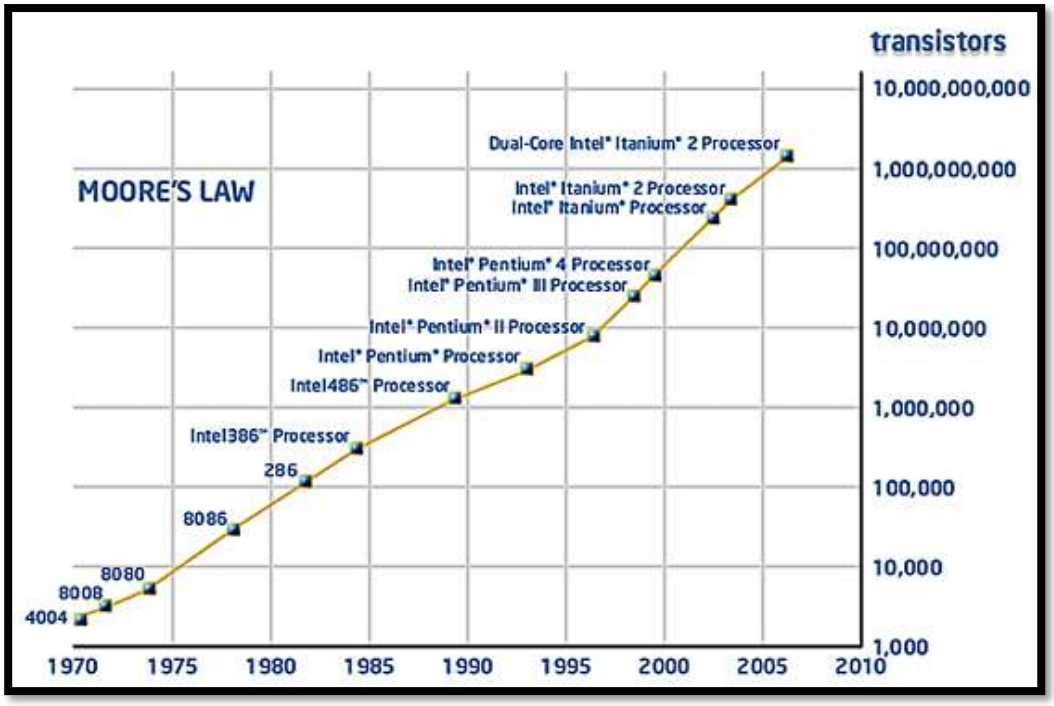


Figure 1 : Evolution de la gamme INTEL suivant la loi de Moore.

Ces systèmes sont largement reconnus pour représenter le prochain marché majeur pour la microélectronique ; il y a intérêt considérable dans le monde entier à développer des méthodes et des outils efficaces pour soutenir le paradigme SoC. Parce qu'il offre de nombreux avantages par rapport aux solutions multi-puces traditionnelles, le système sur puce (SoC) a attiré une grande attention à la fois universitaire et industrielle. En général, les conceptions SoC sont constituées de matériel et de composants logiciels traditionnellement développés séparément et combinés à une étape ultérieure de la conception.

Les Systèmes sur puces peuvent être trouvés dans de nombreux produits tels que : les téléphones mobiles qui utilisent plusieurs processeurs programmables pour gérer le traitement des signaux, les téléviseurs numériques qui utilisent des SoCs sophistiqués pour effectuer la vidéo en temps réel et le décodage audio, les jeux vidéo qui utilisent plusieurs machines de traitement parallèles complexes pour rendre l'action de jeu en temps réel.

1.2. Qu'est-ce qu'un système sur puce ?

Les systèmes sur puces (SoCs) sont la dernière incarnation des technologies VLSI (Very Large Scale Integration). Un SoC peut se définir comme un système mixte matériel/logiciel implanté sur une puce unique. Il représente l'intégration de la plupart ou la totalité des composants et des fonctions d'un système embarqué dans une seule et même puce (fig.2). Une

puce de mémoire peut avoir plusieurs transistors, mais sa structure régulière la rend un composant et pas un système. Les composants assemblés dans un SoC varient selon l'application à réaliser. Les SoCs peuvent inclure différents blocs fonctionnels réutilisables tels que :

- Des éléments de traitements (Microprocesseurs, Microcontrôleurs, DSPs) ;
- des blocs mémoires (RAM, ROM, Flash) ;
- des structures d'interconnexions (Bus, Crossbar, NoC) ;
- des blocs d'entrées/sorties (USB, Ethernet) ;
- des convertisseurs analogique/numérique ;
- des interfaces externes (IP).

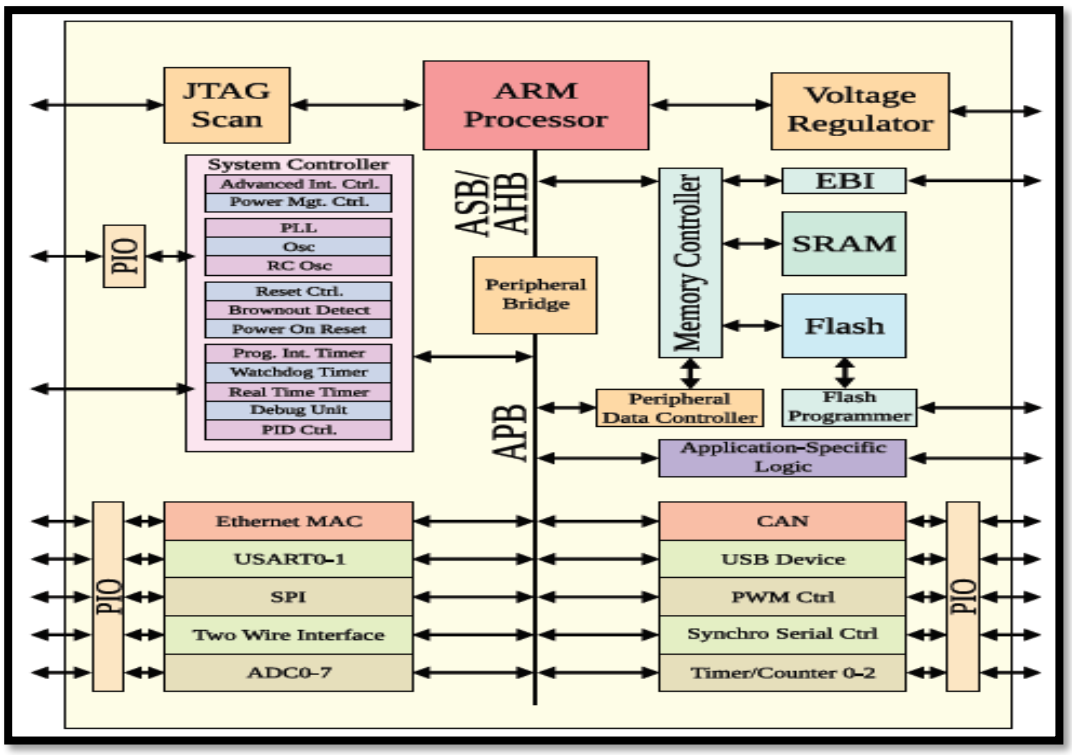


Figure 2 : Architecture d'un système sur puce (SoC).

L'approche SoC a été créée dans un premier temps pour le développement d'ASIC mais a été étendue pour le développement de FPGA. On parle alors de SoPC pour System on Programmable Chip. Le SoPC (acronyme inventé par Synopsys) correspond à l'intégration d'un ou plusieurs cœurs de processeurs et de ses périphériques sur une même puce programmable de type FPGA. Le programme est alors situé soit dans une mémoire interne du circuit FPGA si l'empreinte mémoire le permet, soit dans une mémoire externe le plus souvent [5]. Mais dans les littératures le terme SoC (System on Chip) est généralisé quel que soit le type de circuit (FPGA ou ASIC).

Les systèmes sur puces peuvent être trouvés dans de nombreuses catégories de produits allant des dispositifs de consommation à des systèmes industriels (fig.3) :

- Les téléphones portables utilisent plusieurs processeurs programmables pour gérer les tâches de traitement du signal et de protocole requis par la téléphonie. Ces architectures doivent être conçues pour fonctionner à des niveaux de très faible puissance fournies par des piles ou des batteries.

- Les télécommunications et les réseaux utilisent des SoCs spécialisés, tel que les processeurs de réseau, pour gérer les énormes débits de données offertes par les équipements de transmission moderne.

- Les téléviseurs numériques et les décodeurs utilisent des multiprocesseurs sophistiqués pour effectuer des fonctions de vidéo, de décodage audio et d'interface utilisateur en temps réel.

- Les consoles de jeux vidéo utilisent plusieurs machines de traitement parallèles complexes pour rendre l'action de jeu en temps réel.

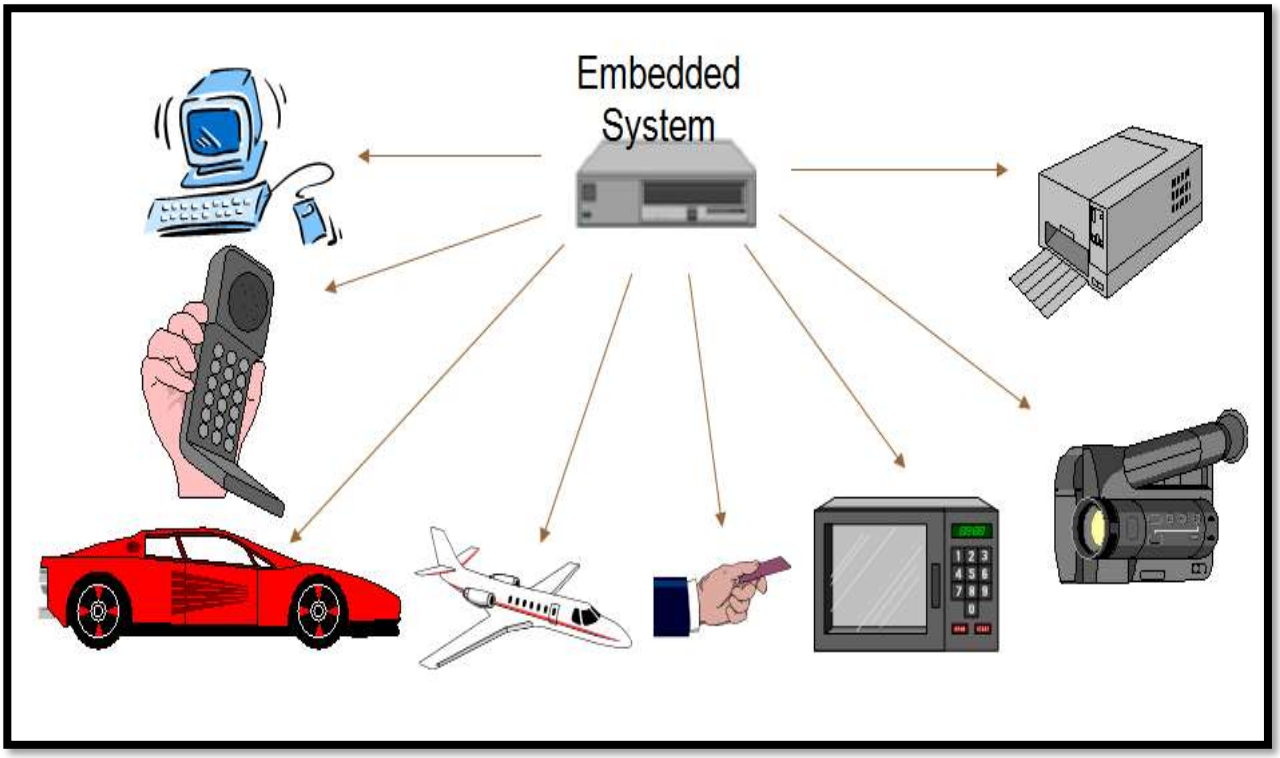


Figure 3 : Exemples d'applications des systèmes sur puce.

1.3. Les différentes structures d'interconnexions.

Les composants, dans un système sur puce, sont reliés entre eux par une structure d'interconnexion qui prend en charge toutes les communications de données entre composants, tant au sein de la puce que des périphériques externes (exemple, les lecteurs de flash externes). Ces structures de communication pour SoC ont été conçues pour avoir une incidence importante sur la performance, la consommation d'énergie, les coûts et les délais de conception de systèmes sur puce.

Dans cette section, nous montrerons les différentes structures d'interconnexions utilisées pour la communication entre les modules d'un SoC.

1.3.1. La connexion point à point.

Ce type d'interconnexion est le plus simple. Il permet de relier chaque deux IPs directement sans aucun protocole de gestion de communication (fig.4). Cette solution offre plusieurs avantages en termes de latence et de débit. De plus, ce type de structure n'a pas besoin d'un arbitrage pour gérer les communications ainsi qu'il ne peut y avoir aucune congestion de données [6]. Cette structure a toutefois des limites au niveau de la flexibilité et de la scalabilité. Par ailleurs, elle ne peut offrir aucune tolérance aux fautes [6, 7] pouvant survenir sur un lien. Donc cette architecture est très limitée et on ne peut pas l'utiliser pour des systèmes compliqués.

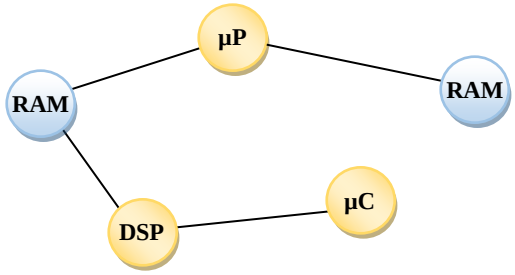


Figure 4 : Connexion point à point.

1.3.2. La connexion à base de bus partagé.

Il s'agit de la structure d'interconnexion la plus utilisée dans les systèmes actuels à cause de sa simplicité. Cette structure est venue pour résoudre les problèmes rencontrés dans l'architecture point à point en termes de flexibilité. Dans les systèmes à base de bus partagé, tous les modules (IPs) sont connectés à un seul médiat de communication ; c'est le bus (fig.5). Cette structure ne permet qu'une seule communication à la fois. Les communications sont gérées par un arbitrage qui décide quel IP a le droit d'accéder au bus pendant un certain temps (en général le temps d'un transfert) et la sélection d'IP se fait par priorité.

Lorsque le nombre d'IP augmente, on a besoin d'étendre le lien qui peut causer plusieurs inconvénients comme : l'augmentation de la consommation énergétique, la réduction de la fréquence de fonctionnement, l'augmentation de la latence ; cela nous donne une perte de temps et diminue la bande passante [7]. Lorsque le nombre de communications augmente, le bus devient un goulot d'étranglement. Dans ce cas, cette architecture ne semble pas fonctionnelle avec les futurs systèmes qui demandent l'intégration de plusieurs éléments pouvant communiquer entre eux en parallèles. Dans cette structure, le nombre d'IPs connectés au bus est limité à une dizaine d'éléments [8].

Citons quelques exemples de bus utilisés dans les systèmes sur puce :

- Le bus AMBA (Advanced Microcontroller Bus Architecture) : ce bus est conçu par la société ARM pour supporter la communication entre les processeurs ARM [9].
- Le bus Sonics SMART Interconnects : de la société Sonics [10]. Plusieurs compagnies tels que Samsung, Toshiba, Broadcom ou encore Texas Instrument utilisent cette structure.
- Le bus CoreConnect : est un bus sur puce développé par la société IBM [11]. Ce type de bus est très utilisé dans les FPGA Xilinx.
- Le STBus : de la société STMicroelectronics. Ce type de bus et dédié au SoC désigné pour des applications qui demandent une grande bande passante comme le traitement de vidéo [12].
- Le bus AVALON : de la société Altera [13].

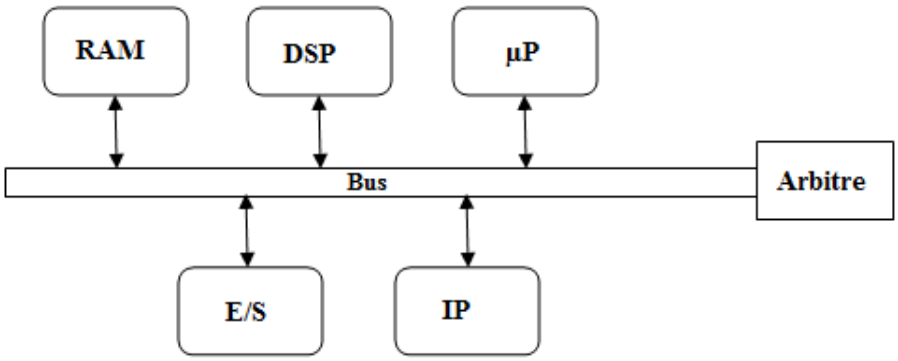


Figure 5 : Structure d'interconnexion Bus partagé.

1.3.3. La connexion à base de bus hiérarchique.

Afin de résoudre les problèmes rencontrés au niveau du bus partagé, l'architecture du bus hiérarchique a été proposée (fig.6). Cette structure permet à plusieurs bus de communiquer entre eux par des ponts (Bridge). Ces structures permettent le parallélisme des communications entre

les IPs qui se trouvent dans des bus différents. Le nombre des IPs reliées à chaque bus est petit, ce qui implique, par rapport au bus partagé, une latence plus petite, une minimisation de la consommation d'énergie et une fréquence de fonctionnement plus élevée. Les communications entre les différents bus se fait à travers le pont ; ce dernier peut jouer un rôle de conversion de protocole dans le cas où les bus n'utilisent pas le même protocole de communication [14].

Cependant, les communications à travers le pont deviennent un goulot d'étranglement, ce qui implique une augmentation de la latence quand deux bus veulent communiquer entre eux. Mais, cette architecture, elle aussi, ne peut pas être fonctionnelle avec les futurs systèmes.

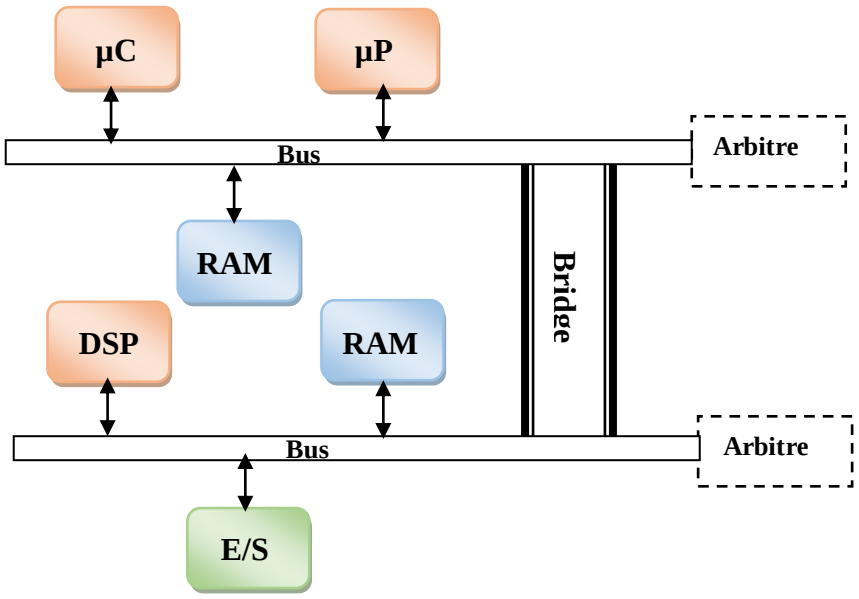


Figure 6 : Structure d'interconnexion Bus hiérarchique.

1.3.4. La connexion à base de Crossbar.

Dans cette structure, tous les modules sont reliés entre eux par un crossbar (fig.7). Ce dernier nous permet des communications en parallèle ainsi que d'augmenter la bande passante car le médiat de transport n'est plus partagé entre tous les modules comme c'est le cas dans les topologies Bus. Cependant, l'inconvénient majeur dans ces architectures est le grand nombre d'interconnexions (câblages) qui croit en fonction du nombre d'IPs intégrées. Le câblage augmente avec le carré du nombre d'éléments communicants : $O(n^2)$.

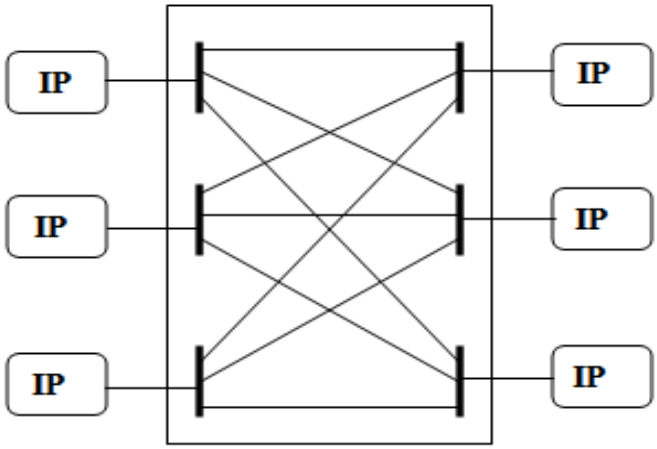


Figure 7 : Structure d'interconnexion Crossbar.

1.3.5. La connexion à base de réseau.

Depuis les années 2000, les concepteurs des SoCs proposent un nouveau paradigme d'interconnexion afin de résoudre les problèmes rencontrés au niveau des structures classiques. Avec le nombre croissant des noyaux intégrés sur une seule puce, la recherche en réseau sur puce (NoC = Network on Chip) devient de plus en plus importante (fig.8). Cette structure est inspirée des réseaux informatiques et adaptée pour les systèmes sur puce. Elle offre une très grande bande passante par rapport aux autres architectures (car elle n'est pas partagée), une diversité des chemins, le parallélisme des communications, la flexibilité, l'extensibilité et la réutilisation facile des IPs.

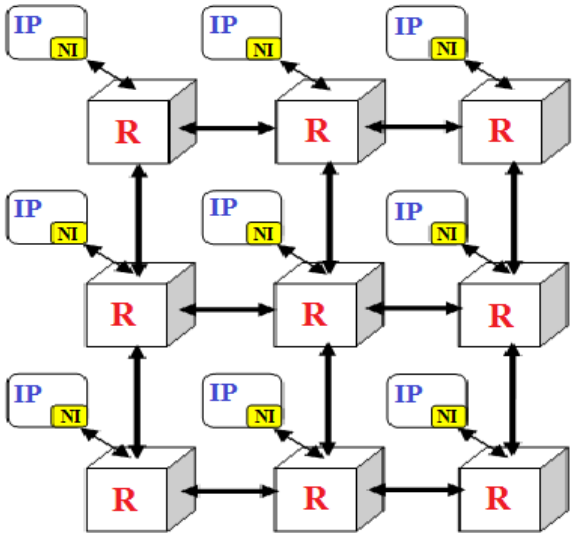


Figure 8 : Structure d'interconnexion NoC.

L'architecture NoC offre plusieurs avantages : scalabilité (cette structure permet d'intégrer de nouveaux nœuds), performance électrique meilleure car tous les liens sont point à point, réutilisation des modules, diversité des chemins, parallélisme des communications, etc. Le tableau 2 résume les caractéristiques de ces structures d'interconnexion.

Tableau 2 : Récapitulatif des structures d'interconnexions.

	Parallélisme	**Consommation**	**Scalabilité**	**Réutilisation**
Point à point	Très bon	Bon	Très mauvais	Très mauvais
Bus partagé	Très mauvais	Très mauvais	Très mauvais	Très bon
Bus hiérarchique	Bon	Mauvais	Mauvais	Très bon
NoC	Très bon	Très bon	Très bon	Très bon

2. Les circuits FPGAs.

Dans le domaine numérique, le choix de la technologie d'implémentation est un point essentiel, que ce soit dans l'utilisation de la logique numérique de fonctionnalité dédiée (fixe ou ASIC), ou dans l'utilisation d'un système logiciel programmé à base de processeur (conçu sur la base d'un microprocesseur, un microcontrôleur, ou un DSP), ou encore dans celle utilisant des dispositifs programmables (PLD, SPLD, CPLD ou FPGA). La figure 9, représente les différents choix de technologies pour la conception des circuits numériques.

Depuis la commercialisation du premier FPGA (Field Programmable Gate Arrays) en 1985, l'utilisation de ce type de circuits ne cesse de s'étendre à des domaines et applications variés, parmi lesquels nous pouvons citer le traitement d'image [15, 16], les réseaux de neurones [17], la comparaison de séquences génétiques [18]. Pendant ce temps, les FPGAs sont devenus de plus en plus complexes et leur capacité d'implémentation des systèmes est passée de quelques milliers à des dizaines de millions de portes logiques.

Initialement, les FPGA ont été conçus pour compléter la conception des circuits ASIC (Application Specific Integrated Circuit) en fournissant la reprogrammation sur la dépense de la dissipation de puissance, la surface de la puce et de la performance. L'intérêt suscité par les FPGAs est dû essentiellement à leurs prix abordables, la facilité de leur mise en œuvre et leur flexibilité. En outre, les coûts fixes et les délais de fabrication, en comparaison avec les circuits

spécifiques (ASIC), sont totalement éliminés. Cependant, ils présentent une faible densité d'intégration de portes logiques et atteignent des fréquences de travail relativement faibles devant les ASICs.

2.1. Les types des circuits logiques programmables.

De nombreux types de circuits logiques programmables sont disponibles (fig.9). Les deux catégories majeures connues sont les PLDs (Programmable Logic Devices) et les FPGAs (Field Programmable Gate Arrays), comme l'indique la figure 9. Les PLDs sont soit SPLDs (Simple PLDs) soit CPLD (Complex PLDs).

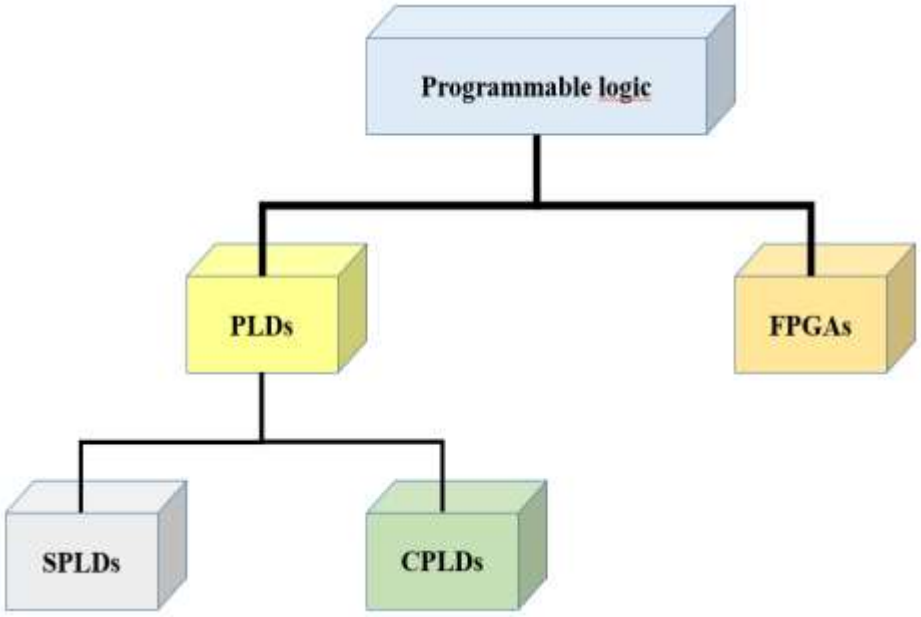


Figure 9 : Différents types de circuits logiques programmables.

2.1.1. Simple Programmable Logic Device (SPLD).

Les circuits SPLDs ont été à l'origine des PLD. Ils sont toujours disponibles pour les applications à petite échelle. Généralement, un SPLD peut remplacer jusqu'à dix circuits intégrés à fonction fixe et leurs interconnexions. La plupart des SPLDs sont représentés par les deux catégories : PAL et GAL. Un PAL (Programmable Array Logic) est un composant qu'on peut programmer une seule fois. Comme représenté dans la figure 10, un PAL est constitué d'un réseau programmable de portes logiques « ET » et d'un réseau fixe de portes logiques « OU ». Un GAL (Generic Array Logic) est un PLA qu'on peut programmer plusieurs fois. Il est constitué d'un réseau programmable de portes logiques « ET » et d'un réseau fixe de portes logiques « OU » avec des sorties programmables, comme présenté dans la figure 10.

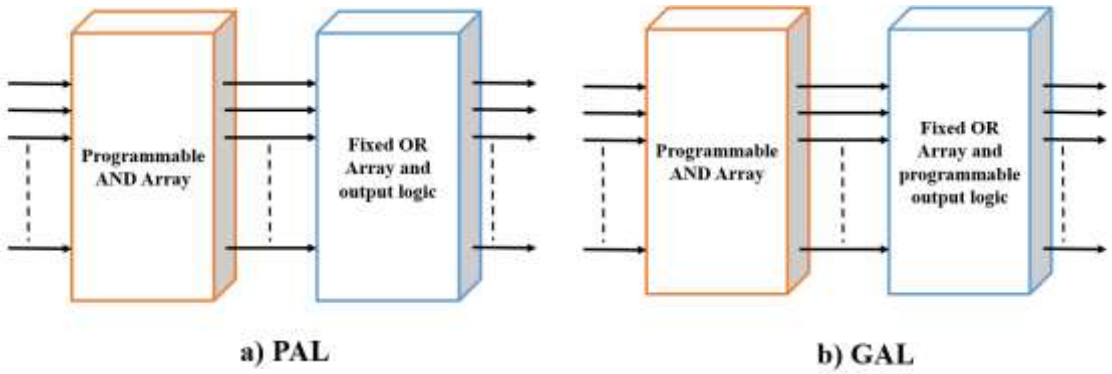


Figure 10 : Schémas synoptiques des SPLDs.

2.1.2. Complex Programmable Logic Device (CPLD).

Comme la technologie a progressé et que la quantité de circuits pouvant être mis sur une seule puce a augmenté (densité d'intégration), les fabricants ont été en mesure de mettre plus d'un SPLD sur une seule puce ; c'est ainsi que le CPLD est né. Essentiellement, le CPLD est un dispositif contenant plusieurs SPLDs et peut remplacer de nombreux circuits intégrés à fonction fixe. La figure 11 montre un schéma synoptique d'un CPLD basique avec quatre LABs (Logic Array Blocks) et une PIA (Programmable Interconnection Array). En fonction de la CPLD spécifique, il peut y avoir de deux à soixante-quatre LABs. Chaque LAB est à peu près équivalent à un SPLD.

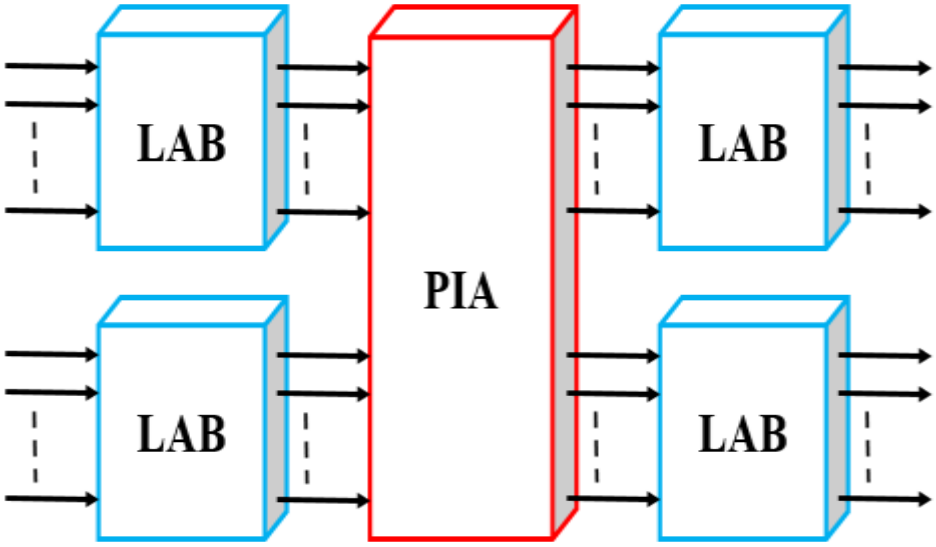


Figure 11 : Schéma synoptique d'un CPLD.

Avec les CPLDs, les concepteurs ont atteint la densité d'intégration maximale pour ce type de circuits ; de nouvelles études ont conduit à l'apparition des FPGAs.

2.1.3. Field Programmable Gate Array (FPGA).

Les circuits FPGAs, ou bien les réseaux logiques programmables, sont des composants électroniques programmables de la famille des PLDs. Un FPGA est un ensemble de blocs logiques élémentaires que l'utilisateur peut interconnecter pour réaliser les fonctions logiques de son choix.

Un FPGA est généralement plus complexe et a une densité beaucoup plus élevée qu'un CPLD, bien que leurs applications puissent parfois se chevaucher. Comme mentionné plus haut, le SPLD et le CPLD sont étroitement liés car le CPLD contient essentiellement un certain nombre de SPLDs. Le FPGA a cependant une structure interne différente comme illustré dans la figure 12. Les FPGAs utilisent des modules logiques plus réduits (par rapport aux CPLDs) mais beaucoup plus nombreux. Les interconnexions entre ces modules ne sont pas centralisées comme dans les CPLDs. Les trois éléments de base dans un FPGA sont les blocs logiques, les interconnexions programmables, et les blocs d'entrée/sortie (fig.12).

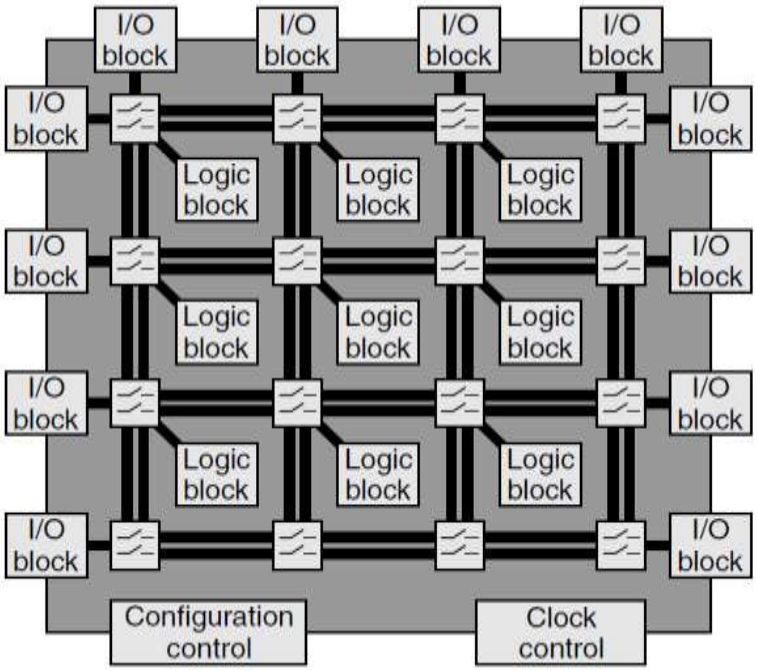


Figure 12 : Architecture de base du circuit FPGA.

2.2. Pourquoi utilise-t-on le FPGA ?

La décision de choisir une technologie spécifique comme un FPGA devrait dépendre principalement des exigences de conception plutôt que d'une préférence personnelle pour une technique plutôt qu'une autre. Par exemple, si la conception nécessite un dispositif programmable

avec de nombreux changements de conception, et des algorithmes utilisant des opérations complexes telles que des multiplications et des bouclages, alors il peut être plus judicieux d'utiliser un dispositif de traitement du signal dédié comme un DSP qui peut être programmé et reprogrammé facilement en utilisant le langage C ou d'autres langages de haut niveau. Si les exigences de vitesse ne sont pas particulièrement strictes, et une plate-forme compacte pas chère est nécessaire, alors un microprocesseur d'usage général comme un PIC serait un choix idéal. Enfin, si les exigences matérielles nécessitent un niveau supérieur de performance, alors un FPGA offre un niveau acceptable de performance, tout en conservant la flexibilité et la réutilisation de la logique programmable.

Les avantages principaux des FPGAs par rapport aux conceptions ASICs peuvent être résumés comme suit :

- Le temps de commercialisation (Time to Market) des FPGAs est largement minime par rapport à celui des conceptions ASIC. Une fois un design entièrement testé est disponible, la conception peut être gravée dans le FPGA et vérifiée, d'où l'ouverture de la phase de production. En conséquence, les FPGAs éliminent le temps d'attente de fabrication.
- Les FPGAs sont d'excellents candidats pour les productions à faible volume car ils éliminent le coût de génération de masque.
- Les FPGAs sont idéaux pour les besoins de prototypage. Le test et la vérification du matériel peuvent être effectués rapidement sur la puce. En outre, les erreurs de conception peuvent être facilement fixées, sans frais de matériel supplémentaires.
- Les FPGAs sont polyvalents et reprogrammables, ce qui leur permet d'être utilisés en plusieurs conceptions sans frais supplémentaires.

2.3. Architecture des circuits FPGAs.

Les FPGA sont des composants logiques de haute densité et reconfigurables qui permettent, après programmation, de réaliser des fonctions logiques, des calculs, et des générations de signaux. En effet, leur structure régulière (comme montré dans la figure 12) en fait des éléments très performants pour les traitements de bas niveaux réguliers. Les principaux éléments composant les FPGAs sont :

- **Les blocs logiques configurables (CLB) :** Les CLBs sont les briques de bases de tous les FPGAs. Ils réalisent une fonction logique élémentaire (exemple : une fonction NAND à 3 entrées). En assemblant plusieurs de ces fonctions élémentaires, on obtient une fonction

complexe. On peut, par exemple, réaliser un additionneur, un multiplieur ou un processeur. Chaque CLB est une cellule constituée d'éléments logiques programmables où l'on trouve des bascules (registres), des LUTs (Look-Up Table), des multiplexeurs et des portes logiques (fig.13). Le LUT est composé de cellules de mémoire (cellules SRAM). Un LUT a typiquement quatre entrées et il est utilisé pour mettre en œuvre une logique combinatoire en stockant la table de vérité de toute équation logique combinatoire jusqu'à quatre variables d'entrée. Il convient de noter, toutefois, que certains FPGAs ont des LUTs avec aussi peu que trois entrées ou jusqu'à six entrées. Le LUT dans certains FPGAs peut également être configuré pour fonctionner comme une petite mémoire vive (RAM) ou registre à décalage. Au cours du temps, et grâce à l'évolution de la technologie, l'architecture de ces cellules a évolué en complexité. Par ailleurs, les FPGAs intègrent de plus en plus de cellules sur une même surface. On trouve par exemple jusqu'à 400000 CLBs appelés ALM (high performance Adaptation Logic Modules) dans les Stratix V d'Altera [19] et 1 956 000 CLBs pour les Virtex 7 de Xilinx [20].

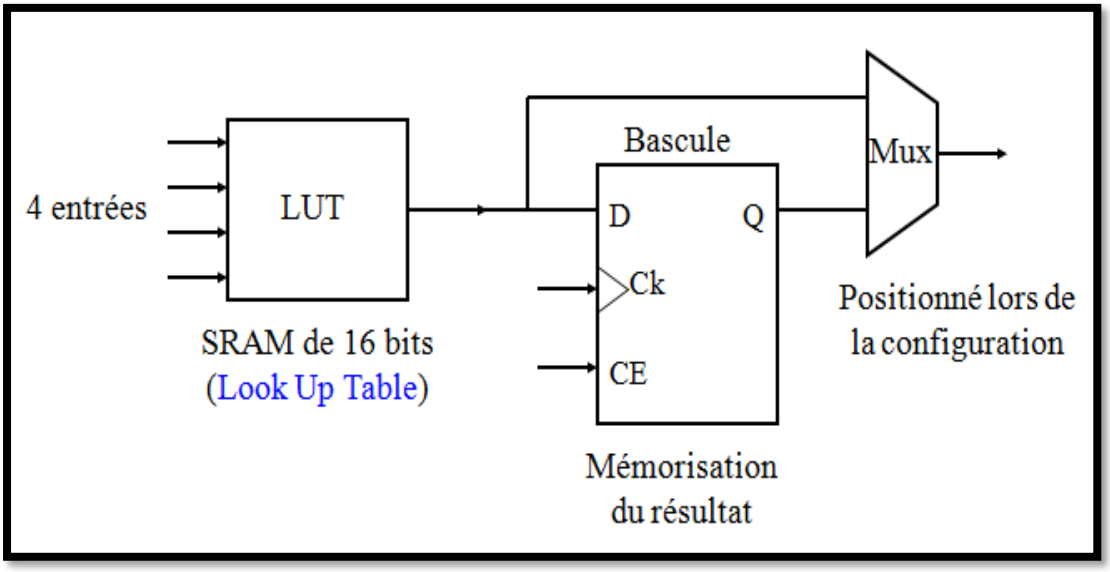


Figure 13 : Architecture d'un CLB.

- **Le réseau d'interconnexions programmables :** Pour pouvoir réaliser des fonctions complexes à partir des fonctions de base que représentent les CLBs, il est nécessaire de disposer de ressources d'interconnexion entre ces différentes cellules. Le réseau d'interconnexions est programmable c'est-à-dire que pour relier une ligne de métal à une autre, on active un transistor qui fonctionne comme un interrupteur.

- **Les blocs d'entrée/sortie :** Ces cellules d'entrées/sorties permettent d'interfacer le FPGA avec l'environnement extérieur. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc d'entrée/sortie contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé.

On a vu que le principal atout des FPGAs par rapport aux ASICs est la flexibilité et le faible coût d'investissement. Ainsi, les FPGAs sont très intéressants pour les produits vendus en faibles volumes. Tandis que pour les forts volumes, les ASICs sont plus économiques. Cependant, les ASICs possèdent encore beaucoup d'avantages par rapport aux FPGAs. En effet, les ASICs sont plus rapides, consomment moins et ont une surface (et donc un coût) plus faible que les FPGAs pour un même circuit. Le fossé qui existe entre les ASICs et les FPGAs est immense [21] :

- Un FPGA demande 20 à 40 fois plus de surface qu'un ASIC.
- Un FPGA est 3 à 4 fois moins rapide qu'un ASIC.
- Un FPGA consomme (consommation dynamique) de 10 à 15 fois plus qu'un ASIC.

De nombreuses recherches existent pour réduire ces écarts de performance [22]. L'une des techniques est d'introduire dans le FPGA des circuits logiques spécifiques c'est-à-dire non programmables. Ce sont des circuits gravés dans le silicium qui accomplissent une fonction précise. On peut choisir de les utiliser ou pas mais on ne peut pas les supprimer. On parle de « hard logic » lorsqu'un bloc n'est pas programmable, contrairement aux blocs logiques programmables, appelés « soft logic » car leur fonctionnement peut être déterminé simplement en programmant des cellules mémoire. Les circuits « hard » implémentés dans les FPGAs sont des circuits très courants dans les systèmes numériques.

Depuis quelques générations de FPGAs, afin d'améliorer leurs performances, les fabricants ont ajouté aux ressources classiques, sur certains modèles, de nombreux éléments tels que :

➢ **Les blocs mémoire :** Les FPGAs classiques peuvent se comporter comme un espace de stockage de variables ; cet espace de stockage est réparti dans tout le FPGA. Ce mode de stockage est tout de même limité en termes d'espace disponible. C'est pourquoi, pour pouvoir stocker une somme plus importante de variables sans avoir à accéder à des mémoires externes, certains fabricants de FPGAs ont introduit des blocs mémoires à l'intérieur des FPGAs, de 10 à 90 Mbits suivant les constructeurs et la taille de FPGA.

➢ **Les multiplieurs :** La logique présente dans les FPGAs permet de réaliser toutes sortes d'opérations arithmétiques (additions, multiplications, …). Ces opérations, et en particulier les multiplications sont très coûteuses en termes de ressources logiques utilisées, d'où l'intérêt de disposer dans les FPGAs de multiplieurs câblés (plus d'une centaine sont en général disponibles sur les modèles haut de gamme). De plus, le temps de calcul pour ces opérations est alors optimisé. Les multiplieurs sont très utilisés dans le traitement de signal.

➢ **Les blocs processeur :** avec l'apparition des très grands FPGAs, il est maintenant possible d'intégrer des algorithmes complexes sur une seule puce. Certains FPGAs disposent aujourd'hui de cœurs de processeurs. Au début des années 2000, plusieurs fabricants proposent des cœurs de processeurs matériels au sein des FPGA (Power PC pour Xilinx ou ARM pour Altera). Cette disposition absente dans les FPGA depuis 2005 apparaît de nouveau dans les prochaines générations de FPGA proposés par Xilinx (ZYNQ 7000). Cette nouvelle famille intègre un ou plusieurs cœurs ARM dans le FPGA.

2.4. Les méthodes de programmation des FPGAs.

Il y a plusieurs constructeurs des circuits FPGAs tels que Xilinx, Altera, Actel, Latice. Ces constructeurs utilisent différentes technologies pour la réalisation des FPGAs. Parmi ces technologies, celles qui assurent une programmation des FPGAs sont les plus intéressantes étant donné qu'elles permettent une grande flexibilité de conception. Les principaux types de programmation sont (Tab.3) :

➢ **La technologie SRAM :** La technologie la plus utilisée est la cellule SRAM. Elle est notamment utilisée dans les FPGAs de Xilinx, Altera et Lattice. Elle est composée de six transistors ce qui est une taille élevée comparée aux autres technologies. Les connexions sont réalisées en rendant les transistors passants. Cette technologie permet une reconfiguration rapide du circuit FPGA. Cependant, son principal inconvénient est la surface nécessaire pour la SRAM. En plus de la taille de la cellule, la SRAM a l'inconvénient d'être volatile, c'est-à-dire qu'elle perd sa donnée lorsqu'elle est hors tension. Il faut donc ajouter une mémoire annexe (une mémoire FLASH) pour stoker la configuration et la charger lors de chaque mise sous tension du FPGA.

➢ **La technologie Anti-fusible :** Elle est notamment utilisée dans les FPGAs de Actel et Quicklogic. Une cellule anti-fusible ne représente aucune surface de silicium. En effet, elle est composée d' une couche de matériau isolant (comme de l'oxyde de silicium) déposée entre deux

couches de métal. Elle permet d'établir une connexion entre deux couches de métal. La programmation se fait en appliquant un fort courant dans la cellule, ce qui rend la cellule passante (résistance entre 20 et 100 ohms [22]). Cette technologie nécessite un circuit supplémentaire pour générer les forts courants lors de la programmation. L'un des inconvénients principaux est le fait que les cellules ne peuvent être programmées qu'une seule fois. De plus, on ne peut pas programmer le FPGA sur son circuit imprimé. Il faut un équipement spécial pour le programmer.

- **La technologie Flash :** La mémoire FLASH est une autre technologie de mémoire utilisée principalement dans les FPGAs de chez Actel. Cette technologie est limitée en nombre de reconfigurations et possède un temps de configuration plus long par rapport à la technologie SRAM. Les FPGAs Flash de chez Actel ne peuvent pas être reprogrammées plus de 500 à 1000 fois [23]. Une cellule mémoire Flash est principalement composée d'un transistor avec une grille flottante, dont l'état « chargé » ou « non-chargé » modifie la tension de seuil. Cette valeur de tension de seuil détermine la valeur stockée dans la cellule. Dans le cas des FPGAs de Actel, la cellule est composée d'un seul transistor pour la programmation et d'un transistor utilisé comme interrupteur. L'avantage de cette technologie est qu'elle garde sa configuration même si l'alimentation est enlevée.

Tableau 3 : Récapitulatif des technologies de configurations.

	SRAM	**Anti-fusible**	**Flash**
Vitesse	Rapide	Rapide	Lente
Densité	Structure large et densité Faible	Structure petite et densité grande	Entre les deux
Volatilité	Volatile	Non volatile	Non volatile
Reconfiguration	Oui	Non	Oui
Sécurité	Moyenne	Très bonne	Très bonne
Consommation	Grande	Faible	Moyenne
Fabricant	Xilinx, Altera, Atmel	Actel, Quicklogic	Actel, Lattice

Afin de programmer un circuit FPGA on doit disposer des éléments suivants (fig.14) :

- Un ordinateur ;
- un logiciel de développement (Xilinx ISE, Quartus, …) ;
- un circuit programmable FPGA ;
- un moyen pour connecter le FPGA à l'ordinateur.

Figure 14 : Eléments essentiels pour programmer les circuits reconfigurables.

Les FPGAs se programment grâce à leurs LUTs et leurs réseaux d'interconnexion. Leur programmation consiste à écrire dans les cellules mémoires de configuration. Pour déterminer la valeur à stocker dans chaque cellule mémoire, le programmeur est aidé d'un logiciel de développement, comme ISE de chez Xilinx ou Quartus de chez Altera, comme le montre la figure 15, à partir d'un fichier-texte écrit en un langage de description de circuits numériques (VHDL ou Verilog) ou d'un schéma (Schématic). L'outil de développement transforme cette description en un fichier de configuration du FPGA en plusieurs étapes. D'abord, le logiciel effectue la synthèse du circuit à implémenter. L'étape de synthèse consiste à convertir les fichiers-textes (décrivant le circuit) en un fichier RTL (Register Transfer Level) qui décrit le circuit au niveau porte logique puis la dernière étape qu'est l'implémentation. Le processus d'implémentation prend quatre étapes pour convertir le netlist à un fichier de programmation finale : Translate, Map, Place and Route, et de générer le fichier de programmation.

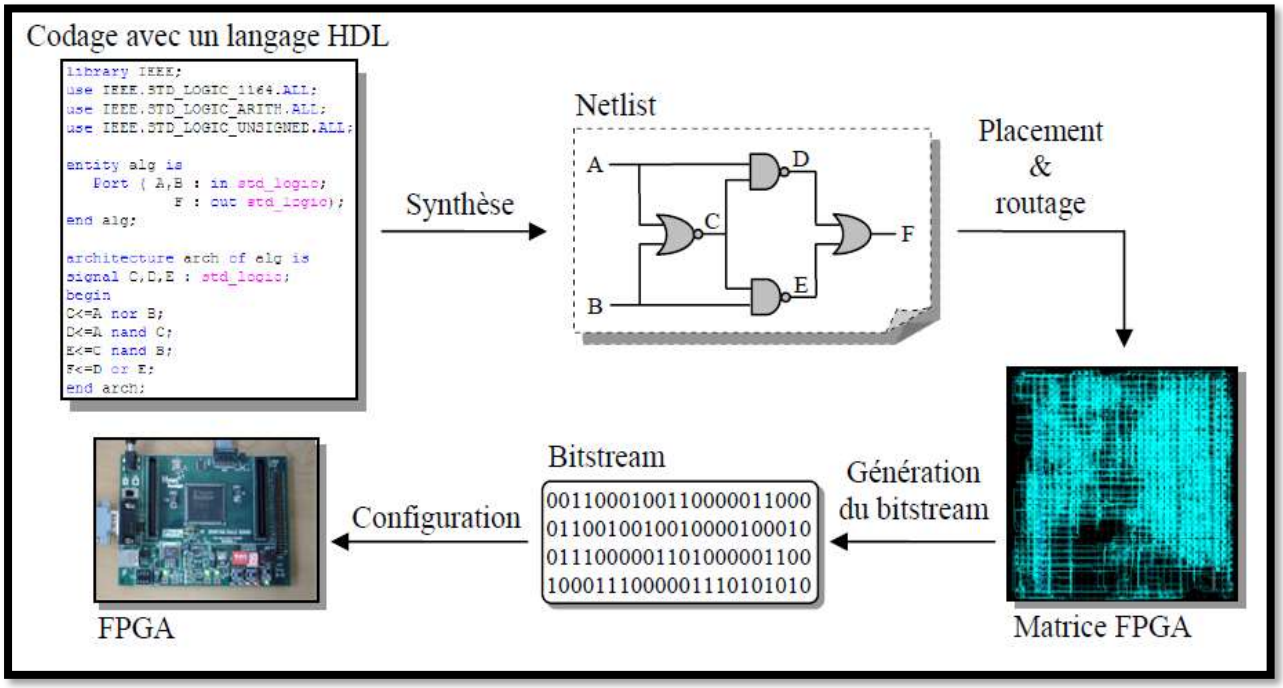


Figure 15 : Etapes réalisées par les outils de développement afin de programmer un FPGA.

3. Conclusion.

A court terme, les systèmes sur puce domineront de plus en plus le marché des circuits intégrés. Leur complexité ainsi que leurs performances deviendront de plus en plus élevées. Aujourd'hui, la conception des SoCs est de plus en plus utilisée dans les systèmes à haute performance. Dans de tels systèmes, le choix de la structure d'interconnexion est très important afin d'augmenter la performance. Dans ce chapitre, nous avons donné un aperçu sur quelques architectures d'interconnexions et leurs limites. La structure d'interconnexion deviendra à terme le goulot d'étranglement du développement des systèmes. Les architectures fondées sur des bus atteignent dès aujourd'hui leurs limites ; aussi de nouvelles architectures fondées sur des réseaux commencent-elles à être envisagées.

Nous avons commencé par la connexion point à point ; pour résoudre les problèmes rencontrés dans cette structure, les concepteurs ont proposé l'architecture à base de bus partagé qui ne supporte pas les communications en parallèle. Pour diminuer les problèmes de bus partagé au niveau de la latence, la bande passante et le parallélisme, une nouvelle structure est apparue ; c'est le bus hiérarchique, il n'a pas résolu ces problèmes mais il les a diminués et a permis en plus des communications en parallèle se déroulant dans des bus différents. La structure crossbar offre une très grande passante car chaque deux IPs sont reliés entre eux par un lien dédié, mais le

problème majeur dans ces architectures reste le grand nombre de liens qui augmente suivant le nombre d'IPs intégrés.

Dans ce premier chapitre, nous avons aussi présenté une étude comparative entre les différents types de circuits reconfigurables et les circuits ASICs ainsi que les techniques de configuration principales (SRAM, Flash et Anti-fusible) utilisées par les constructeurs des circuits FPGAs (Xilinx, Altera, Actel, …). En plus, nous avons présenté les éléments constitutifs de l'architecture FPGA.

CHAPITRE 2

Etat de l'art sur les réseaux sur puce

Grâce à la densité d'intégration continuellement croissante des circuits numériques, les systèmes sur puces sont capables de répondre aux besoins d'aujourd'hui qui consistent à regrouper plusieurs applications hétérogènes sur un même support. Comme nous l'avons mentionné dans le chapitre précédent, l'interconnexion entre les différents modules est l'un des problèmes majeurs dans la conception des systèmes sur puce. Ainsi, les performances du système dépendent en grande partie de la structure de communication, notamment en ce qui concerne le débit, la surface et la puissance consommées.

L'architecture d'interconnexion de type bus partagé est l'architecture d'interconnexion la plus utilisée dans les SoCs actuels. Cette structure ne permet qu'une seule transaction de communication à la fois en fonction du résultat d'arbitrage. La bande passante moyenne de la communication de chaque élément de traitement est en proportion inverse du nombre total d'IPs dans un système. Les systèmes d'aujourd'hui, qui peuvent intégrer plusieurs unités de traitement dans une seule puce demandant un débit élevé et une faible latence, ne peuvent pas être satisfaits par l'intégration d'un bus partagé comme une structure d'interconnexion.

Depuis les années 2000, le réseau sur puce (NoC) a été proposé au cours des dernières années comme une solution prometteuse pour résoudre les problèmes rencontrés au niveau des interconnexions classiques [24, 25]. Cette nouvelle structure d'interconnexions est encore en cours d'étude afin de pouvoir évaluer leurs performances, les comparer entres elles et guider les choix de leur conception.

Le but de ce premier chapitre est d'introduire le paradigme des réseaux sur puce ainsi que les notions qui lui sont associées. A cet effet, nous détaillerons les différents éléments de base, qui constitues cette structure, puis nous décrirons les points de caractérisation des réseaux sur puce, à savoir les métriques de performances, les différentes topologies, les algorithmes de routage, les protocoles de communication, le format de paquet, la qualité de service, le contrôle de flux et de congestion. Nous présenterons aussi quelques NoCs académiques et industriels.

1. Architecture des réseaux sur puce.

1.1. Les éléments de base d'un NoC.

Un NoC typique se compose de trois éléments principaux. Chaque réseau est composé principalement des routeurs qui permettent d'acheminer les données à partir d'une source vers une destination, des liens qui permettent de connecter les nœuds et de transférer les données entre elles (ils jouent un rôle important dans la performance de l'architecture des NoCs) et des interfaces réseaux (NIs) (fig.16). Les IPs (Intellectual Properties) pourraient être n'importe quels

composants tels qu'un microprocesseur, un circuit intégré spécifique à une application (ASIC), une mémoire, ou une combinaison de composants reliés entre eux. Ces IPs sont connectées aux routeurs de réseau par l'intermédiaire d'une interface de réseau.

Dans cette section nous montrerons les différents éléments de base du NoC : les routeurs, les liens et les interfaces réseau. Nous présenterons aussi les différentes caractéristiques comme les topologies les plus utilisées, les algorithmes de routage, les stratégies de commutation, …

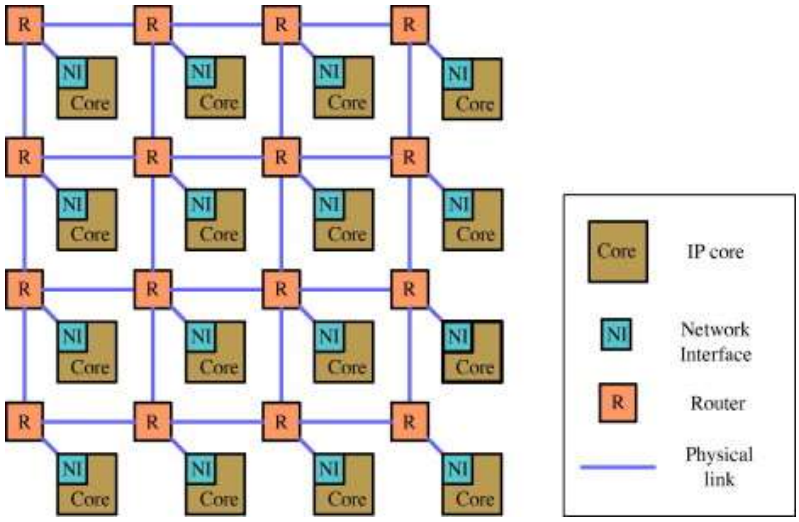


Figure 16 : Les éléments de base d'un NoC.

1.1.1. Le routeur.

La conception du routeur joue un rôle important pour décider de la performance des systèmes à base du NoC. Le routeur est responsable de la communication entre le nœud associé et le reste des nœuds à travers la couche réseau. L'architecture d'un routeur dans un NoC dépend de la topologie du réseau. Généralement, un routeur est constitué d'un ensemble de buffers FIFO à chaque port, un crossbar qui relie les ports d'entrée avec ceux de sortie, une unité de routage et d'arbitrage (fig.17).

- **Les buffers FIFO.**

Le buffer est un élément principal dans l'architecture du routeur ; son rôle est de stocker temporairement les informations transmises (généralement des paquets). La taille de la FIFO dans le routeur est très importante car elle influe directement sur le délai du routage, la perte de paquets, la consommation d'énergie et la surface occupée. Cette taille dépend du mode de commutation choisi [26]. Les buffers peuvent contenir plusieurs paquets, un seul paquet ou encore une partie du paquet. Concernant la largeur du buffer, celle-ci ne pourra pas être plus

petite que celle d'une unité de contrôle de flux. Les buffers peuvent être placés en entrée, en sortie, ou encore en sortie virtuelle.

- File d'attente en entrée : Un seul buffer est positionné à chaque port d'entrée du routeur. Cette technique a l'avantage de réduire l'espace mémoire, mais dans un autre sens elle peut induire une saturation causée par le blocage de l'entête de ligne. Ce phénomène peut se produire lorsque l'entête de la file ne peut accéder au port de sortie associé, bloquant ainsi les autres paquets de la file même si leur sortie est libre.
- File d'attente en sortie : Dans cette technique, chaque port de sortie a autant de files d'attente que le port d'entrée. Cette technique offre de meilleures performances par rapport à la technique précédente, mais elle nécessite beaucoup de fils de connexion, un espace mémoire important et forcément une surface importante.
- File d'attente en sortie virtuelle : L'idée de cette technique est de combiner les avantages des deux précédentes techniques. Le principe est que chaque port d'entrée possède plusieurs files d'attente servant à répartir les paquets entrant en fonction de leur destination ou bien de leur priorité. On crée ainsi des canaux virtuels. Malgré l'augmentation de la surface et de la latence, les canaux virtuels possèdent plusieurs avantages tels que : l'évitement des inter-blocages, l'optimisation de l'utilisation des liens, etc.

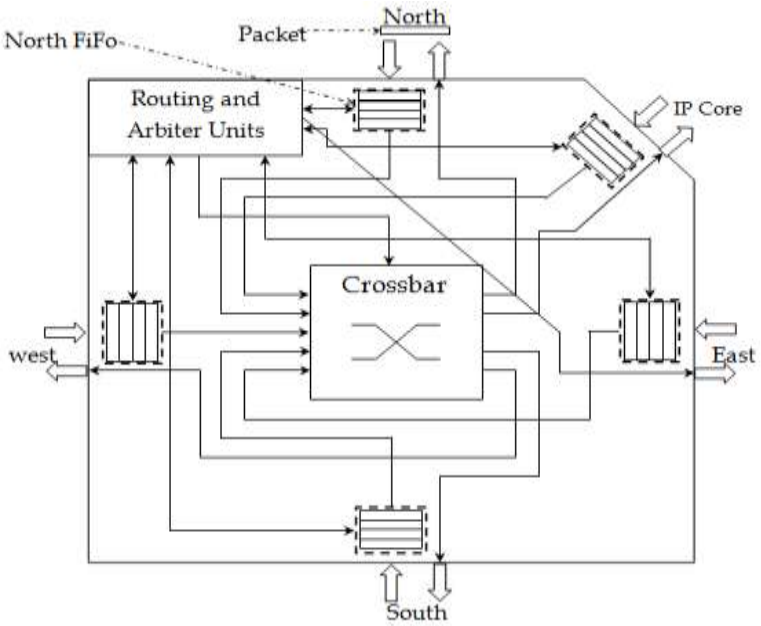


Figure 17 : Architecture d'un routeur d'une topologie Mesh 2D.

- **Le Crossbar.**

Le Crossbar est un commutateur non bloquant qui permet la connexion entre les entrées et les sorties d'un routeur. Il est utilisé pour l'acheminement des paquets du buffer d'entrée au port de sortie approprié en se basant sur l'adresse incluse dans l'entête du paquet.

- **L'unité de routage.**

Cette unité est responsable du décodage de l'unité d'information fournie par le message entrant. Elle est basée sur l'algorithme de routage et la destination du message. Son rôle est de sélectionner les ports de sortie les plus appropriés pour la transmission du message.

- **L'unité d'arbitrage.**

Un composant arbitre est utilisé pour déterminer quel paquet obtiendra la priorité lorsque plusieurs paquets provenant de différentes sources sont en lice pour le transfert sur le même lien d'interconnexion. Ainsi les paquets traversent de multiples liens et passent par un ou plusieurs routeurs dans le réseau NoC comme ils se déplacent de la source à la destination.

Les principales politiques d'arbitrage que l'on peut trouver dans les NoCs sont le Time Division Multiple Access (TDMA), le tourniquet (ou Round-Robin) et la priorité fixe.

TDMA : Cette politique est basée sur la partition du temps en des périodes courtes appelées plages de temps. Un ou plusieurs slots de temps sont attribués à chaque élément communicant. La ressource partagée sera réservée entièrement à l'élément auquel le slot de temps est associé.

Round-Robin : Dans cette politique, l'accès est donné à un paquet provenant d'une entrée différente à chaque cycle.

Priorité fixe : L'émetteur donne à chaque paquet une priorité fixe avant son entrée au réseau ; cette priorité est utilisée tout le long du chemin.

1.1.2. Les liens.

Le rôle des liens est de transporter les informations entre les routeurs et entre l'interface réseau et son routeur. Le lien est celui qui offre la bande passante pour les communications entre une source et une destination. Les liens peuvent être unidirectionnels ou bidirectionnels comme ils peuvent se composer de plusieurs canaux physiques ou logiques afin de réduire le taux de congestion lors des communications [27].

1.1.3. L'interface réseau.

Le troisième élément dans les réseaux sur puce est l'interface réseau (NI). Cet élément est l'intermédiaire entre l'IP et le réseau. L'interface réseau est très importante car elle sépare les calculs (au niveau des IPs) et les communications (au niveau du réseau). Elle permet aussi la réutilisation des cœurs et des infrastructures de communication indépendamment l'une de l'autre. Les interfaces réseaux servent à adapter les éléments de calculs à la structure de communication. Pour cela, l'interface réseau fait l'adaptation de protocole entre la ressource et le réseau, donc elle convertit le protocole d'entrée/sortie des nœuds de calcul vers le protocole de communication utilisé dans le NoC [28].

Afin d'être échangés efficacement à travers le réseau, La NI divise les messages en un ou plusieurs paquets. Chaque paquet représente un segment du message auquel est rajouté un entête de paquet. Cet entête contient les informations de routage qui sont nécessaires à la propagation du paquet dans le réseau.

Les flits associés à un paquet de données sont constitués d'un flit entête (head), d'un flit de queue et d'un certain nombre de flits du corps entre les deux (fig.18). L'entête comporte les informations de routage et de séquençage. La charge utile représente les données réelles qui doivent être transmises. Enfin, le paquet possède également une partie qui indique la fin du paquet, c'est la queue ; elle contient généralement le code de vérification d'erreur.

Le bloc (NI) est composé de deux parties : la première est la partie back-end qui est liée au routeur ; elle est indépendante de l'unité de traitement. La deuxième est la partie front-end connectée à la ressource de traitement qui sert d'adaptateur entre le protocole de l'unité de traitement et celui de la structure de communication.

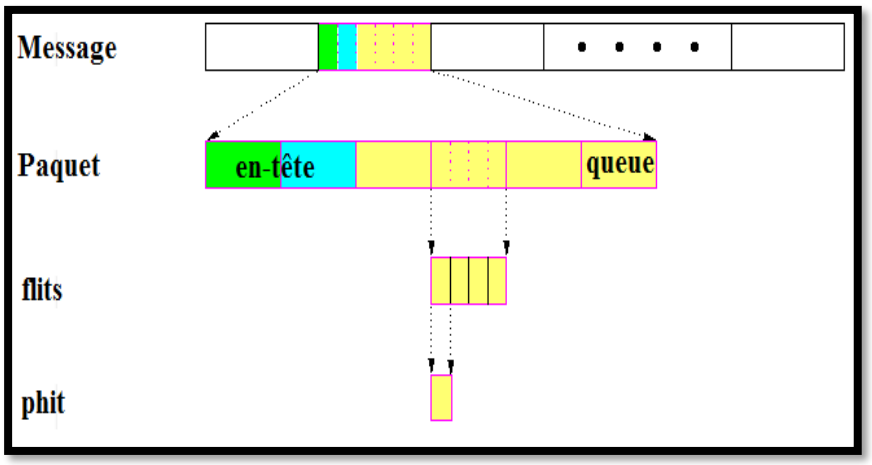


Figure 18 : Structure d'un message.

2. Les caractéristiques d'un NoC.

Chaque réseau est caractérisé par une topologie, un mode de commutation, un algorithme de routage et une politique de contrôle de flux.

2.1. La topologie.

La topologie des réseaux sur puce détermine la disposition physique et les connexions entre les nœuds et les liens dans le réseau [29]. La topologie est la première étape dans la conception des réseaux car à partir de la sélection de la topologie on choisit l'algorithme de routage qui convient ainsi que la méthode de contrôle de flux. La topologie est un point important de la performance des NoCs. Elle est représentée par un graphe G (N, C), où N est l'ensemble des routeurs qui composent le réseau et C l'ensemble des canaux de communication. Le NoC peut être organisé en différentes topologies. Plusieurs topologies ont été proposées et mises en œuvre pour les systèmes multi-cœurs, y compris maille, tore, anneau, papillon et des réseaux d'interconnexion irrégulière [29, 30] (fig.19). La complexité d'implémentation de la topologie dépend de deux facteurs : le nombre de liens à chaque nœud et la facilité de poser la topologie sur une puce. Comme les routeurs et les liens traversés par un paquet encourent énergie, l'effet d'une topologie sur le nombre de sauts affecte aussi directement la consommation d'énergie du réseau.

Une topologie de réseau peut être classée comme étant soit directe, soit indirecte soit encore hybride. Pour les topologies directes (Mesh, Tore, Anneau), chaque élément de calcul (IP) est associé à un routeur ; donc tous les routeurs sont des sources et des destinations. Les paquets sont transmis directement entre les nœuds terminaux. Dans une topologie indirecte (Papillon, Arbre élargi), les routeurs sont différents des éléments de calcul ; seuls les éléments de calcul sont les sources et les destinations du trafic ; les nœuds intermédiaires changent simplement le trafic vers et à partir des éléments de calcul [31]. Pour les réseaux hybrides, cette topologie est basée sur une combinaison entre les deux précédentes topologies (directe et indirecte).

On peut aussi classer les topologies selon leur régularité ; il y a des topologies régulières et d'autres irrégulières. On dit qu'un réseau ou une topologie est régulière lorsque tous les nœuds ont le même degré (tous les nœuds ont le même nombre de voisins). Dans le cas contraire, on dit que la topologie est irrégulière. Le routage dans les réseaux irréguliers est très compliqué et difficile. Les topologies irrégulières permettent plus de liberté et ainsi de dimensionner précisément le réseau requis. Elle peut être issue d'une topologie régulière qui a été retaillée au plus juste de façon à enlever les éléments non utilisés [27]. Une topologie irrégulière nécessite en revanche une plus grande attention pour le routage car les règles à appliquer ne sont plus triviales.

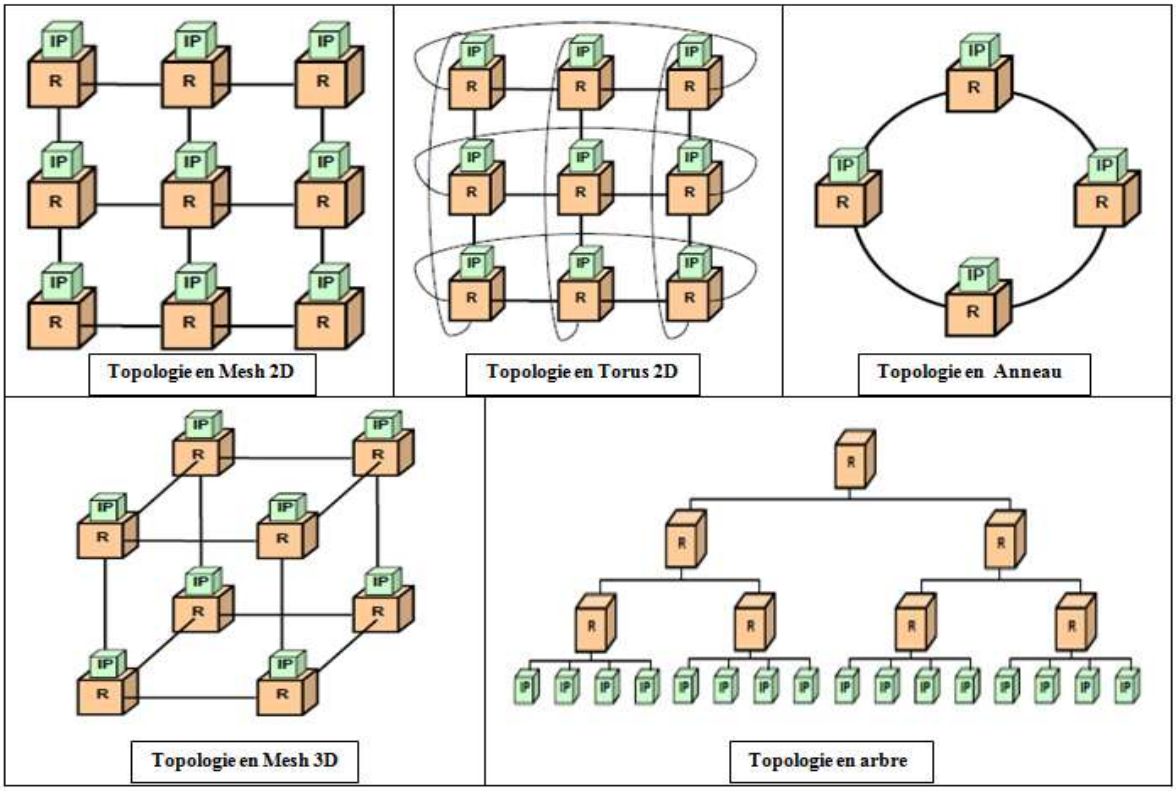


Figure 19 : Quelques topologies pour les NoCs.

Les NoCs de la littérature utilisent généralement des topologies régulières. La topologie la plus utilisée est celle en grille 2D (2D mesh). Cette topologie est la plus utilisée car son arrangement physique est régulier ainsi qu'elle est facile à implémenter sur des circuits reconfigurables (FPGA) et elle offre une grande scalabilité (capacité d'accroître facilement la structure matérielle pour répondre à une exigence de performances). Elle permet d'utiliser des stratégies de routage simple et donc peu coûteuses.

Cette topologie se compose de « m » colonnes et « n » lignes. Les routeurs sont situés dans les intersections de deux fils, et les ressources de calcul sont proches des routeurs. Les adresses des routeurs et des ressources peuvent être facilement définies comme coordonnées XY en maille. Les nœuds le long de la périphérie du réseau, dans une topologie Mesh, ont un degré plus bas que les nœuds dans le centre du réseau. Cette topologie est la plus utilisée car son arrangement physique est régulier ainsi qu'elle est facile à implémenter sur des circuits reconfigurables (FPGA).

On peut comparer les topologies selon plusieurs critères. Parmi ces critères on cite les plus courants :

Le degré du routeur : Désigne le nombre de liens à chaque nœud.

Le diamètre : C'est la distance maximale entre une paire de nœuds dans le réseau.

La distance moyenne : C'est le nombre moyen de liens entre deux ressources.

La diversité de chemins : Chaque paire source/destination a plusieurs chemins en son sein. Cette métrique permet au réseau d'être tolérant aux fautes.

La régularité : On dit qu'un réseau est régulier lorsque tous les routeurs ont le même degré.

La connectivité : Présente le nombre de voisins directs pour chaque nœud 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.

2.2. Les algorithmes de routage.

Comme nous l'avons décrit précédemment, la topologie définit l'organisation physique du réseau composé par les nœuds. L'algorithme de routage est l'étape qui suit la détermination de la topologie du réseau. En fait, une topologie donnée définit les chemins disponibles entre l'ensemble des nœuds. L'algorithme de routage est responsable de décider quel chemin le message doit prendre afin d'être acheminé efficacement de sa source à sa destination. Le choix de l'algorithme de routage devient d'une très grande importance pour augmenter la performance du réseau. Le but de l'algorithme de routage est de répartir le trafic de la même manière entre les chemins fournis par la topologie du réseau, améliorant ainsi la latence du réseau et le débit.

Dans cette section, nous discuterons brièvement de différentes classes d'algorithmes de routage. L'algorithme de routage peut être classé en plusieurs catégories selon plusieurs propriétés : le routage source quand l'émetteur est responsable de définir le chemin de routage, le routage distribué si chaque routeur a sa propre décision du choix du routeur de destination du paquet, le routage déterministe lorsqu'il dépend uniquement de l'émetteur et du destinataire, le routage oblivious et le routage adaptatif quand on tient compte des zones de congestion et des pannes [30]. Il y a aussi le routage minimal ou non minimal qui dépend de la longueur du chemin.

2.2.1. Les routages déterministes et oblivious.

Les algorithmes de routage déterministes choisissent toujours le même chemin pour router les paquets à partir d'un certain point « A » à un certain point « B ». Les algorithmes déterministes sont utilisés dans les réseaux réguliers et irréguliers. Dans le cas le plus simple, chaque routeur dispose d'une table de routage qui comprend les routes à tous les autres routeurs

du réseau. Lorsque la structure du réseau change, chaque routeur doit être mis à jour. Ce type de routage ne tient pas compte de la diversité de chemins du réseau et n'est pas sensible à l'état du réseau. Cela peut entraîner des déséquilibres de charge dans le réseau, mais il est simple et peu coûteux à mettre en œuvre. L'avantage du routage déterministe est sa capacité à éviter le problème de blocage (deadlock).

L'algorithme de routage à plus court chemin est l'algorithme de routage déterministe le plus simple. Les paquets sont toujours acheminés le long du chemin le plus court possible. Les deux algorithmes de routage, vecteur de distance (distance vector) et à état de lien (link state), sont des algorithmes de routage à plus court chemin.

En Distance Vector Routing, chaque routeur dispose d'une table de routage qui contient des informations sur les routeurs voisins et tous les destinataires. Les routeurs échangent les informations de la table de routage entre eux, et de cette façon ils gardent les tableaux à jour. Les routeurs acheminent les paquets en comptant le plus court chemin à partir de leurs tables de routage puis envoient des paquets vers le routeur suivant. Le routage en vecteur de distance est une méthode simple car chaque routeur n'est pas tenu de connaître la structure de l'ensemble du réseau.

Le routage à état de lien est une modification de routage à vecteur de distance. L'idée de base est la même que dans le routage à vecteur de distance ; mais dans le routage à état de lien, chaque routeur partage sa table de routage avec tous les autres routeurs dans le réseau [32]. Link State Routing dans les systèmes à base de réseau sur puce est un peu personnalisé de la version traditionnelle. Les tables de routage qui couvrent l'ensemble du réseau sont déjà stockées dans la mémoire du routeur pendant la phase de production. Les routeurs utilisent leurs mécanismes de mise à jour de la table de routage chaque fois qu'il y a un changement remarquable dans la structure du réseau ou si des défauts apparaissent [33].

Dans le routage source (source routing), c'est l'émetteur qui prend toutes les décisions concernant le chemin de routage d'un paquet. Toutes les informations du trajet sont stockées dans l'entête du paquet avant de l'envoyer, et les routeurs font le routage exactement comme le fait l'émetteur tout le long du chemin.

Dans le routage oblivious, les chemins de routage sont choisis sans tenir compte de l'état du réseau. En n'utilisant pas d'informations sur l'état du réseau, ces algorithmes de routage peuvent être simples. Ils sont simples à mettre en œuvre et simples à analyser. L'algorithme de routage déterministe est un sous-ensemble du routage oblivious. L'algorithme de routage le plus simple

dans le routage oblivious est l'algorithme de routage qui fait un minimum de tours possible, comme par exemple Dimension Ordred Routing (DOR).

Parmi les algorithmes les plus connus, il y a le routage en XY. Dans l'algorithme de routage XY, le paquet est toujours acheminé en premier dans la direction des « X » jusqu'à ce qu'il atteigne le nœud qui a la même coordonnée « X » du nœud de destination. Ensuite, il est acheminé dans la direction des « Y » jusqu'à atteindre le nœud de destination. En présence des défaillances au niveau des liens et/ou des routeurs, le routage en XY ne peut pas acheminer les paquets.

2.2.2. Le routage adaptatif (dynamique).

Le routage adaptatif est l'algorithme de routage le plus sophistiqué, dans lequel le chemin qu'un message doit prendre d'un routeur « A » à un routeur « B » dépend de la situation du trafic de réseau. Par exemple un message, peut être d'abord transmis suivant la route XY et trouver une congestion et/ou une défaillance au niveau du nœud et/ou du lien ; le message doit alors choisir un nouveau chemin à prendre vers la destination.

Il y a deux types de routage adaptatif : le routage adaptatif minimal (minimal adaptif routing) et le routage entièrement adaptatif (fully adaptif routing). L'algorithme de routage adaptatif minimal route toujours les paquets suivant le chemin le plus court. L'algorithme est efficace lorsqu'il existe plusieurs chemins courts entre l'émetteur et le récepteur. L'algorithme utilise la route la moins encombrée. Un algorithme de routage entièrement adaptatif utilise toujours une route non encombrée. Cet algorithme ne prend pas en compte la longueur du chemin entre l'émetteur et le récepteur. Typiquement, un algorithme de routage adaptatif choisit toujours les chemins libres de la congestion. Le chemin le plus court est le meilleur.

Aussi, il y a deux classes d'algorithmes de routage : minimales et non minimales. Les algorithmes de routages non-minimaux sélectionnent les chemins qui peuvent augmenter le nombre de sauts entre la source et la destination. Ce type d'algorithmes peut entraîner ou éviter l'apparition de deadlock et livelock. Un routage minimal garantit que le chemin le plus court vers la destination est toujours choisi.

2.3. Les problèmes de routage.

2.3.1. Deadlock (Inter-blocage statique).

Un blocage se produit quand un ou plusieurs paquets ne peuvent pas avancer vers leurs destinations parce que le buffer demandé par le message est saturé, bloqué par un autre message

qui attend aussi (fig.20). En effet, cela se produit lorsqu'un routeur est en attente d'envoyer un paquet vers un autre routeur qui est lui aussi en attente et ainsi de suite, jusqu'au routeur initial.

La technique de commutation de paquet Wormhole est la plus encline à l'inter-blocage statique (deadlock). On peut éviter ce genre de problème par l'application d'un algorithme de routage approprié par exemple le fait d'interdire certains virages dans une topologie Mesh [28], ou encore grâce à l'application des canaux virtuels.

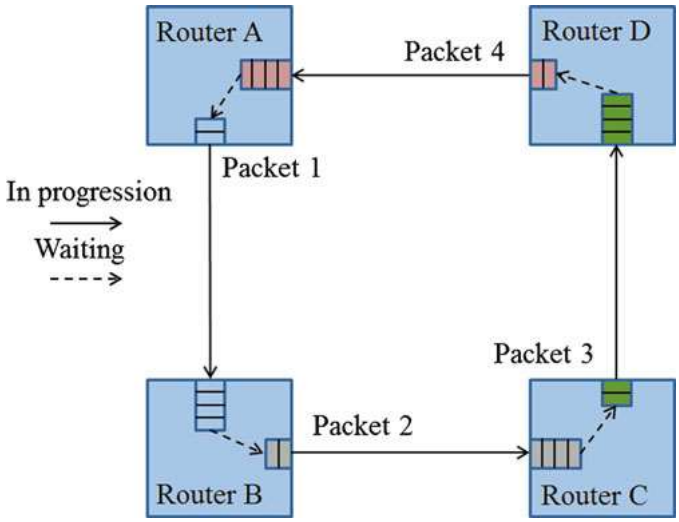


Figure 20 : Inter-blocage statique (Deadlock).

2.3.2. Livelock (Inter-blocage dynamique).

Livelock se produit quand un paquet continue de tourner autour de sa destination sans jamais l'atteindre parce que les liens nécessaires pour ce faire sont toujours réservés par d'autres messages. Ce problème existe surtout quand on utilise des algorithmes de routage non-minimal. Heureusement, ce problème peut être facilement résolu en limitant le nombre de fois qu'un message peut être mal acheminé. Une autre solution est de donner une priorité à des paquets basée sur l'ancienneté du paquet. Le paquet le plus ancien obtient toujours la plus haute priorité et sera acheminé vers sa destination.

2.3.3. Starvation.

L'utilisation des priorités différentes peut entraîner une situation où certains paquets moins prioritaires n'atteindront jamais leurs destinations. Cela se produit lorsque les paquets avec une priorité plus élevée réservent les ressources tout le temps. La starvation peut être évitée en utilisant un algorithme de routage ou juste de réserver de la bande passante uniquement pour les paquets à faible priorité.

2.4. Les stratégies de commutation.

La stratégie de commutation définit la manière dont le message va se transmettre de la source vers la destination. C'est la manière dans laquelle les ressources sont allouées afin de transmettre le message. Il y a deux grandes stratégies de commutation : commutation de circuit et commutation de paquet.

2.4.1. Commutation de circuit.

La commutation de circuit [34] réserve un chemin de bout en bout dédiée provenant de la source vers la destination avant de commencer à transmettre les données. Le chemin peut être un circuit réel ou virtuel. Un petit message de configuration est envoyé dans le réseau et se réserve les liens nécessaires pour transmettre le message en entier (ou plusieurs messages) de la source à la destination. Une fois ce message de configuration atteint la destination (avoir alloué avec succès les liens nécessaires), un message d'accusé de réception sera transmis à la source. Lorsque la source reçoit le message d'accusé de réception, il sera communiqué le message qui peut alors se déplacer rapidement à travers le réseau. Une fois que le message a terminé son parcours, les ressources sont libérées [35]. La commutation de circuit est souvent utilisée lorsque l'on vise des garanties de services.

2.4.2. Commutation de paquet.

Dans cette stratégie, les messages à échanger sont découpés en des paquets qui sont émis indépendamment dans le réseau. Lors de la transmission, ces paquets peuvent prendre différents chemins et sont réassemblés à l'arrivée. Dans une telle stratégie, des paquets risquent d'être perdus. Dans ce cas, il faut une retransmission du paquet. Elle aussi offre une meilleure latence et améliore les performances du système [24].

Dans le cas de la commutation de paquet, il faut tout d'abord choisir une politique de mémorisation, c'est-à-dire la façon dont les données vont transiter d'un nœud à l'autre. Les trois politiques principales sont : Store and Forward (SAT), Virtual Cut Through (VCT), Wormhole (WH) et canaux virtuels.

- **Store and Forward (SAF).**

Dans cette technique simple de commutation, Store and Forward [35], un paquet est envoyé d'un routeur au routeur suivant uniquement si le dernier a l'espace nécessaire afin de stocker tout le paquet. Chaque nœud attend jusqu'à ce que l'ensemble du paquet soit reçu totalement dans le buffer du port d'entrée avant de transmettre n'importe quelle partie du paquet vers le nœud

suivant. Donc, les buffers dans une telle technique doivent avoir une taille égale ou supérieure à celle du paquet. Quand le flit de la queue arrive à chaque nœud, le flit d'entête peut allouer le lien suivant.

- **Virtual Cut-Through (VCT).**

La commutation SAF est basée sur la réception totale du paquet avant de le transmettre au routeur suivant. Par contre dans la VCT, chaque routeur peut émettre le paquet avant même de le stocker intégralement dans le buffer d'entrée du routeur. Mais avant de transmettre le paquet, le routeur attend la garantie que le paquet sera accepté entièrement dans le prochain routeur. Cette technique permet de réduire le temps de latence du routeur par rapport au SAF en transmettant le premier flit d'un paquet dès que l'espace pour le paquet entier est disponible dans le routeur suivant.

Les autres flits suivent le premier flit sans aucun retard. Cependant, si aucun espace n'est disponible dans le buffer de réception, aucun flit n'est envoyé, et l'ensemble du paquet est mis en buffer du routeur transmetteur. Les exigences en mémoire du buffer sont les mêmes que pour SAF. Le VCT n'est pas non plus fréquemment utilisé dans les NoCs.

- **Wormhole (WH).**

Les deux techniques précédentes, SAF et VCT, ont besoin d'une taille de mémorisation égale à au moins la taille du paquet. Par contre, dans cette technique, les besoins de mémorisation sont réduits en un flit au lieu d'un paquet entier. Un flit d'un paquet est transmis au routeur de réception si l'espace pour ce flit est disponible dans le routeur. S'il n'y a pas suffisamment d'espace dans le prochain routeur pour stocker le paquet entier, les parties du paquet seront réparties entre deux ou plusieurs routeurs. Une telle répartition des paquets entre plusieurs routeurs peut entraîner le blocage des liens, ce qui conduit à une congestion plus élevée que dans SAF et VCT. Néanmoins, la plupart des architectures NoC utilisent la commutation WH (exemple, SPIN [36, 37]) ou une combinaison de la commutation WH et la commutation de circuit virtuel (exemple, MANGO [38], Æthereal [39]).

- **Canaux virtuels.**

Pour surmonter le problème de la contention induite par la technique Wormhole, la technique des canaux virtuels [40] a été proposée. Par conséquent, si un message réserve le canal, aucun autre message ne pourra utiliser le même canal physique si son port de sortie demandée est disponible ; il reste bloqué au niveau du routeur courant à cause de la saturation du réseau. Ce

problème est connu comme blocage de l'entête de ligne (head-of-line blocking). Lors de l'utilisation des canaux virtuels, les buffers au port d'entrée sont divisés en différents buffers virtuels et le canal est partagé par tous les buffers virtuels. Les canaux virtuels peuvent être utilisés pour améliorer la latence des messages et le débit du réseau. Leur inconvénient majeur est que la bande passante disponible du lien est répartie sur tous les canaux virtuels partageant un lien physique, entraînant des vitesses inférieures.

2.5. Contrôle de flux.

La transmission d'un flit entre les ports d'entrée et de sortie dans un routeur est une tâche exécutée par la technique de commutation. Le contrôle de flux, cependant, a en charge l'administration de l'avancement de l'information entre les routeurs. Les techniques de contrôle de flux ont en charge de déterminer le moment où les flits peuvent être transmis pour évaluer la capacité des buffers et la bande passante du lien, ainsi que d'allouer les ressources du réseau pour les paquets traversant un NoC.

Il y a plusieurs techniques de contrôle de flux. Parmi ces techniques nous présentons les plus utilisées (ACK/NACK, Stop & Go et Credit-Based).

2.5.1. ACK/NACK.

Dans la technique ACK / NACK, lorsque les flits sont envoyés sur un lien, une copie est conservée dans un buffer par l'émetteur. Quand un flit arrive à un buffer du routeur qui suit le routeur émetteur, si le buffer de ce dernier a l'espace disponible, le flit est accepté et un signal d'accusé de réception (ACK) est renvoyé à l'émetteur. Au lieu de cela, s'il n'y a pas d'espace disponible, le flit est supprimé et un acquittement négatif (NACK) est envoyé. Le flit doit être conservé à son origine jusqu'à ce qu'il reçoive un accusé de réception positif. Quand un ACK est reçu par l'émetteur, il supprime sa copie du flit de son buffer. Quand un NACK est reçu, l'émetteur rembobine sa file d'attente de sortie et commence à renvoyer des flits à partir de celle qui est corrompue. ACK / NACK peut être mis en œuvre soit de bout en bout entre l'émetteur et le récepteur, soit de routeur en routeur.

2.5.2. Stop & Go.

Dans cette technique, chaque buffer a deux seuils : Stop et Go. Lorsque l'espace occupé dans le buffer récepteur atteint le seuil d'arrêt (Stop), un signal est envoyé à l'émetteur afin d'arrêter la transmission, en prenant en compte ce qui reste encore de la taille de la mémoire du buffer si elle est suffisante pour les flits qui sont toujours transmis par l'émetteur. Lorsque

l'occupation du buffer diminue jusqu'à devenir inférieure ou égale au deuxième seuil, go, un autre signal est envoyé pour réactiver le flux de flits.

2.5.3. A base de crédit.

Le contrôle de flux à base de crédit est l'une des techniques les plus populaires. Dans ce mécanisme, les paquets sont autorisés à quitter l'émetteur seulement s'il ya un espace pour eux dans le buffer récepteur. Un compteur de crédit est installé au niveau de chaque routeur qui suit l'espace libre disponible dans le buffer du récepteur. Le compteur est initialisé à la capacité de la mémoire buffer du récepteur et il décrémente chaque fois qu'un paquet est envoyé. Lorsque le récepteur supprime un paquet de ses buffers d'interface, il envoie un crédit à l'expéditeur, ce qui ajoute alors à son compteur de crédit.

3. Quelques NoCs académiques et industriels.

Dans cette section, nous présentons quelques NoCs académiques et industriels trouvés dans la littérature (Tab.4). Un certain nombre de sociétés existe déjà qui ont commencé à mettre la technologie NoC à des produits commerciaux comme Arteris [41], Silistix [42] et iNoCs [43].

- **Ætheral**

Le réseau Ætheral est développé par la société Philips [44] dont le but principal est d'offrir une garantie de qualité de service. Le réseau Ætheral définit deux classes de qualité de service : la classe Guaranteed Service et la classe Best Effort. Les mécanismes de garantie de services sont implémentés au niveau des routeurs et se basent sur une technique d'allocation statique des ressources (TDMA = Time Division Multiple Access) du réseau. L'allocation se fait par des tables, appelées « Slot Tables », configurées lors de la conception du réseau, où chaque ligne correspond à un intervalle temporel (slot), un slot étant une période de quelques cycles allouée pour une communication entre deux composants, et chaque colonne correspond à un port de sortie du routeur.

Ce NoC a une instance d'un routeur à 6 ports et une interface réseau avec 4 ports IP. Les files d'attente sont placées à l'entrée et sont toutes de types FIFO (First Input First Output) ; elles ont une largeur de 32 bits et une profondeur de 8 mots.

La topologie adoptée dans ce type de réseau est une topologie indirecte et le routage utilisé est de type source sans contention basé sur la technique TDMA. La stratégie de mémorisation utilisée est la commutation de paquet Wormhole.

- **Nostrum.**

Nostrum est un projet de recherche visant à développer une architecture de réseau sur puce [45]. Il est développé par le Département Electronics, Computer and Software System (ECS) à KTH, Stockholm. Le réseau dispose d'une topologie Mesh 2D. Pour pouvoir communiquer avec les autres IPs, chaque IP possède un commutateur lui permettant de se connecter au réseau.

Le réseau Nostrum fait l'usage d'un algorithme de routage de déviation (hot-potato) avec une commutation de paquet SAF et offre un soutien pour la répartition des charges de commutation, une garantie de la bande passante (GB) et une multidiffusion. GB est réalisé en utilisant ce qu'on appelle des conteneurs en boucle. Ces conteneurs sont mis en œuvre en boucle par des circuits virtuels, à l'aide d'un mécanisme TDM.

- **STNoC.**

Le STNoC [46] a été introduit par la société STMicroelectronics. Le STNoC est un réseau sur puce flexible à commutation de paquet. Ce réseau utilise une topologie Spidergon (fig.21) supportant des algorithmes de routage minimaliste nécessitant un minimum de ressources matérielles. Il utilise des liens de 32 bits et une stratégie de mémorisation où les buffers sont placés en entrée. L'interface réseau du STNoC accepte n'importe quel genre du protocole IP comme l'AXI, l'OCP ou le STBus à convertir en paquets de communication.

- **SPIN.**

Le réseau SPIN (Scalable Programmable Integrated Network on chip) [36], développé par le laboratoire LIP6 (Laboratoire d'informatique de Paris 6), implémente une topologie en arbre élargi. Il utilise la stratégie de mémorisation Wormhole, le routage adaptatif et le contrôle de flux à base de crédit.

Le bloc de construction de base du réseau SPIN est le routeur de RSPIN, qui comprend huit ports ayant une paire de canaux d'entrée et de sortie compliant avec le lien SPIN.

En 2003, un réseau SPIN à 32 ports a été mis en œuvre dans un CMOS de 0,13 µm, la superficie totale est de 4,6 mm^2 (0,144 mm^2 par port), pour une bande passante cumulée d'environ 100 Gbits/s.

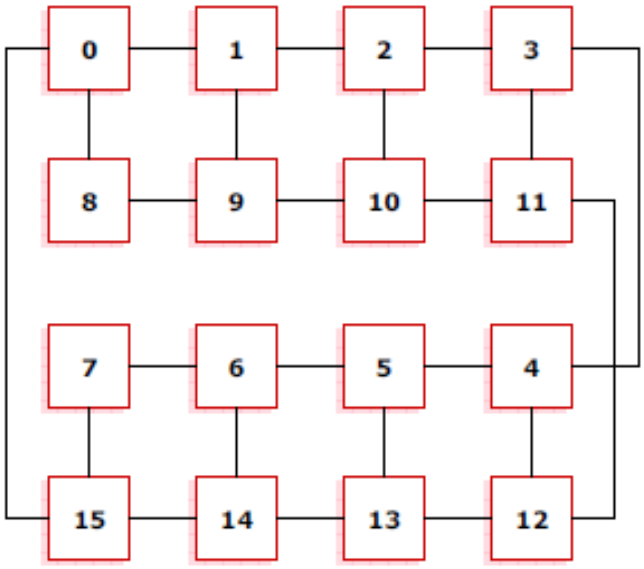


Figure 21 : Topologie du réseau STNoC.

- **HERMES.**

Le réseau HERMES [47, 48] est développé à Faculdade de Informatica PUCRS, Brésil. C'est un réseau direct avec une topologie maillée 2D dont la taille peut être fixée en fonction des préférences de l'utilisateur. Il utilise la commutation Wormhole comme une stratégie de mémorisation et le XY minimal comme algorithme de routage. Dans ce réseau, chaque routeur a 5 ports bidirectionnels (4 reliés aux routeurs voisins, et un à l'IP) ; les buffers sont placés en entrée. La taille du flit est de 8 bits, les deux premiers flits contiennent l'entête. Cette dernière contient l'adresse de la destination et le nombre de flits dans le paquet. Une dernière caractéristique importante du réseau Hermes est qu'il est compatible avec une implémentation matérielle (ex : FPGA); en effet le code VHDL généré est complètement synthétisable.

Tableau 4 : Comparaison entre quelques NoCs.

	Commutation	Topologie	Routage	Liens	Support de la QoS
Ætheral	Circuit et Paquet	2D variable	Source	32 bits	Oui
STNoC	Paquet	Spidergon	Déterministe	32 bits	Oui
SPIN	Wormhole	Arbre élargi	Distribuée et adaptatif	32 bits donnée + 4 bits contrôle	-
HERMES	Wormhole	Mesh 2D	XY partiellement adaptatif	8 bits donnée + 2 bits contrôle	-
Nostrum	Paquet	Mesh 2D	Hot-potato	128 bits donnée	Oui

4. Conclusion.

Dans ce chapitre, nous avons exposé dans un premier temps le concept des réseaux sur puce ainsi que les notions et la terminologie associées. Nous avons ensuite détaillé les mécanismes internes et les éléments constitutifs de l'architecture d'un NoC. Par la suite, nous avons détaillé les protocoles de communication nécessaires à la transmission de l'information au sein des NoCs. Par ailleurs, nous avons vu que, pour gérer les flux de communications, différents mécanismes peuvent être mis en place. C’est ainsi que, grâce à leur flexibilité et leur extensibilité, les réseaux sur puce permettent de découpler les communications et les traitements au sein d'un SoC. Il faut bien noter toutefois, que la conception d'un système doit être faite en fonction du trafic généré par l'application ciblée. Afin d'obtenir des performances optimales (en termes de surface, débit, latence), l'analyse des communications permet en effet de définir l'architecture du NoC la mieux adaptée à l'application. Enfin, nous avons présenté quelques implémentations de NoCs qui nous semblaient les plus significatives.

CHAPITRE 3

Modélisation des Systèmes sur Puce en utilisant l'UML.

Les Systèmes sur Puce sont devenus de plus en plus nécessaires dans notre vie, soit professionnelle ou encore personnelle. Les flots de développements et les outils de conception de SoCs, utilisés aujourd'hui, n'ont pas été aussi prompts à suivre les grandes évolutions matérielles ainsi que logicielles. La grande densité d'intégration des modules, le temps de fabrication et la complexité de la conception des futurs SoCs impliquent une très grande difficulté de conception des systèmes en bas niveau (RTL) où il nous faut décrire le moindre détail du comportement des composants.

En raison de la croissance exponentielle continue de la complexité de la conception des SoCs, il est absolument nécessaire de trouver de nouvelles méthodologies et d'outils pour gérer les aspects de co-conception des SoCs. Pour réduire les temps et les coûts de développement, deux aspects orthogonaux se révèlent indispensables : la montée dans les niveaux d'abstraction pour la spécification du système et la réutilisation des blocs IPs prédéfinis.

Afin d'accélérer la conception de SoCs, des approches de spécification orientées modèles sont alors apparues. La modélisation permet d'abstraire les aspects les plus importants pour les communiquer, les analyser et les valider avant toute implémentation. Parmi ces approches, on trouve l'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) en anglais.

Dans ce chapitre, nous présenterons les bases de l'ingénierie dirigée par des modèles et comment elle peut faciliter le développement des SoCs, le langage de modélisation UML et les différents profils utilisés pour la modélisation des SoCs, la modélisation des SoC en utilisant le profile UML/MARTE, l'environnement de conception des systèmes embarqués GASPARD2, les transformations des modèles et la génération automatique du code.

1. L'Ingénierie Dirigée par les Modèles (IDM).

On sait tous à quel point il est difficile de lire, comprendre et modifier un code afin de le réutiliser quel que soit le langage de programmation choisi et malgré les lignes de commentaire qui nous facilitent la tâche de compréhension du fonctionnement du code.

Des approches de spécification orientées modèles sont apparues afin d'accélérer la conception des SoCs. Avant toute implémentation, la modélisation permet d'abstraire les aspects les plus importants pour les communiquer, les analyser et les valider. Parmi ces approches, on trouve l'Ingénierie Dirigée par les Modèles (IDM).

L'ingénierie dirigée par les modèles est une méthodologie de développement de logiciels qui se concentre sur les modèles, les méta-modèles, et les transformations de modèles. Elle a

permis plusieurs améliorations significatives dans le développement des systèmes en permettant de se concentrer sur une préoccupation plus abstraite que la programmation classique, suivant le principe de : "tout est un modèle" [49]. Un modèle est la représentation simplifiée d'un système qui met en évidence les propriétés d'intérêt pour un but ou un point de vue donné.

L'approche IDM est devenue prometteuse, non seulement pour le génie logiciel, mais pour le matériel ainsi que l'ingénierie de système ; elle a attiré beaucoup d'attention dans l'industrie et le milieu universitaire. Avant tout, l'IDM joue un rôle très important qui contribue à la modélisation, la génération automatique de codes et une passerelle entre différentes technologies.

De façon générale, l'IDM peut être définie autour de trois concepts de base : les modèles, les méta-modèles et les transformations des modèles.

1.1. Modèles.

Les modèles sont des représentations, à différents niveaux d'abstraction et selon plusieurs vues, d'une réalité ou d'un système. Un modèle contient donc plusieurs informations sur un système. Ces informations ont toutes vocations à faciliter d'une manière ou d'une autre la production du code de système. Il est important de noter que le modèle décrit ce qu'un système est censé faire. Il ne dit pas comment implémenter le système. Un modèle dispose de deux éléments clés : les concepts et les relations. Les concepts représentent des choses et les relations sont les liens entre ces choses dans la réalité.

Un modèle peut, entre autres choses, préciser les différentes données manipulées par l'application (vue des données), préciser les différentes fonctionnalités directement offertes aux utilisateurs de l'application (vue des utilisateurs) et préciser les technologies, telles que Java, utilisées pour réaliser l'application (vue technique).

En d'autres termes, un modèle est composé de plusieurs vues sur une application. Chacune de ces vues contient certaines informations sur l'application. Ces vues peuvent cibler différents niveaux d'abstraction, et elles doivent être cohérentes [50]. La figure 22, schématise un modèle d'application composé de trois vues.

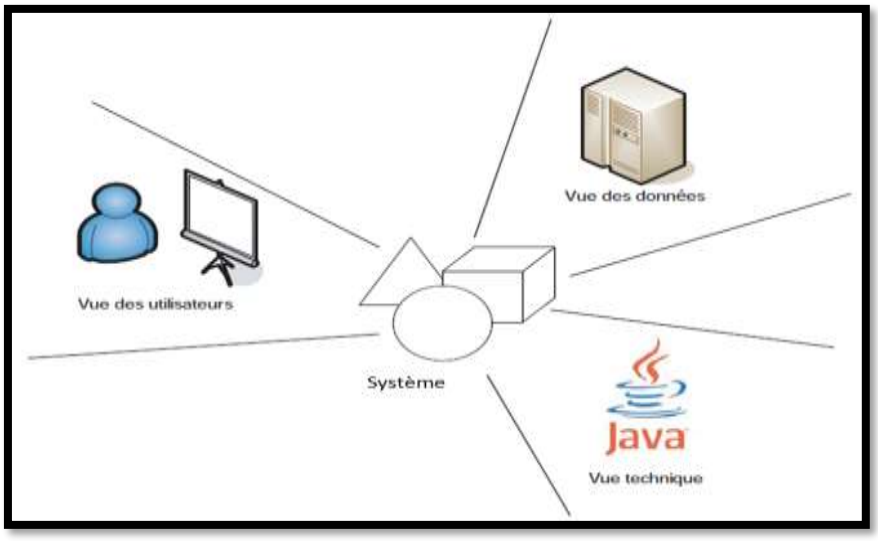


Figure 22 : Modèle d'un système.

1.2. Méta-modèles.

Méta-modèle signifie littéralement modèle du modèle. Il peut être défini comme la représentation d'un point de vue particulier sur des modèles. Un méta-modèle qui se décrit lui-même est dit réflexif. Cette propriété permet d'avoir une représentation complète des modèles sans régression infinie : modèle, méta-modèle, méta-méta-modèle…

Afin d'être interprétable par une machine, l'expression avec laquelle un modèle est représenté est formellement prédéfinie. Ceci est réalisé par un méta-modèle. Dans l'IDM, Un méta-modèle est une collection de concepts et de relations pour décrire un modèle en utilisant un langage de description de modèle ; il est utilisé pour la définition de la syntaxe d'un modèle. Un méta-modèle peut être considéré comme une représentation interne dans un flux de synthèse de haut niveau.

La figure 23 représente la relation entre les modèles et les méta-modèles. Un des méta-modèles les plus connus est le méta-modèle UML. Le niveau « **M0 »** est la représentation d'une réalité (un programme informatique). Dans cet exemple, plusieurs variables (nombre et balance) prennent des valeurs qui leur sont affectées. Le niveau **« M1 »** est le niveau d'abstraction le plus bas (premier niveau d'abstraction), où les concepts peuvent être manipulés par les développeurs. Pour cet exemple, le modèle contient la déclaration des variables utilisées dans « **M0 »** et la notion de Compte. Un modèle de niveau « **M1 »** est conforme à un méta-modèle de niveau « **M2 »**. Les concepts manipulés par les développeurs à « **M1 »** sont définis et situés à ce niveau. Dans cette exemple, *Compte* est une classe, alors que les déclarations de variables sont des attributs clos dans la classe. Enfin, un méta-modèle au niveau **« M2 »** est conforme à une méta-

méta-modèle (au niveau de **M3**). Celui-ci se conforme à lui-même. Dans l'exemple, les concepts tels que les classes et les attributs sont des méta-classes, tandis que la relation est une métarelation. Le méta-méta-modèle peut se décrire lui-même, par exemple ; la méta-classe et la métarelation restent des méta-classes ; et les relations telles que la source et la destination sont des métarelations.

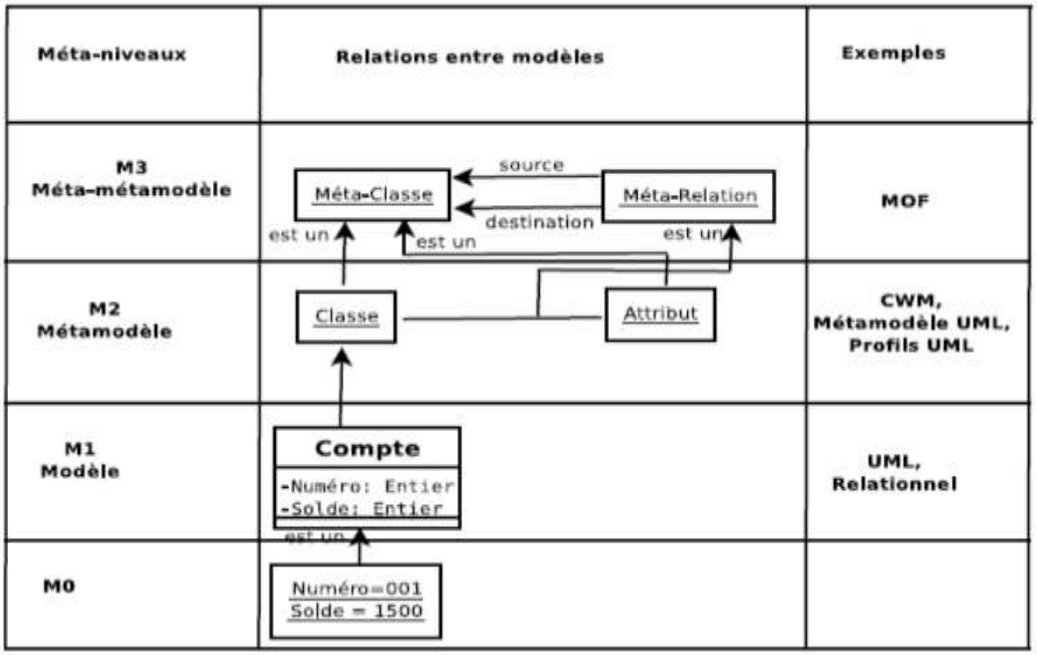


Figure 23 : Différents niveaux de modélisation en IDM.

1.3. Transformations des modèles.

La transformation des modèles présente un autre point clé de l'IDM. Elle permet de passer d'un modèle source décrit à un certain niveau d'abstraction à un modèle destination décrit éventuellement à un autre niveau d'abstraction. Ces modèles source et destination sont conformes à leur méta-modèle respectif (méta-modèle source et destination) et le passage de l'un à l'autre est décrit par des règles de transformation. Ces règles sont exécutées sur les modèles source afin de générer les modèles destination, comme illustré dans la figure 24.

De nombreux outils de transformation de modèles existent sur le marché, dans différents contextes technologiques. Parmi ces outils nous noterons les outils ATL (Atlas Transformation Language) [51].

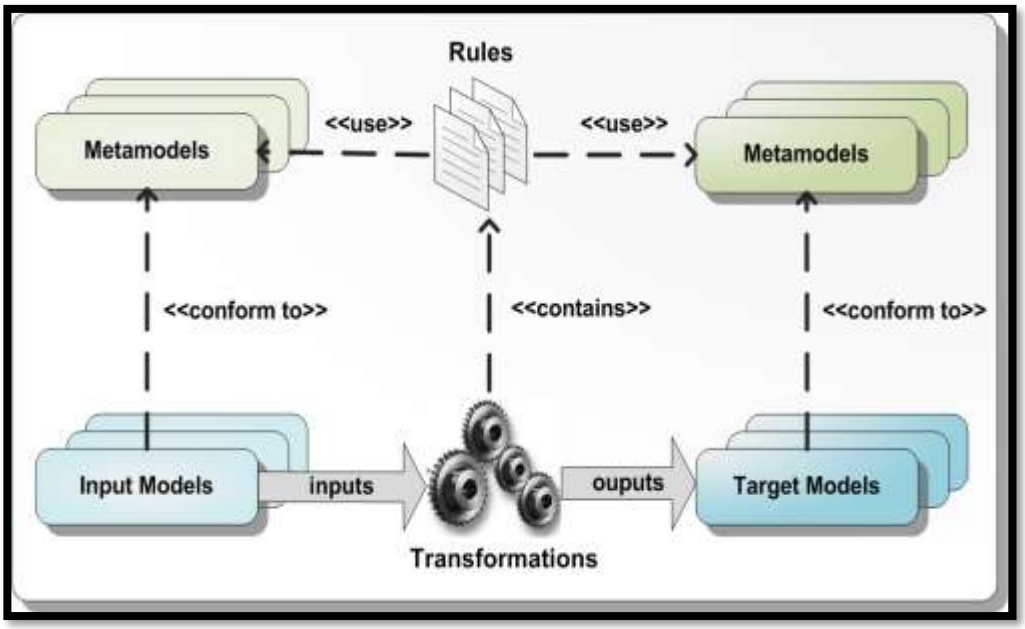


Figure 24 : Principe de transformation de modèle.

2. Le langage UML.

Pour répondre aux besoins de documentation et de spécification de logiciels, de nombreux langages graphiques ont vu le jour, en particulier au sein de la communauté des objets. Afin de privilégier la réutilisation de composants, d'architectures, ou de favoriser l'interconnexion entre des systèmes modélisés avec des notations différentes, il apparaît évident qu'une notation consensuelle est nécessaire. De ce constat est née la notation UML [52].

Le langage de modélisation unifié (Unified Modeling Language) est un des outils les plus excitants et utiles dans le monde du développement du système. C'est parce que l'UML est un langage de modélisation visuelle qui permet aux développeurs de créer des modèles qui capturent leurs visions dans une norme, facile à comprendre et fournit un mécanisme pour partager et communiquer ces visions avec les autres d'une manière efficace.

Le langage UML est considéré comme le formalisme le plus utilisé dans l'approche IDM. Il est un langage de modélisation visuel utilisé pour documenter, spécifier et visualiser graphiquement les aspects d'un système logiciel. Il est standardisé et maintenu par l'OMG (Object Management Group) depuis 1997 [53]. L'OMG est un groupement d'industriels dont l'objectif est de standardiser autour des technologies objet, afin de garantir l'interopérabilité des développements. Comme son nom l'indique, ce langage est né de la fusion de plusieurs langages de modélisation qui ont été utilisés depuis les années 1990. Ces langages sont OMT [année 91], Booch [année 91] et OOSE [année 92] (fig.25). Ils sont développés dans différentes organisations

au travers des années 1980 jusqu'au début des années 1990 par Grady Booch, James Rumbaugh and Ivan Jacobson. L'UML a rassemblé les meilleures caractéristiques des différents langages de modélisation à objet en une seule notation. Comme l'UML est très répandu dans l'industrie ainsi que dans le milieu universitaire, à des fins de modélisation, un grand nombre d'outils ont été développés pour le soutenir.

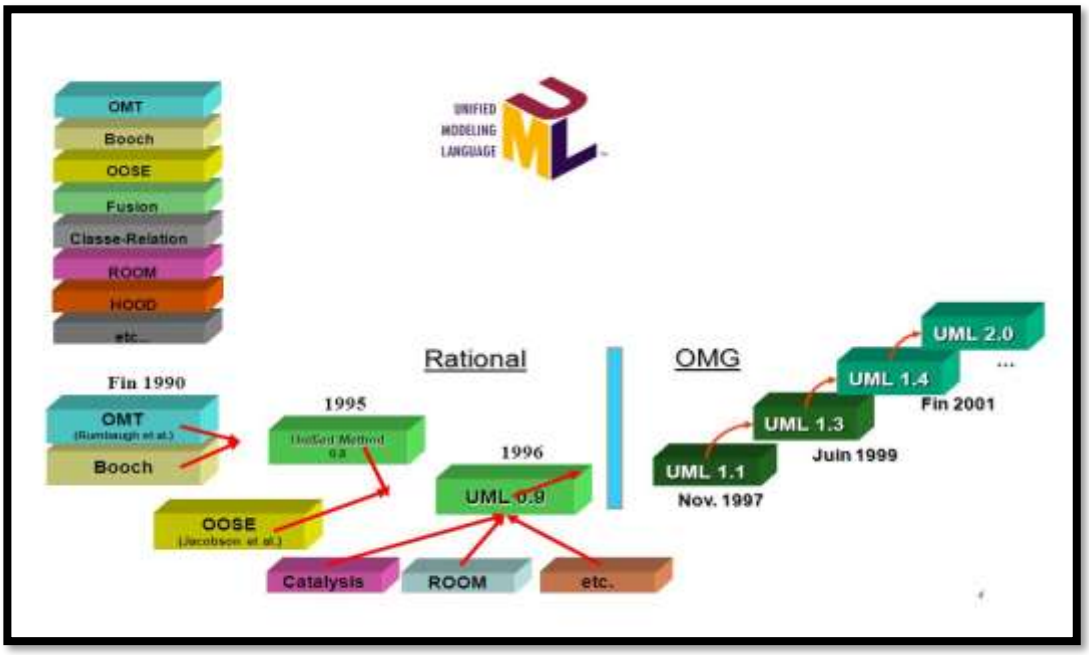


Figure 25 : Historique de l'UML.

2.1. Pourquoi modéliser?

Le recours à la modélisation est depuis longtemps une pratique indispensable au développement logiciel, car un modèle est prévu pour arriver à anticiper les résultats du codage. Un modèle est en effet une représentation abstraite d'un système destiné à en faciliter l'étude et à le documenter. C'est un outil majeur de communication entre les différents intervenants au sein d'un projet. Chaque membre de l'équipe, depuis l'utilisateur jusqu'au développeur, utilise et enrichit le modèle différemment. En outre, les systèmes devenant de plus en plus complexes, leur compréhension et leur maîtrise globale dépassent les capacités d'un seul individu. La construction d'un modèle abstrait aide à y remédier. Le modèle présente notamment l'atout de faciliter la traçabilité du système, à savoir la possibilité de partir d'un de ses éléments et de suivre ses interactions et liens avec d'autres parties du modèle.

Associé au processus de développement, un modèle représente l'ensemble des vues sur une expression de besoins ou sur une solution technique. Pris à un niveau de détail pertinent, il décrit ou conçoit la cible de l'étape en cours. Le modèle sert donc des objectifs différents suivant l'activité de développement et sera construit avec des points de vue de plus en plus détaillés :

- Dans les activités de spécification des exigences, il convient premièrement de considérer le système comme une boîte noire à part entière afin d'étudier sa place dans le système métier plus global qu'est l'entreprise. On développe pour cela un modèle de niveau contexte, afin de tracer précisément les frontières fonctionnelles du système.
- Dans les activités d'analyse, le modèle commence à représenter le système vu de l'intérieur. Il se compose d'objets représentant une abstraction des concepts manipulés par les utilisateurs. Le modèle comprend par ailleurs deux points de vue, la structure statique et le comportement dynamique. Il s'agit de deux perspectives différentes qui aident à compléter la compréhension du système à développer.
- Dans les activités de conception, le modèle correspond aux concepts informatiques qui sont utilisés par les outils, les langages ou les plates-formes de développement. Le modèle sert ici à étudier, documenter, communiquer et anticiper une solution. Il est en effet toujours plus rentable de découvrir une erreur de conception sur un modèle, que de la découvrir au bout de milliers de lignes codées sans méthode. Pour la conception du déploiement enfin, le modèle représente également les matériels et les logiciels à interconnecter.

Le modèle en tant qu'abstraction d'un système s'accorde parfaitement bien avec les concepts orientés objet. Un objet peut en effet représenter l'abstraction d'une entité métier utilisée en analyse, puis d'un composant de solution logicielle en conception. La correspondance est encore plus flagrante lorsque les langages de développement sont eux-mêmes orientés objet. Cela explique le succès de la modélisation objet ces dernières années pour les projets de plus en plus nombreux utilisant C++, Java ou C#.

2.2. Les diagrammes UML.

Un modèle UML se compose des éléments tels que les packages, les classes et les associations. Les diagrammes UML correspondant à ces éléments sont des représentations graphiques des parties du modèle UML. Plus exactement, les diagrammes UML contiennent des éléments graphiques (nœuds reliés par des relations -arcs-) qui représentent les éléments dans le modèle UML.

Chaque diagramme permet de donner une vue du système. Toutefois, il est impossible de donner une seule représentation graphique complète d'un logiciel, ou de tout autre système complexe. De même qu'il est impossible de représenter entièrement une sculpture à trois dimensions par des photos à deux dimensions. Mais il est possible de donner sur un système des vues partielles, semblables chacune à une photo d'une sculpture, et dont l'union donnera une idée utilisable en pratique.

Le langage de modélisation UML propose dans la standardisation « 1.3 » un ensemble de 9 diagrammes ; dans la « 2.0 » il y a 13 types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis par l'OMG en deux grands groupes : six diagrammes structurels (Structur Diagram) et sept diagrammes comportementaux (Behavior Diagram). Le paragraphe qui suit décrit brièvement les diagrammes les plus courants de l'UML et les concepts qu'ils présentent (Fig.26).

2.2.1. Les diagrammes structurels.

- **Diagramme de classes :** Une classe est une catégorie ou un groupe de choses qui ont les mêmes attributs et les mêmes comportements. Dans le développement orienté objet, le diagramme de classe est considéré comme le point centrale. Ce diagramme décrit les classes d'une application et leurs relations statiques. Il définit la structure (relation, attributs, classification) des entités manipulées par l'utilisateur. Ce diagramme est utilisé pour décrire l'architecture d'une application.
- **Diagramme d'objets :** Le diagramme d'objet permet la représentation d'instances des classes et des liens entre instances à l'exécution. C'est probablement le diagramme le moins utilisé de l'UML.
- **Diagramme de packages :** Le diagramme de package donne une vue d'ensemble du système structuré en paquetage. Chaque paquetage représente un ensemble homogène d'éléments du système (classes, composants…).
- **Diagramme de composants :** Le diagramme de composant représente les différents constituants du logiciel au niveau de l'implémentation d'un système.
- **Diagramme de structure composite :** Un diagramme de structure composite est un graphique représentant la structure interne d'une ou plusieurs classes. Une classe sur un diagramme de composite contient un ensemble de parties reliées par des connecteurs. Une partie possède un type et une multiplicité. Les parties peuvent être typées par des composants. Un composant possède un ensemble d'interfaces et une ou plusieurs implémentations.

- **Diagramme de déploiement :** Ce diagramme décrit l'architecture technique d'un système avec une vue centrée sur la répartition des composants dans la configuration d'exploitation.

2.2.2. Les diagrammes comportementaux.

- **Diagramme des cas d'utilisation :** Ce diagramme est destiné à représenter les besoins des utilisateurs par rapport au système. Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude. Il constitue un des diagrammes les plus structurants dans l'analyse d'un système.
- **Diagramme de vue d'ensemble des interactions :** Ce diagramme fournit une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activités.
- **Diagramme de séquence :** Ce diagramme permet de décrire les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.
- **Diagramme de communication :** Ce diagramme est une autre représentation des scénarios des cas d'utilisation qui met plus l'accent sur les objets et les messages échangés.
- **Diagramme de temps :** Ce diagramme permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.
- **Diagramme d'activité :** Un diagramme d'activité est un graphique de nœuds et de flots qui matérialise le flot de contrôle ou de données, à travers les étapes d'un calcul. L'exécution de ces étapes peut être à la fois simultanée et séquentielle. Une activité présente à la fois des constructions de synchronisation et de branchement. Dans la définition d'une activité, on trouve des nœuds d'activité. Ils représentent l'exécution d'une instruction dans une procédure ou d'une étape dans un enchaînement d'activités et sont connectés par des flots de contrôle et de données. L'exécution d'un nœud d'activité commence à l'endroit où l'on trouve des jetons (indicateurs de contrôle) sur chacun de ses flots d'entrée. Lorsque l'exécution du nœud s'achève, elle se poursuit vers les nœuds qui se trouvent sur ses flots de sortie.
- **Diagramme d'état :** Un diagramme de machine à état est un graphique représentant des états et des transitions. Il est habituellement relié à une classe et il décrit la réponse d'une instance de la classe aux événements qu'elle reçoit. On peut également rattacher les machines à états à des comportements, des cas d'utilisation et à des collaborations pour décrire leur

exécution. Une machine à états est un modèle de tous les états possibles que peut prendre un objet de classe. Elle résume toute influence en provenance du reste du monde comme un événement. Lorsqu'un objet détecte un événement, il y répond en fonction de son état actuel. Cette réponse peut comprendre l'exécution d'un effet et d'un changement vers un autre état.

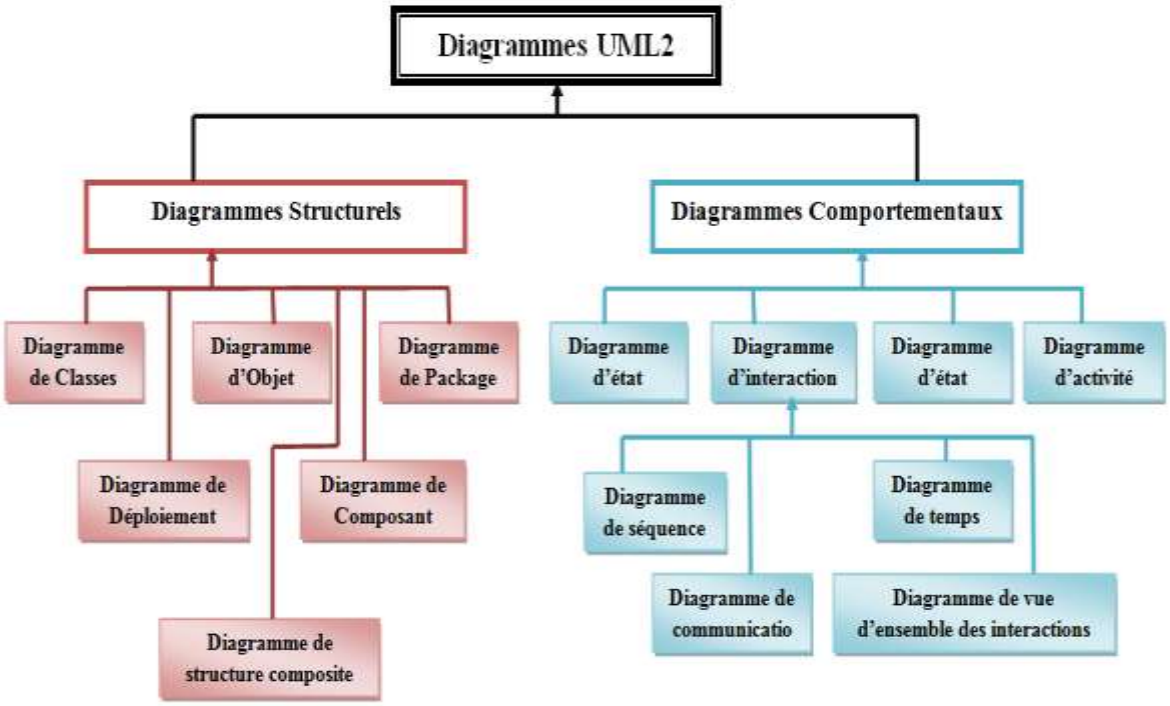


Figure 26 : Diagrammes UML.

2.3. Les profile UML pour la modélisation des SoCs.

Un profil UML permet d'adapter UML à un domaine particulier ou à une problématique spécifique. Les profils mettent en jeu le concept central de stéréotype. Un stéréotype est une sorte d'étiquette nommée que l'on peut coller sur tout élément d'un modèle UML. Par exemple, coller un stéréotype « continuous » sur un flot dans un diagramme d'activité signifie que le flot en question n'est plus un flot UML standard, c'est-à-dire discret, mais qu'il est continu. SysML ajoute ainsi un certain nombre de concepts à UML 2 en définissant un ensemble de stéréotypes. Mais il simplifie également le langage de modélisation en laissant de côté certains concepts UML jugés non indispensables.

Dans la littérature, plusieurs travaux ont été réalisés ces dernières années pour étudier la possibilité d'utiliser UML comme langage de base pour la spécification des systèmes embarqués [54, 55, 56].

2.3.1. SysML (System Modeling Language).

SysML est un langage de modélisation graphique à usage général qui supporte l'analyse, la spécification, la conception, la vérification et la validation d'un large éventail de systèmes hétérogènes complexes qui ne sont pas forcément basé sur le logiciel [57]. Il réutilise un sous-ensemble de l'UML 2 et fournit des extensions supplémentaires nécessaires à l'ingénierie des systèmes. SysML permet aux ingénieurs de modéliser les exigences du système, son comportement ainsi que sa structure.

SysML s'articule autour de neuf types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système. Ces types de diagrammes sont répartis par l'OMG en trois grands groupes (fig.27) :

- Quatre diagrammes comportementaux :

1. Le diagramme d'activité qui montre l'enchainement des actions et décisions au sein d'une activité complexe ;
2. le diagramme de séquence qui montre la séquence verticale des messages passés entre blocs au sein d'une interaction ;
3. le diagramme d'états qui montre les différents états et transitions possibles des blocs dynamiques ;
4. le diagramme de cas d'utilisation qui montre les interactions fonctionnelles entre les acteurs et le système à l'étude.

- Un diagramme transverse : le diagramme d'exigences montre les exigences du système et leurs relations.
- Quatre diagrammes structurels :

1. Le diagramme de définition de blocs qui montre les briques de base statiques : blocs, compositions, associations, attributs, opérations, généralisations, etc ;
2. le diagramme de bloc interne qui montre l'organisation interne d'un élément statique complexe;
3. le diagramme paramétrique qui représente les contraintes du système, les équations qui le régissent ;
4. le diagramme de packages qui montre l'organisation logique du modèle et les relations entre packages.

2.3.2. UML-SystemC.

Ce profile est développé par l'Université de Catania et STMicroelectronics [58, 59]. Il prend les avantages de l'UML 2 et le langage SystemC suivant les principes du MDA. Ce profil UML définit un langage qui permet de spécifier, analyser, concevoir, construire, visualiser et documenter les artefacts logiciels et matériels dans un flot de conception de SoC.

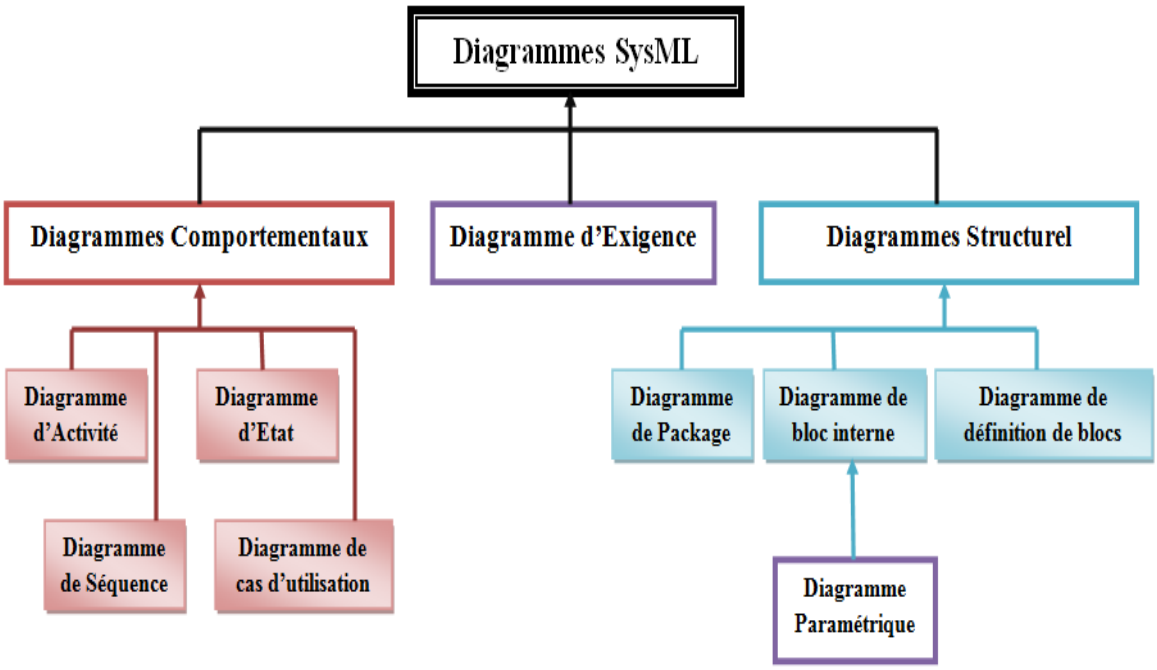


Figure 27 : Diagrammes du profile SysML.

2.3.3. UML-SoC.

Il est développé par Fujitsu Limited et Fujitsu Laboratories [60]. Ce profil vise à décrire l'information spécifique du système sur puce utilisant l'UML. Ce profil envisage de décrire les informations spécifiques d'un système sur puce en utilisant l'UML. Il intègre des concepts de SoC et permet la génération automatique du code pour le matériel (par exemple : VHDL, SystemC), couvrant les niveaux d'abstraction du niveau de la modélisation transactionnel (Transactional Level Modeling = TLM) au Register Transfer Level (RTL). UML-SoC est axé sur le schéma de structure de l'UML 2. Il propose les stéréotypes qui permettent la modélisation structurelle, la modélisation de la communication et la modélisation de l'opération et de la propriété.

Le tableau 5 donne la correspondance entre certains stéréotypes SoC et des constructions UML [60].

Tableau 5 : Un sous ensemble des stéréotypes du profil UML-SoC.

Les éléments du modèle SoC	Stéréotype	Les méta-classe UML
Module	SoCModule	Class
Process	SoCProcess	Operation
Data	Data	Class
Controller	Controller	Class
Protocol Interface	SoCInterface	Interface
Channel	SoCChannel	Class
Protocol	SoCProtocol	Collaboration
Port	SoCPort	Port/Class
Part	SoCModuleProperty	Property
Channel Part	SoCChannelProperty	Property

2.3.4. UML-SPT (Schedulability, Performance, and Time).

L'UML-SPT [61] est défini comme support à UML pour modéliser certains domaines temps réel. Il introduit une notion quantifiable du temps et de ressources. Il décrit ainsi les éléments d'un système avec des concepts quantitatifs de temps, de performance et d'ordonnançabilité permettant l'analyse de systèmes temps réel. Cependant, le SPT ne considère qu'une échelle de temps chronométrique ne faisant référence qu'au temps physique.

2.3.5. UML-MARTE (Modeling and Analysis of Real-time and Embedded systems).

Le profile UML-MARTE est considéré comme un remplacement et un raffinement du profile SPT [62]. Il est dédié à la modélisation des systèmes embarqués temps réel. Il offre la possibilité de modéliser des applications, des architectures et les relations entre elles. Il offre aussi des extensions pour faire de l'analyse de performances et d'ordonnancements en introduisant une notion quantifiable de temps physique et logique. Nous détaillerons ce profile dans le paragraphe qui suit.

3. Le profile MARTE.

Vu la complexité des SoCs d'aujourd'hui, une abstraction permet la simplification, la compréhension et la décomposition du système en sous-systèmes faciles à gérer. Dans le domaine de la modélisation, tous les systèmes sont considérés comme des modèles ; il décrit certaines propriétés du système indépendamment de la technologie et des détails d'implémentation. Donc la modélisation permet de masquer les détails qui ne sont pas pertinents et qui peuvent ralentir le

processus de conception. En se basant sur cette approche d'abstraction, les concepteurs peuvent se concentrer sur un domaine particulier à l'aide d'une modélisation visuelle qui améliore la communication entre le système et son concepteur.

Le profile MARTE (Modeling and Analysis of Real-time and Embedded systems), qui a été normalisé par l'OMG en Juillet 2007, est un profile UML dédié à la modélisation des systèmes embarqués à temps réel. Ce profil vise à étendre l'UML pour être utilisé dans un développement dirigé par les modèles des applications temps réel et des systèmes embarqués, y compris leurs aspects logiciels et matériels. Il offre la possibilité de modéliser les applications, les architectures et les relations entre elles. Ce nouveau profil est venu pour remplacer le profil UML existant Profile for Schedulability Performance and Time (SPT), qui est basé sur une version antérieure de UML (UML 1.x). Le profil MARTE est structuré autour de deux préoccupations, l'une pour modéliser les contraintes du temps réel et des systèmes embarqués et l'autre pour annoter des modèles d'application afin de supporter l'analyse des propriétés de système.

La figure 28 montre l'architecture globale du profile MARTE selon une décomposition en paquetages : le package fondations (Foundations Package) qui définit les concepts de base dans le domaine des systèmes embarqués, le modèle de conception MARTE (MARTE Design Model) qui fournit un langage spécifique au domaine de la modélisation de phénomènes spécifiques à RTES, le modèle d'analyse MARTE (MARTE Analysis) qui permet l'analyse des performances des systèmes modélisés ; en outre, MARTE définit de nombreuses constructions de modélisation transversales complémentaires qui sont collectées dans le paquet MARTE annexes qui comprend les opérations prédéfinies couramment utilisés dans les systèmes temps-réel. Dans cette architecture, la séparation entre la modélisation des concepts de système et l'annotation des modèles permet l'analyse des propriétés du système. Cette séparation est faite par les deux paquetages MARTE design model et MARTE analysis model.

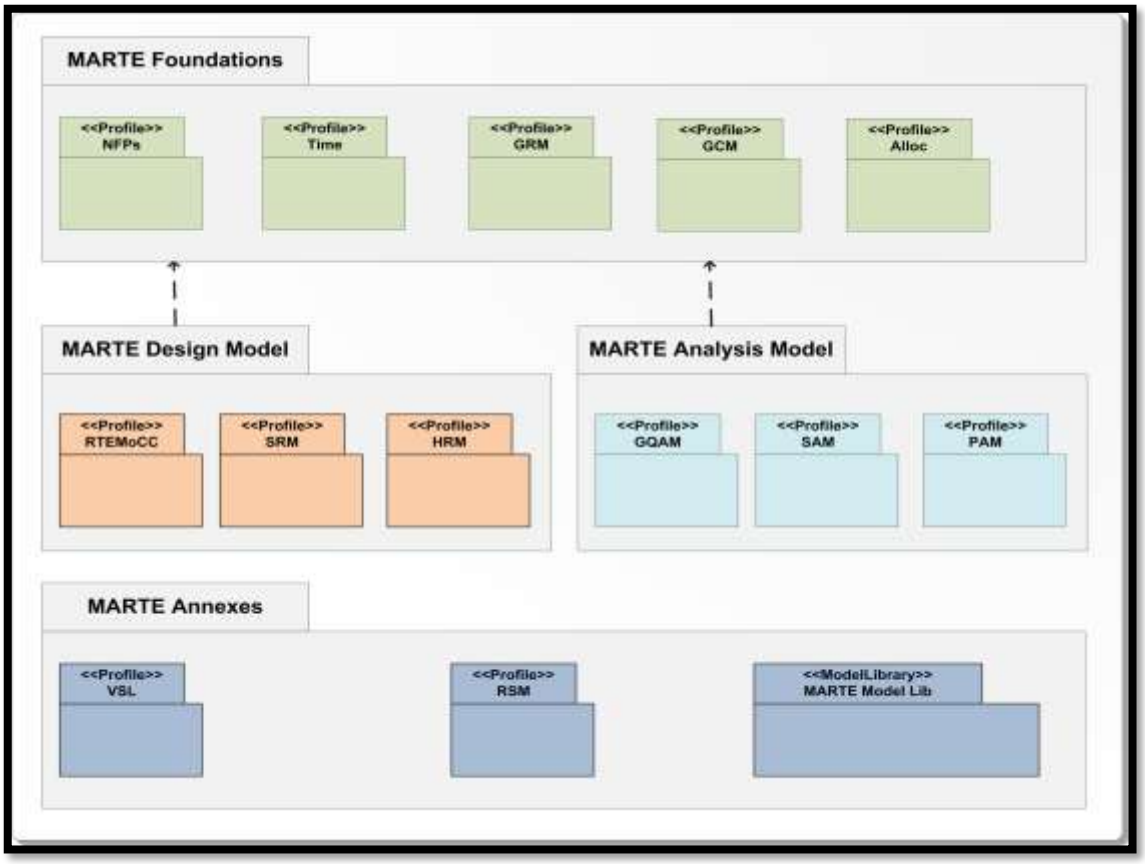


Figure 28 : Architecture globale du profil MARTE.

3.1. Les paquetages du profile MARTE.

Nous donnons, ci-dessous, brièvement un résumé global des concepts MARTE présents à la fois dans le méta-modèle MARTE et le profil MARTE.

- **Foundation package :**

- **Core Elements :** Décrit les concepts de base dans les spécifications MARTE, tels que les modèles, les éléments de modèle et leurs comportements associés. Les comportements peuvent être n'importe quelle chose comme les machines d'état, les diagrammes d'activité, les interactions (diagrammes de séquence / chronogrammes), les automates, etc.

- **Non Functional Properties (NFPs) :** Ils permettent de décrire les propriétés qui ne sont pas liées à des aspects fonctionnels tels que la consommation d'énergie, l'utilisation de la mémoire, etc. Il s'agit d'un aspect crucial pour une description détaillée d'un RTES.

- **Time :** Le paquet MARTE Time permet les concepts utilisés principalement dans le domaine synchrone ainsi que dans les systèmes en temps réel discrets tels que les FPGA. La

notion la plus importante dans ce paquet est l'utilisation de contraintes de temps sur les modèles comportementaux d'UML tels que les diagrammes de séquence et de machines d'état.

- **Generic Resource Modeling (GRM) :** Ce paquetage introduit le concept de ressources et de services de ressources. Les ressources peuvent être classées en différents types, tels que le calcul, le stockage et les ressources de synchronisation. Les ressources peuvent également être gérées et négociées et peuvent être planifiées. Cette notion permet de modéliser les ressources partagées et mutuellement exclusives. Ce paquetage contient les notations utiles pour la modélisation de plateformes d'exécution des systèmes embarqués.
- **Generic Component Modeling (GCM) :** Ce paquetage englobe les concepts de base pour la modélisation en utilisant la notion de composant. Il permet de définir des concepts tels que les composants, les ports et les instances.
- **Allocation :** Le paquetage d'allocation permet l'affectation du modèle d'application dans le modèle d'architecture. L'allocation peut être soit spatiale ou temporelle dans la nature.

➢ **MARTE design model :**

- **High Level Application Modeling (HLAM) :** Ce paquetage permet la description des caractéristiques et des fonctionnalités des systèmes à temps-réel. Son objectif est de fournir des concepts de modélisation de haut niveau pour la modélisation des caractéristiques temps-réel et embarquées.
- **Software Resource Modeling (SRM) :** Ce paquetage fournit des concepts utiles pour la modélisation des plates-formes logicielles d'exécution. Il permet de décrire les ressources et les services de systèmes d'exploitation temps réel (RTOS) d'une manière commune.
- **Hardware Resource Modeling (HRM) :** Les concepts de matériel dans MARTE permettent de représenter des architectures matérielles dans plusieurs points de vue. Les points de vue peuvent être soit fonctionnels, soit physiques, soit hybrides.

➢ **MARTE analysis :**

- **Generic Quantitative Analysis Modeling (GQAM) :** Le paquetage GQAM permet aux concepteurs de se concentrer sur l'analyse via le profil MARTE. L'analyse peut être faite pour le comportement du logiciel (comme de l'ordonnancement et de la performance) ainsi que pour d'autres aspects telles que la puissance, l'énergie, la tolérance aux pannes, etc. Le paquetage GQAM donne la description de la façon dont le comportement du système utilise les ressources disponibles. La mémoire et l'utilisation d'énergie sont également abordées dans ce paquetage.

- **Schedulability Analysis Modeling (SAM) :** Ce paquetage étend celui de GQAM dans le cadre d'analyse de l'ordonnancement. L'ordonnancement d'un système peut être lié au système lui-même ou à un sous-module, par exemple pour répondre à certaines contraintes comme celles liées au temps (par exemple, deadlines, miss rations). L'analyse de l'ordonnançabilité aide également à l'optimisation du système. Un système peut être analysé selon des scénarios ou des valeurs d'entrée différentes afin d'observer les différences.
- **Performance Analysis Modeling (PAM) :** Le paquetage PAM étend le paquetage GQAM pour l'analyse des propriétés temporelles de systèmes embarqués en temps réel.

➢ **MARTE annexes :**

- **Value Specification Language (VSL) :** Le langage qui doit être utilisé dans les contraintes NFP. VSL définit les types de données, des paramètres, des constantes, des énumérations et des expressions. Ces expressions VSL peuvent être utilisées pour spécifier les paramètres non fonctionnels, les valeurs, les opérations, les valeurs et les dépendances entre les différentes valeurs dans un modèle.
- **Repetitive structure package (RSM) :** Permet aux expressions compactes de représenter les applications massivement parallèles et des architectures. Les topologies en grille et en cube peuvent être exprimées également, ainsi que des topologies d'interconnexions présentes dans les NoCs et les réseaux d'interconnexion à plusieurs étages [63, 64].
- **Clock Handling Facilities :** Cette annexe fournit la syntaxe abstraite pour spécifier les dépendances d'horloge et les valeurs synchronisées.
- **MARTE Libraries :** Cette annexe définit les bibliothèques MARTE prédéfinies pour les types primitifs, des types de données étendus, ainsi que d'une bibliothèque de temps qui définit les énumérations pour les concepts de temps.

3.2. La modélisation des structures répétitives.

Les domaines d'application tels que le traitement du signal, le traitement d'images, ou des appareils mobiles, exigent habituellement des calculs intensifs de données à effectuer, éventuellement de façon parallèle, et avec l'aide de plusieurs unités de calcul. Dans le domaine des systèmes embarqués, nous appelons ce genre de systèmes " les systèmes embarqués à calcul intensif ". Le but de cette section est de présenter des constructions de modélisation de haut niveau qui permettent de tenir compte de ce genre de systèmes. Plus précisément, il décrit une manière compacte pour exprimer la régularité de la structure ou de la topologie d'un tel système. Les structures considérées sont composées de répétitions d'éléments structurels, reliés entre eux

par un motif de liaison régulière. Nous appelons ce type de constructions "structures répétitives" [65].

Le profile MARTE présente la possibilité de modéliser les architectures ayant une structure répétitive régulière telle que les topologies des NoCs en utilisant l'extension RSM (Repetitive Structure Modeling). Ce paquetage est basé sur les notions du langage Array-OL (Array Oriented Language) développé par Thales Underwater Systems [66] pour permettre la description de parallélisme de tâches et de données, de manière compacte. Il est également utilisable pour décrire des topologies et structures régulières au niveau matériel. Ce langage est dédié aux applications de traitement de signal intensif. Les structures répétitives dans le RSM sont décomposées en sous-composants interconnectés entre eux par l'intermédiaire des motifs réguliers.

Comme illustré dans la figure 29, les mécanismes dans le paquetage RSM sont classés en deux sortes : la spécification de la forme (Shape) et la représentation des liens topologiques. Une Shape est décrite avec un vecteur d'entiers strictement positifs représentant le nombre d'éléments dans chaque dimension du tableau. Comme exemple, afin de modéliser une instance de composant répétée 64 fois sous forme d'un tableau bidimensionnel 8x8, nous représentons une seule instance avec une Shape de (8 ; 8).

Les topologies de lien sont utilisées pour exprimer la connexion entre les éléments répétés sous une forme compacte. Les trois concepts Tiler, InterRepetition et Reshape sont des exemples de ces topologies.

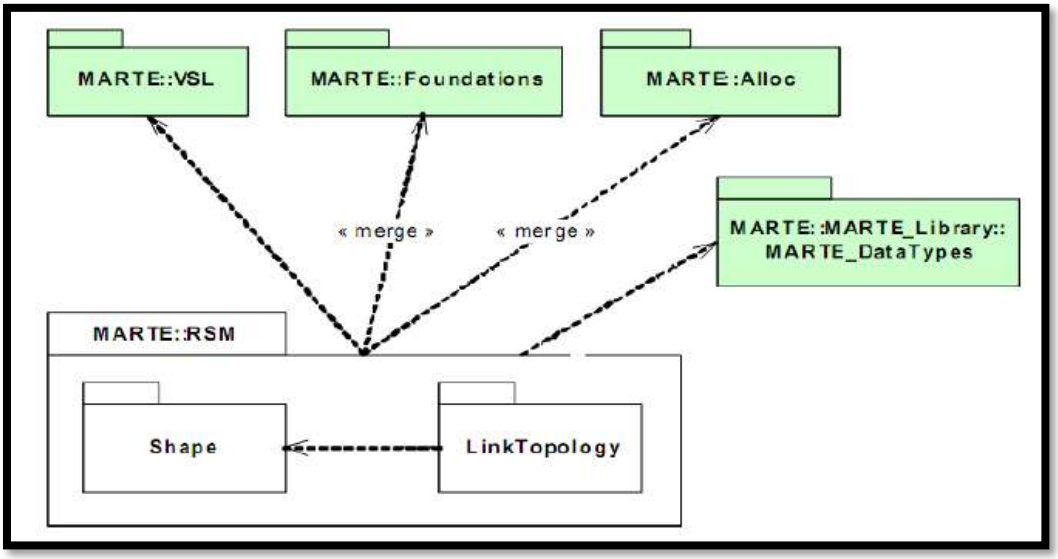


Figure 29 : Architecture globale du paquetage RSM.

4. L'environnement GASPARD.

GASPARD2 (Graphical Array Specification for Parallel and Distributed Computing version 2) [67, 68] est un outil de développement de systèmes embarqués développé par l'équipe DaRT (Dataparallelism pour Real-Time) du Laboratoire d'Informatique Fondamentale de Lille (LIFL). Il est un environnement, basé sur l'Ingénierie Dirigé par des Modèles (IDM), qui permet de modéliser une architecture SoC entier en utilisant le langage UML (plus précisément le profil MARTE) qui répond à la représentation en Y de Gajski et Kuhn [69, 70]. Le modèle Y dans Gaspard2 est principalement défini autour de trois méta-modèles : l'application qui permet de définir la fonctionnalité du système, l'architecture du matériel qui sera utilisé pour modéliser la plate-forme matérielle supportant l'exécution de l'application et exécuter sa fonctionnalité, et l'association qui permet de spécifier le mappage de l'application sur une architecture matérielle donnée. Le déploiement des IPs logicielles et matérielles peut être aussi modélisé en Gaspard2. Il permet d'attribuer les bouts de code, mis dans la bibliothèque, à chaque composant.

Le but de l'environnement GASPARD est de générer le code de simulation dans divers langages, tels que le VHDL qui permet la synthèse des accélérateurs matériels et le SystemC permettant la simulation [71], à partir d'une modélisation en MARTE. L'avantage majeur de modéliser en suivant le flot GASPARD est de faire la modélisation indépendamment du code à produire, qui veut dire qu'il permet la génération du code à partir de l'emplacement d'une application sur une architecture. Cet environnement est basé sur la plateforme eclipse. Gaspard part d'une description générale du système en UML puis, par raffinement et transformation de modèles successifs, génère le code de l'application et de l'architecture matérielle.

Comme illustré dans la figure 30, l'environnement Gaspard2 consiste en six paquetages principaux permettant la modélisation en suivant le modèle en Y. Ces paquetages sont : - component ; - factorisation ; - architecture matérielle ; - application ; - contrôle ; - association.

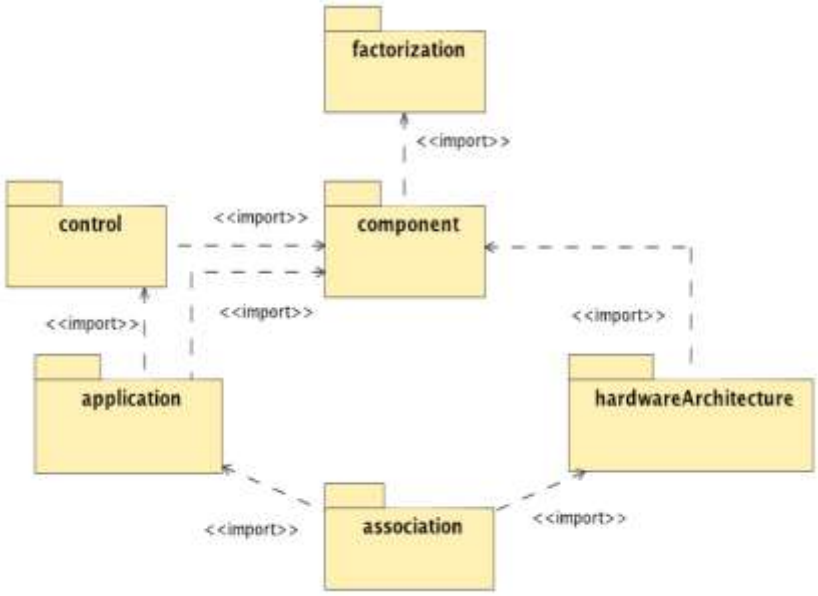


Figure 30 : Les paquetages principaux de l'environnement GASPARD.

La Figure 31 représente l'architecture globale de la méthodologie de co-conception des SoCs proposée par l'environnement Gaspard2. Cette méthodologie consiste à modéliser l'application et l'architecture séparément [72, 73].

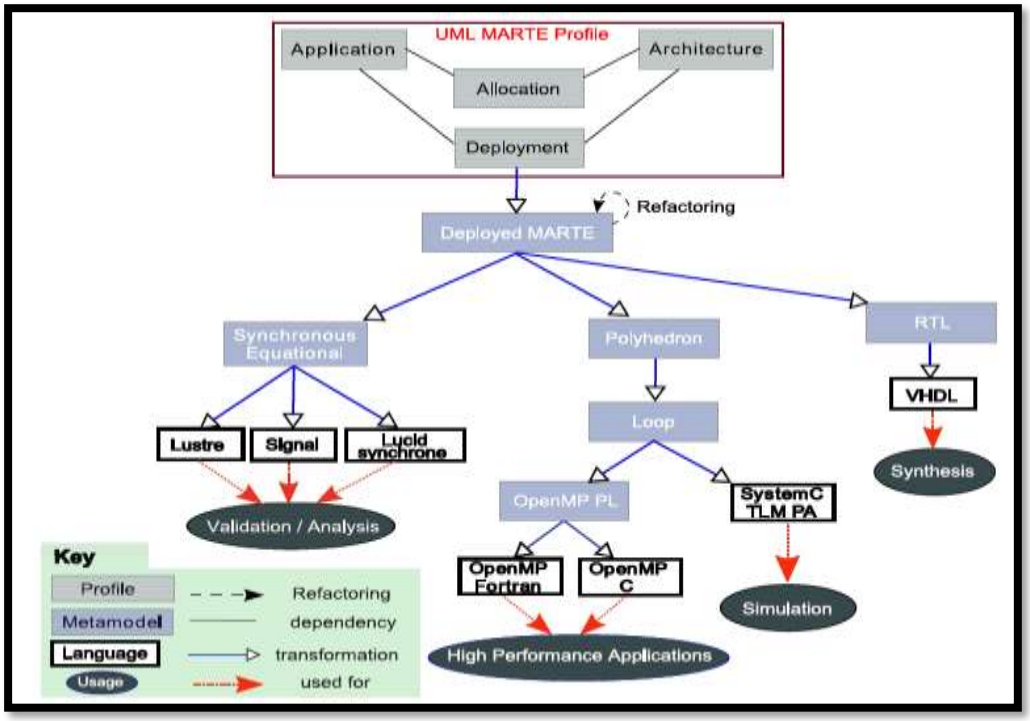


Figure 31 : Architecture globale de la méthodologie de co-conception de GASPARD.

4.1. Application

La modélisation de l'application est spécifiée par l'utilisation du paquetage RSM. Le parallélisme de tâches ou de données, sur GASPARD, peut exprimer par une modélisation compacte de la composante répétée. Deux concepts sont très importants à utiliser lors de la modélisation sur GASPARD : - un composant qu'il est une boîte noire qui représente la description abstraite d'une fonctionnalité ; - une instance de composant. L'ensemble d'instance de composant forme le composant global, donc un composant peut contenir plusieurs instances mais le sens inverse n'est pas autorisé. Il y a 3 types de composants : - *élémentaire* qu'il ne possède aucune instance de composant ; - *composé* qu'il est composé d'instances de composant connectées via des ports ; - *répétitif* qu'il contient une instance de composant qui est répétée une ou plusieurs fois.

4.2. Architecture.

L'aspect le plus intéressant pour nous est la modélisation des topologies d'un NoC régulières de type 2D, 3D et 2D à base de cluster d'une façon compacte. Dans l'architecture globale de MARTE il y a le paquetage HRM qui décrit les concepts associés à la modélisation d'une architecture matérielle cible. Il fournit des ressources à différents aspects, par exemple on trouve les unités de calculs (processsor), les unités de stockage (Hw_RAM) et les unités de communication (Hw_bus).

4.3. Association ou l'allocation.

Une association en GASPARD permet d'allouer des tâches sur des processeurs ou l'allocation des composants VHDL sur des composants en VHDL. L'association en GASPARD peut être vue comme une allocation ou une distribution. Le profil MARTE fournit les stéréotypes nécessaires.

4.4. Déploiement

Dans les phases précédentes, la modélisation est faite indépendamment d'une technologie cible. Cette phase dépend des objectifs du concepteur. La phase de déploiement consiste à implémenter les modèles de l'application et de l'architecture, dans un langage suivant les objectifs du concepteur afin d'obtenir un code exécutable. En effet, tous les composants élémentaires de l'application et de l'architecture devront être liés à des IPs de la bibliothèque. Suivant la chaîne de GASPARD2 on peut générer : - un code SystemC ; - un code VHDL ; - un code OpenMP/Fortran.

Les différentes implémentations des IPs sont rassemblées dans un concept appelé VirtualIP. La bibliothèque qui regroupe ces IPs est appelée GaspardLIB [74, 75].

5. Conclusion.

Dans ce chapitre, nous avons présenté le langage de modélisation de haut niveau UML ainsi que les profils les plus répandus dans le domaine des SoCs et des systèmes embarqués. Plusieurs approches de spécification orientées modèles sont apparues afin d'accélérer la conception des SoCs. Parmi ces approches, nous avons présenté l'Ingénierie Dirigée par les Modèles (IDM) qui se concentre sur les modèles, les méta-modèles, et les transformations de modèles. Elle a permis plusieurs améliorations significatives dans le développement des systèmes en permettant de se concentrer sur une préoccupation plus abstraite que la programmation classique, suivant le principe de : "tout est un modèle".

Par la suite, nous avons traité le langage UML et ses différents diagrammes. L'UML est un langage de modélisation visuelle qui permet aux développeurs de créer des modèles qui capturent leurs visions dans une norme facile à comprendre ; il fournit un mécanisme pour partager et communiquer ces visions avec les autres d'une manière efficace.

L'UML a rassemblé les meilleures caractéristiques des différents langages de modélisation à objet en une seule notation. Comme l'UML est très répandu dans l'industrie ainsi que dans le milieu universitaire, dans un but de modélisation, un grand nombre d'outils ont été développés pour le soutenir. Parmi ces outils, on trouve le GASPARD qui est un outil de développement de systèmes embarqués développé par l'équipe DaRT du Laboratoire d'Informatique Fondamentale de Lille. Cet environnement est basé sur le profile MARTE. Il est un profile UML dédié à la modélisation des systèmes embarqués à temps réel. Ce profil vise à étendre l'UML pour être utilisé dans un développement dirigé par les modèles des applications temps réel et des systèmes embarqués, y compris leurs aspects logiciels et matériels.

L'environnement GASPARD2 permet : - la conception des SoCs dans une démarche mixte application/architecture indépendamment l'une de l'autre ; - la modélisation à haut niveau, en utilisant le profil UML/MARTE, de l'application de l'architecture et de l'allocation indépendamment de la technologie cible ; - l'abstraction d'un système dans différents niveaux ; - la génération de code exécutable dans un domaine précis, le code généré peut être synthétisé ou simulé.

CHAPITRE 4

Architecture de l’approche proposée et résultats.

Généralement, les structures d'interconnexion actuelles ne répondent pas aux exigences des futurs systèmes qui intègrent un très grand nombre d'IPs. Pour s'adapter à cette hausse de complexité, le NoC a donc émergé comme un nouveau paradigme de communication pervasive pour connecter les ressources IPs. Il a été proposé comme une solution prometteuse pour résoudre les problèmes rencontrés au niveau des interconnexions classiques généralement basés sur des bus partagés ou des crossbars, qui ne supportent pas les débits élevés. Ces interconnexions classiques manquent de flexibilité et ne seront pas adaptées pour les systèmes futurs implémentant plusieurs centaines de composants IPs. En-effet, les concepteurs des SoCs embarquent de plus en plus de composants IPs afin de répondre aux besoins de nouvelles applications. Cette augmentation d'intégration au niveau d'un NoC, généralement d'une topologie Mesh 2D, provoque une augmentation de la surface de la puce. Cette augmentation au niveau de la taille provoque elle-même plusieurs inconvénients, comme la réduction au niveau de la bande passante et la fréquence de fonctionnement et une augmentation au niveau de la latence et la consommation de l'énergie [35].

Ce chapitre est organisé en trois parties. Dans la première partie, nous présenterons l'approche proposée à base d'une topologie Mesh 2D et à base de clustering. Le but fixé est de résoudre les problèmes rencontrés dans les architectures classiques. Cette approche est basée sur une combinaison entre une stratégie de placement des modules, un routage XY à deux niveaux et enfin une technique de clustering basée sur le taux de communication entre les modules. L'approche proposée induit surtout une consommation des ressources logiques minimales, une consommation d'énergie réduite et fournit une meilleure latence par rapport aux techniques existantes pour les réseaux Mesh 2D classique, Mesh 3D classique et Mesh 3D modifié.

Dans la deuxième partie de ce chapitre, nous présenterons la modélisation de l'approche proposée en utilisant le profile UML-MARTE. La modélisation est réalisée en deux parties : la modélisation structurelle, dans laquelle nous modélisons la structure des concepts du réseau réalisé en utilisant des diagrammes structurels comme le diagramme de composant et le diagramme de composite ; et la modélisation comportementale, dans laquelle nous représentons le comportement du réseau réalisé en utilisant les diagrammes de séquence, d'activité et d'état. Nous finirons par la génération du code VHDL.

La troisième partie de ce chapitre est consacré à l'évaluation de la performance du système implémenté. Pour évaluer ce système, nous avons appliqué l'approche proposée à un réseau

d'une topologie Mesh 2D de différentes tailles (4x4, 8x8 et 12x12). Les résultats obtenus sont comparés avec ceux de la topologie Mesh 2D classique. Par la suite, nous avons comparé les quatre approches (2D classique, 3D classique, 3D modifié et notre architecture à base de clusters) constituées de 64 coeurs, donc des réseaux d'une taille de 8x8, 4x4x4, 4x4x4 modifié et 4x4x4 à base de clusters.

1. La technique de clustering.

Les algorithmes de clustering sont utilisés dans de nombreux problèmes de réseau. Ils consistent à diviser le réseau en un ensemble de nœuds sous certaines conditions. En particulier, il permet au protocole de routage de fonctionner plus efficacement en réduisant le trafic de contrôle dans le réseau et en simplifiant les données de processus de commutation. Ces algorithmes ont des caractéristiques différentes et sont conçus pour satisfaire certains objectifs en fonction du contexte dans lequel le regroupement est déployé (routage, sécurité, économies d'énergie,). Plusieurs études [77, 78, 79] s'intéressent à l'établissement d'une classification des approches existantes de clustering. Ces algorithmes ont différentes caractéristiques telles que le nombre des nœuds dans chaque groupe, le nombre moyen de clusters formés dans le réseau, les distances entre les nœuds et les chefs des clusters.

La technique de clustering consiste à organiser les nœuds du réseau en des groupes virtuels appelés "clusters". Les nœuds, géographiquement voisins, sont regroupés dans un même cluster selon certaines règles (distance, application, synchronisation, charge de communication …). Dans un cluster, on retrouve généralement trois types de nœuds comme le montre la figure 32 : "cluster-head", nœuds "passerelles" et nœuds "ordinaires" dits aussi membres. Dans chaque cluster, un nœud est élu comme chef de groupe, appelé cluster-head, qui possède des fonctions supplémentaires. Une passerelle est un nœud non-cluster-head qui possède des liens inter-clusters et peut donc accéder à des clusters voisins et acheminer les données entre eux, tandis qu'un nœud ordinaire est un nœud non-cluster-head qui ne possède pas des liens avec les autres clusters. Dans notre cas, nous utilisons seulement deux types (chef et ordinaire). Le clustering représente une solution prometteuse pour les réseaux à base de circuits reconfigurables (FPGA).

Le clustering représente une solution prometteuse pour les réseaux à grand nombre de nœuds [80, 81]. Cette technique de structuration du réseau possède plusieurs avantages. Les nœuds de chaque cluster sont supervisés par leur cluster-head qui peut coordonner l'accès au canal, épargnant ainsi les ressources gaspillées dans la retransmission due aux collisions. La

structure de cluster fait apparaître le réseau plus petit et plus stable aux yeux de chaque nœud du réseau.

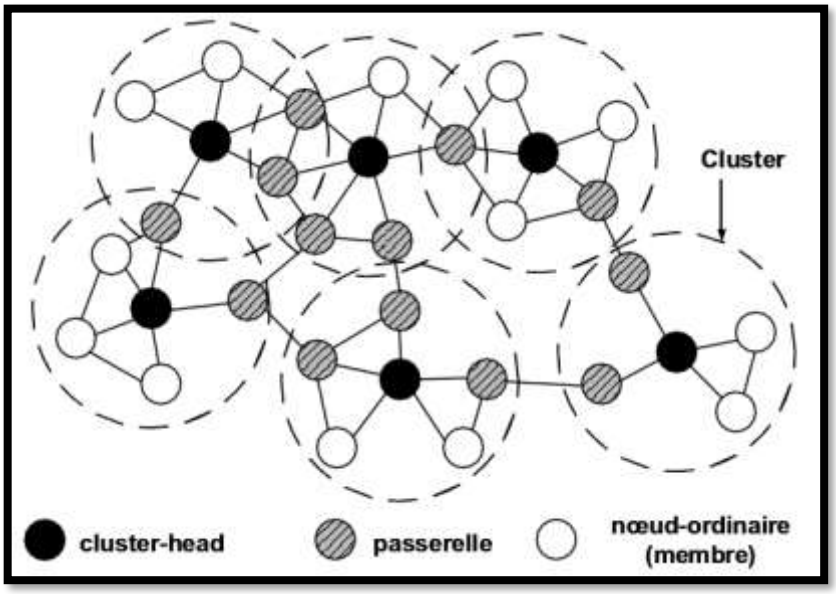


Figure 32 : Exemple de structure de clusters.

La littérature présente plusieurs mécanismes pour le clustering [77, 78, 79, 82]. Celui-ci peut être basé sur plusieurs critères. Une des classifications s'appuie sur la sélection du chef de cluster. Une autre classification peut prendre en compte le nombre de sauts qui séparent un nœud ordinaire d'un nœud chef du cluster auquel il est attaché.

Nous présentons ci-dessous les critères de classification des approches existantes de clustering. Plusieurs métriques ont été proposées dans la littérature pour élire l'ensemble des cluster-heads. Nous présentons ci-dessous les métriques de sélection des cluster-heads.

❖ **Sans métrique :** Regroupe les algorithmes qui déclarent les cluster-heads sans avoir recours à aucune métrique de sélection.

❖ **Métrique arbitraire :** Représente une valeur choisie arbitrairement et généralement non significative. Dans cette classe, nous trouvons les algorithmes qui utilisent des valeurs aléatoires ou l'identifiant des nœuds.

❖ **Métriques liées à la topologie :** Dans cette classe nous groupons tous les algorithmes qui utilisent une métrique issue de la topologie du réseau. Parmi les métriques qui appartiennent à cette classe nous citons : le degré de connectivité, le k-degré de connectivité et la k-densité.

❖ **Métriques combinées :** Plusieurs algorithmes combinent plusieurs métriques de différents types pour élire l'ensemble de cluster-heads.

2. Architecture du réseau réalisé.

2.1. Le routeur utilisé.

La figure 33 représente l'architecture du routeur utilisé pour la réalisation de l'approche proposée. Il se compose de quatre ports bidirectionnels Est, Ouest, Nord et Sud. Ce routeur contient quatre blocs principaux :

- L'unité de routage : l'algorithme de routage utilisé pour l'approche proposée est divisé en deux parties : - routage intra-cluster ; - routage inter-cluster.
- L'unité d'arbitrage : l'arbitrage est utilisé pour gérer la priorité entre les différentes demandes du transfert ; nous avons choisi l'arbitrage Round-Robin en utilisant un compteur modulo 6 pour les topologies Mesh 3D et Mesh 3D modifiée et un compteur modulo 4 pour les topologies Mesh 2D et Mesh 2D à base de cluster de notre structure proposée.
- Finite State Machine (FSM) : pour gérer les signaux de contrôle et de commande.
- DATA-PATH : c'est le chemin de données nécessaire pour le transfert de données.

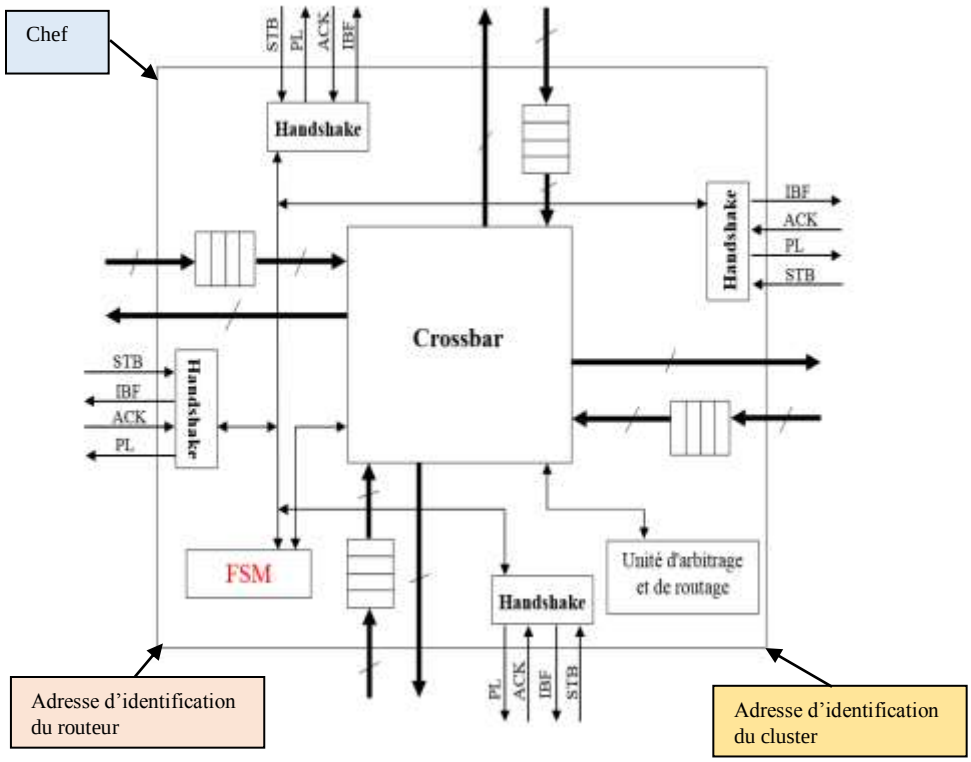


Figure 33 : Architecture du routeur implémenté.

2.1.1. Les FIFOs utilisées :

Ce type de mémoire est formé par un arrangement de registres à décalage. Le terme FIFO désigne le fonctionnement de base de ce type de mémoire, dans lequel le premier bit de données écrit dans la mémoire est le premier à être lu. Dans l'architecture du routeur proposé, nous avons utilisé une FIFO à 8 niveaux avec une largeur de 8 bits par niveau (fig.34).

Les demandes d'écriture et de lecture sont conditionnées par l'état du FIFO (Full, Empty, Half). L'écriture sur la FIFO se fait quand son état est vide ou moyenne (quand elle n'est pas pleine). La lecture se fait quand l'état du FIFO moyenne ou pleine (quand elle n'est pas vide).

Les données entrantes sont mémorisées lorsque le signal de demande d'écriture est validé et qu'il y a de l'espace disponible. Quand les flags de lecture et d'écriture sont égaux à zéro, ça veut dire qu'il y a certaines données disponibles dans le buffer. Les signaux S1 et S7 indique si le buffer est vide ou plein (S1 = 1 c'est-à-dire le buffer est vide, S7 = 1 c'est-à-dire le buffer est plein). Lorsque le premier octet est écrit dans la FIFO rien ne se passe sur le bus « Do » jusqu'à ce que le signal « Rd » soit pulsé haut pendant au moins un cycle d'horloge. Une fois qu'un octet a été écrit dans la FIFO, le drapeau vide sera bas. Pour lire l'octet suivant de la mémoire FIFO, le signal « Rd » soit haut pour un cycle d'horloge et le prochain octet de données sera disponibles pour lecture sur le cycle d'horloge suivant. Lorsque le dernier octet de données est poussé sur le bus « Do », le drapeau vide devient haut.

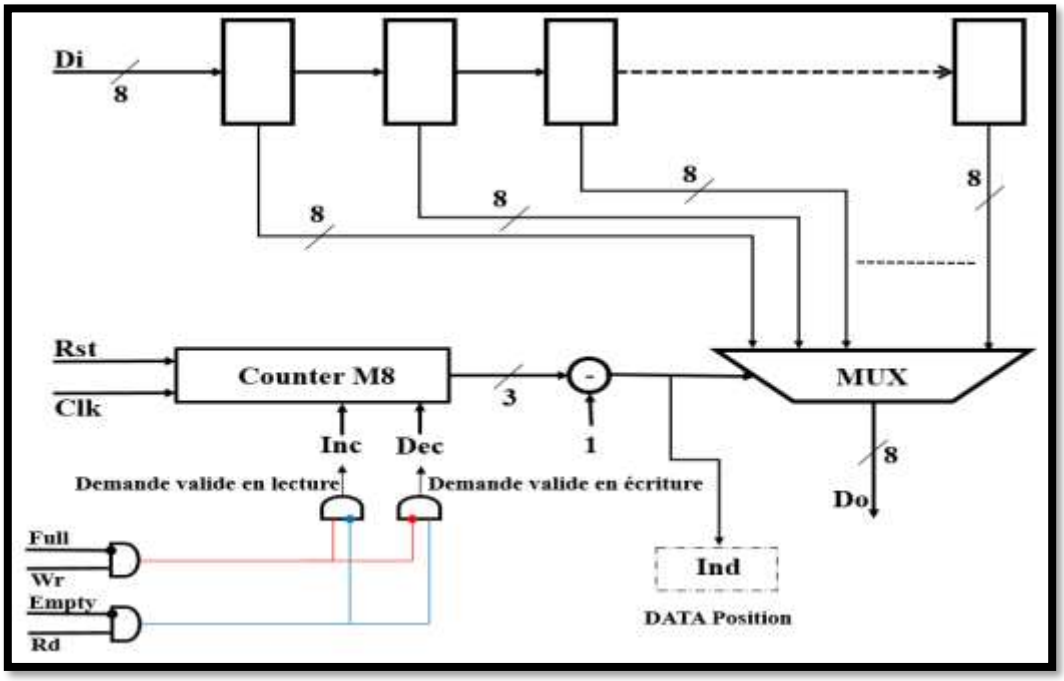


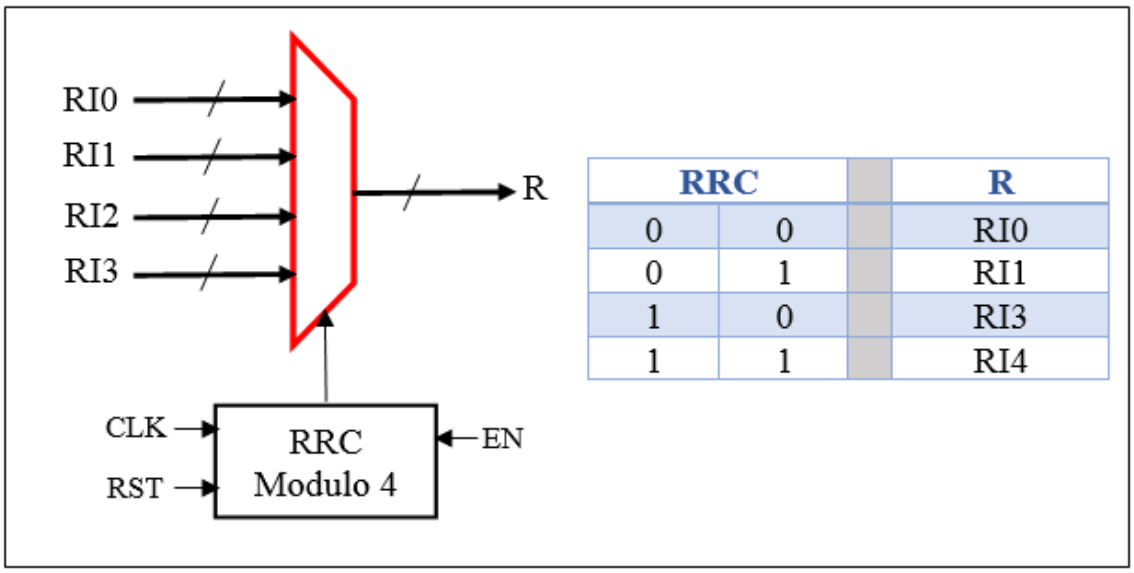
Figure 34 : Architecture du FIFO utilisée.

2.1.2. Logique de contrôle :

Deux modules mettent en œuvre la logique de contrôle : le routage et l'arbitrage, comme nous l'avons présenté dans la Figure 33. Lorsqu'un commutateur reçoit un flit d'en-tête, l'arbitrage est exécuté et si la requête de paquet entrant est accordée, un algorithme de routage XY est exécuté pour connecter les données du port d'entrée au port de sortie correcte. Dans cette partie nous détaillerons ces deux unités.

- **L'unité d'arbitrage :**

Le protocole d'arbitrage utilisé par le routeur réalisé est de type Round-Robin où nous utilisons un compteur modulo 4 afin de choisir le port d'entrée qui a la priorité de sortir le premier (fig.35, fig.36). La logique d'arbitrage Round-Robin analyse les demandes d'entrée d'une manière cyclique à partir de la position qui a la plus haute priorité, et accorde la première demande active. Pour le cycle d'arbitrage suivant, les points de vecteur prioritaire à la position suivante à l'entrée accordée. De cette manière, l'entrée accordée reçoit la priorité la plus faible dans le cycle d'arbitrage suivant.



RRC			R
0	0		RI0
0	1		RI1
1	0		RI3
1	1		RI4

Figure 35 : Multiplexeur du port d'entrée.

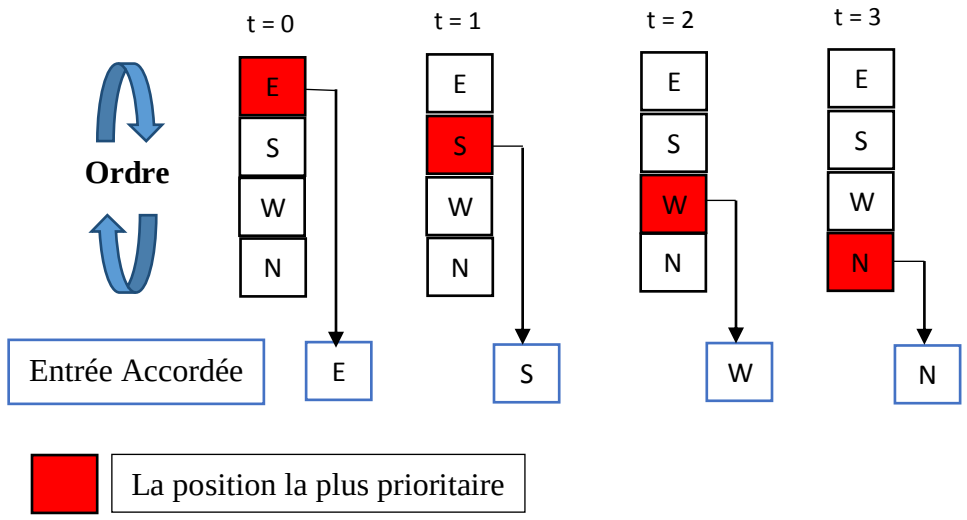


Figure 36 : Arbitrage Round-Robin à 4 états.

- **L'unité de routage :**

Pour les NoCs d'une topologie Mesh de n-dimension, le routage DOR (Dimention Order Routing) produit des algorithmes de routage sans deadlock. Ces algorithmes sont très populaires, comme le routage XY pour Mesh 2D et XYZ pour Mesh 3D.

L'algorithme de routage utilisé dans cette conception est un routage XY à deux niveau. Quand on a un adressage local (intra-cluster) le routage choisi est "XY" classique. Par contre quand on a un adressage global, l'algorithme choisi est "XY" selon cluster. Selon cet algorithme, un paquet doit toujours être acheminé le long de l'axe horizontal X du Mesh jusqu'à ce qu'il atteigne la même colonne de la destination. Ensuite, il doit être acheminé le long de l'axe vertical Y en direction de l'emplacement de la ressource destinataire. Nous détaillerons cet algorithme par la suite.

2.1.3. Protocole de communication :

Comme illustré dans la figure 37, pour faire l'interaction entre les routeurs, c'est le protocole de communication Handshake qui est utilisé. Dans le cas où les données sont mises sur la ligne, l'existence des données est informée au routeur suivant. Le routeur suivant prend les données de la ligne et transmet sa confirmation au routeur expéditeur. De plus, les signaux PL, ACK, STB et IBF sont nécessaires.

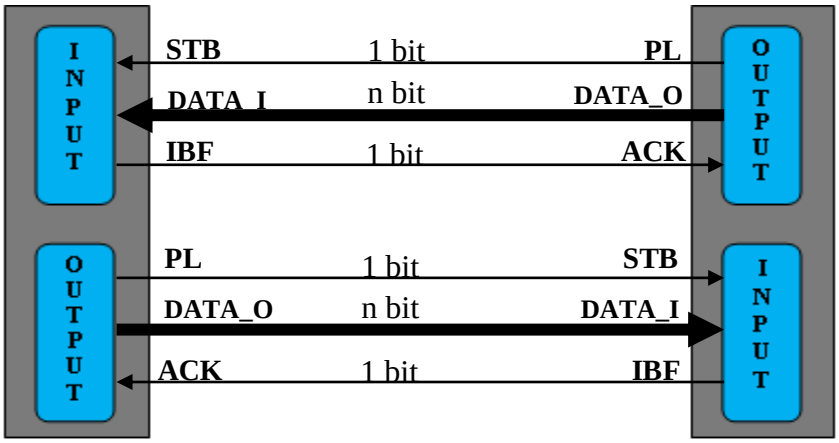


Figure 37 : Signaux du protocole de communication Handshake.

Les signaux de communication dans le protocole de Handshake entre deux routeurs sont :

- DATA_I : les données à recevoir (n bits) ;
- DATA_O : les données à transmettre (n bits) ;
- PL : signal de contrôle indiquant la disponibilité des données pour sortir (1 bit) ;
- STB : signal de contrôle indiquant la disponibilité des données en entrée (1 bit) ;
- ACK : signal de contrôle indiquant la réception réussite de données (1 bit) ;
- IBF : signal de contrôle indiquant l'état du buffer (1 bit) (IBF = 1 => buffer full, IBF = 0 => buffer empty).

Dans ce protocole, lorsque le commutateur a besoin d'envoyer des données à un commutateur voisin, il les met dans le signal DATA_O et active le signal PL. Une fois que le commutateur voisin a stocké les données à partir du signal DATA_I, il affirme que les données sont bien reçues par un signal ACK et la transmission est terminée.

2.1.4. L'unité FSM :

La machine à état (FSM = Finite State Machine) est une machine abstraite composée d'un ensemble d'états (y compris l'état initial), un ensemble d'événements d'entrée, un ensemble d'événements de sortie et une fonction de transition d'état. C'est une machine qui fait des transitions d'un état à un autre basé essentiellement sur deux conditions : l'état dans lequel nous sommes actuellement et la valeur de certains intrants (inputs). La structure générale d'une FSM, comme le montre la figure 38, se décompose en trois parties : - le décodeur d'état futur (Combinatoire) ; - le registre ou mémoire (Séquentiel) ; - le décodeur de sortie (Combinatoire).

Il y a deux types de machines d'état ; machine de Mealy et machine de Moore. Ces machines sont illustrées par la figure 37. Les deux types de systèmes sont déclenchés par une horloge unique. L'état suivant est déterminé par une fonction combinatoire des entrées et l'état actuel. La différence entre les deux modèles est que, dans la machine de Moore les sorties dépendent uniquement de l'état actuel, tandis que dans la machine de Mealy les sorties dépendent de l'état actuel et des entrées. Les deux machines de Moore et de Mealy sont communément appelées machines d'état. C'est-à-dire qu'elles ont un état interne qui change.

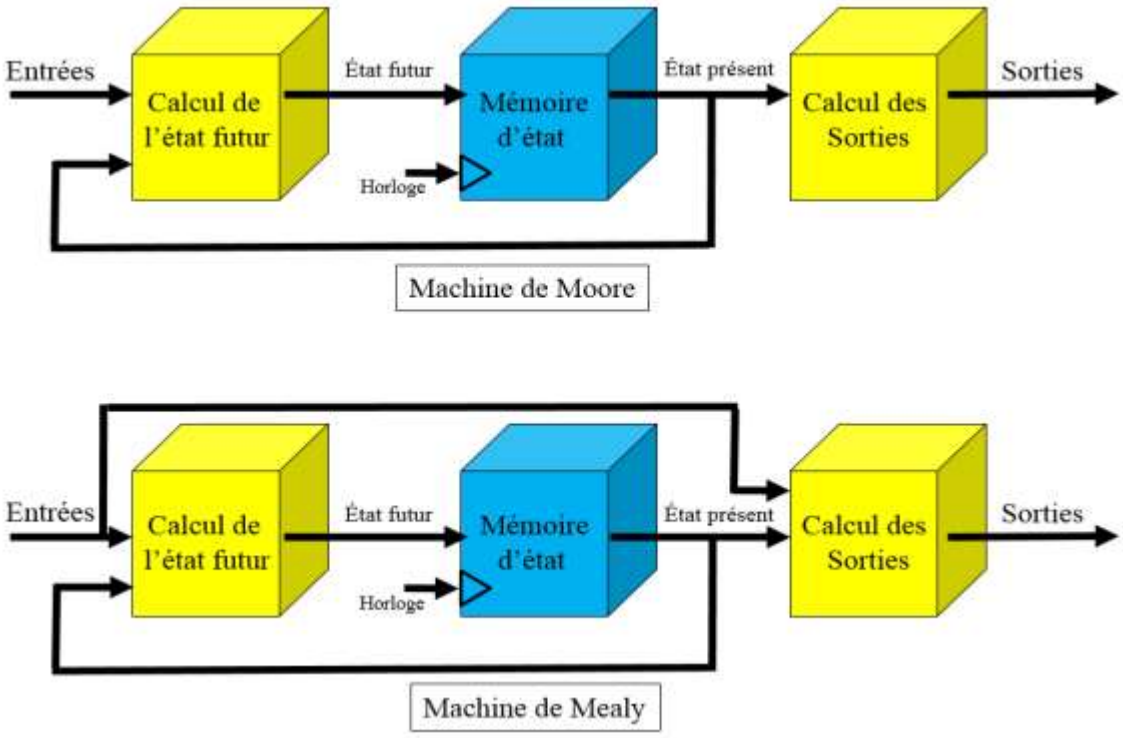


Figure 38 : Les machines de Moore et de Mealy.

Le FSM utilisé dans l'architecture du routeur implémenté est celui de Moore. Dans la structure réalisée, le FSM est utilisé pour gérer les signaux de contrôle du protocole Handshake. Comme le montre la figure 39, il y a quatre états dans architecture implémentée (IBFs, FdRs, ACKs et TRTs). La figure 40 présente l'organigramme de notre FSM.

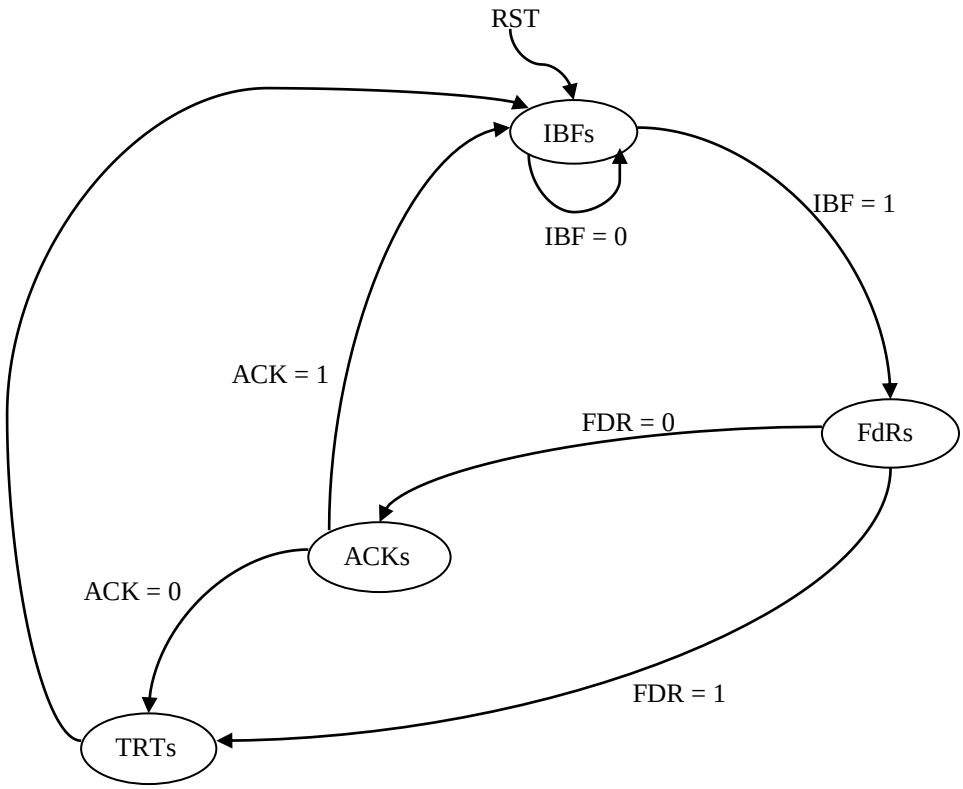


Figure 39 : Les états du FSM.

2.2. Présentation de l'approche proposée.

L'augmentation du nombre d'IPs intégrés dans un réseau d'une topologie Mesh 2D provoque une augmentation de la taille de ce réseau. Cette augmentation au niveau de la taille provoque elle-même plusieurs inconvénients, comme la réduction au niveau de la bande passante et la fréquence de fonctionnement et l'augmentation au niveau de la latence et la consommation de l'énergie.

Afin de résoudre ces limites, des chercheurs ont proposé les architectures 3D (Mesh 3D). C'est vrai que ces architectures offrent de meilleurs résultats au niveau des paramètres ci-dessus mentionnés, mais il reste toujours des problèmes au niveau de la consommation énorme des ressources logiques, la complexité des algorithmes appliqués et la difficulté au niveau de l'implantation sur des circuits reconfigurables.

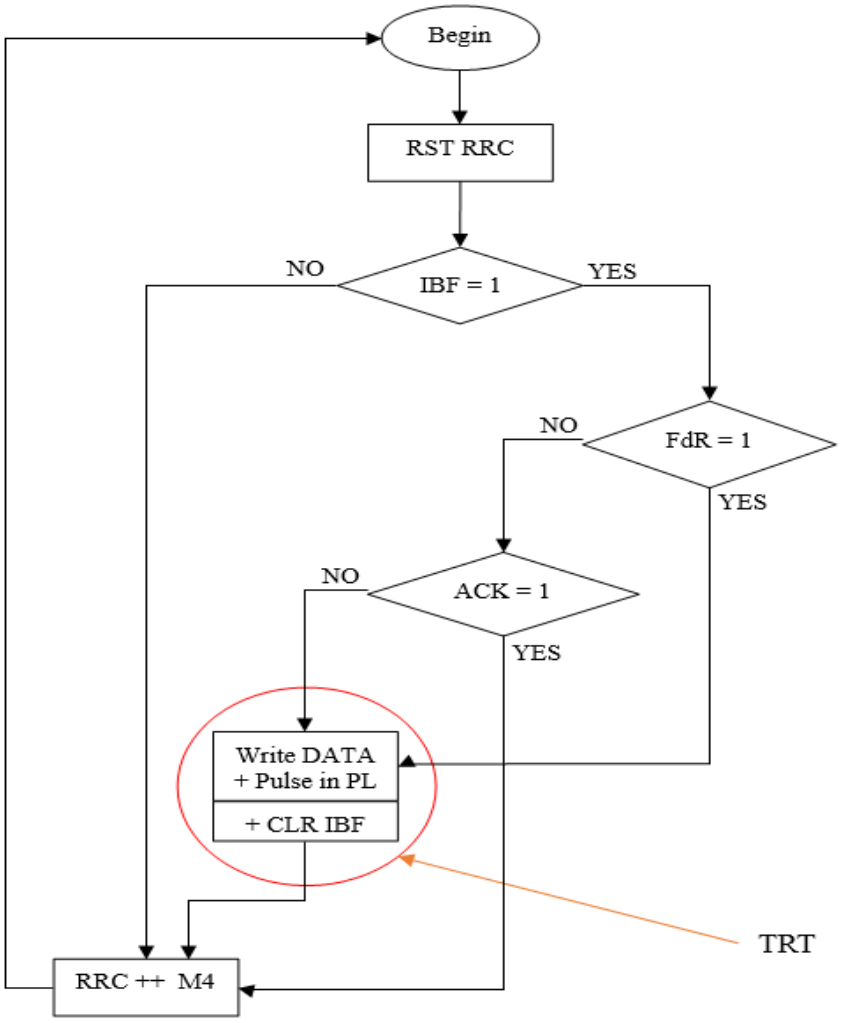


Figure 40 : Organigramme du FSM.

D'autres chercheurs ont proposé une autre approche. Ils ont choisi un routeur chef au niveau de chaque couche. Ces couches sont reliées entre elles à travers les chefs uniquement ; c'est-à-dire que toutes les autres connexions sont éliminées. Cette architecture offre un meilleur résultat au niveau de la consommation logique par rapport à la structure Mesh 3D classique. Mais il reste toujours un autre problème, la dissipation thermique. Comme on le sait, les processeurs dégagent une très grande quantité de chaleur. Cette chaleur provoque des erreurs et des pannes permanentes. Par ailleurs, dans ces structures il est difficile de protéger les éléments de cette chaleur dégagée.

L'objectif principal de l'approche proposée est de résoudre les problèmes rencontrés dans les structures mentionnées précédemment : - éviter le problème de la dissipation thermique par l'utilisation d'une topologie plane (2D) ; - implantation facile sur les circuits reconfigurables ; - minimiser la latence moyenne en utilisant une technique de placement basée sur la charge de

communication entre chaque deux nœuds ; - minimiser la consommation de l'énergie par ce que les modules les plus communicants sont dans le même cluster et cela nous donne une minimisation du nombre de sauts ; - et plus particulièrement, minimiser la consommation des ressources logiques sur la plateforme utilisée qui est, dans notre cas, le FPGA.

Afin d'améliorer la performance du réseau en termes de paramètres mentionnés précédemment, nous proposons un algorithme de routage XY modifié et une technique de placement pour les structures NoC à base d'une topologie Mesh 2D, qui sont combinés avec une technique de clustering appropriée.

Dans l'approche proposée, comme le montre la figure 40, l'ensemble du réseau est découpé en quatre sous-réseaux par l'utilisation d'une technique de clustering où les nœuds les plus communicants sont regroupés dans le même cluster. Le calcul de la densité de la charge de communication entre chaque paire de nœuds est effectué en fonction de l'application à réaliser.

L'architecture proposée est constituée de deux types de routeurs dans chaque cluster (fig.41) : un routeur chef pour chaque cluster et des routeurs ordinaires. Chaque routeur, que ce soit ordinaire ou chef, est identifié par ses coordonnées (x, y) pour le routage local, où « x » est la coordonnée horizontale et « y » la coordonnée verticale. En plus des coordonnées, chaque routeur a un bit test (bit chef) qui nous montre si le routeur est un chef ou non (chef = 0 => routeur ordinaire, chef = 1 => routeur chef). Les routeurs chefs ont des adresses supplémentaires pour gérer les communications entre les différents clusters (inter-clusters).

La figure 42 présente le format du paquet que nous avons utilisé pour la transmission des données. Comme on le voit, on a un bit « A/R » pour définir si le routage est à l'intérieur du même cluster (interne ou intra-cluster) ou entre différents clusters (externe ou inter-clusters). Le bit « A/R = 0 » indique que le routage est intra-cluster et « A/R = 1 » indique que le routage est inter-clusters. Le deuxième champ dans le paquet est réservé aux informations concernant l'adresse du cluster source et celle du cluster destination, dans le cas du routage global ; nous n'avons besoin de ces informations ou de ce champ que dans le cas d'émission d'un paquet vers une destination dans un cluster différent de celui de la source. Le troisième champ est réservé aux informations à propos du routage local, au niveau du même cluster. Le quatrième et le dernier champ contient la charge utile (données à transmettre).

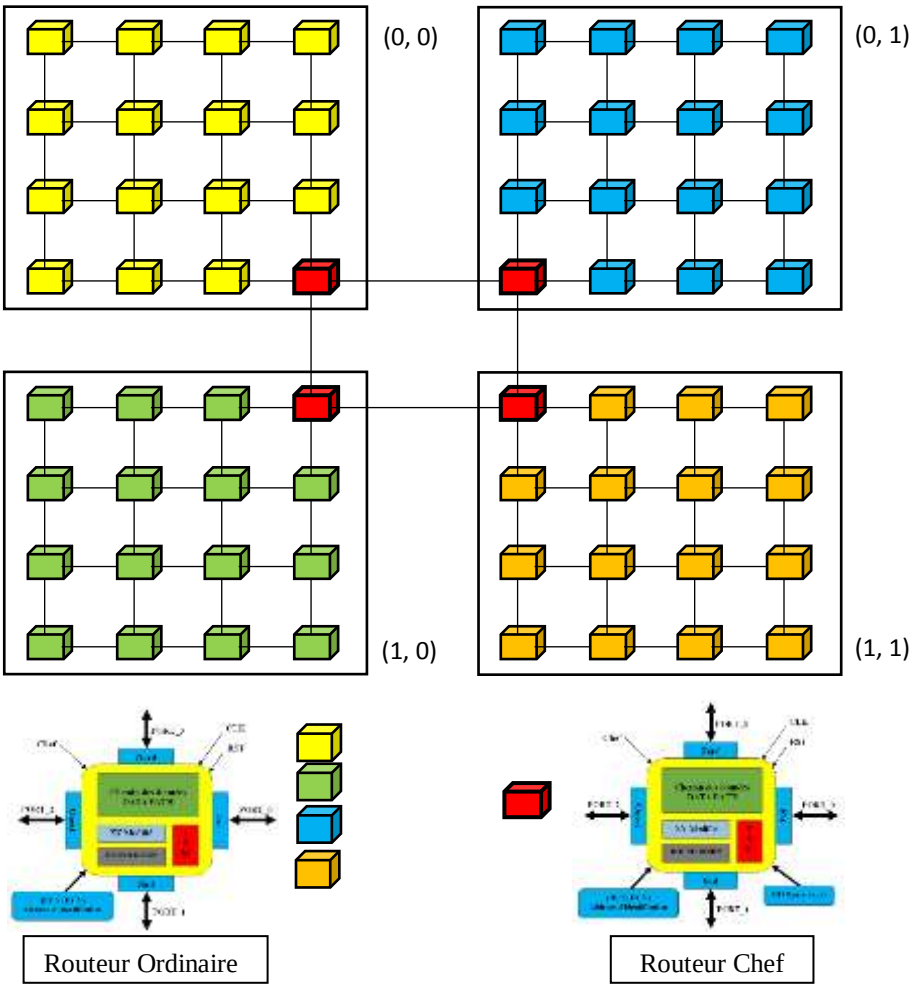


Figure 41 : Regroupement de nœuds dans des clusters.

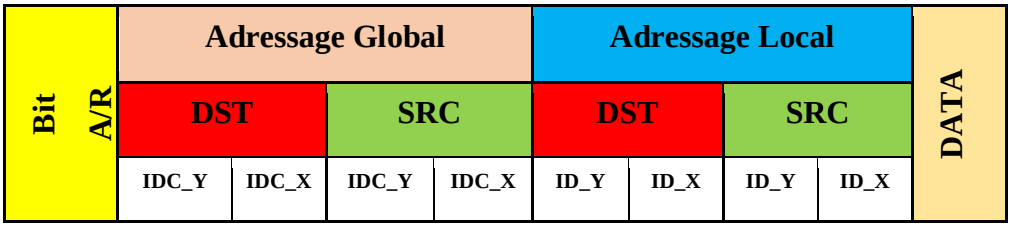


Figure 42 : Format du paquet.

Les étapes de l'approche proposée sont les suivantes (fig.43) :

Etape 1 : Dans un premier temps, calculer la densité de charge de communication entre chaque paire de nœuds pour une application choisie pour sélectionner les nœuds les plus communicants dans le système.

Étapes 2 et 3 : Après avoir obtenu la densité de charge de communication, mettre les nœuds les plus communicants dans le même cluster ; le nombre de nœuds dans chaque cluster ne dépasse pas un certain seuil. Dans notre cas, chaque cluster est constitué de deux types de routeurs : un routeur ordinaire et un routeur chef de cluster. La gestion des clusters est effectuée de manière à obtenir un minimum de communication inter-clusters (entre les différents clusters), où le cluster « 0 » communique plus avec les clusters « 1 » et « 2 » qu'avec le cluster « 3 », raison pour laquelle nous mettons le cluster « 0 » au voisinage des clusters « 1 » et « 2 » et loin du cluster « 3 ». La même opération est effectuée pour tous les autres clusters.

Étape 4 : Dans cette étape, nous sélectionnons un cluster-head pour chaque cluster. Les chefs des clusters contiennent des informations supplémentaires sur le routage dans tout le réseau en ajoutant des instructions à l'algorithme de routage afin de réaliser les communications entre les éléments des différents clusters, tandis que les autres membres du cluster utilisent un routage XY simple. Le rôle du cluster-head est de gérer les communications entre les membres de différents clusters. Comme vu plus haut, la sélection du cluster-head se fait selon certains critères. Cependant, dans notre cas, nous avons choisi comme chef le nœud qui a la plus petite densité de charge de communication dans le cluster. Ce faisant, nous réduisons le nombre de trafics qui passent à travers le chef et minimisons les chances qu'il tombe en panne.

Étape 5 : La dernière étape est la sélection de l'algorithme de routage. L'algorithme de routage proposé est constitué de deux niveaux : un niveau où les communications se font au sein du même cluster (intra-cluster) et un autre où celles-ci se font entre les différents clusters (inter-clusters). Nous détaillerons ci-dessous l'algorithme de routage à utiliser.

Au niveau intra-cluster nous utilisons l'algorithme de routage déterministe pour les structures à deux dimensions XY. Au niveau inter-clusters, l'algorithme de routage choisi est un routage XY modifié. Dans cette technique de routage, chaque routeur est identifié par une adresse unique sur les deux axes « X » et « Y », ID_X et ID_Y, ainsi que l'adresse du cluster qu'on trouve au niveau des routeurs chefs, IDC_X et IDC_Y.

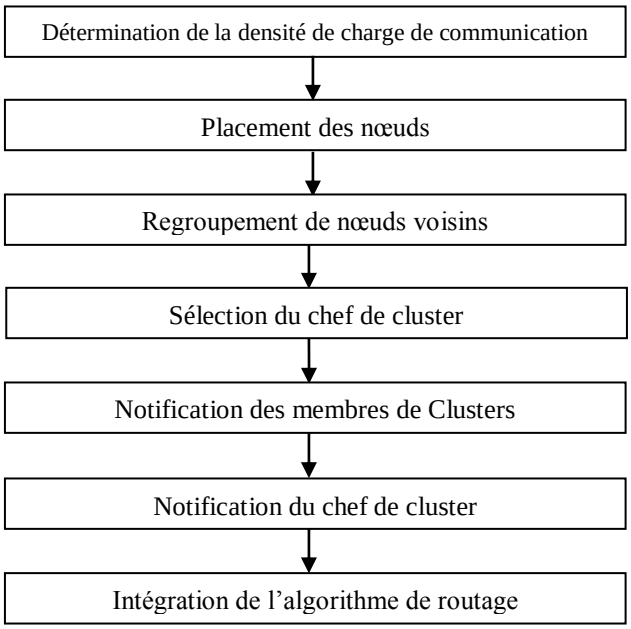


Figure 43 : Les étapes de l'approche proposée.

2.3. Algorithme de routage en mode cluster.

Le paquet circule dans le réseau en utilisant un algorithme de routage approprié pour l'architecture du réseau utilisé. Ce paquet, comme nous l'avons présenté précédemment, consiste en quatre champs, parmi ces champs il y a l'adressage local et l'adressage global qui contiennent les valeurs des composantes « ID_X », « ID_Y », « IDC_X » et « IDC_Y » où ID_X et ID_Y sont les coordonnées en « x » et en « y » de la source et de la destination du paquet à transmettre pour le routage local, IDC_X et IDC_Y sont les coordonnées en « x » et en « y » de la source et de la destination du paquet à transmettre pour le routage global.

Le routage, dans l'approche proposée, est divisé en deux niveaux ; routage intra-cluster et routage inter-clusters. Si le bit « A/R = 0 », cela veut dire que le routage choisi est le routage XY classique suivant l'adressage local, sinon le routage choisi est divisé en trois parties suivant une permutation entre les adressages global et local : - un routage « XY classique » vers le chef du cluster source ; - un routage « XY selon cluster » vers le routeur chef du cluster de destination ; - enfin un retour vers un routage « XY classique » au niveau du cluster de destination.

Cet algorithme commence par tester le bit « A/R » pour décider si le routage est interne ou externe. Dans le cas « A/R = 0 », l'algorithme de routage fait une comparaison entre les adresses du routeur source et du routeur de destination qu'on trouve dans le champ d'adressage local. Le routage se fait selon XY classique vers la destination. Dans le cas « A/R = 1 », l'algorithme de routage fait une comparaison entre les adresses du routeur source et son routeur chef suivant le

routage XY classique et selon un adressage caché qu'on trouve au niveau de chaque routeur. Comme mentionné précédemment, on peut savoir si le routeur courant est le routeur chef ou non par un test du bit « chef ». Quand le paquet arrive au routeur chef du cluster source, le routage se transforme en routage inter-clusters suivant le champ d'adressage global où se trouvent l'adresse du routeur chef du cluster source et celle du routeur chef du cluster destination ; l'adresse d'identification du cluster est définie dans chaque routeur chef. Quand le paquet arrive au routeur chef du cluster de destination, le routage revient vers le routage intra-cluster et le paquet se transmet vers sa destination finale.

Le déroulement du routage intra-cluster (Paquet 1) et inter-clusters (Paquet 2) sera expliqué dans l'exemple illustré par la figure 44.

***Paquet 1* :** Le premier bit dans le paquet 1, qui représente le bit « A/R », est égal à « 0 » ; ceci signifie que le routage est intra-cluster. Dans ce cas, le champ qui nous intéresse est celui de l'adressage local où l'adresse de la source est « ID_X = 0, ID_Y = 0 » et l'adresse de destination est « ID_X = 1, ID_Y = 1 ». Le routage se fait selon XY classique jusqu'à ce que le paquet atteigne sa destination finale.

***Paquet 2* :** Le premier bit dans le paquet 2, qui représente le bit « A/R », est égal à « 1 » ; ceci signifie que le routage est inter-clusters. Au niveau du même cluster, on a une nouvelle destination ; cette destination est le chef du cluster qui contient le routeur source. L'adressage de la nouvelle destination est intégré dans chaque routeur du cluster et cette information n'apparait pas au niveau du paquet. L'étape suivante consiste à tester le bit chef ; dans le cas de l'exemple présenté, le bit « chef = 0 », le routeur courant n'est donc pas le chef. Suivant le routage XY classique le paquet est transmis vers le prochain routeur jusqu'à arriver au routeur chef du cluster source (chef = 1). Le transfert du paquet est fait sans rien changer dans le paquet original. Quand le paquet arrive au routeur chef source, le routage se fait selon le champ d'adressage global qui contient des informations sur le routage entre les différents clusters. Dans cet exemple, l'adresse de la source est « IDC_X = 0, IDC_Y = 0 » et l'adresse de destination est « IDC_X = 1, IDC_Y = 1 ». Le routage se fait selon XY classique entre les chefs des clusters jusqu'à ce que le paquet atteigne le routeur chef du cluster de destination. Quand l'adresse du routeur chef du cluster source (IDC_X, IDC_Y) est égale à l'adresse du routeur chef du cluster de destination (IDC_X, IDC_Y), le bit A/R devient égal à « 0 » et le routage se fait selon l'adressage local du paquet jusqu'à ce qu'il atteigne sa destination finale.

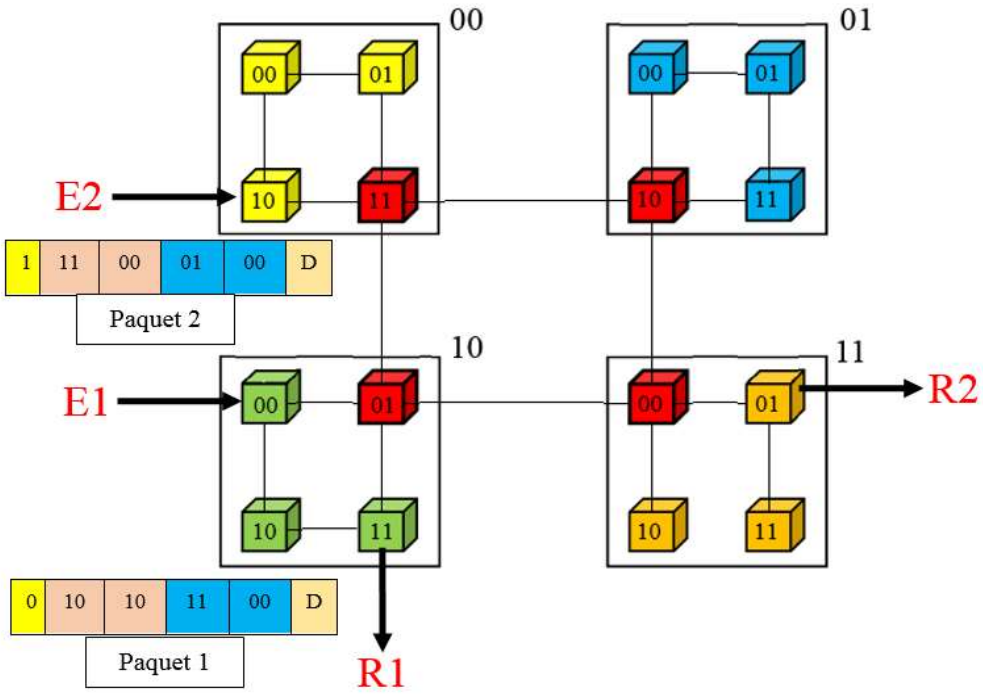


Figure 44 : Exemple de routage intra-cluster et inter-clusters.

La figure 45 présente l'algorithme de routage utilisé dans l'architecture à base de clusters proposé. Dans cet algorithme nous montrons les deux niveaux de routage : intra-cluster et inter-clusters.

3. Modélisation de l'approche proposée.

Les méthodes traditionnelles, de conception des systèmes, consistent à décrire les accélérateurs au niveau RTL en utilisant un langage de description du matériel comme le VHDL ou le Verilog. La conception des SoC (y compris les NoC) en décrivant tous les détails au niveau RTL, prend généralement un temps long. En effet, il est nécessaire de valider la bonne fonctionnalité de chaque sous-ensemble du système et d'assurer le bon fonctionnement de l'association de ces sous-ensembles. Les accélérateurs restent couramment utilisés dans le domaine du traitement du signal intensif, malgré ces difficultés de conception. Cependant, plutôt que d'être directement décrits au niveau RTL, les accélérateurs sont conçus à l'aide d'outils comme MATLAB-Simulink, Xilinx-Vivado, ou encore l'environnement de développement, récemment offert par Xilinx, le SDSoC. Afin d'accélérer la conception des SoCs, des approches de spécification orientées modèles sont alors apparues. La modélisation permet d'abstraire les aspects les plus importants pour les communiquer, les analyser et les valider avant toute implémentation. Parmi ces approches, on trouve l'Ingénierie Dirigée par les Modèles (IDM).

```
If (A/R = 0)
        % Routage local
        If (X_ID = X_DST)
                % Priorité selon « y »
                If (Y_ID > Y_DST)
                        Y_SRC ← Y_ID – 1 ;
                Else
                        Y_SRC ← Y_ID + 1 ;
                Endif
                X_SRC ← X_ID ;
        Else
                % Priorité selon « x »
                If (X_ID > X_DST)
                        X_SRC ← X_ID – 1 ;
                Else
                        X_SRC ← X_ID + 1 ;
                Endif
                Y_SRC ← Y_ID ;
        Endif
Else
        % Routage global
        If (chef = 0)
                % Slave
                If (X_ID = X_DSTChef)
                        % Priorité selon « y »
                        If (Y_ID > Y_DSTChef)
                                Y_SRC ← Y_ID – 1 ;
                        Else
                                Y_SRC ← Y_ID + 1 ;

                        Endif
                        X_SRC ← X_ID ;
                Else
                        % Priorité selon « x »
                        If (X_ID > X_DSTChef)
                                X_SRC ← X_ID – 1 ;
                        Else
                                X_SRC ← X_ID + 1 ;

                        Endif
                        Y_SRC ← Y_ID ;
                Else
                        % (chef = 1) Master
                        If (X_IDC = X_DSTC)
                                % Priorité selon "y"
                                If (Y_IDC > Y_DSTC)
                                        Y_SRCC ← Y_ID – 1 ;
                                Else
                                        Y_SRCC ← Y_ID + 1 ;
                                Endif
                                X_SRCC ← X_IDC ;
                        Else
                                % Priorité selon "x"
                                If (X_IDC > X_DSTC)
                                        X_SRCC ← X_ID – 1 ;
                                Else
                                        X_SRCC ← X_ID + 1 ;
                                Endif
                                Y_SRCC ← Y_IDC ;
                        Endif
                                If (Y_SRCC = Y_DSTC && X_SRCC = X_DSTC)
                                        A/R ← 0;
                                Else
                                        A/R ← 1;
                                Endif
                Endif
Endif
```

Figure 45 : L'algorithme de routage proposé.

Nous pouvons représenter un système dans l'IDM selon plusieurs vues. Philippe Kruchten a proposé un modèle pour décrire un système selon cinq vues différentes (fig.46), appelé le modèle 4+1. Quand vous essayez de regarder l'architecture des systèmes complexes, il est utile de décomposer le tout du système en un ensembles de pièces connexes. Ce modèle est utilisé pour décrire l'architecture des systèmes en fonction de plusieurs vues simultanées.

Le modèle 4+1 se compose des vues suivantes : Development view, Logical view, Physical view, Process view et Use Case view (ou les Scenarios). Les vues sont utilisées pour décrire le système du point de vue des différentes parties prenantes, tels que les utilisateurs, développeurs et gestionnaires de projets. Ces vues sont concurrentes et chacune offre sa propre perspective significative sur l'architecture du système. Chacune de ces vues représente un aspect du système et chacune possède des types particuliers de diagrammes UML qui lui sont associés.

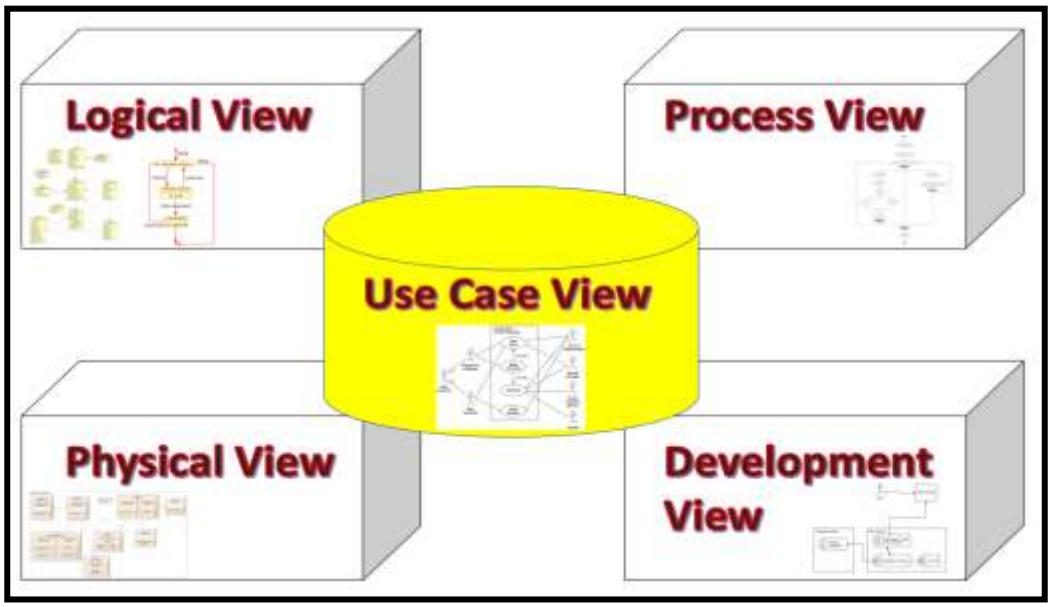


Figure 46 : Le modèle 4+1 de Philippe Kruchten.

La vue logique concerne la fonctionnalité que le système fournit aux utilisateurs. Elle montre les parties qui composent le système, ainsi que leurs interactions. Il représente un ensemble d'abstractions et met l'accent sur les classes et les objets. Les diagrammes UML utilisés pour représenter la vue logique comprennent : le Diagramme de classes, le Diagramme d'états, le Diagramme d'objets, le Diagramme de séquences, le Diagramme de communications et le Diagramme de structure composite.

La vue de processus décrit des processus d'un système, comment ces processus communiquent, et se concentre sur le comportement d'exécution du système. Elle est particulièrement utile lorsqu'un système aura un certain nombre de threads simultanés des processus. Le diagramme UML qui représente la vue de processus est le diagramme d'activité.

La vue physique (ou la vue de Déploiement) représente le système d'un point de vue d'un ingénieur de système. C'est là que nous cartographions les artefacts logiciels sur le matériel qui les héberge. Le diagramme UML utilisé pour représenter cette vue est le diagramme de déploiement.

La vue de développement (ou la vue d'Implémentation) décrit les modules du système, ou les composantes, en incluant des paquets, des sous-systèmes et des bibliothèques de classe. Les diagrammes UML utilisés pour représenter cette vue sont : le diagramme de composant et le diagramme de package.

La vue de cas d'utilisation montre la fonctionnalité du système ; en d'autres termes, ce point de vue illustre ce que le système est censé faire. Il capture les objectifs de l'utilisateur et les scénarios, et offre une perspective extérieure sur le système. Il est utile pour définir et expliquer les structures et les fonctionnalités dans les quatre autres vues. Ainsi, cette vue fournit un guide pour les modèles dans les quatre vues précédentes. Le diagramme UML « cas d'utilisation » fournit la vue de cas d'utilisation.

Le modèle 4+1 présente une façon de décomposer la modélisation d'un système entier en une série de vues. Une autre façon de représenter ces systèmes consiste à les décomposer selon des modèles statiques et dynamiques. Les modèles statiques montrent les caractéristiques structurelles du système ; les modèles dynamiques montrent les caractéristiques comportementales du système. Les modèles de mise en œuvre (ou d'implémentation) montrent les éléments nécessaires au déploiement du système.

3.1. Modélisation des concepts du NoC implémenté en UML.

Pour représenter un système en UML, nous le répartissons en plusieurs vues : statique, comportementale et de déploiement. Les diagrammes statiques fournissent aux développeurs des informations détaillées sur les classes, les types de données, les paramètres et les espaces de noms. Ils sont considérés comme la colonne vertébrale du modèle UML. Cependant, les diagrammes dynamiques, tels que les diagrammes de séquence et de communication, nous aident à définir les objets et leur interaction. La pratique de modélisation agile recommande de développer des modèles en parallèle, en travaillant sur des modèles statiques et dynamiques en même temps.

Dans cette section, nous montrons comment modéliser les concepts de base nécessaires à la construction d'un NoC : - la topologie ; - la structure du réseau implémenté ; - l'algorithme de routage. Pour modéliser la topologie, il va falloir utiliser forcément le paquetage RSM du profil MARTE ; pour la modélisation de la structure de l'approche proposée, nous utilisons le diagramme de structure composite ; pour la modélisation de l'algorithme de routage, nous utilisons le diagramme d'activité. Nous détaillons dans les sections ci-dessous la modélisation des concepts du NoC cités précédemment.

3.1.1. Modélisation de la topologie.

Le paquetage RSM du profile MARTE permet la modélisation des structures répétitives régulières. L'objectif est donc de modéliser la topologie d'un NoC en se basant sur le paquetage RSM.

Les mécanismes fournis par le paquetage RSM du MARTE sont basés sur deux aspects : - la possibilité de décrire la forme (Shape) d'une structure répétée par une forme multidimensionnelle et aussi l'expression de l'ensemble des liens comme un tableau multidimensionnel ; - la modalité suivant laquelle représenter des informations topologiques sur des relations entre des entités. Un exemple d'application de Shape est de prendre une instance de composant ou un port et de spécifier la multiplicité à l'aide d'un vecteur strictement positif qui indique le nombre d'éléments dans chaque dimension du tableau. Ce paquetage, qui est basé sur RSM, utilise les concepts Tiler, Reshape, Inter-repetition pour modéliser les liens qui existent entre les composants (fig.47).

Un Tiler représente un connecteur spécial, utilisé dans un contexte de répétition. Il permet de construire les motifs à partir des informations du tableau d'entrée, ou la façon de ranger les motifs dans un tableau. Ces informations sont : - l'origine du motif de référence ; - la matrice de pavage qui décrit l'espacement régulier des tuiles dans le tableau ; - la matrice d'ajustage qui donne une information sur l'espacement régulier des points d'une tuile. Le Tiler est un connecteur qui permet d'identifier des sous-tableaux (motifs) à l'intérieur d'un tableau défini par une forme (Shape). La tuile est l'assemblage d'un ensemble de points du tableau utilisés pour créer le même motif. L'ensemble de la tuile est construit par des points régulièrement espacés, et les tuiles sont aussi régulièrement espacées.

Le concept Reshape [83] permet la représentation des topologies de liens plus complexes dans lesquelles les éléments d'un tableau multidimensionnel sont redistribués dans un autre tableau. Il est défini comme un connecteur construit par deux Tilers mis bout à bout : - le premier rassemble dans un motif intermédiaire les éléments par motif depuis le tableau d'entrée ; - le deuxième répartit les éléments du motif intermédiaire sous une nouvelle forme dans le tableau de sortie. Cette construction est essentielle pour la modélisation en MARTE. Elle permet l'expression des liens dans les topologies complexes.

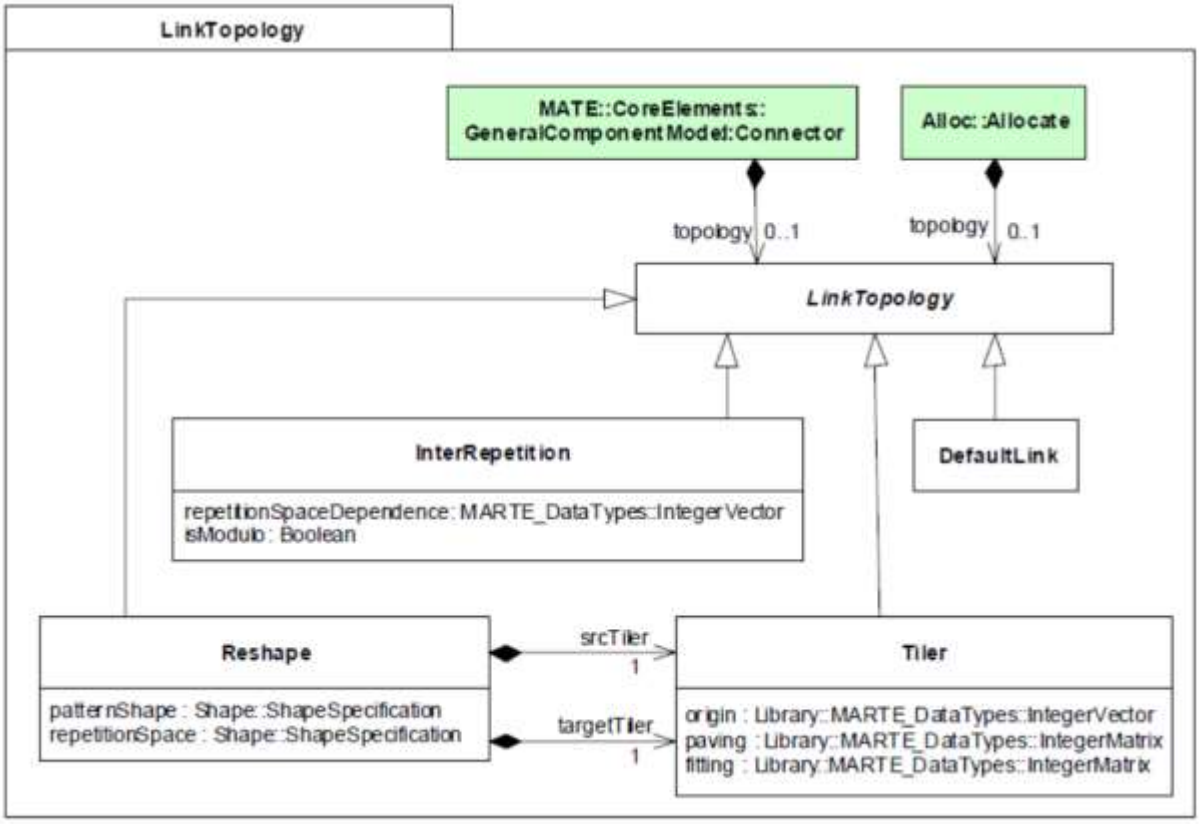


Figure 47 : Le paquetage RSM du profil MARTE.

La dépendance d'inter-répétition connecte un port de sortie d'un composant répété avec un de ses ports d'entrée, sachant que la forme des deux ports connectés doit être semblable. Un connecteur InterRepetition est utilisé pour la modélisation des liens entre les répétitions d'un même composant GASPARD2 dans une structure répétitive. Comme exemple de structures répétitives, on cite les topologies des réseaux sur puce. Nous verrons, dans ce qui suit, un exemple d'utilisation de ce concept pour la modélisation des topologies du NoC Mesh 2D. Comme la topologie proposée est basée sur la structure Mesh 2D, il s'ensuit que ce concept nous sera très utile.

Les difficultés rencontrées lors de la modélisation en utilisant le paquetage RSM sont relatives au savoir-faire, c'est-à-dire qu'à un certain moment nous ne savons pas comment commencer, par quoi commencer et quels sont les concepts, fournis par le RSM, que nous devons utiliser pour modéliser d'une façon compacte la topologie proposée.

En premier temps, il faut définir la topologie en précisant la taille, le degré, le nombre des ports de chaque routeur. Par la suite, préciser le type du routeur suivant le degré. Dans l'étape suivante, effectuer un test sur le nombre des ports afin de spécifier le type du connecteur (Tiler,

Reshape, inter-répétition). Finalement, si tous les routeurs ont le même nombre de ports, alors nous choisirons le connecteur et nous modélisons. Sinon, nous faisons une reconstruction de la topologie en regroupant les routeurs de la même forme et en les classant suivant leurs degrés afin de modéliser d'une façon compacte.

La topologie Mesh 2D, par exemple, est construite par une matrice de N lignes et M colonnes de routeurs interconnectés à travers des liens physiques. Cette topologie est caractérisée par un ensemble de routeurs qui n'ont pas le même degré. La brique de base dans cette topologie est un routeur de quatre ports (Est, Ouest, Sud et Nord) bidirectionnels. La structure de ce réseau est décrite comme étant une répétition de cette unité élémentaire sur deux dimensions « X » et « Y ». La valeur de répétition sur chaque dimension est donnée par la valeur du « tagged value : Shape » du stéréotype « Shaped ». Chaque routeur est connecté à ses voisins situés Est, Ouest, Nord et Sud. Le connecteur "InterRépétition" spécifie les liens entre les routeurs. Dans notre cas, Shape = {8, 8} qui signifie que nous avons huit routeurs dans chaque dimension.

A remarquer aussi que cette topologie présente une dépendance non cyclique entre les liens. Cela se traduit, dans l'expression de la dépendance d'Inter-Répétition, par la sémantique : isModulo = false. Cette valeur indique que les routeurs de bord ne sont pas connectés entre eux comme dans le cas de la topologie Torus. La figure 48 illustre la modélisation d'un exemple d'une topologie Mesh 2D d'une taille 8x8 sous l'environnement GASPARD2.

L'attribut « repetitionSpaceDependence » est un vecteur de conversion sur l'espace du tableau multidimensionnel. Il indique la position d'un routeur voisin dans la direction spécifique. Sa valeur est égale à {0,1} indiquant que, suivant la direction verticale, un routeur de position {i,j} est connecté à un de ses voisins de position{i, j+1}, tandis que le deuxième connecteur possède un vecteur de translation qui est égal à {1,0} indiquant que, suivant la direction horizontale, un routeur de position {i,j} est connecté à un de ses voisins de position {i+1, j}.

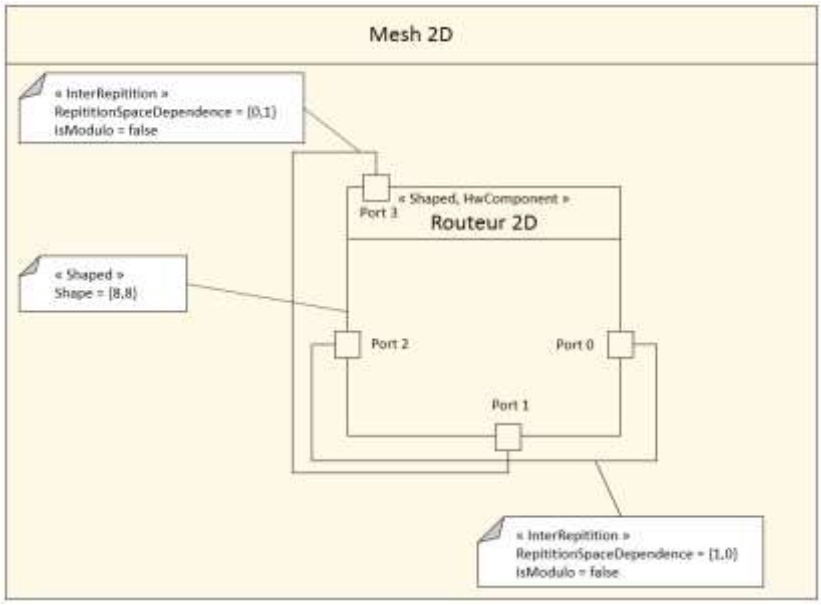


Figure 48 : Modélisation de la topologie Mesh 2D.

La topologie à base de clusters proposée est inspirée des réseaux Ad-Hoc. Le but de cette structure est de résoudre les problèmes et les limites rencontrées au niveau des structures classiques. Afin de résoudre ces limites, des chercheurs ont proposé, en premier temps, les architectures 3D (Mesh 3D) qui offrent de meilleurs résultats au niveau des paramètres mentionnés plus haut, mais il reste toujours des problèmes au niveau de la consommation énorme des ressources logiques, la complexité des algorithmes appliqués et la difficulté au niveau de l'implantation sur des circuits reconfigurables. D'autres chercheurs ont proposé l'architecture 3D modifiée. Mais il reste toujours le problème de la dissipation thermique, l'interférence électrique et la diaphonie. Nous avons proposé l'architecture Mesh 2D à base de clusters afin de résoudre ces problèmes comme détaillé précédemment.

Le bloc principal de construction dans n'importe quel réseau, quel que soit sa topologie, est le routeur. Comme détaillé précédemment, dans l'architecture réalisée, nous avons deux types de routeurs (ordinaire et chef). Chaque routeur a quatre ports d'entrées/sorties : Est, Ouest, Nord et Sud.

La topologie proposée consiste en quatre sous-réseaux de type Mesh 2D. Donc, nous modélisons une topologie Mesh 2D classique sous l'environnement GASPARD2 et relions les différents clusters à travers les chefs. Le connecteur "Inter-Répétition" spécifie les liens entre les routeurs. La valeur booléenne "isModulo" indique que les routeurs de bord sont connectés entre eux, comme dans le cas de Torus, ou pas.

3.1.2. Modélisation de la structure du réseau implémenté.

Les modèles statiques d'un système montrent la structure du système. Ils mettent en évidence les parties qui composent le système. Ces modèles sont utilisés pour définir les noms des classes, les attributs, les méthodes et les package. Parmi les diagrammes UML utilisés dans les diagrammes statiques, nous trouvons : Class Diagram, Package Diagram, Use Case Diagram, Composite Structure Diagram.

Afin de modéliser la structure de l'approche proposée, nous avons utilisé le diagramme de structure composite. La figure 49 montre la structure du réseau implémenté en termes de classes et les relations entre ces classes mais pas les actions. Donc, le diagramme de structure composite présente la façon dont les choses sont mises ensemble dans le système.

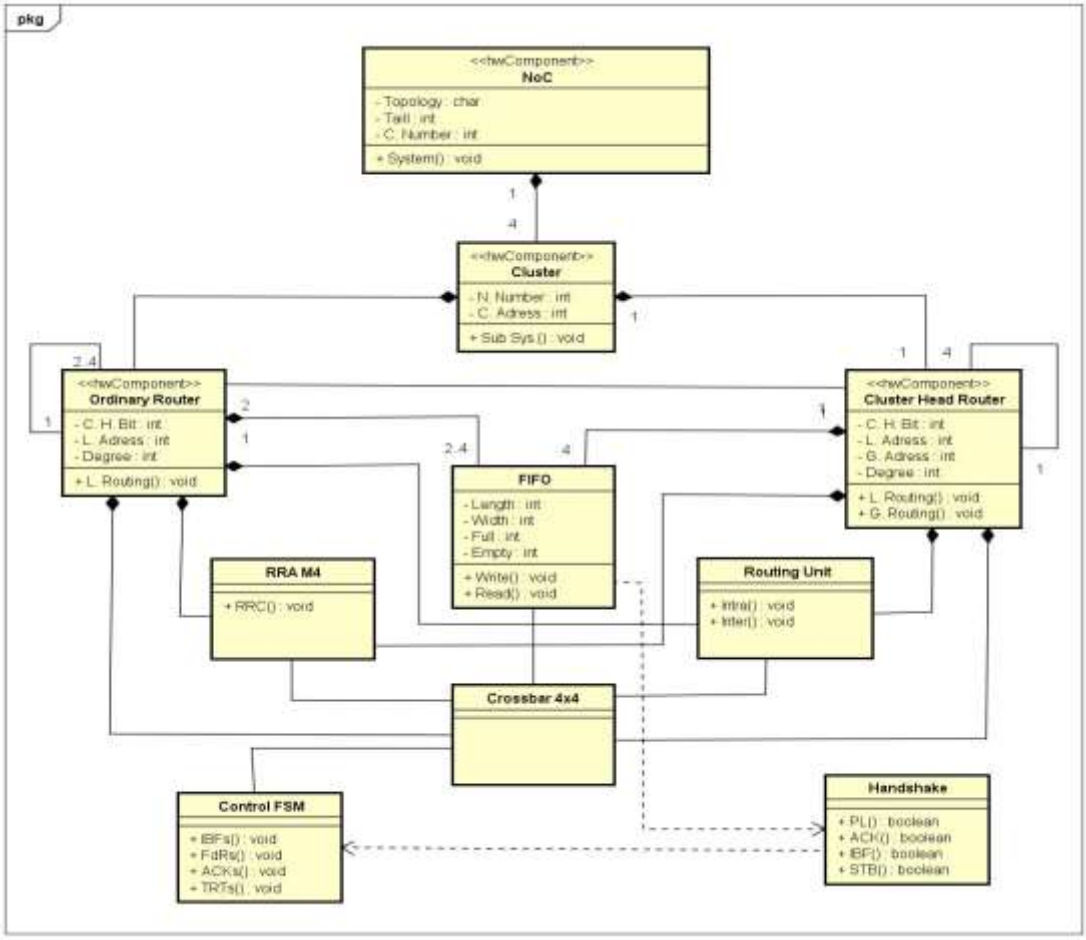


Figure 49 : Structure du NoC implémenté en utilisant Composite Structure Diagram.

Gaspard2 adopte une approche à base de composants basés sur le profil MARTE ; c'est pour cela qu'il utilise le paquetage GCM du profile MARTE comme sa brique de base. Le paquetage component représente le concept principal pour la modélisation du système. Un

composant peut-être de type : - élémentaire, qui ne possède aucune instance de composants ; - composé, qui est constitué d'instances de composants connectées via des ports ; - répétitif, qui contient une instance de composants répétée une ou plusieurs fois.

Un composant structuré dans le paquet GCM peut être défini comme le concept UML classificateur et repose principalement sur les classes structurées d'UML. Ainsi, il peut être utilisé pour la modélisation des deux structures de classes et de composants dans leurs diagrammes respectifs. Un composant structuré peut également contenir plusieurs propriétés telles que : l'Assembly Connectors et l'Interaction Port. L'assembly Connectors a comme rôle de connecter des ports d'un composant structuré à des ports de sous-composants. Le concept Interaction Port définit l'intéraction qui peut exister entre les différents composants à l'aide des connecteurs d'assemblage. Un port d'intéraction peut être classé en deux types principaux : - Flow Port, qui définit l'orientation de la communication si elle est entrée, sortie ou entrée / sortie ; - Message Port, qui est utilisé pour orienter les messages (request/reply) lors de la communication. La figure 50 montre un exemple de composant répétitif (un routeur) à base de composants élémentaires.

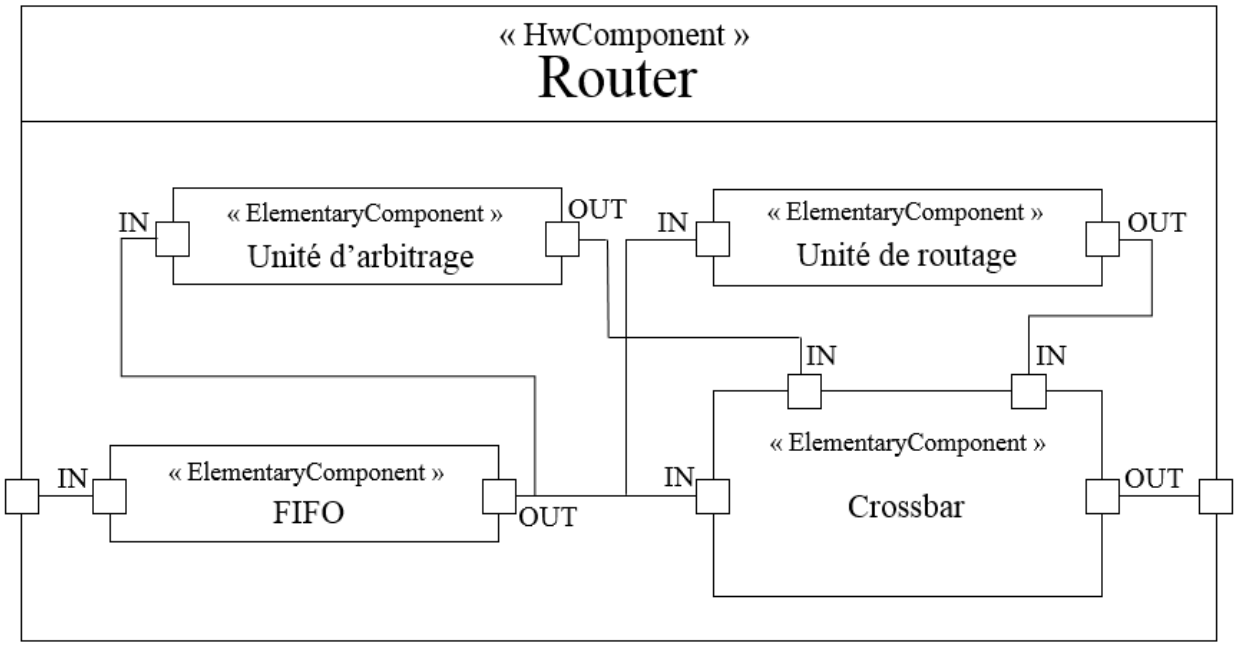


Figure 50 : Modélisation du routeur comme composant composé.

3.1.3. Modélisation du comportement du réseau implémenté.

Les modèles dynamiques d'un système montrent le comportement du système ; par exemple, comment le système se comporte en réponse à des événements externes. Il permet

d'identifier les objets nécessaires et comment ces objets fonctionnent ensemble à travers des méthodes et des messages. Ces modèles sont utilisés pour désigner la logique et le comportement du système. Parmi les diagrammes UML utilisés pour les modèles dynamiques, on trouve : le diagramme de séquence, le diagramme d'activité, le diagramme de communication, le diagramme d'état.

3.1.3.1. Modélisation de l'algorithme de routage.

Dans ce document, l'algorithme de routage est divisé en deux groupes : routage intra-cluster (local) et routage inter-clusters (global) (fig.51). Nous avons modélisé le comportement de cet algorithme en tant que diagramme d'activité.

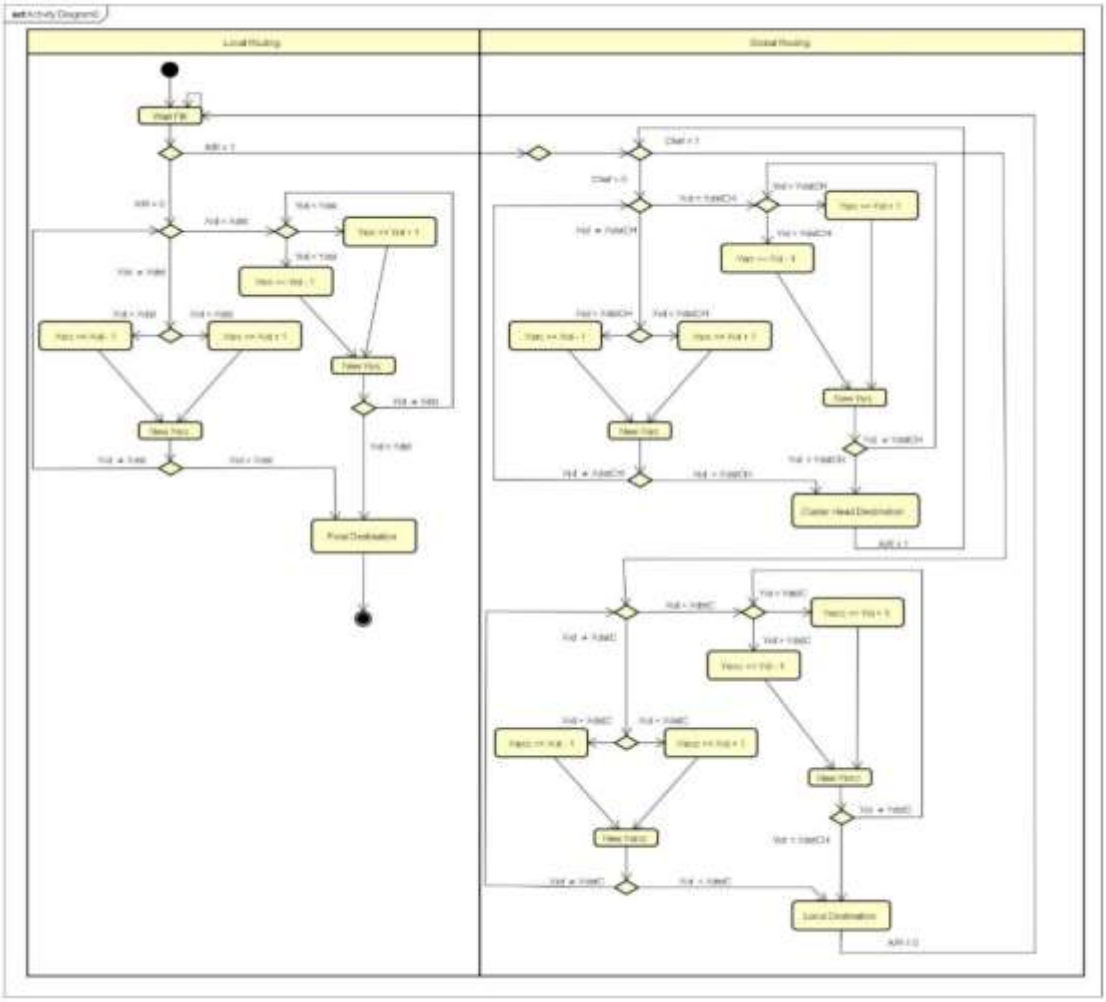


Figure 51 : Modélisation de l'algorithme de routage proposé en utilisant un diagramme d'activité.

3.1.3.2. Modélisation du protocole de communication et du fonctionnement de réseau.

Les éléments d'un système travaillent ensemble pour atteindre les objectifs du système, et un langage de modélisation doit avoir un moyen d'être représenté. Il y a deux types de

diagrammes qu'on peut utiliser pour représenter les communications entre l'ensemble des objets du système : - le diagramme de séquence ; - le diagramme de communication. Dans les figures 52 et 53, nous réaliserons un diagramme de communication et un autre de séquence particulier afin de répertorier tous les messages de contrôle entre un routeur source de type « Ordinary Router » et un routeur destinataire (ou intermédiaire) de type « Cluster Head Router ».

Le diagramme de séquence et le diagramme de communication montrent les interactions entre les objets. Pour cette raison, l'UML se réfère à eux collectivement comme des diagrammes d'interaction.

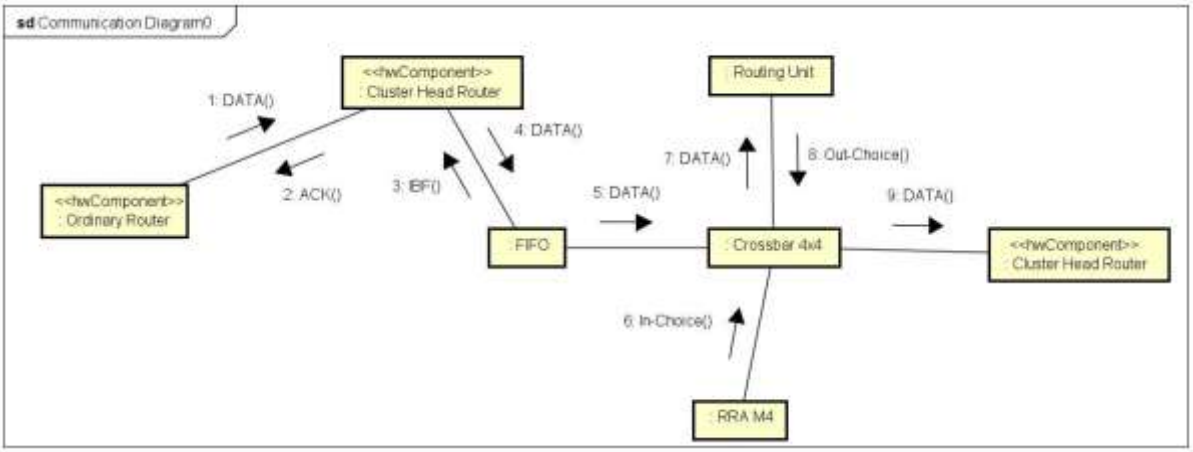


Figure 52 : Messages de contrôle en tant que diagramme de communication.

Dans un système de fonctionnement, cependant, les objets interagissent les uns avec les autres ; ces interactions se produisent au cours du temps. Le diagramme de séquence UML montre la dynamique basée sur le temps des intéractions.

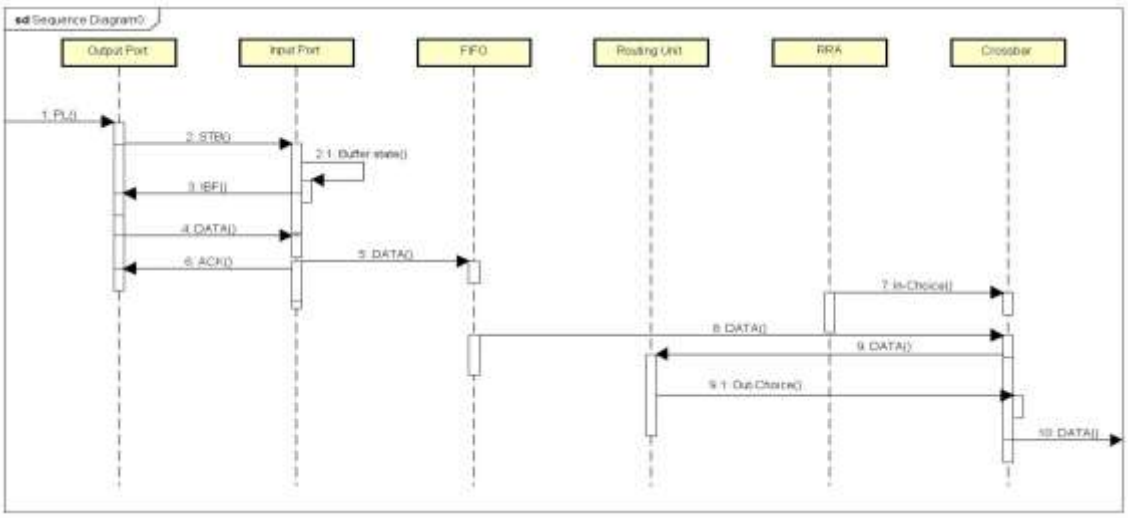


Figure 53 : Messages de contrôle en tant que diagramme de séquence.

La modélisation du comportement du bloc FSM qui gère les messages de contrôles du protocole de communication Handshake se fait à partir d'un diagramme d'état (fig.54). À un moment donné, un objet est dans un état particulier. La figure 54 montre que le module de protocole de communication (Handshake) passe d'un état à un autre.

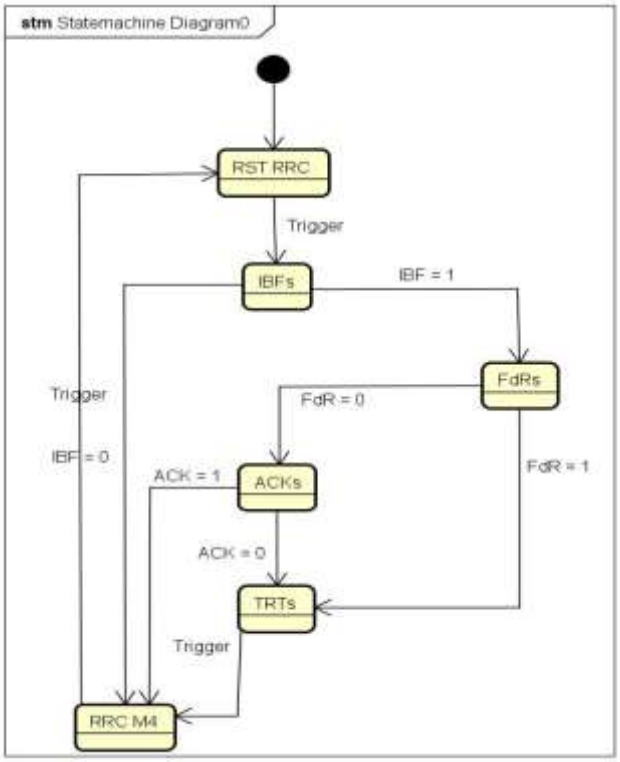


Figure 54 : Modélisation du comportement du module FSM en tant que diagramme d'état.

La figure 55 montre la modélisation du contrôleur FIFO en tant que deux diagrammes d'états. Le premier présente les états plein ou non du FIFO, tandis que le deuxième présente les états vide ou non du FIFO. La combinaison entre les deux diagrammes nous donne les trois états : vide, plein et moyen.

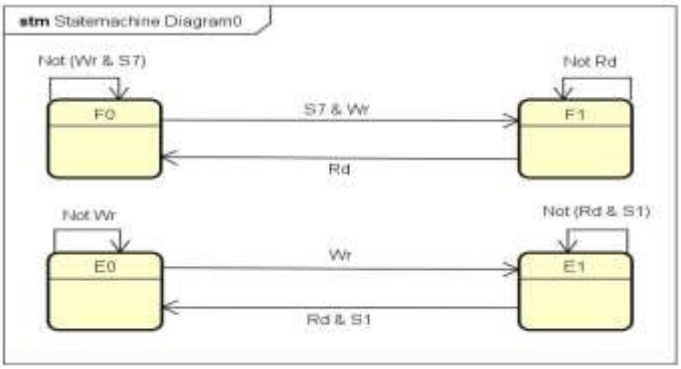


Figure 55 : Modélisation des états du FIFO en tant que diagramme d'état.

3.2. Du modèle vers la génération du code VHDL.

Selon les principes fournis par MDE, la génération de code peut être considérée comme la réécriture d'un modèle sous une forme textuelle. Pour une génération efficace de code, il faut s'assurer que le niveau d'abstraction du modèle d'entrée est proche du texte généré. Ainsi, la génération de code peut être considérée comme une transformation un à un, dans laquelle chaque concept du modèle génère une partie du texte global. La génération de code consiste donc à produire, pour chaque concept disponible, la syntaxe correspondant à ce concept dans le langage ciblé.

La figure 56 montre la vision globale du flux de conception de l'environnement Gaspard2 tout en respectant les principes de l'IDM. Dans chaque chaîne de transformation de Gaspard 2, la première étape est toujours la même. Cette étape consiste en une transformation du méta-modèle « UML + profil MARTE » vers le méta-modèle Deployed. Ce méta-modèle est une composition de plusieurs méta-modèles (application, architecture, association et déploiement). Une chaîne de transformations vise alors un méta-modèle RTL à partir duquel le code VHDL peut être généré.

Les transformations de modèles UML2MARTE permet la conversion du modèle UML en un modèle MARTE Déployé conforme au méta-modèle MARTE Déployé. Par la suite, les transformations de Deployed2RTL convertissent le modèle Deployed en un modèle RTL, qui correspondent à leur propre méta-modèle. Le modèle RTL est considéré comme une abstraction bas niveau ; il fournit des détails relatifs aux accélérateurs matériels et les fonctions de commande qui peuvent être utilisés pour la génération du code final. Enfin, nous générons le code VHDL en utilisant la transformation modèle vers texte (RTL2VHDL).

Habituellement, chaque accélérateur doit être personnalisé séparément sans réutilisation possible. Cela conduit à des délais de production longs et des coûts de conception élevés. En utilisant cette chaîne de transformation, il est possible de produire automatiquement des architectures matérielles qui permettent de résoudre ces deux problèmes et de réduire les erreurs résultant de l'intervention humaine. La génération d'un accélérateur matériel repose sur le partitionnement matériel-logiciel spécifié dans les modèles MARTE de haut niveau.

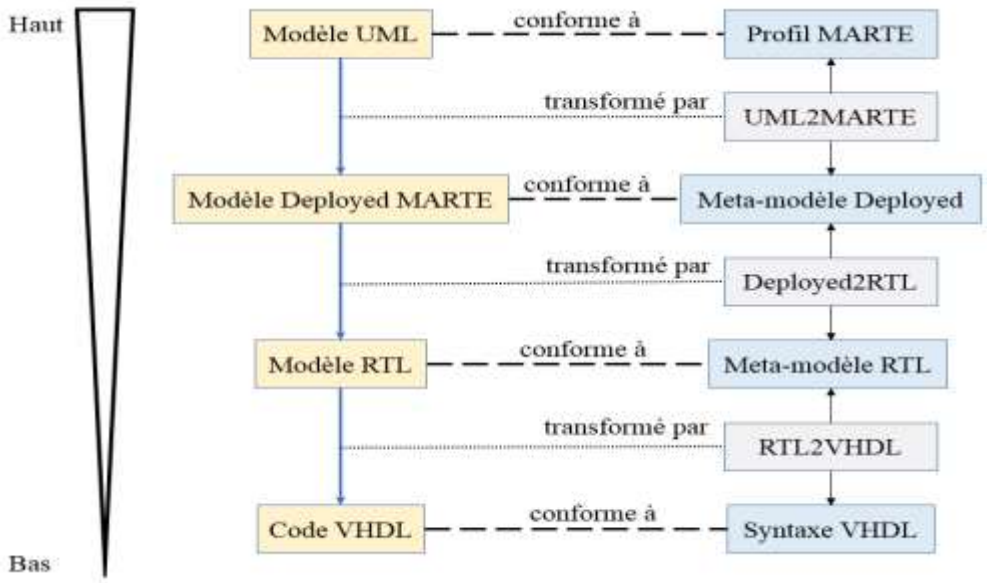


Figure 56 : Flux de conception de l'environnement GASPARD2.

- **Transformation UML2MARTE :**

La transformation UML2MARTE permet de transformer un modèle UML en un modèle MARTE intermédiaire, comme indiqué dans la figure 31. Le modèle UML est conforme au méta-modèle UML et le profil MARTE, tandis que le modèle MARTE respecte le méta-modèle MARTE étendu.

- **Transformation Deployed2RTL :**

Le méta-modèle RTL de Gaspard2 correspond à un niveau intermédiaire entre la modélisation d'une application de Gaspard2 qui est spécifiée en utilisant le profil MARTE ainsi que son déploiement, et la génération automatique de code, permettant l'implémentation finale sur un circuit FPGA.

Un accélérateur dans le méta-modèle RTL est automatiquement généré à partir d'un modèle d'application conforme au méta-modèle Deployed du Gaspard par une transformation de modèles appelée Deployed2RTL. La transformation du modèle Deployed2RTL est en mesure de produire un modèle RTL à partir d'un modèle Deployed.

La transformation de modèles Deployed2RTL est en mesure de produire un modèle RTL à partir d'un modèle Deployed. Comme illustré dans la figure 57, la transformation de Deployed2RTL prend le modèle MARTE en entrée et génère deux modèles de sortie : - un modèle RTL, contenant les concepts liés à l'accélérateur matériel / contrôle ; - un modèle RTL PortType, contenant les types de données et de contrôle présents dans le modèle MARTE.

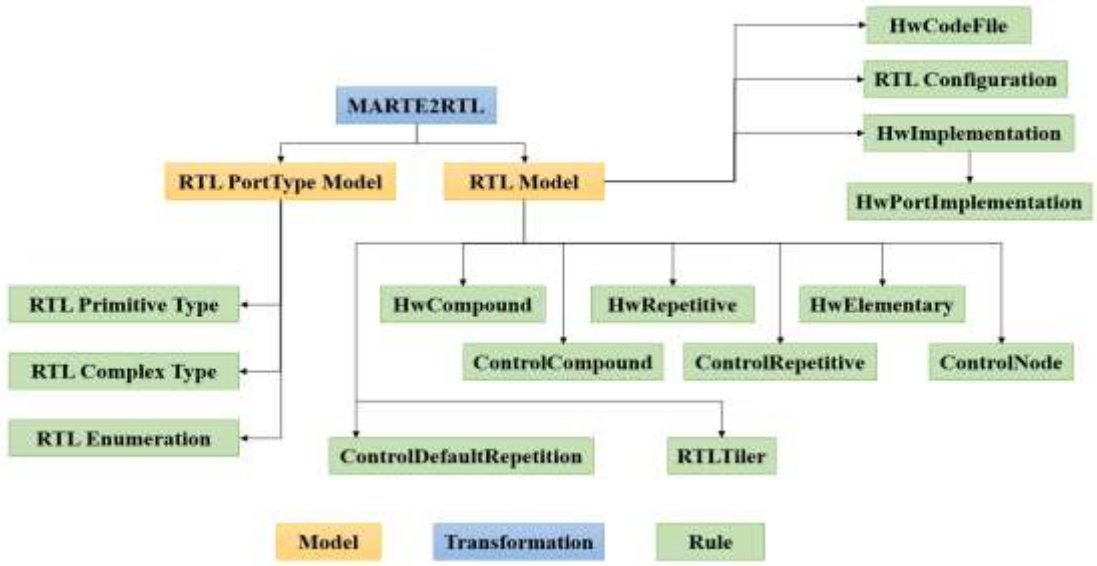


Figure 57 : Vue d'ensemble de la transformation du modèle MARTE2RTL.

Il reste encore que le modèle RTL ne soit pas directement exploitable par les outils de simulation et de synthèse, contrairement au code VHDL. Le méta-modèle RTL, défini dans le paquetage RTL Gaspard2, recueille les concepts nécessaires pour décrire l'architecture matérielle au niveau RTL, ce qui permet l'exécution matérielle de la fonctionnalité. Le méta-modèle RTL est indépendant de toute syntaxe HDL comme le VHDL et le Verilog, mais son niveau de description permet la génération du code VHDL synthétisable.

- **Transformation RTL2VHDL :**

Actuellement, un grand nombre d'outils de transformation de modèle à texte existent dans la littérature. De plus, l'OMG a proposé la norme MOF2Text pour ces types de transformations. Dans la norme, un méta-modèle a été défini qui aide dans le développement de ces transformations par la présence de modèles de génération de code. Comme pour le QVT, les outils respectant cette norme devraient être capables de lire avec succès un modèle avant de l'interpréter dans le code éventuel. Néanmoins, il n'existe pas actuellement un outil unique qui implémente complètement la norme, mais l'idée commune dans chaque outil est la présence des modèles de génération de code. De même, des outils tels que Acceleo, MOFScript et M2T [84] sont également basés sur ces modèles de code.

RTL2VHDL est une transformation modèle vers texte. Elle génère à la fois le code VHDL d'un accélérateur décrit par un modèle RTL et le code correspondant à son placement sur FPGA.

Le code généré marque la fin de l'IDM et le début de l'utilisation des outils classiques comme Xilinx ISE et ModelSim pour la synthèse et la simulation.

L'environnement Gaspard2 supporte les transformations de modèle vers texte en utilisant le JET (Java Emitter Templates). La figure 58 illustre le fonctionnement du JET lors de la transformation de modèle vers un code. Le JET Generator permet de charger le modèle d'entrée et d'appliquer JET Genlets avant d'enregistrer le code généré dans un fichier. JET GENLET est une classe Java avec une méthode GENERATE qui prend en argument le modèle (ou une partie du modèle) et qui retourne une chaîne de caractères. Les JET GENLETS sont générées à partir des JET TEMPLATE, scripts qui définissent les relations entre un concept dans un modèle et le code.

Dans les modèles JET, on peut trouver deux types d'éléments : le texte qui est écrit directement dans le code de sortie requis (dans notre cas le VHDL) et le code Java. Le code Java peut être lui-même de deux natures différentes :

- <% SCRIPT %> : le script est directement reproduit dans le code Java de la classe générée. Ce script permet par exemple d'évaluer des boucles dont la taille dépend du modèle d'entrée.

- <%= EXPRESSION %> : l'expression dépend en général du modèle d'entrée. Il est par exemple possible d'écrire le nom du concept a (attribut name) par le biais de l'expression <%= A.GETNAME() %>.

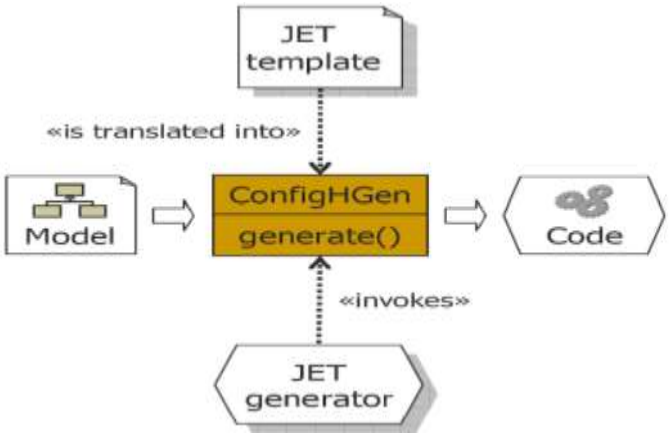


Figure 58 : Flot de conception de JET pour la génération du code à partir d'un modèle en Ecore.

Dans l'environnement Gaspard2, nous n'utilisons pas directement le JET, mais nous pouvons profiter d'une couche supplémentaire qui lie à Ecore, pour simplifier l'utilisation du JET. Cette couche logicielle a été développée au sein de l'équipe DaRT et a été appelée MoCodE

(Model to Code Engine). Cette couche permet également la génération du code de sortie dans plusieurs fichiers séparés, ce qui n'est pas possible avec JET.

4. Résultats expérimentaux.

Nous avons effectué des expériences afin d'évaluer la performance de l'approche proposée et de faire une comparaison avec des réseaux d'une topologie Mesh 2D classique de tailles 4x4, 8x8 et 12x12, largement utilisées dans les NoCs. Nous avons implémenté l'approche proposée en VHDL au niveau RTL (Register Transfer Level) et l'avons synthétisée et simulée en utilisant les outils Xilinx ISE Design Suite 14.5 et ModelSim respectivement. Le réseau a été prototypé dans le circuit VIRTEX 5 XC5VLX110T-3FF1136.

Le routeur est le composant le plus important dans un réseau sur puce. Il est la colonne vertébrale des systèmes à base de NoC. Donc, il doit être conçu avec une efficacité maximale. Comme nous l'avons détaillé dans le chapitre 4, le routeur utilisé est constitué de quatre ports d'entrée/sortie (Est = Port 0, Sud = Port 1, Ouest = Port 2 et Nord = Port 3). Chaque routeur est défini par une adresse d'identification, un bit chef et une adresse d'identification de cluster au niveau du routeur chef. On a aussi les deux signaux CLK pour l'horloge et RST pour le remise à zéro (Reset). Les éléments constitutifs du routeur utilisé sont au nombre de quatre : - le DATA PATH ou le chemin de données ; - l'unité FSM pour gérer les signaux utilisés dans le protocole de communication Handshake ; - l'unité de routage pour calculer le chemin à emprunter par un paquet ; - l'unité d'arbitrage (Round-Robin) pour gérer la priorité entre les différentes demandes du transfert.

Les FIFOs (First In, First Out) sont essentiellement des buffers de mémorisation utilisés pour stocker temporairement des données jusqu'à ce qu'un autre processus soit prêt à lire. Comme leur nom l'indique, le premier octet écrit dans un FIFO sera le premier à apparaître sur la sortie. Ce FIFO a un port de données d'entrée, une horloge, une remise à zéro actif bas, une permission de lecture, une permission d'écriture, un port de données de sortie, des signaux de contrôle vides et pleins. Le modèle de boîte noire VHDL généré avec des entrées et des sorties est détaillé dans la figure 59. Les signaux utilisés dans la FIFO implémentée sont : - DI pour les données entrantes ; - DO pour les données sortantes ; - WR le signal qui permet l'écriture au niveau du FIFO ; - RD le signal qui permet la lecture à partir du FIFO ; - EMPTY et FULL pour définir l'état, vide ou plein, du buffer ; - RST pour le Reset ; - CLK pour l'horloge.

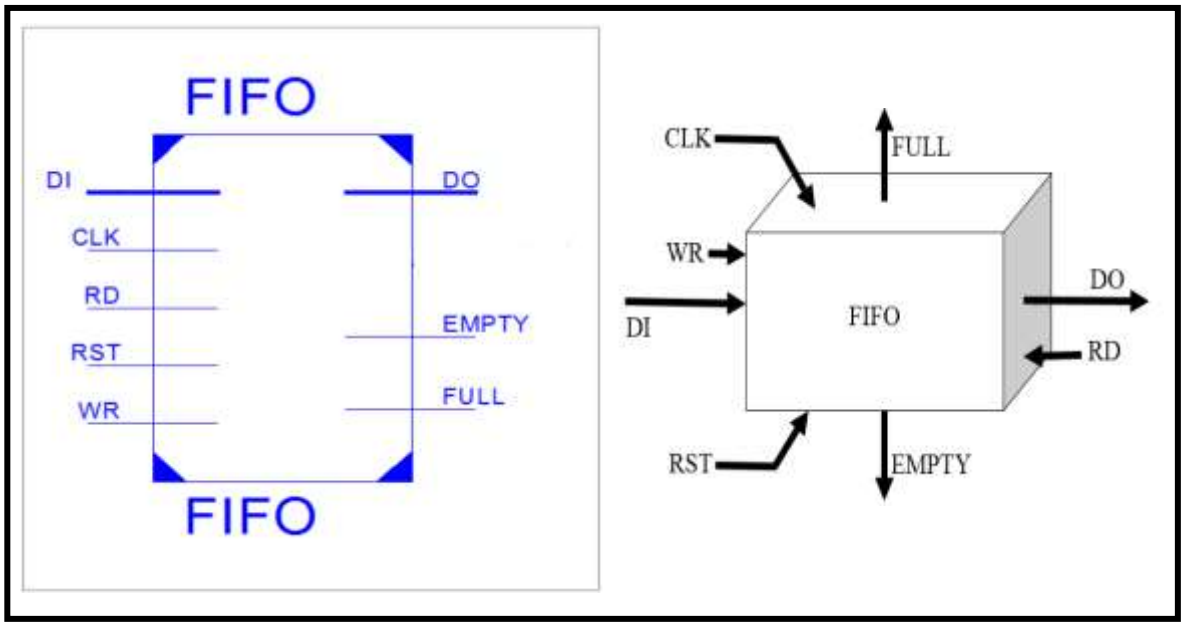


Figure 59 : Signaux du buffer utilisé.

L'objectif principal d'un routeur dans un réseau est de fournir un transfert correct des messages entre les routeurs et/ou les IPs. Le routeur a une unité logique de contrôle et de routage et quatre ports bidirectionnels : Est, Ouest, Nord et Sud. Chaque port a un buffer d'entrée pour le stockage temporaire des paquets. La figure 55 présente la simulation du routeur utilisé ; nous considérons un routage du paquet entrant à partir du port de direction Est, qui doit être acheminé vers le port de direction Nord.

La figure 60 montre un aperçu de la simulation d'un routeur ordinaire. A l'instant t = 0 ns, débute une décision de routage d'un paquet de données en provenance du port Est vers le port Nord. Le transfert des données ne se fait qu'à l'étape de traitement (TRT) en plus de ce que le buffer soit vide (IBF = 0).

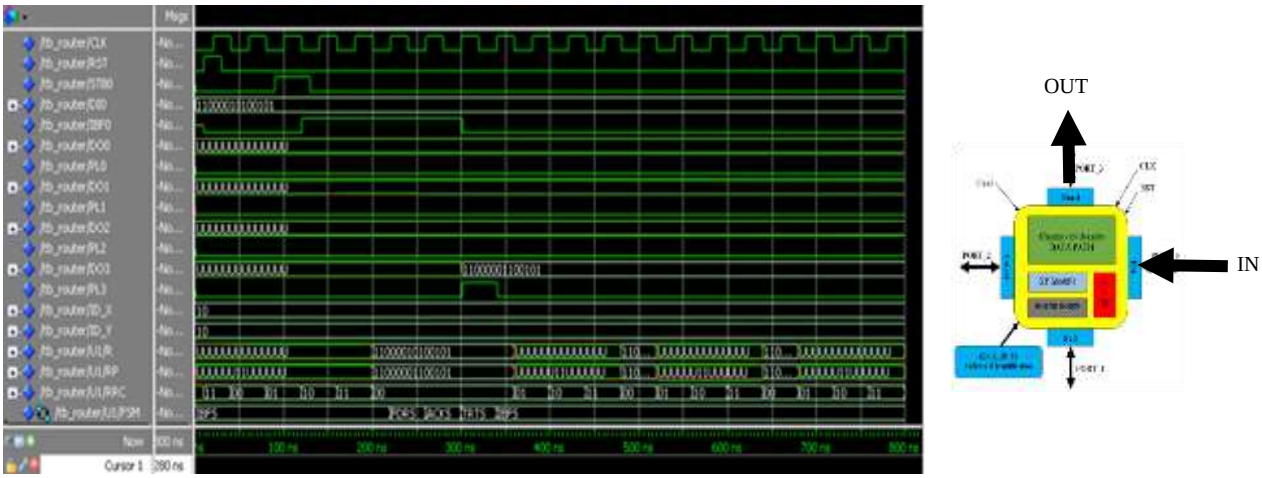


Figure 60 : Simulation du routeur utilisé.

La figure 61 montre que le nombre des slices LUTs utilisés dans un routeur ordinaire est diminué de 14 %, le nombre de slices registres est réduit de 20 %, le nombre d'entrées/sorties est réduit de 9 % et la fréquence de fonctionnement est plus grande de 5 % par rapport à un routeur chef.

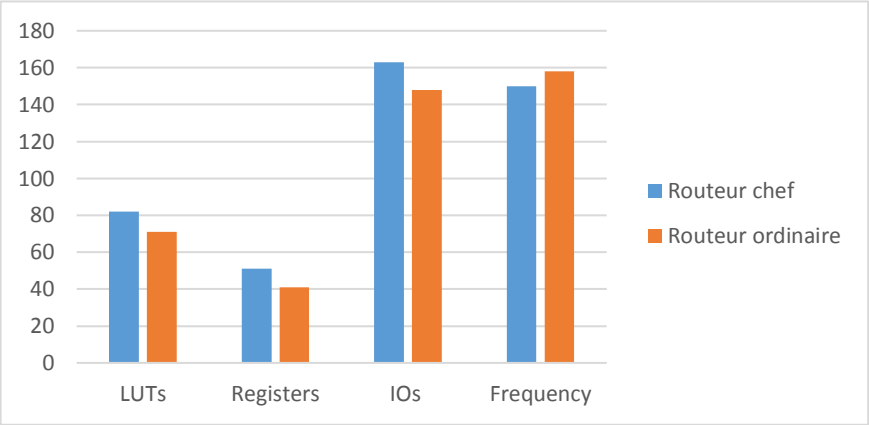


Figure 61 : Comparaison de la consommation en ressources et en fréquence de fonctionnement maximal d'un routeur ordinaire par rapport à un routeur chef.

Dans ce que suit, nous présenterons une étude comparative entre deux types de topologies. La première est de type Mesh 2D classique et l'autre de type Mesh 2D à base de clusters (l'approche proposée). L'étude est résumée dans les tableaux 6, 7 et 8. Nous avons fait la comparaison entre ces deux architectures en utilisant trois tailles différentes pour chaque type dont le nombre de nœuds constitutifs est le même. Nous avons adopté les tailles suivantes : - 4x4 contre 2x2x4 (quatre cluster d'une taille 2x2) où le nombre de nœuds est égal à 16 ; - 8x8 contre 4x4x4 où le nombre de nœuds est égal à 64 ; - 12x12 où le nombre de nœuds est égal à 144.

Comme le montre le tableau 6, les ressources supplémentaires sont d'environ 232 registres, 44 IOs et 140 LUTs entre la topologie Mesh 2D classique d'une taille 4x4 et la topologie proposée à base de clusters (la même architecture du routeur est utilisée pour les deux topologies en appliquant respectivement l'algorithme de routage XY statique et l'algorithme de routage XY à deux niveaux proposé). Le NoC 4x4 classique utilise 16 % plus de registres, 17 % plus d'IOs et 20 % plus de slices LUTs par rapport au NoC 2x2x4 à base de clusters. La fréquence de fonctionnement dans l'approche proposée (2x2x4) est supérieure de 7 % à celle de la topologie (4x4) classique. Cette réduction au niveau des ressources logiques utilisées du FPGA est due à la réduction du nombre de liens connectant les différents clusters à une connexion au lieu de deux.

Tableau 6 : Comparaison entre l'approche proposée 2x2x4 et la topologie 4x4 classique.

	4x4	2x2x4	Disponible	%
LUTs	691	551	69120	20
Registers	1456	1224	69120	16
IOs	258	214	440	17
Frequency	143,696	153,139		7

Comme le montre le tableau 7, le réseau Mesh 2D classique d'une taille 8x8 utilise 18 % plus de registres, 11 % plus d'IOs et 21 % plus de slices LUTs par rapport au NoC 4x4x4 à base de clusters. La fréquence de fonctionnement dans l'approche proposée (4x4x4) est supérieure de 6 % à celle de la topologie (8x8) classique. Cette réduction au niveau de des ressources logiques utilisées du FPGA est due à la réduction du nombre de liens connectant les différents clusters à une connexion au lieu de quatre.

Tableau 7 : Comparaison entre l'approche proposée 4x4x4 et la topologie 8x8 classique.

	8x8	4x4x4	Disponible	%
LUTs	3201	2522	69120	21
Registers	8544	7056	69120	18
IOs	642	571	440	11
Frequency	137,88	144,399		6

Le tableau 8 montre que le NoC d'une topologie Mesh 2D classique d'une taille 12x12 utilisant 20 % plus de registres, 8 % plus d'IOs et 22 % plus de slices LUTs par rapport au NoC 4x4x4 à base de clusters. La fréquence de fonctionnement dans l'approche proposée (6x6x4) est supérieure de 6 % à celle de la topologie (12x12) classique. Cette réduction au niveau des ressources logiques utilisées du FPGA est due à la réduction du nombre de liens connectant les différents clusters à un lien au lieu de six.

Tableau 8 : comparaison entre l'approche proposée 6x6x4 et une topologie 12x12.

	12x12	6x6x4	Disponible	%
LUTs	8346	6501	69120	22
Registers	23932	19023	69120	20
IOs	1154	1061	440	8
Frequency	134,675	143.303		6

Nous avons appliqué cette approche sur l'un des NoCs les plus connues depuis 2004 [47], le réseau Hermes. Par l'application de l'approche proposée sur le NoC Hermes, nous avons obtenu de meilleurs résultats au niveau de la consommation des ressources logiques, la fréquence de fonctionnement et la consommation de l'énergie. Le temps passé à livrer des paquets croît linéairement avec le nombre de sauts. Pour les tailles de tampon de six ou plusieurs positions, le temps reste presque constant, la croissance de 10 cycles d'horloge par hop pour le réseau Hermes et de 8 cycles d'horloge pour le réseau proposé. Cela se produit parce que chaque commutateur dépense des cycles d'horloge pour exécuter les algorithmes d'arbitrage et de routage. A partir de l'équation suivante : *Temps total sans collision par paquets = (Hc + (Nf-1) * 2) * NP*, où :

- *Hc* : Nombre de cycles d'horloge pour exécuter les algorithmes d'arbitrage et de routage, 10 dans le NoC Hermes et 8 pour l'approche proposée ;
- *Nf* : Nombre de flits, 20 dans cette expérience ; Le facteur -1 est utilisé parce que le premier flit (en-tête) est traité en *Hc* ;
- **2* : Chaque flit passe deux cycles d'horloge à transmettre au commutateur suivant
- *Np* : Nombre de paquets, 20, 40 et 60 dans cette expérience.

Les résultats sont présentés dans la figure 62.

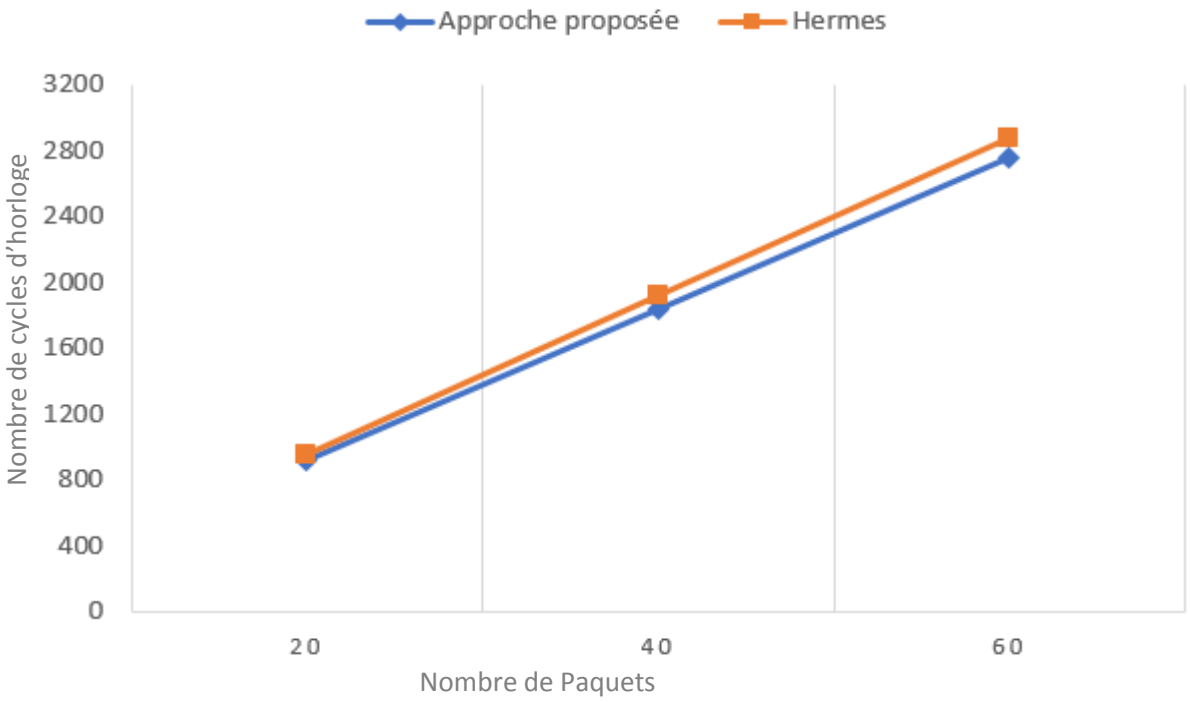


Figure 62 : Temps total, en cycles d'horloge, pour livrer différents nombres de paquets d'une longueur de 20 flits ; pour NoC Hermes et le NoC proposé.

La figure 63 montre que l'augmentation de la taille du réseau à base de clusters (16, 64 et 144 nœuds) provoque une réduction continue au niveau de la consommation des registres et des LUTs et une augmentation au niveau du nombre d'IOs, mais il reste toujours inférieur à celui de Mesh 2D classique. Donc, plus la taille du réseau est grande, plus la réduction au niveau du nombre de LUTs et de registres du FPGA utilisés augmente.

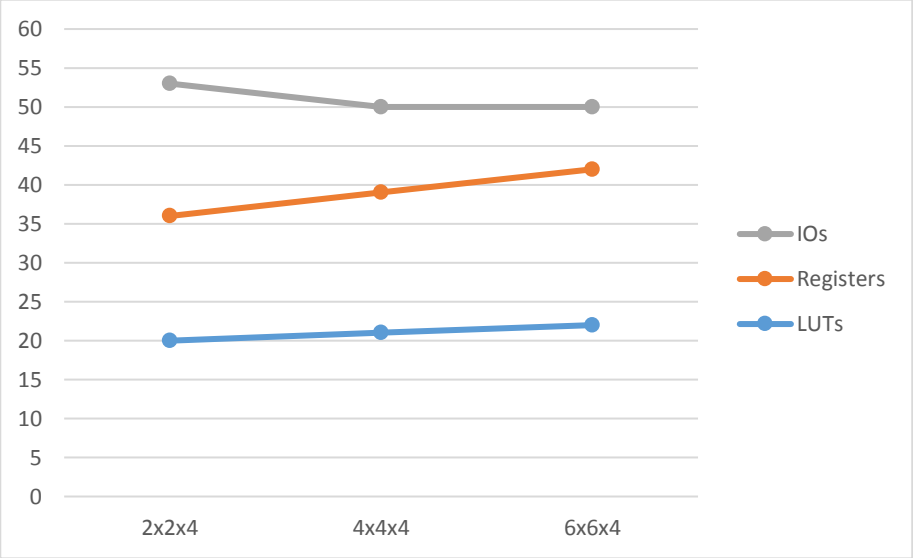


Figure 63 : Consommation des ressources logiques à réduire avec l'augmentation de la taille du réseau.

La figure 64 montre une étude comparative entre l'architecture à base de clusters proposée, la Mesh 2D et la Mesh 3D. nous remarquons, à partir de la figure 58, que l'architecture Mesh 3D offre de meilleurs résultats au niveau de la fréquence de fonctionnement, la latence et la consommation d'énergie. La fréquence de la structure 3D est plus grande de 41 % par rapport à celle de 2D à base de clusters et de 44 % par rapport à celle de 2D classique, ce qui nous donne effectivement une fréquence meilleure par rapport à celle de Mesh 2D classique ou encore à l'approche proposée à base de clusters. Mais comme on le voit dans la figure 62, le nombre des ressources logiques utilisées dans la structure 3D est énorme ; elles sont 65 % plus nombreuse au niveau de nombre de LUTs, 45 % au niveau de nombre de registres et 71 % au niveau de nombre des IOs. En plus de ces inconvénients, l'implémentation des structures 3D sur les circuits reconfigurables est très difficile ainsi que la complexité des algorithmes appliqués.

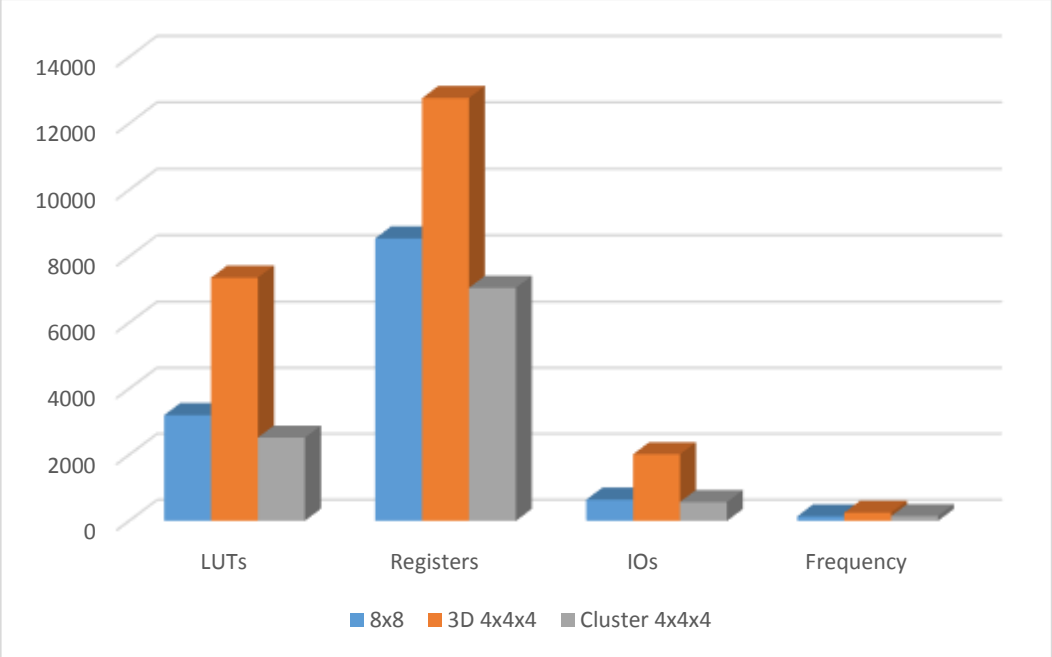


Figure 64 : Comparaison de la consommation en ressources logiques et en fréquence de fonctionnement entre l'approche proposée 4x4x4 et Mesh 2D 8x8 et Mesh 3D 4x4x4 classique.

La figure 65 montre une étude comparative entre l'architecture proposée à base de clusters, Mesh 3D et Mesh 3D modifiée proposé par A. Benhaoues et al. [85]. Comme dans n'importe quel approche, l'approche proposée dans ce travaille offre des avantages et des inconvénients par rapport aux autres approches mentionné précédemment. Nous remarquons que l'architecture Mesh 3D modifiée offre meilleurs résultats au niveau de la fréquence de fonctionnement. La fréquence de la structure 3D modifiée est plus grande de 6 % par rapport au 3D classique et de 45 % par rapport à celle de 2D de l'approche proposée ; ceci nous donne effectivement une fréquence meilleure de celle de Mesh 3D classique ou encore de l'approche proposée à base de clusters.

Comme le montre la figure 65, le nombre de ressources logiques utilisées dans la structure 3D classique, par rapport à celle de 3D modifiée, est plus grand de 28 % au niveau du nombre de LUTs, de 27% au niveau du nombre de registres et de 32 % au niveau d'IOs. Mais quand nous le comparons à celui de l'approche proposée, nous trouvons que le nombre de ressources logiques utilisées est plus grand de 52 % au niveau du nombre de LUTs, de 24 % au niveau du nombre de registres et de 59 % au niveau du nombre des IOs.

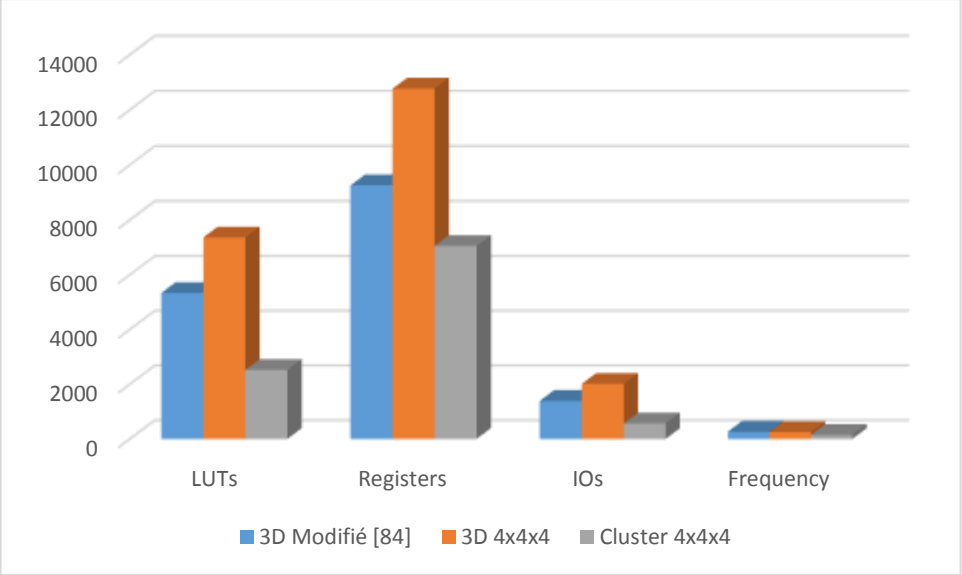


Figure 65 : Comparaison de la consommation en ressources logiques et en fréquence de fonctionnement entre l'approche proposée 4x4x4 et Mesh 3D 4x4x4 et Mesh 3D [85] 4x4x4.

En utilisant une technique de placement de modules à base du taux de communications entre chaque deux nœuds, et de regroupement des nœuds les plus communicants dans le même cluster, nous obtenons obligatoirement une réduction au niveau du nombre de routeurs à traverser par un paquet pour atteindre sa destination, qui provoque une réduction de la latence moyenne et de la consommation d'énergie.

5. Conclusion.

Afin d'améliorer les performances du réseau en termes de paramètres mentionnés précédemment dans la première partie de ce chapitre, nous avons proposé un algorithme de routage XY modifié et une technique de placement pour les structures NoC à base d'une topologie Mesh 2D, qui sont combinés avec une technique de clustering appropriée. Nous avons aussi détaillé l'architecture de l'approche proposée. L'objectif de cette approche est de réduire la latence moyenne et les coûts de ressources logiques en respectant les contraintes de la bande passante et la consommation d'énergie de communication du réseau.

Dans la deuxième partie de ce chapitre, nous avons présenté en détails la modélisation des concepts des NoCs. La modélisation est réalisée en deux parties : la modélisation structurelle, dans laquelle nous modélisons la structure des concepts du réseau réalisé ; et la modélisation comportementale, dans laquelle nous représentons le comportement du réseau réalisé. Nous avons aussi détaillé la chaine de transformation utilisée et la génération du code VHDL.

La dernière partie est consacrée à la présentation des résultats expérimentaux obtenus. Les résultats montrent, par rapport à la topologie Mesh 2D classique qui utilise un routage XY statique, que des améliorations de performances significatives peuvent être réalisées en utilisant l'approche proposée avec un coût supplémentaire acceptable dans le routeur cluster-head.

D'après les résultats obtenus, dans la troisième partie de ce chapitre, l'approche proposée nous offre plusieurs avantages tels que :

- éviter le problème de la dissipation thermique, rencontré au niveau des structure 3D, par l'utilisation d'une topologie plane (2D) ;

- implanter facilement sur les circuits reconfigurables par rapport aux architectures 3D ;

- minimiser la latence moyenne en utilisant une technique de placement basée sur la charge de communication entre chaque deux nœuds ;

- réduire la consommation de l'énergie par ce que les modules les plus communicants sont dans le même cluster et cela nous donne une minimisation du nombre de sauts ;

- minimiser la consommation des ressources logiques sur la plateforme utilisée qui est, dans notre cas, le FPGA.

Comme travail futur, nous proposons d'appliquer l'approche proposée sur diverses applications multimédia et la développer davantage afin d'améliorer nos résultats.

Conclusion Générale

De nos jours, la conception des SoCs est de plus en plus utilisée dans les systèmes à haute performance. Avec le taux d'intégration croissant, leur complexité deviendra de plus en plus élevée. Afin d'augmenter la performance de ces systèmes, le choix de la structure d'interconnexion est primordial. Nous avons donné un aperçu sur quelques architectures d'interconnexions et leurs limites. Le bus partagé est l'architecture d'interconnexion la plus utilisée dans les SoCs actuels. Mais cette structure finira par atteindre ses limites face aux besoins des futurs systèmes qui intègrent un très grand nombre d'éléments. Ainsi, il est très difficile d'ajouter de nouveaux éléments communicants à un bus car d'une part, la bande passante allouée à chaque élément diminue lorsque le nombre d'éléments augmente et, d'autre part, le contrôle du bus se complexifie, limitant alors l'extensibilité de cette structure de communication. Ces limitations rendent les bus incompatibles avec les futures générations de circuits intégrant un très grand nombre de ressources de traitement. Le paradigme de réseau sur puce (NoC) a été proposé face aux défis rencontrés au niveau des interconnexions classiques.

Dans un premier temps, nous nous sommes intéressés au concept des réseaux sur puce ainsi qu'aux notions et aux terminologies qui leur sont associées. Par la suite, nous avons détaillé les mécanismes internes et les éléments constitutifs de l'architecture d'un NoC. En effet, partant du contexte dans lequel les NoCs ont émergé, nous avons analysé les différentes caractéristiques principales de ces réseaux, les topologies utilisées, les algorithmes de routage associés, leurs protocoles de communication et les mécanismes de contrôle de flux.

Ensuite, nous avons présenté les limites des méthodologies de conception des systèmes sur puce. En face de ces limites, de nouvelles méthodologies et technologies sont continuellement proposées pour répondre aux exigences de conception des SoCs. La conception à base de réutilisation des composants et la conception à base de l'élévation de niveaux d'abstraction, en particulier, sont fortement encouragés. Dans le cadre de cette thèse, pour aborder les SoCs reconfigurables, nous avons adopté une approche de conception basée sur la méthodologie à base de l'approche IDM. Nous avons présenté comment modéliser la topologie à base de cluster, par l'utilisation du paquetage RSM, et la modélisation d'un algorithme de routage XY à deux niveaux en se basant sur le diagramme d'activité du langage UML. Cette modélisation a été réalisée à l'aide de l'environnement Gaspard2 qui est un outil de développement de systèmes embarqués développé par l'équipe DaRT du laboratoire LIFL.

Dans le présent travail, nous avons proposé une nouvelle architecture pour gérer les communications dans un NoC d'une topologie Mesh 2D modifié grâce à la combinaison entre les techniques de routage, de placement et de clustering. L'objectif de cette approche est de réduire la latence moyenne et surtout les coûts de ressources logiques, en respectant les contraintes de la bande passante et la consommation d'énergie. Par comparaison avec une topologie Mesh 2D classique, nous avons montré que l'approche proposée permet de réduire le temps de détection et de rapports des échecs et des pannes des liens et/ou des routeurs, tout en optimisant l'utilisation des ressources. Les résultats expérimentaux montrent, par rapport à la topologie Mesh 2D classique qui utilise un routage XY statique, que des améliorations de performances significatives peuvent être réalisées en utilisant l'approche proposée avec un coût supplémentaire acceptable dans le routeur cluster-head.

Dans nos futurs travaux à court terme, nous projetons, de manière générale, de faire des simulations et des évaluations analytiques supplémentaires afin de perfectionner et de comparer nos approches proposées dans cette thèse avec d'autres approches existantes dans la littérature.

Premièrement, nous voudrions utiliser un algorithme de routage dynamique afin d'améliorer davantage les performances des architectures personnalisées par l'ajout de liens. Un algorithme de routage dynamique est actuellement en cours de développement.

Deuxièmement, nous ajouterons des algorithmes de détection et correction des erreurs. Nous sommes entrain de développer des algorithmes de détection d'erreur à base du CRC et de correction à base de Hamming.

Publication de l'auteur.

Revues :

A. Bouguettaya, S. Toumi, M. T. Kimour et A. Boudjedra, « A new 2D Mesh routing approach for networks on chip », Revue Synthèse, numéro : 32, Avril 2016, pages 98-105, ISSN : 1111-4924.

Conférences internationales :

A. Bouguettaya, M. T. Kimour et S. Toumi, « Dynamic Routing in FPGA », International Conference on Embedded Systems in Telecommunications and Instrumentation Annaba-Algeria, « ICESTI'12 » November 05-07, 2012.

A. Bouguettaya, S. Toumi, A. Boudjedra et M. T. Kimour, « A new fault tolerance routing approach for FPGA architecture », 2nd International Conference on Signal, Image, Vision and their Applications Guelma-Algeria, « SIVA'13 » November 18-20, 2013.

A. Bouguettaya, M. T. Kimour, S. Toumi, A. Boudjedra et Y. Diaf, « Routage dans les réseaux sur puce », 2ème Conférence Internationale sur l'Electronique, l'Electrotechnique et l'Automatique Oran-Algeria, « CIEEA'13 », 2013.

A. Bouguettaya, A. Boudjedra, S. Toumi, M. T. Kimour et S. El-Akrmi, « Synthèse et simulation d'un module UART basé sur VHDL », International Conference on Embedded Systems in Telecommunications and Instrumentation Annaba-Algeria, « ICESTI'14 » October 27-29, 2014.

Listes des autres conférences :

S. Boudjema, K. Saouchi et **A. Bouguettaya**, « Sine Generator Based of Parallel CORDIC », 1st International Conference on Nanoelectronics, Communications and Renewable Energy Jijel-Algeria, « ICNCRE'13 » September 22-23, 2013.

S. Derouiche, B. Djedou, **A. Bouguettaya** et A. Rafraf « Implementation of denoising algorithm to ensure the intelligibility of the signal at the hearing impaired with cochlear implant », 1st International Conference on Nanoelectronics, Communications and Renewable Energy Jijel-Algeria, « ICNCRE'13 » September 22-23, 2013.

A. Boudjedra, S. Toumi, M. Boutalbi, **A. Bouguettaya** et S. El-Akrmi, « Routage dans les réseaux sur puce », 2ème Conférence Internationale sur l'Electronique, l'Electrotechnique et l'Automatique Oran-Algeria, « CIEEA'13 », 2013.

S. El-Akrmi, S. Toumi, A. Hassainia, **A. Bouguettaya**, A. Boudjedra, H. Triki et A. El-Akrmi, « Spectral properties of linearly chirped apodized fiber Bragg gratings for dispersion cancellation in optical communications systems », International Conference on Embedded Systems in Telecommunications and Instrumentation Annaba-Algeria, « ICESTI'14 » October 27-29, 2014.

Références.

[1] G. E. Moore, 1998, "Cramming More Components onto Integrated Circuits," in proc. of the IEEE, Volume 86 , Issue 1, pages 82-85.

[2] Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS), 2001. http://public.itrs.net.

[3] D. G. Bailey, 2011, "Design for Embedded Image Processing on FPGAs," John Wiley & Sons (Asia) Pte Ltd, ISBN: 978-0-470-82849-6.

[4] S. Heath, 2003, "Embedded Systems Design," Elsevier Science Publisher, second edition, ISBN 0-7506-5546-1.

[5] K. Messaoudi, 2012, "Traitement des Signaux et Images en Temps Réel « Implantation de H.264 sur MPSoC »," Thèse de doctorat, Université d'Annaba.

[6] C. Zeferino, M. Kreutz, L. Carro, and A. Susin, 2002, "A study on communication issues for systems-on-chip, " in Proc. 15th Symposium on Integrated Circuits and Systems Design, pp. 121-126.

[7] J. Delorme, 2007, "Méthodologie de modélisation et d'exploration d'architecture de réseaux sur puce appliquée aux telecommunications, " Thèse de doctorat, Institut national des sciences appliquées de Rennes.

[8] William J. Dally et Brian Towles, 2001, "Route packets, not wires : on-chip inteconnectoin networks," In Proceedings of the 38th conference on Design automation, (New York, NY, USA), pages 684–689.

[9] ARM limited, 1999, "AMBA Specification", www.arm.com.

[10] Sonics Inc. SMART Inteconnect Solutions. http//www.sonicsinc.com.

[11] International Business Machines Corporation, 1999, "The CoreConnect™ Bus Architecture," http://www-01.ibm.com.

[12] P. Wodey, G. Camarroque, F. Baray, R. Hersemeule et J. P. Cousin, 2003, "LOTOS code generation for model checking of STBus based SoC: the STBus interconnection," In Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, Pages 204 – 213.

[13] Q. Zhou, Y. Song, D. Zhang et G. M. Du, 2011, "A design of multi-core system based on Avalon bus," International Conference on Computer Science and Network Technology, Volume 3, pages 1456 – 1459.

[14] E. Salminen, 2010, “On design and comparison of On-Chip Networks,” Thèse de l’Université de Technologie de Tampere.

[15] H. Moussa, 2009, “Architectures de réseaux sur puce pour décodeurs canal multiprocesseurs,” Thèse de doctorat, l’école nationale supérieure des télécommunications de Bretagne.

[16] P. Mondal, P. K. Biswal et S. Banerjee, 2016, “FPGA based accelerated 3D affine transform for real-time image processing applications,” Computers & Electrical Engineering, Volume 49, Pages 69–83

[17] C. González, S. Sánchez, A. Paz, J. Resano, D. Mozos et A. Plaza, 2013, “Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing,” Integration, the VLSI Journal, Volume 46, Issue 2, Pages 89–103.

[18] A. Guellal, C. Larbes, D. Bendib, L. Hassaine et A. Malek, 2015, “FPGA based on-line Artificial Neural Network Selective Harmonic Elimination PWM technique,” International Journal of Electrical Power & Energy Systems, Volume 68, Pages 33–43.

[19] C. Bernardeschi , L. Cassano , M. G.C.A. Cimino et A. Domenici, , 2013, “GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs,” Journal of Systems Architecture, Volume 59, Issue 10, Part D, Pages 1243–1254.

[20] Altera. Stratix II devices, http://www.altera.com/products/devices/stratix2/st2-index.jsp, 2005.

[21] http://www.xilinx.com.

[22] I. Kuon et J. Rose, 2006, “Measuring the gap between FPGAs and ASICs,” International Symposium on Field Programmable Gate Arrays, Monterey, California, pp. 21–30.

[23] O. Gonçalves, 2013, “Conception sur mesure d’un FPGA durci aux radiations à base de mémoires magnétiques,” Thèse de doctorat, Université de Grenoble.

[24] http://www.itrs.net/Links/2010ITRS/2010Update/ToPost/ERD_ERM_2010FINALReport MemoryAssessment_ITRS.pdf.

[25] X. Jiang, W. Wolf, J. Hankel et S. Charkdha, 2005, “A Methodology for design, modeling and analysis for networks-on-Chip,” in Proc. of IEEE International Symposium on Circuits and Systems, pages 1778-1781.

[26] M. Palesi et M. Daneshtalab, 2014, “Routing Algorithms in Networks-on-Chip,” Springer Edition ISBN: 978-1-4614-8273-4.

[27] A. Leroy, 2006, “Optimizing the on-chip communication architecture of low power systems on chip in deep sub-micron technology,” Thèse de doctorat, Université Libre de Bruxelles.

[28] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, 2005, “Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC,” in Proc. of the Symposium on Integrated Circuits and Systems Design, pages 178-183.

[29] H. Moussa, 2009, “Architectures de reseaux sur puce pour decodeurs canal multiprocesseurs,” thèse de doctorat, l'école nationale supérieure des télécommunications de Bretagne.

[30] P. P. Pande, C Grecu, A Ivanov et R Saleh, 2005, “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” IEEE Trans Computer.

[31] W. J. Dally et B. Towles, 2004, “Principles and Practices of Interconnection Networks,” Morgan Kaufmann Edition.

[32] W. Dally and B. Towles, 2001, “Route packets, not wires : on-chip interconnection networks,” in Proc. Design Automation Conference, pages 684-689.

[33] V. Rantala, T. Lehtonen, J. Plosila, 2006, “Network on Chip Routing Algorithms,” TUCS Technical Report, No 779.

[34] M. Ali, M. Welzl, S. Hellebrand, 2005, “A Dynamic Routing Mechanism for Network on Chip,” 23rd NORCHIP Conference.

[35] P. T. Wolkotte, G. Smit, K. G. Rauwerda et T. L. Smit, 2005, “An energy-efficient reconfigurable circuit switched network-on-chip,” In Proceedings of the 19th IEEE international parallel and distributed processing symposium.

[36] N. E. Jerger et L. S. Peh, 2009, “Network On chip,” Morgan & Claypool Edition, ISBN: 9781598295849.

[37] A. Andriahantenaina, H. Charlery, A. Greiner et al., 2003, “SPIN: a scalable, packet switched, on-chip micro–network,” In: Design automation and test in Europe conference and exhibition (DATE), pp 70–73.

[38] M. A. Abd El Ghany, M. A. El-Moursy and M. Ismail, 2009, “High throughput architecture for high performance NoC,” IEEE International Symposium on Circuits and Systems, pp 2241-2244.

[39] T. Bjerregaard, 2005, “The MANGO clockless network-on-chip: Concepts and implementation,” Thèse de doctorat, Informatics and Mathematical Modelling, Technical University of Denmark.

[40] K. Goossens , J. Dielissen and A. Radulescu , 2005, “Athereal network on chip: Concepts, architectures and implementations ,” IEEE Design and Test of Computers , Vol. 22 , No. 5 , pages 414 – 421.

[41] M. O. Gharan et G. N. Khan, 2014, “Packet-based Adaptive Virtual Channel Configuration for NoC Systems,” In proc. of the International Workshop on the Design and Performance of Network on Chip, Volume 34, Pages 552–558.

[42] http://www.arteris.com/flex_noc.php

[43] http://www.silistix.com

[44] http://www.inocs.com

[45] K. Goossens, J. Dielissen et A. Radulescu, 2005, “Athereal network on chip: Concepts, architectures and implementations,” IEEE Design and Test of Computers , Vol. 22 , No. 5, pages 414–421.

[46] M. Millberg, E. Nilsson, R. Thid et A. Jantsch, 2004, “Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network-on-chip, ” in Proceedings of IEEE Design, Automation and Testing in Europe Conference, pp. 890 – 895.

[47] F. Dubois, J. Cano, M. Coppola, J. Flich et F. Petrot, 2011, “Spidergon STNoC design flow,” in the Fifth IEEE/ACM International Symposium on Networks on Chip NoCS, Pages 267-268.

[48] F. Moraes, N. Calazans, A. Mello, L. Moller et L. Ost, 2004, “HERMES: an infrastructure for low area overhead packet-switching networks on chip”, Integration, the VLSI Journal, Volume 38, Issue 1, Pages 69-93.

[49] L. Ost, A. Mello, J. Palma, F. Moraes et N. Calazans, 2005, “MAIA—A framework for networks on chip generation and verifi cation,” in Proceedings of the IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 49 – 52.

[50] J. Bézivin, 2005, “On the unification power of models,” Software and Systems Modeling, Volume 4, Issue 2 , pages 171-188.

[51] X. Blanc et I. Mounier, 2006, “UML 2 pour les développeurs cours avec exercices corrigés,” Edition Eyrolles, ISBN : 978-2-212-12029-5.

[52] F. Jouault et I. Kurtev, 2005, “Transforming models with ATL,” In Proceedings of the Model Transformations in PracticeWorkshop, Montego Bay, Jamaica, Pages 128-138.

[53] F. Boutekkouk, 2010, “Aide à la Conception des Systèmes Multiprocesseurs Mono-puce à partir de Spécification de haut niveau,” Thèse de doctorat, Université de Constantine.

[54] Object Management Group Inc., 2007, Uml 2 infrastructure (final adopted specifcation). http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.

[55] P. Green, M. Edwards et S. Essa, 2001, “UML for System-Level Design,” In Forum on Design Languages, Proceedings of FDL 2001, Lyon, France.

[56] G. Martin, L. Lavagno et J. Louis–Guerin, 2001, “Embedded UML: a merger of real-time UML and co-design,” In Proceedings of CODES 2001, Copenhagen, pages 23-28.

[57] R. Chen, M. Sgroi, L. Lavagno, G. Martin, A. Sangiovanni-Vincentelli et J. Rabaey, 2003, “UML AND PLATFORM-BASED DESIGN,” In "UML for Real", Edited by B. Selic, L. Lavagno, G. Martin, pages 107-126, Kluwer Academic Publishers.

[58] Object Management Group Inc., 2006, Final adopted omg sysml specification. http://www.omg.org/cgibin/doc?ptc/06-0504.

[59] E. Riccobene, P. Scandura, A. Rosti et S. Bocchino, 2005, “A SOC Design Methodology Involving a UML2.0 Profile for SystemC,” In Proceedings of the Design, Automation and Test in Europe Conference end Exhibition.

[60] E. Riccobene, P. Scandurra, A. Rosti et S. Bocchino, 2006, “Amodel-driven design environment for embedded systems,” In Proceedings of the 43rd annual Design Automation Conference, pages 915–918.

[61] OMG, 2006, UML profile for System on Chip (SoC), Version 1.0.1.

[62] OMG, 2005, UML profile for Schedulability, Performance and Time (SPT), Version 1.1.

[63] OMG. Modeling and analysis of real-time and embedded systems (MARTE). http://www.omgmarte.org/.

[64] I. R. Quadri, P. Boulet, S. Meftali, and J.-L. Dekeyser, 2008, “Using an mde approach for modeling of interconnection networks,” In The International Symposium on Parallel Architectures, Algorithms and Networks Conference.

[65] I. R. Quadri, S. Meftali, and J.-L. Dekeyser, 2010, “Dynamic Reconfigurable Network-on-Chip Design: Innovations for Computational Processing and Communication,” chapter From MARTE to Reconfigurable NoCs: A model driven design methodology.

[66] Object Management Group, 2007, UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE), version 1.0Beta1, ptc/07-08-04. http://www.omg.org/cgi-bin/doc?ptc/2007-08-04.

[67] P. Boulet, 2008, “Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing,” Technical Report 6467, INRIA, France. http://hal.inria.fr/inria-00261178/en.

[68] A. Gamatié, S. Le Beux, E. Piel, A. Etien, R. B. Atitallah, P. Marquet et J.-L. Dekeyser, 2008, “A model driven design framework for high performance embedded systems,” Research Report RR-6614, INRIA. http://hal.inria.fr/inria-00311115/en.

[69] DaRT team, 2009, Graphical Array Specification for Parallel and Distributed Computing (GASPARD2). http://www.gaspard2.org/.

[70] D. Gajski et al., 1991, “High-Level Synthesis : Introduction to Chip and System Design,” Kluwer Academic Publishers.

[71] D. Gajski et R. H. Kuhn, 1983, “New VLSI tools,” IEEE Computer, pages 11-14.

[72] M. Elhajji, 2012, “Co-Design de l’application H264 et implantation sur un NoC-GALS,” Thèse de doctorat, Université de Monastir, Tunisie.

[73] I. R. Quadri, S. Meftali et J.-L. Dekeyser, 2009, “High level modeling of dynamic reconfigurable FPGA,” Int. J. Reconfig. Comput., Pages 1-15.

[74] S. I. Han, S. I. Chae, L. Brisolara, L. Carro, K. Popovici, X. Guerin, A. A. Jerraya, K. Huang, L. Li et X. Yan, 2009, “Simulink based heterogeneous multiprocessor SoC design flow for mixed hardware/software refinement and simulation,” Integr. VLSI J. Volume 42,Issue 2, pages 227-245.

[75] R. Ben Atitallah, 2008, “Modèles et simulation des systèmes sur puce multiprocesseurs Estimation des performances et de la consommation d’énergie,” Thèse de l’université de Lille.

[76] GaspardLib. GaspardLib: An open platform for modelling and simulation of multi-processors system on chip, 2009. https://www.GaspardLib.fr/trac/dev/wiki.

[77] M. Elhaji, P. Boulet, A. Zitouni, R. Tourki, J. L. Dekeyser et S. Meftaly, 2011, “Modeling Networks-on-Chip at System Level with the MARTE UML profile,” In Proceedings of the Model Based Engineering for Embedded Systems Design (M-BED2011).

[78] Abbasi and M. F. Younis, 2007, “A Survey on Clustering Algorithms for Wireless Sensor Networks,” In Computer Communications, Volume 30, Issues 14–15 pages 2826–2841.

[79] D. Wei and H. A. Chan, 2006, “Clustering Ad Hoc Networks: Schemes and Classifications,” In Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, pages 920 - 926.

[80] J. Y. Yu and P. H. J. Chong, 2005, "A Survey of Clustering Schemes for Mobile Ad Hoc Networks," In IEEE Communications Surveys and Tutorials, Volume 7, Issue 1, pages 32 - 48.

[81] H. Wu et A. A. Abouzeid, 2004, "Cluster-based routing overhead in networks with unreliable nodes," In proc. of the Wireless Communications and Networking Conference, volume 4, pages 2557 – 2562.

[82] B. Al-Sharaa, 2013, "The Extended Clustering Ad Hoc Routing Protocol (ECRP)," International Journal of Computer Networks & Communications (IJCNC), Volume 5, Numéro 3.

[83] R. Agerwal and M. Matwani, 2009, "Survey of Clustering Algorithms for MANET," In International Journal on Computer Science and Engineering, Volume 1, Issue 2, pages 98-104.

[84] C. Glitia, 2009, "Optimisation des applications de traitements systématique intensives sur system on chip," thèse de doctorat à l'université des sciences et technologies de Lille.

[85] I. R. Quadri, 2011, "MARTE based model driven design methodology for targeting dynamically recon_gurable FPGA based SoCs," Thèse de l'Université des Sciences et Technologies de Lille.

[86] A. Benhahoues, E. Bourennane, S. Toumi, C. Tanougast, M. Hichem et K. Messaoudi, 2014, "Implementation of Universal Digital Architecture using 3D-NoC for Mobile Terminal," Conference: IEEE CoDIT'14, 2nd International Conference on Control, Decision and Information Technologies, Metz, France.