REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT D'ELECTRONIQUE

Mémoire de Magister

Présenté par

Melle LANI Fatiha

Modélisation des Spécifications d'un NoC sous SDL

Jury d'examen

Président	**Mr M. HADDADI**	**Professeur**	**ENP**
Rapporteur	**Mr R.SADOUN**	**Chargé de Cours**	**ENP**
Examinateurs			
	Mme L. HAMAMI	**Maître de conférences**	**ENP**
	Mr H. BOUSBIA-SALAH	**Docteur**	**ENP**
	Mr O.STIHI	**Chargé de Cours**	**ENP**

2007

مُلخّص

يَتمَحورُ الهَدفُ مِن العَمل المقدَّم في أطرُوحَةِ الماجِستِير هَذِه، حَولَ إدْمَاج نُظُم الشَّبكَاتِ المعْلُومَاتِية فِي رُقَاقاتٍ الِكترُونِية (مَا يُصطلحُ عَليهِ فِي التَّراجِم الإنجْلِيزِية باسْم "NoC" : *Network on Chip*)، وهَذَا بالاعتِمادِ طبْعًا عَلى مخْتَلفِ المفاهِيم وَ الآلِياتِ المعْمُول بهَا فِي عَالمِ الشَّبكَاتِ المعْلومَاتية، والتِي قُمنَا بتَحلِيلِهَا والعَمل عَلى أقْلَمَتِهَا لِتسْتَجِيبَ وَ مُتطَلَّبَاتِ إنشَاء SoCs.

لِتَحقِيقِ الهَدفِ المنْشُود، يتَوجَّبُ المُرورُ بمرْحَلتيْن أسَاسِيتَين: وَضعُ النَّمُوذج ثمَّ المصَادَقةِ عَليْه. مِنْ خِلاَلِ عَمَلِنَا هَذَا، تطرَّقنَا بالتَّحِديد، إلى أولى المرْحلتين وَالتِي تهْدفُ إلى إعْطَاءِ نمُوذجٍ للعُقَدِ دَاخِلِ NoCs. لِهَذَا، قُمنَا بِتقْدِيم اِقتِرَاحٍ لِكَيْفِية الإنشَاء، تعتَمدُ أسَاسًا عَلى تِقنية Top Down وَالتِي بدَورِهَا ترتَكزُ عَلى مجمُوعَةٍ مِنَ القوانِين وَالاصْطِلاَحَات، لاَ تُؤَمِّنُ فقطْ تمَرِيرَ المعْطَيات الدَاخِلةِ والخَارجَةِ مِنْ وَ إلى العُقْدَة، بَلْ وتسْمَحُ أيْضًا بسِتْر وَ تورية تَفَاصِيلِ التَّنفِيذ لأيِّ عَمَلِيةٍ ضَرُورِيةٍ تَتمُّ فِي المسْتوَيَاتِ الدُّنيَا. وَ بالاسْتنَادِ إلى هَذَا العَمَل، يمُكِنُنَا التَّأكِيدُ بالقول أنَّ تطْوِيرَ وبنَاءَ قاعِدةِ اِتِّصَالٍ وَ تَواصُلٍ خَاصَّة بأنظِمَة SoCs قد أضْحَى ممُكِنًا تحْقِيقُه مُستَقبَلاً.

الكَلمَات المفتَاحية: NoC، طُرقُ وَ كَيفيات الإنشَاء، SDL، تِقنِيَة المسْتَوَيَاتِ المتَعدِّدَة، النَّمُوذج OSI.

RÉSUMÉ

L'objectif du travail qui nous a été proposé dans ce mémoire de Magister porte sur l'intégration des systèmes de réseaux informatiques sur des puces électroniques (appelés dans la littérature Anglo-Saxonne "NoC" pour *Network on Chip*) et ce, en se basant, bien évidement, sur les différents concepts et mécanismes connus dans le monde des réseaux informatiques, et que nous les avons analysé et adapté pour répondre aux exigences et aux diverses contraintes de la conception des SoCs.

Afin de remplir cet objectif, nous devons passer par deux phases essentielles: la modélisation et la validation. Nous nous somme intéressés plus précisément par la première étape qui vise la définition d'un model de nœuds pour les NoCs. Pour cela, nous avons proposé une méthodologie de conception basée sur l'approche Top Down reposant sur une pile protocolaire, assurant non seulement la circulation de données entrant-sortant du nœud, mais aussi, permettant le masquage et l'abstraction des détails d'exécution de toute opération requise s'effectue en bas niveaux. Et grâce au présent travail, on se permet de dire que l'implémentation future d'une plateforme de communication propre aux SoCs est alors devenue envisageable.

MOTS CLÉS: NoC, Méthodologie et approche de conception, SDL, Approche en couches, modèle OSI.

ABSTRACT

The objective of the work presented in this Magister dissertation, concerns the integration of Network systems on Chips (NoC). The work was carried out based on various concepts and mechanisms well known in the world of computer networks, which we analyzed and adapted to meet the requirements and various constraints in the design of SoCs.

In order to carry out this work, we must pass through two essential steps: modeling and validation. In this study we concentrated our efforts more on the first step, in which we have tried to definite a model of nodes for the NoCs. For that, we have proposed a design methodology, based on the Top Down approach and relying on a pile protocol. This will not only ensure in and outgoing data circulation at the node, but will also allow the masking and abstraction of all execution details of any operation carried out at low levels. As a result of this work, we can assert that in the future, the implementation of a "SoCs" communication platform seems possible.

KEY WORDS: NoC, Approach and methodology of design, SDL, Multi layer approach, OSI Model.

SOMMAIRE

LISTE DES FIGURES

LISTE DES TABLEAUX

Introduction Générale

Les systèmes intégrés constituent de nos jours un défi majeur dans l'industrie électronique et de télécommunication, que ce soit dans le large public ou dans les secteurs spécialisés. Ces systèmes ont permis la réduction du poids, taille et consommation électrique des équipements, et leur utilisation survole des sommets jamais atteints auparavant. L'accroissement des applications faisant appel aux systèmes intégrés, incite les développeurs à chercher des solutions qui puissent rehausser leurs performances, pour répondre à des critères de marché de plus en plus exigeant.

Les densités actuelles d'intégration des circuits intégrés permettent l'assemblage de nombreux cœurs sur la même puce. Le problème d'interconnexion se fait de plus en plus sentir, à tel point que les structures de communications classiques utilisées aujourd'hui (bus et point à point) ne suffisent plus.

Ces interconnexions physiques actuelles telles que les bus sont des facteurs limitant les performances des SoCs (longueur des interconnexions, bande passante, consommation d'énergie). Les technologies sur silicium doivent s'adapter à de nouvelles contraintes comme la synchronisation complète d'un système sur puce par une ou plusieurs horloges. Le paradigme de synchronisation probablement utilisé tendra vers un concept Globalement Asynchrone Localement Synchrone (ou GALS) utilisant plusieurs sources d'horloges. Les composants initialiseront donc le transfert des données automatiquement en fonction de leurs besoins.

Les perturbations électriques dues aux couplages capacitifs (« crosstalk »), les interférences électromagnétiques et les charges d'inductions peuvent être des erreurs de données. La transmission de données numériques sur des fils sera donc assujettie à une probabilité d'erreur. Il est donc indispensable de proposer dans une nouvelle architecture d'interconnexion.

Les réseaux d'interconnexion sont étudiés depuis plusieurs années. C'est le cas des réseaux informatiques et de téléphonie. Cependant depuis une dizaine d'années, nous avons vu une évolution rapide de la technologie d'interconnexion des systèmes sur puce. Et ce après avoir proposé de remplacer les bus par des réseaux sur puce avec des interconnexions à base de routeurs. On parle alors de nouveau paradigme réseau sur puce ou NOC (Network on Chip) susceptible de proposer des solutions, efficaces aux problèmes d'intégrations complexes des systèmes sur puce.

Ces architectures d'interconnexions devront faire face ; tout de même à de nombreuses contraintes: consommation d'énergie électriques, occupation de surface de silicium, performances, synchronisation.

Trois objectifs distincts sont étudiés au travers de ce travail consistant en:

- Identification de la problématique relative à l'architecture de communication utilisée dans les SoC afin d'y percevoir de la nécessité d'introduire le concept de plateforme NoC

- L'étude et le dégagement de tous les mécanismes réseau pouvant être exploités pour une conception d'une architecture de communication sur puce.

- Après cette analyse, notre objectif est de proposer un model simple d'un Nœud NOC reposant sur l'étude qui a été déjà faite. Pour ce faire; on est ramené à choisir une méthodologie de modélisation adaptée à notre système.

Ce mémoire est agencé de la manière suivante :

Le chapitre 1 présente le contexte et la problématique de l'étude. Il détaille les avantages et les inconvénients des circuits synchrones, en rappelant combien un circuit synchrone sera difficile à mettre en place dans les futurs systèmes sur puce. Les particularités et les exigences des circuits asynchrones sont également présentées ainsi que celles des architectures globalement asynchrones localement synchrones. Enfin, les différents Principes de base de la conception des systèmes mono puce sont passés en revue.

Le chapitre 2 détaille les différents types de réseaux sur puce, en allant des simples réseaux à médium partagés (bus) aux réseaux hybrides hiérarchiques en présentant différentes topologies. Une première comparaison qualitative de ces architectures est proposée. Un état de l'art des réseaux sur puce NoC industriels et autre est aussi présenté pour enfin retenir un modèle de conception de réseau sur puce qu'on appelle Nostrum qui servira de modèle de référence pour notre projet .

Le chapitre 3 introduit les approches de conception, ce chapitre se propose alors en vue de présenter notre approche de conception de tels systèmes avec les outils associés et le flux de conception possible.

Le chapitre 4 expose notre cahier des charges élaboré en utilisant les mécanismes réseau (présentés dans deuxième chapitre) pour modéliser notre Nœud réseau sur puce. Le but alors est de valider le modèle que nous avons décrit sous SDL après la génération du réseau sur puce associé à ce modèle. Une architecture d'évaluation de performance sera présentée pour proposer enfin les simulations et les résultats associés.

Chapitre I

Contexte et Problématique

1. Contexte

L'interconnexion de composants électroniques a été de tout temps une préoccupation majeure dans la réussite et la conception d'un système électronique. Il y a plus d'une décennie, l'interconnexion se faisait plus au niveau composant élémentaire par l'intermédiaire des PCB pour Printed Circuit Boards (circuits imprimés). Avec l'évolution technologique associée à la conception et la réalisation des circuits intégrés. Nous sommes face à une nouvelle problématique, où l'interconnexion se fait à l'intérieur du circuit (intra-chip) avec des dizaines de cœurs (IP Intelectual Property) à interconnecter. De ce fait, le goulot d'étranglement dans ces circuits, appelés communément SoC (System On Chip), réside dans la gestion de leurs intercommunications ; ce qui montre que l'approche antérieurs n'est plus suffisante, d'où l'émergence de nouvelle approche d'interconnexion des composants.

2. Système sur puce

Avant d'aborder la problématique visée, il est nécessaire de définir ce qu'est un system on chip (SoC) (on l'appelle aussi un système sur puce ou encore système sur silicium).

Un système peut être modélisé comme un ensemble de composants de base connectés autour d'un réseau de communication.

Un système monopuce [1] intègre de la mémoire nécessaire au fonctionnement d'un processeur, de la mémoire partagée pour la communication entre plusieurs composants ou de la mémoire tampon pour cacher la latence du système et amortir les variations des flux de données. Il peut comprendre également des coprocesseurs matériels pour le contrôle du système ou pour des opérations de calcul rapide. Il peut comprendre aussi des périphériques d'entrée-sortie pour assurer la liaison avec l'environnement extérieur.

Un sous-système processeur est un processeur éventuellement accompagné d'un ensemble de composants de base nécessaires à son fonctionnement (mémoire ROM, RAM, contrôleur mémoire, contrôleur d'interruptions, ...).

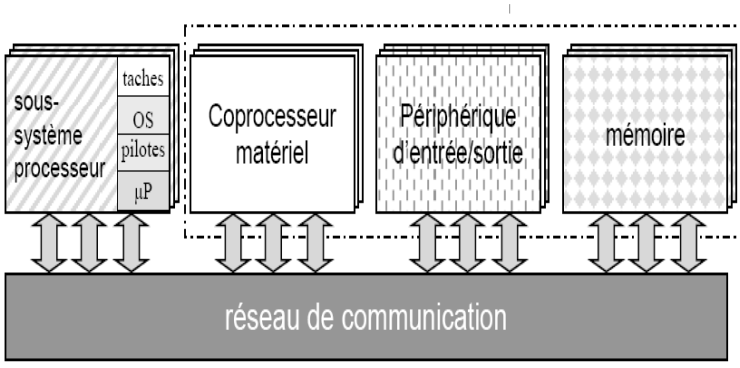


Figure 1.1 : Un système sur puce [2]

Le réseau de communication est devenu le cœur de la conception des systèmes sur silicium. La plupart des grands acteurs du monde de la conception en microélectronique

offrent aujourd'hui dans leur portefeuille de composants, un réseau de communication intégré. Ce réseau consiste en un ensemble de bus partagés, interconnectés par des ponts.

2.1. Un exemple de SoC : Nomadik

L'architecture de ST Nomadik [3] est conçue pour des applications multimédia portables. Ceci exige le support de codage et de décodage MPEG4. Il exige également un support à une vaste gamme des tailles d'affichage pour des téléphones portables (160 x 160) à PDA (320 x 240 jusqu'à 680 x 480).

La figure (Figure 1.2) montre la première implémentation de l'architecture hétérogène du système monopuce multiprocesseurs de ST Nomadik. L'unité centrale de traitement principale pour le système est un ARM926E-JS. Ce processeur est utilisé pour le contrôle et la coordination des tâches, tandis que la plupart des fonctions sont exécutées par les accélérateurs spécifiques à l'application. L'ensemble des composants est interconnecté via un bus AMBA.

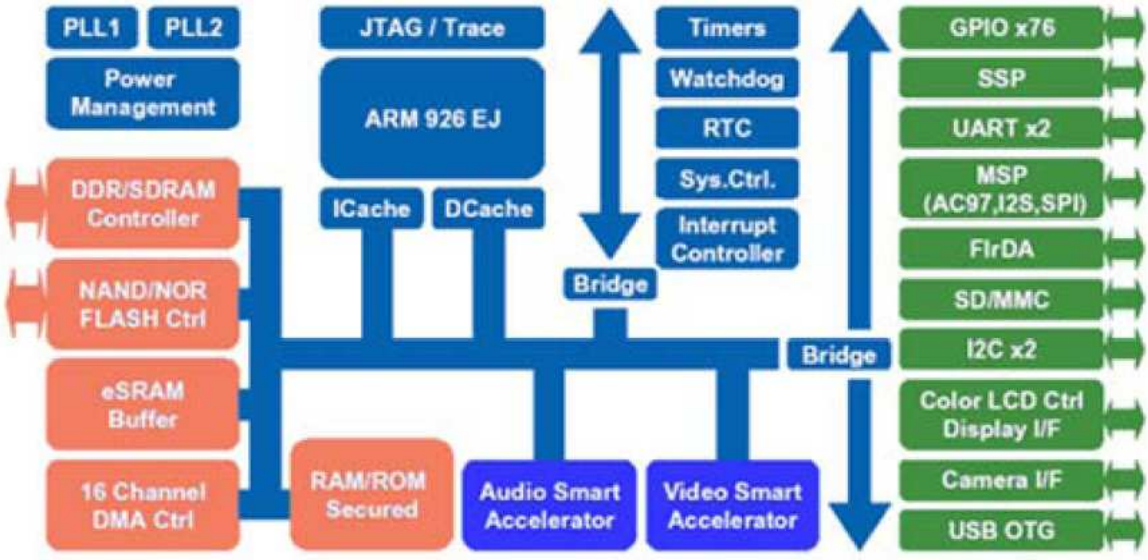


Figure 1.2 : Système monopuce multiprocesseurs de ST Nomadik

3. Pourquoi conçoit-on et utilise-t-on des systèmes monopuce

Traditionnellement, ces circuits étaient jusqu'alors réalisés avec une approche hybride en intégrant des processeurs et des ASIC sur une carte [2]. Aujourd'hui, la technologie permet d'intégrer ces composants dans une même puce afin de miniaturiser les systèmes. C'est ce qui a amené développement des systèmes mono puce, assemblage de processeurs pour leur **flexibilité** et de blocs matériels pour leurs **performances**.

En effet, L'utilisation de processeurs permet d'avoir des circuits programmables, qui sont **évolutifs et flexibles**. La conception du système peut alors commencer avant la finalisation d'une norme ou avant la définition finale du produit. Il est envisageable de faire évoluer le système tout au long de sa conception car la modification du logiciel est beaucoup plus facile et représente un effort moindre que le changement du matériel. Le système peut évoluer non seulement pendant les étapes de sa conception mais aussi tout au long de la vie

du produit. Il est ainsi possible de le faire évoluer avec l'évolution d'une norme, ajouter de nouvelles fonctionnalités ou le support de nouvelles normes, ou bien tout simplement améliorer la qualité du produit avec par exemple : le perfectionnement d'algorithmes de traitement d'images. Cette évolution est beaucoup plus rapide qu'elle ne le serait s'il fallait modifier le circuit. Cette flexibilité permet également la différenciation du produit final. En permettant de laisser une plus grande liberté d'utilisation du circuit qu'un ASIC. Le concepteur de l'application peut ainsi adapter le produit à son application afin de le différencier d'un produit concurrent utilisant le même circuit. La différenciation s'effectue à la fois en terme de fonctionnalités disponibles mais aussi de qualité du produit (traitement d'images par exemple).

Si la réalisation d'applications en logiciel a l'avantage de la flexibilité, cette solution se heurte néanmoins au traditionnel problème du logiciel. L'utilisation de composants matériels spécifiques, ou ASIC ("Application Specific Integrated Circuit"), offre de **meilleures performances** pour une puissance plus faible. Une approche purement matérielle ou purement logicielle n'est pas toujours bien adaptée à certaines applications. Certaines applications ne peuvent pas être réalisées entièrement en logiciel, mais une réalisation complètement matérielle n'est pas totalement satisfaisante car pas assez flexible.

Les systèmes monopuce avec leurs méthodes et outils associés d'une part et de la flexibilité s'avèrent des solutions pour maîtriser la complexité de la transposition de ces applications sur des circuits tout en respectant les contraintes sévères de temps de mise sur le marché.

4. Influence de l'évolution des technologies de fabrication des circuits intégrés

Les capacités d'intégration des technologies de fabrication autorisent le développement d'applications de complexité croissante. Cela répond à une double demande d'avoir des circuits intégrés plus performants et offrant plus de fonctionnalités pour une surface moindre (miniaturisation) [2]. Même si l'élément de base à intégrer est resté le même, c'est-à-dire le transistor, pour le concepteur il ne s'agit plus tellement d'intégrer des transistors mais des processeurs, mémoires ou bus partagés sur une même puce. Les circuits actuels allant jusqu'à plusieurs centaines de millions de transistors, concevoir ces circuits en décrivant manuellement l'assemblage de chaque transistor représenterait une masse de travail excessive.

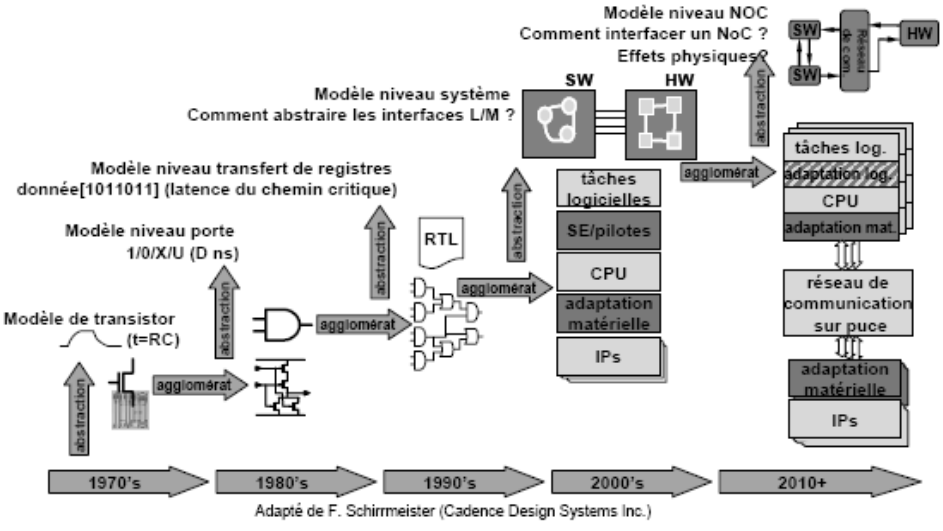


Figure 1.3 : Evolution des technologies de fabrication [4]

Pour gérer la complexité des circuits intégrés traditionnellement évaluée en nombre de transistors, les méthodologies de conception ont toujours reposé sur trois principes de base: la **décomposition du problème**, la **réutilisation** de composants et l'**abstraction**.

La décomposition du circuit en plusieurs parties plus simples à concevoir permet de simplifier un problème trop compliqué à maîtriser en le décomposant en plusieurs sous problèmes que l'on sait gérer.

La réutilisation de composants pour concevoir ces différentes parties permet d'accélérer la conception des circuits. Pour maîtriser la complexité des circuits, la nature des composants réutilisés a continuellement évolué, du transistor au processeur en passant par des portes logiques et des registres.

Cette évolution a pu se faire grâce à l'utilisation de modèles de spécification du circuit avec des niveaux d'abstraction croissants. Si les caractéristiques électriques des transistors sont nécessaires pour les assembler, l'étude analogique d'un circuit de plusieurs millions de transistors serait ingérable. Il est donc nécessaire d'abstraire les problèmes pour se concentrer sur les problèmes significatifs. Comme la montre la figure (Figure 1.2), on a commencé par abstraire les zones de dopage et les masques, pour concevoir le circuit comme un assemblage de transistors, puis comme un assemblage de portes logiques et finalement comme de la logique combinatoire et des registres avec la synthèse logique.

Les méthodes de conception ont donc continuellement évolué pour pouvoir gérer la complexité croissante des circuits intégrés. Une complexité devenue telle qu'il est impossible de continuer à les concevoir au niveau RTL, où il faut préciser chaque détail du comportement des composants. Le grand défi en ce moment pour les ingénieurs est de réussir à maîtriser la complexité lors de la conception de ces systèmes et d'arriver à une conception rapide des systèmes monopuce sous de fortes contraintes de qualité et de temps de développement.

L'**abstraction** permet de concevoir des circuits plus rapidement, ouvrant la voie à l'intégration d'un plus grand nombre de composants, jusqu'à ce qu'on soit limité par de nouveaux problèmes. On assiste aujourd'hui à un **problème de productivité** des concepteurs, et à un **accroissement des temps de conception**. Face à la complexité des systèmes mono puce, les méthodes de conception existantes et l'expérience du concepteur sont insuffisantes pour respecter des temps de mise sur le marché extrêmement courts. Pour comprendre comment concevoir des systèmes monopuce de plus en plus complexes dans des délais raisonnables en gardant le contrôle sur le résultat final, il nous faut comprendre les difficultés rencontrées pour leur conception afin de passer à un niveau d'abstraction supérieur.

5. Contraintes physiques et limites des méthodes classiques de conception

5.1. Limites physiques

L'augmentation des performances des circuits intégrés est liée à la finesse de gravure des circuits. Sa diminution a permis d'intégrer un nombre croissant de transistors sur une même surface de silicium, tout en permettant d'augmenter les fréquences de fonctionnement. Avec les finesses de gravure atteintes par les nouvelles technologies, un certain nombre de limites physiques [2] pose des problèmes en termes de **fiabilité** et de **puissance consommée**. Les **interférences électromagnétiques** existant entre deux lignes proches, dues à leur couplage capacitif ou à leur inductance mutuelle, sont des sources de

bruit. Avec l'augmentation en fréquence des circuits et la plus grande finesse de gravure, elles ne sont plus négligeables. De même, le circuit est plus sensible aux rayons cosmiques qui risquent de charger ou décharger inopinément une capacité ("soft errors" en anglais).

5.1.1. Synchronisation

a. Circuit synchrone

Un circuit synchrone [5] est basé sur le fait que tous les composants sont synchronisés par une horloge globale. Les transitions du circuit sont temporellement prédictibles. Un système synchrone permet d'assurer un fonctionnement déterministe, comme par exemple le séquencement des instructions. Il assure également une fiabilité des communications. En revanche, un circuit synchrone pose des problèmes de fonctionnement et de conception de plus en plus complexe en termes de fonctionnement, la synchronisation de l'ensemble des fonctions d'un système par une horloge unique engendre des délais de synchronisation qui limitent certaines applications. En effet, l'horloge est synchronisée à la fréquence du cœur le plus lent du système. Les autres cœurs pouvant fonctionnés à une fréquence plus élevée sont ralentis, ce sont les délais de synchronisation. La liste ci-dessous présente les difficultés de réalisation rencontrées dans un circuit synchrone :

- Le **temps de propagation** d'une horloge dans un système de taille conséquent peut entraîner un déphasage de l'horloge et des retards de synchronisation.

- La **consommation d'énergie** : tous les éléments d'un circuit s'activent au front d'horloge, même s'ils ne réalisent pas de fonctions utiles. De plus, les lignes d'horloge ont une capacité électrique élevée.

- La commutation synchrone de toutes les lignes d'horloge et les pointes de courants induites par les commutations simultanées peut entraîner une **pollution électromagnétique** dans le circuit.

Il existe des techniques de distribution d'horloge dont l'usage est souvent impératif : la boucle à verrouillage de délai, la distribution de l'horloge par une grille ou le réseau d'horloge distribué par PLL sont des exemples de techniques utilisées pour distribuer l'horloge.

L'étude suivante permet de visualiser l'évolution des circuits synchrones dans les systèmes sur puce. La figure (Figure 1.4) illustre l'évolution de la zone isochrone en fonction de la fréquence d'horloge du système pour deux technologies (50 nm et 100 nm). La zone isochrone est la plus longue distance (en millimètres) séparant deux composants pouvant être synchronisés par une seule horloge.

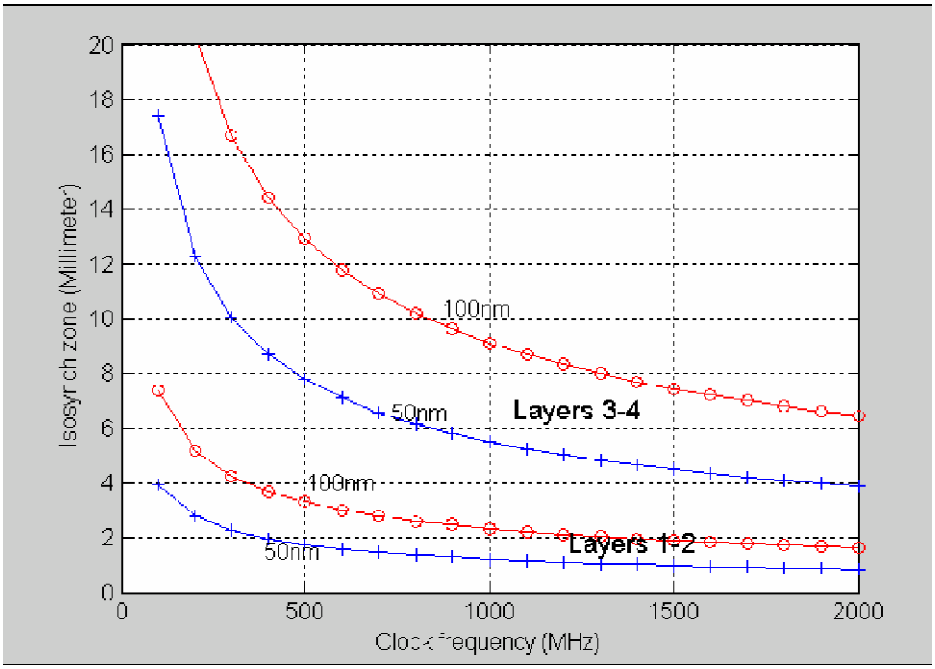


Figure 1.4: Evolution de la zone isochrone en fonction de la fréquence d'un système [6]

	low power SoC			high performance MPU/SoC		
	2001	2004	2010	2001	2004	2010
Longueur de canal (nm)	130	90	45	90	53	25
Tension d'alimentation	1.2	1	0.6	1.1	1	0.6
Nombre de transistors (M)	3.3	8.3	40	276	553	2212
Taille du chip (mm²)	100	120	144	310	310	310
Fréquence (ghz)	0.15	0.3	0.6	1.7	2.4	4.7
Niveaux de métal	6	7	9	7	8	10
Puissance max (W)	0.1	0.1	0.1	130	160	218

Tableau 1.1 : Verrous technologiques [7]

Aux vues du tableau et du graphique ci-dessus, il devient impossible de distribuer une seule horloge sur toute la puce. Deux solutions sont envisageables pour résoudre ce problème : soit rendre le système complètement asynchrone, c'est-à-dire ne plus avoir d'horloge, soit le rendre globalement asynchrone et localement synchrone (GALS).

b. Circuit asynchrone

Contrairement à un circuit synchrone [5], dans un circuit asynchrone, la date de transition des signaux n'est pas prédictible. Le contrôle et le séquencement des

communications et des instructions ne sont pas assurés par un signal d'horloge global mais par d'autres méthodes de communications. Le système asynchrone présente de nombreux avantages :

- Les protocoles asynchrones ont été démontrés comme étant robustes.
- La pollution électromagnétique liée à la commutation synchrone des lignes d'horloge et aux pointes de courants induites par les commutations simultanées des portes est largement diminuée.
- La fréquence moyenne de travail du système augmente lorsque chaque bloc travaille à sa fréquence limite et non plus à la fréquence limite de tout le système.
- La consommation diminue notamment par la suppression de l'arbre d'horloge.

En revanche, les systèmes asynchrones présentent les inconvénients suivants:

- Coûts supplémentaires en logique de contrôle induisant généralement un coût additionnel en surface.
- La métastabilité : la capacité pour un état non équilibré de persister pendant une longue période de temps. Cela arrive lorsqu'un signal est échantillonné à une valeur intermédiaire qui ne permet pas de déterminer son niveau logique.
- Le manque de maturité. Malgré d'énormes progrès en cours dans des groupes de recherche, les techniques de conception asynchrones (outils de conception, librairies, langages) sont bien moins avancés que les techniques pour la conception synchrone.

Dans l'état actuel, les protocoles asynchrones permettant l'échange de données entre cœurs de grandes tailles (et non plus au niveau de cellules élémentaires) nous semblent un compromis intéressant pour la conception de systèmes sur puce. Ce concept est détaillé dans le paragraphe suivant.

C. Circuits Globalement Asynchrone Localement Synchrone

Le paradigme GALS [5] est l'assemblage sur une même puce de plusieurs composants synchronisés par des horloges indépendantes. Un système globalement asynchrone localement synchrone est divisé en sous blocs cadencés par des horloges indépendantes. Chaque bloc communique au moyen de protocoles asynchrone, c'est-à-dire que la synchronisation ne se fait pas par une horloge globale mais par des protocoles de communication locaux.

Ce protocole est dit synchrone dans le domaine informatique car il synchronise deux processus concurrents. Pour les concepteurs en microélectronique, la communication est dite asynchrone pour la distinguer de la communication synchrone, alors qu'il s'agit de distinguer une synchronisation par protocoles de communication d'une synchronisation par horloge.

La synchronisation GALS combine les avantages des deux techniques de synchronisation :

- Une fréquence propre à Chaque bloc, tout en étant cadencé par une horloge. Chaque bloc est conçu à l'aide d'outils traditionnels de conception synchrone.
  Les protocoles de communications asynchrones sont robustes.

Cette méthode de synchronisation semble une alternative intéressante pour les systèmes sur puce de demain. L'interconnexion de demain devra donc explorer la voix d'une synchronisation GALS.

5.1.2. Interconnexions sur puce

Les réseaux d'interconnexions [5] jouent un rôle majeur dans les performances des systèmes sur puce. Il y a plusieurs facteurs pouvant affecter le choix d'une architecture appropriée puisqu'une interconnexion de système sur puce doit satisfaire certaines contraintes pour être viable. Elle doit être performante, flexible, simple, physiquement implantable et respecter les contraintes de coût du cahier de charges. Quelques une de ces contraintes sont détaillées ci-dessous :

a. Le coût d'une interconnexion

Ce coût se mesure en surface, en énergie consommée et en performances. Nous considérons que les performances d'un système regroupent son débit et sa latence, notions définies par la suite. Nous adoptons ici le point de vu de l'interconnexion et considérons la latence non pas comme un coût mais comme une performance. Ainsi, plus la latence est petite, meilleures sont les performances.

b. Les performances d'une interconnexion

Les performances d'un réseau sur puce représentent l'efficacité du réseau à acheminer les données d'une source vers une cible. Afin d'évaluer ces performances, nous mesurons deux caractéristiques : la latence moyenne mesurée en cycles d'horloge et le débit mesuré en mots par unité de temps (dans notre cas par cycle d'horloge).

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

Le message est acquitté par le cœur destinataire. La latence correspond alors au temps entre l'émission du paquet et la réception de son acquittement par le cœur initiateur.

Le message est acquitté par un intermédiaire entre la source et la cible. La latence correspond alors au temps entre l'émission du paquet et sa réception à la cible.

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

c. La flexibilité d'une interconnexion

La **flexibilité** d'un réseau est le paramètre le plus difficile à apprécier. La flexibilité d'un médium exprime sa capacité à s'intégrer à un système. Cette flexibilité est composée de deux paramètres : la portabilité du médium et sa « scalabilité » décrits ci-après.

Le temps de conception d'un élément est un paramètre important dans l'élaboration d'un composant. En effet, la durée de vie des composants étant de plus en plus faible, leur conception doit se faire en un temps réduit. Les concepteurs utilisent donc de plus en plus de librairies afin de les aider à gagner du temps. L'interconnexion d'un système sur puce doit être conçue dans la même optique: elle doit être facilement réutilisable et donc reconfigurable. La capacité d'une interconnexion à être adaptée à un nouveau système est sa **portabilité**.

La « **scalabilité** » d'un médium correspond à sa capacité à évoluer en fonction du nombre de cœurs. Si l'ajout de cœurs fait augmenter les performances et le coût de l'interconnexion de façon proportionnelle, le médium est alors « scalable ». En revanche, s'il devient le goulot d'étranglement du système à cause de l'ajout de cœurs supplémentaires,

le composant ne pourra pas être utilisé. Pour savoir si un médium est flexible (« scalable »), il faut mesurer l'évolution de son coût et de ces performances en fonction du nombre de cœurs connectés.

d. La qualité de service

La **qualité de service** ou QoS (pour Quality of Service) présuppose qu'un travail est effectué et permet de définir la qualité avec laquelle ce service est rendu. La qualité d'un travail est jugée en fonction de son application. Dans une interconnexion, nous considérons que le service rendu est l'acheminement des paquets jusqu'au cœur destinataire et la qualité est le temps nécessaire à ce travail.

Donc, imposer une qualité de service dans une interconnexion revient à prédire la latence des messages et donc borner le débit de l'interconnexion.

Les limites physiques influent donc sur la conception du système. Elles ne se limitent pas à un problème technologique mais doivent être prise en compte dès les premières étapes de la conception du système.

5.2. Problème de productivité

Les systèmes mono puce sont des systèmes multiprocesseurs hétérogènes. Les flots de développement logiciel et matériel sont généralement séparés. Le logiciel est conçu une fois le matériel terminé, ce qui **allonge les délais**. Ce problème s'accentue dans la mesure où les systèmes en se complexifiant deviennent plus difficiles à programmer. Cela pose également des problèmes pour prévoir si le système final est capable de fournir les performances attendues car on ne peut évaluer ses performances qu'une fois le matériel conçu. Compte tenu du coût de conception d'un tel système, il n'est pas acceptable d'attendre que le système soit entièrement conçu pour s'apercevoir que le système ne respecte pas la spécification. Dans la mesure où le résultat est incertain, il n'est également pas possible d'optimiser le système, ce qui est un frein pour améliorer la qualité des systèmes.

Les méthodes de conception n'ayant pas radicalement évolué depuis l'apparition de la synthèse logique, les temps de conception sont devenus excessifs. On parle maintenant de **"lacune de productivité"** *(design productivity gap* en anglais) [8] pour se référer à l'écart existant entre l'évolution du nombre de transistors que l'on peut mettre sur une puce et le nombre moyen de transistors d'un circuit qu'un ingénieur peut concevoir en une journée de travail. Pour contrebalancer ce manque de productivité, les équipes de conception doivent être très grandes, ce qui implique un coût très important. De plus, le coût des masques devient d'autant plus élevé que l'on diminue la finesse de gravure des transistors.

6. Principes de base de la conception des systèmes mono puce

6.1. Découplage calcul/communication

La communication entre différents composants hétérogènes est une des difficultés de la conception des systèmes mono puce. Les communications dans les systèmes mono puce se complexifient et sont d'une importance décisive pour les performances du système. Le problème de productivité des concepteurs requiert un nouveau paradigme tenant compte des problèmes physiques posés par la communication dans les systèmes mono puce.

Le découplage du calcul et de la communication [2] sépare les problèmes afin de mieux maîtriser cette complexité. Pour le concepteur de l'application, il est important de savoir comment envoyer des données d'une tâche à une autre ou d'une tâche à un coprocesseur matériel. Il est nécessaire de connaître les fonctions à appeler pour envoyer une donnée, mais il est inutile de connaître comment la communication se fait. Les composants de calcul sont conçus avec l'hypothèse que le réseau de communication fournit les performances requises. Cela facilite leur conception mais aussi la conception du système qui consiste alors à connecter ces composants à un réseau de communication. Du point de vue du concepteur du réseau, les informations importantes sont les besoins en bande passante/latence d'un composant. Le type et la signification des données transportées ne sont pas d'une grande importance pour lui. Les ressources de communication peuvent donc être conçues indépendamment des ressources de calcul.

On parle alors de conception basée sur les interfaces ("interface-based design" en anglais), car on ne s'intéresse plus à la manière dont est réalisé le composant, mais seulement à son interface pour savoir s'il peut-être connecté au réseau de communication. La conception de systèmes mono puce consiste alors à assembler des composants autour d'un réseau de communication.

6.2. Réutilisation

Pour réduire les coûts et les temps de conception, un concept important dans les systèmes mono puce est celui de la réutilisation [2] des composants. Des modèles de composants matériels (au niveau masque ou transfert de registres/RTL) sont considérés comme de la "propriété intellectuelle", ou IP ("Intellectual Properly" en anglais). On parle donc d'IP pour se référer à un composant ou bloc matériel réutilisable. Les processeurs sont également réutilisables. L'utilisation de processeurs et d'IP permet de réduire le temps de conception des SoC. Un système monopuce peut donc être représenté par le schéma ci-après.

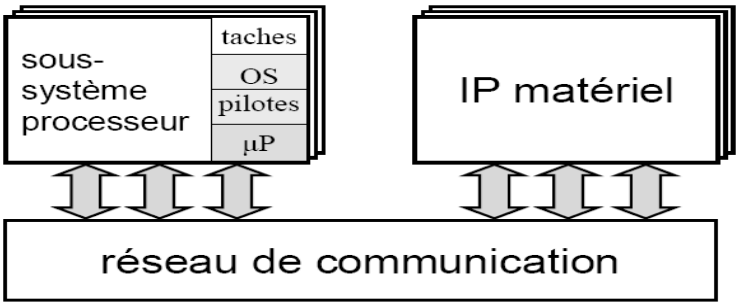


Figure 1.5 : Système monopuce: un réseau de composants réutilisables

Si l'on conçoit les systèmes mono puce à partir de composants réutilisables, le développement des systèmes consiste alors à assembler les composants pour les intégrer dans un même circuit. Le système peut être vu comme un ensemble de nœuds de calcul logiciel (processeur) ou matériel (IP) communiquant à travers un réseau de communication. La conception des SoC consiste alors à choisir les différents nœuds de calcul et à les faire communiquer ensemble. Les composants de communication peuvent aussi être déjà conçus et réutilisés après configuration.

6.3. Abstraction

Concevoir des systèmes mono puce au niveau transfert de registres reste difficile même si l'on utilise des composants réutilisables. Il est aujourd'hui nécessaire d'élever [2] le niveau d'abstraction des flots de conception classiques pour maîtriser la complexité des SoC. Dans les premières étapes de la conception, l'abstraction dispense le concepteur de la gestion des problèmes liés à l'hétérogénéité du système et aux limites physiques des interconnexions. Pour cela, des modèles de haut niveau ont été proposés pour abstraire les communications dans systèmes mono puce.

Pour aborder la conception du système dans une seule et même approche cohérente, les composants aussi bien logiciels que matériels sont modélisés avec un modèle unique. Des flots de conception séparés du logiciel et du matériel ne permettent ni de respecter les délais de conception, ni de garantir que le système répondra aux exigences avant son prototypage.

La conception conjointe du matériel et du logiciel est un défi pour la réalisation d'un flot de conception système, préambule à l'obtention de circuits de meilleure qualité en un temps plus court.

7. Conclusion

Nous venons de voir que l'interconnexion des systèmes sur puce doit répondre à certains critères architecturaux. Un des premiers critères est la synchronisation d'un système sur puce devant fonctionner dans un environnement **GALS**. Ensuite, l'interconnexion doit satisfaire des critères de performances, de consommation d'énergie, de surface de silicium, tout en proposant une qualité de service optimale. Le problème est de savoir si les interconnexions d'aujourd'hui répondent à ces critères.

Les nouvelles possibilités de la technologie et les effets physiques qui vont en découler, orientent la conception dans le même sens **: "le besoin de prendre en compte les communications"**. Cette tendance a conduit à l'utilisation de bus de communication de plus en plus compliqués, et les réseaux de communication vont probablement devenir encore plus compliqués comme nous le verrons dans le chapitre 2. La communication est alors beaucoup moins prévisible qu'avec l'utilisation uniquement d'interconnexions dédiées et doit donc être traitée différemment.

Ces systèmes sont donc complexes et sont soumis à de fortes contraintes, il est donc nécessaire d'établir une méthodologie systématique permettant au concepteur de maîtriser la complexité des SoC, afin de **réduire les temps et les coûts** de conception tout en **améliorant la qualité** des systèmes.

A cette fin, les principes de base de la conception des circuits intégrés doivent être redéfinis en fonction des problèmes posés par la conception des systèmes, à cet effet le deuxième chapitre se propose afin d'introduire les nouveaux principes de conception on parle alors du nouveau paradigme.

« Network On Chip »

Chapitre 2

Rétrospective des Réseaux Sur Puce (NoC)

1. Introduction

Le but de ce chapitre est de présenter en détail le nouveau paradigme introduit dans le premier chapitre I 'Network On Chip'. On examinera d'abord les différents types d'architectures de réseau ; en analysant les avantages et les inconvénients des divers mécanismes associés, et ce afin de pouvoir prendre des décisions relatives à notre modélisation.

Pour pouvoir traiter un tel paradigme des prérequis réseaux informatiques doivent être préalablement introduits, parmi eux le modèle de référence OSI.

Les réseaux sur puce peuvent être représentés par le modèle en couche OSI (« Open Systems Interconnexion », modèle de référence d'interconnexion de systèmes ouverts).Ce modèle a été défini pour éviter la multiplication des solutions d'interconnexion d'architectures hétérogènes. Il décrit les concepts utilisés et la démarche suivie pour normaliser l'interconnexion de systèmes ouverts.

Le modèle OSI comporte 7 couches :

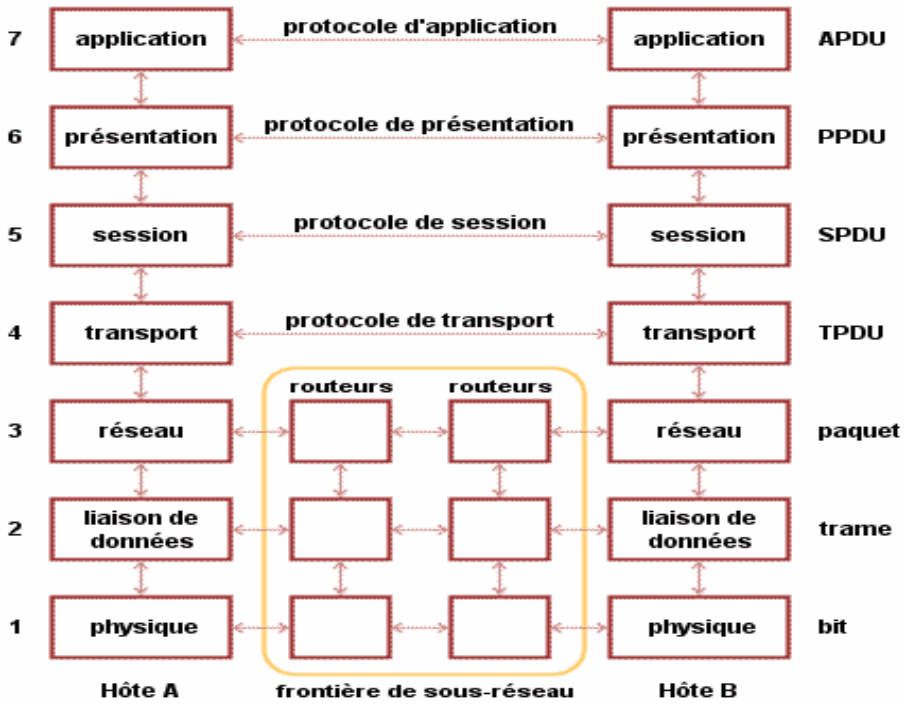


Figure 2.1 : Modèle OSI

Les principes qui ont conduit à ces 7 couches sont les suivants :

- une couche doit être créée lorsqu'un nouveau niveau d'abstraction est nécessaire,
- chaque couche a des fonctions bien définies,
- les fonctions de chaque couche doivent être choisies dans l'objectif de la normalisation internationale des protocoles,
- les frontières entre coucfies doivent être choisies de manière à minimiser le flux d'information aux interfaces,

- le nombre de couches doit être tel qu'il n'y ait pas cohabitation de fonctions très différentes au sein d'une même couche et que l'architecture ne soit pas trop difficile à maîtriser.

Les couches basses (1, 2, 3 et 4) sont nécessaires à l'acheminement des informations entre les extrémités concernées et dépendent du support physique. Les couches hautes (5, 6 et 7) sont responsables du traitement de l'information relative à la gestion des échanges entre systèmes informatiques. Par ailleurs, les couches 1 à 3 interviennent entre machines voisines, et non entre les machines d'extrémité qui peuvent être séparées par plusieurs routeurs. Les couches 4 à 7 sont au contraire des couches qui n'interviennent qu'entre hôtes distants.

2. Modèle OSI

Lors d'une émission, les données « descendent » chacune des couches au niveau de la machine émettrice. À chaque couche traversée, une information (un en-tête *H -éventuellement nul-) nécessaire à sa transmission est ajoutée au paquet de données : c'est l'encapsulation.

Lors d'une réception, les données « remontent » chacune des couches au niveau de la machine réceptrice. À chaque couche traversée, une information encapsulée est lue et utilisée (désencapsulation) ; à la réception les données retrouvent donc leurs états.

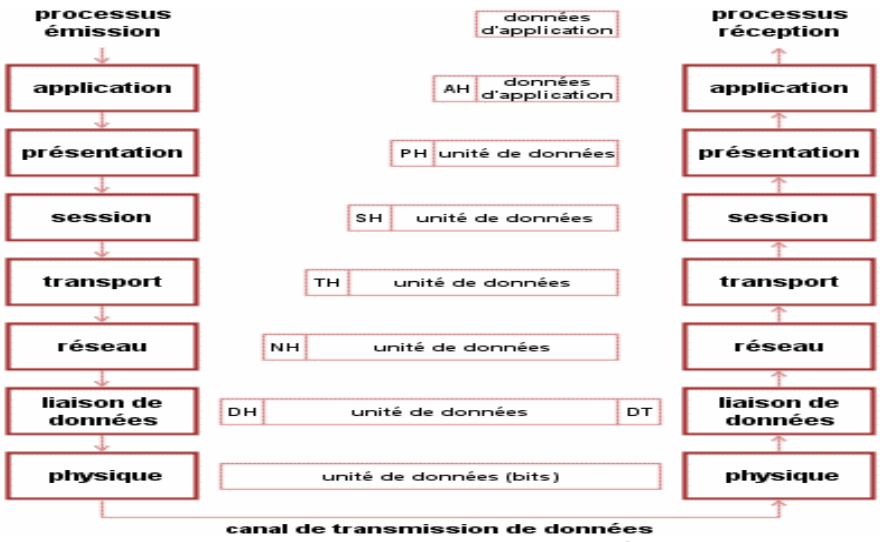


Figure 2.2 : Transmission de données

Le concept important est le suivant : il faut considérer que chaque couche est programmée comme si elle était vraiment horizontale, c'est à dire qu'elle dialoguait directement avec sa couche paire réceptrice. Au moment de dialoguer avec sa couche paire, chaque couche rajoute un en-tête et l'envoie (virtuellement, grâce à la couche sous-jacente) à sa couche paire.

3. Types de réseau

Les architectures réseau sont habituellement regroupées en quatre catégories:

3.1. Réseaux à médium partagé – les bus

Cette structure d'interconnexion est beaucoup utilisée dans des systèmes sur puce qui ne permet qu'a nœud a la fois d'utiliser le bus pouvant fonctionner avec un multiplexage temporel ou spatial [5]: dans ce cas plusieurs cœurs peuvent utiliser le même médium et la bande passante est alors allouée dynamiquement. Chaque cœur doit recevoir un acquittement avant d'émettre un message sur le bus. L'acquittement est distribué suivant un protocole d'arbitrage permettant de résoudre les éventuels conflits du réseau. Un des avantages de l'interconnexion à médium partagé et qu'elle supporte très bien le « broadcast ».

En revanche, cette architecture a une extensibilité limitée à une dizaine de cœurs maîtres [9]. La consommation d'énergie est aussi un point critique des interconnexions à medium partagé.

Les systèmes sur puce sont passés d'une dizaine à une centaine de cœurs nécessitant donc des bandes passantes importantes. Pour ces systèmes, l'interconnexion à médium partagé est le goulot d'étranglement en termes de performance et de consommation d'énergie.

Exemple de Bus

Le bus « backplane » est le type de bus le plus utilisé dans les systèmes sur puce avec les technologies actuelles [5]. Les processeurs, les éléments mémoires et les « entrées-sorties » cohabitent sur ce médium composé de 3 types de lignes. Les lignes de données transportent les informations entre les sources et les cibles. L'adresse de destination des données se situe sur la ligne d'adresse. Les lignes de contrôle servent aux requêtes, aux acquittements et aux transferts d'information concernant les données.

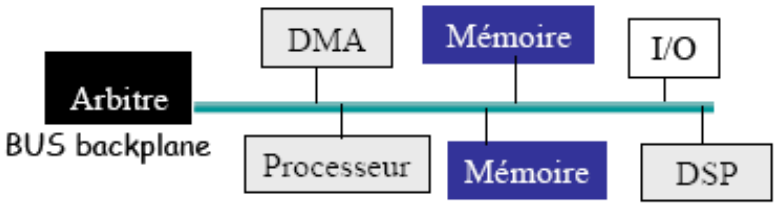


Figure 2.3 : Bus « backplane »

3.2. Réseaux directs

Les réseaux directs permettent de résoudre les problèmes d'extensibilité (ou « scalabilité ») des bus[10]. Un nœud peut contenir un ou plusieurs éléments processeur, mémoire et entrée/sortie.

Ces éléments sont interfacés avec le réseau via un routeur. Les routeurs sont directement connectés aux routeurs voisins, comme l'illustre la Figure 2.4. Le routeur possède plusieurs canaux d'entrée/sortie en direction des routeurs voisins et un canal entrée/sortie local pour le cœur.

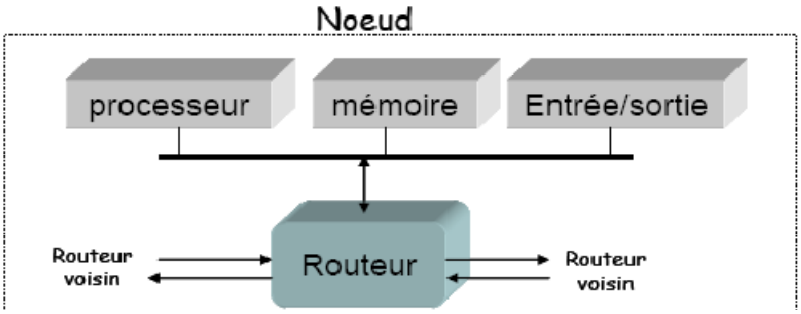


Figure 2.4 : Le nœud d'un réseau direct

Le réseau direct est caractérisé par trois facteurs: sa topologie, son algorithme de routage et le type d'acheminement des données [5].

L'algorithme de routage et les différents types d'acheminement sont décrits dans le paragraphe 5.

La topologie définit la façon dont sont agencés les nœuds les uns par rapport aux autres. Une topologie possible pour un réseau direct est de connecter tous les nœuds entre eux, comme l'illustre la Figure 2.5. C'est la connexion en point à point. La communication point à point permet un parallélisme contrôlé. En effet, chaque connexion est dédiée et spécifique à la communication entre deux cœurs. Cela permet ainsi d'établir des connexions à fort débit, uniquement entre deux cœurs et d'optimiser éventuellement la consommation d'énergie.

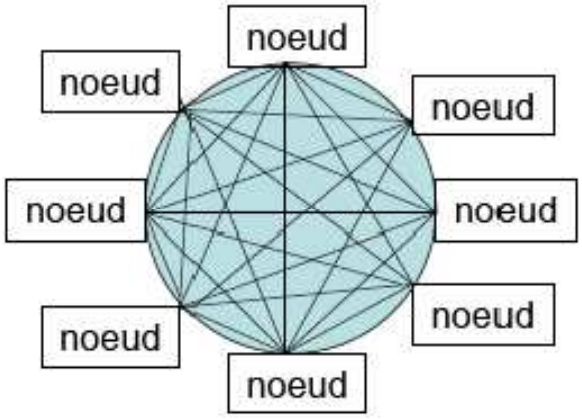


Figure 2.5 : Topologie point à point pour un réseau direct

Le coût d'une telle interconnexion est prohibitif [5]. De plus, les problèmes essentiels de ce type d'approche sont d'une part la faible possibilité de réutilisation (du fait de sa spécificité) et d'autre part le manque de « scalabilité ». Ce type de connexion étant déterministe, il est aisé de garantir une qualité de service optimale. En revanche, le taux d'utilisation du médium est faible. La connexion point à point n'est donc pas parfaitement adaptée aux systèmes de demain.

Différentes topologies, où un message peut traverser plusieurs nœuds intermédiaires, sont donc proposées dans la littérature en essayant d'équilibrer les performances et le coût du réseau. Nous présentons certaines de ces topologies implantées dans des réseaux sur puce :

La topologie la **plus employée** est le maillage à 2 dimensions. Les nœuds sont organisés en maille. Cette topologie est la plus utilisée car les **algorithmes de routage**

sont **simples à instaurer**, de plus elle offre une « **scalabilité** » importante, notamment en vue d'une implantation sur silicium.

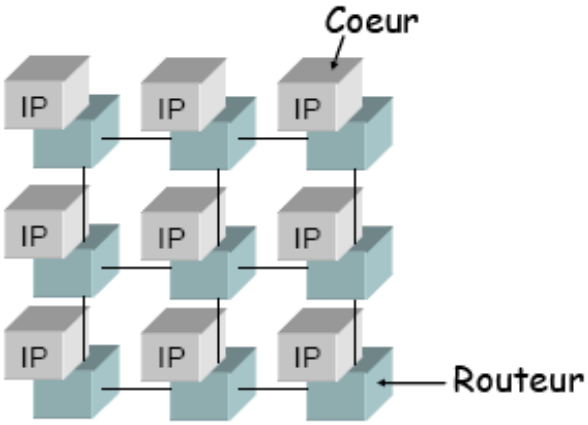


Figure 2.6 : Topologie maillée 2 dimensions

Le maillage à trois dimensions et le torus représentés Figure 2.7. sont des topologies dérivées du maillage à 2 dimensions. Elles présentent bien souvent des **bandes passantes supérieures** au maillage à 2 dimensions et les **algorithmes de routage** restent **assez simples**. Pourtant, ces topologies sont moins utilisées, du fait d'une « **scalabilité** » **limitée**. Au final, les problèmes électriques des bus sont retrouvés dans ces topologies.

La topologie torus est aussi utilisée par quelques réseaux dont le schéma de l'interconnexion est représenté ci-dessous.

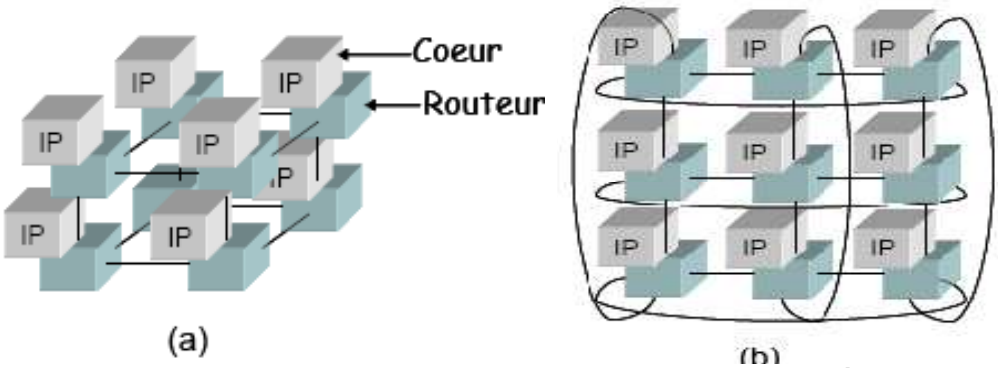


Figure 2.7 : Topologies sur la base du maillage : (a) maillage à 3 dimensions (b) torus

D'autres topologies, tels que les topologies en anneaux, en octogone et en nid d'abeille sont représentées Figure 2.8 Dans une topologie en anneau le routeur est relié uniquement à ses deux voisins par des liens bidirectionnels. C'est une topologie **facile à mettre en œuvre** au niveau de l'algorithme de routage et de l'implantation physique mais qui présente quelques inconvénients. Par exemple, pour un réseau de grande taille, le message doit traverser un nombre important de routeurs. **Limitant** ainsi **la bande passante**.

La topologie octogone a été proposée pour une application **spécifique**. Elle permet de connecter 8 nœuds avec 12 liens bidirectionnels comme l'illustre la Figure 2.8. L'avantage de cette topologie est qu'une communication entre n'importe quel nœud ne traverse jamais plus de deux routeurs pour atteindre sa destination.

Un routeur dans un réseau en nid d'abeille possède sept ports bidirectionnels, un local pour le cœur et six autres vers les routeurs voisins. **L'algorithme de routage** est

plus **complexe** à mettre en place car il existe un grand nombre de chemins possibles pour rejoindre une destination.

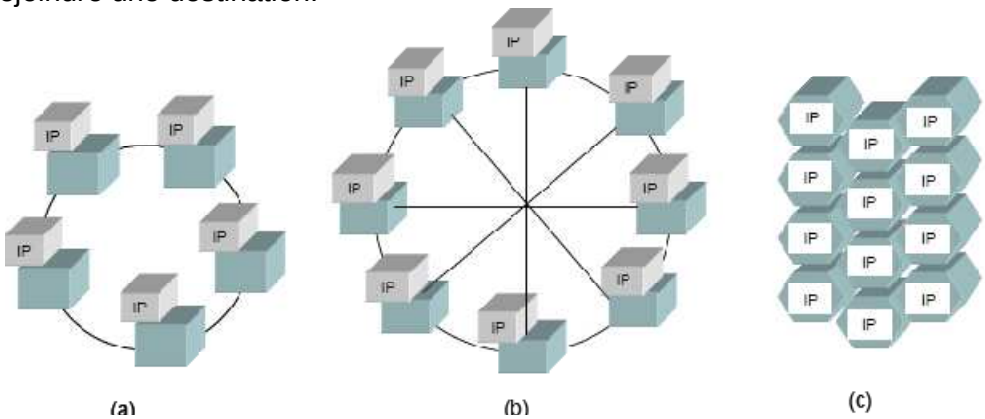


Figure 2.8 : Topologies sur la base d'un cercle (a) en anneau (b) en octogone (c) en nid d'abeille

La dernière topologie présentée parmi les réseaux directs est l'arbre. Une propriété caractéristique des topologies en arbre est que chaque nœud, sauf le nœud racine, possède un parent. L'inconvénient majeur des arbres est que le nœud source et les nœuds adjacents deviennent le goulot d'étranglement du réseau.

Leiserson propose de résorber ce goulot d'étranglement en allouant une bande passante plus importante aux liens situés vers le nœud source notamment en dupliquant les liens. Cette topologie s'appelle le « fat-tree ».

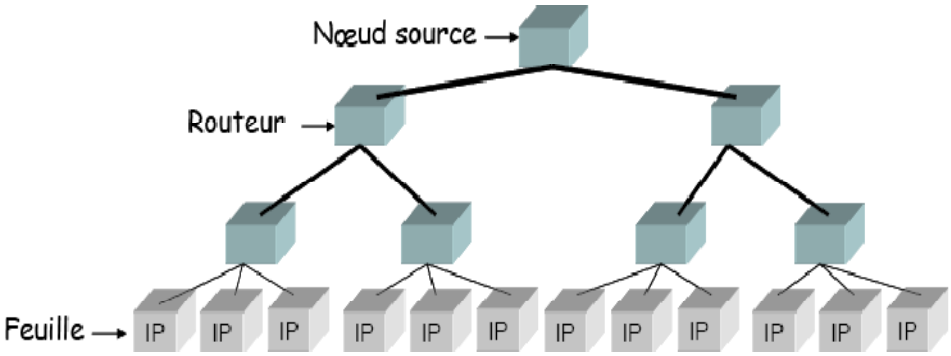


Figure 2.9: Topologie en arbre « fat-tree »

3.3. Réseaux indirects

La notion de réseau direct et indirect provient essentiellement du fait que dans les réseaux indirects le routage des messages est prédéterminé à la source [5]. A titre d'exemple dans les réseaux indirects une des topologies est de connecter tous les nœuds à un seul commutateur (« Switch ») « crossbar » à N entrées et N sorties. Il existe diverses architectures de commutateur « crossbar » dans la littérature.

Nous présentons ici l'arbre Banyan exemple caractéristique de systèmes d'interconnexions « crossbar ». Cette interconnexion est par exemple utilisée dans des réseaux optiques pour les télécommunications est auto-routable (comme tous les commutateurs des réseaux indirects). L'arbre de Banyan a un haut degré de parallélisme. Son inconvénient majeur est qu'il peut y avoir des contentions lorsque deux données veulent traverser le même commutateur simultanément avec des destinations différentes.

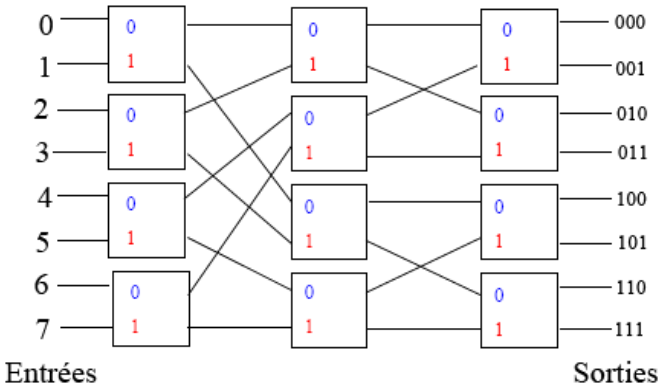


Figure 2.10 : Switch « crossbar » Banyan 8x8

Le coût de ces architectures devient important pour des réseaux de taille importante. Lorsque le nombre de liens augmente, les contraintes physiques limitent l'utilisation de cette topologie, notamment pour des implantations sur silicium.

3.4. Réseaux hybrides

Cette section présente les réseaux hybrides [5] qui regroupent toutes les topologies ne rentrant pas dans les classes décrites ci-dessus. En général, les réseaux hybrides combinent les 3 types de réseaux d'écrits précédemment : ce sont des topologies utilisant des **« ponts ».**

Un réseau hiérarchique est composé de plusieurs sous réseaux pouvant avoir des topologies, des performances et des formats de messages différents. Avec une topologie en anneaux hiérarchiques, chaque sous réseau est connecté à un anneau bidirectionnel. Les sous réseaux sont connectés entre eux par un anneau intermédiaire, comme l'illustre la Figure 2.11.

Une étude compare les performances des topologies maillées et en anneaux hiérarchiques dans les réseaux d'interconnexions. La conclusion de cette étude est que les **réseaux maillés sont plus « scalable » que les réseaux en anneaux hiérarchiques** car il est plus simple d'ajouter un nouveau cœur dans un maillage que dans un cercle.

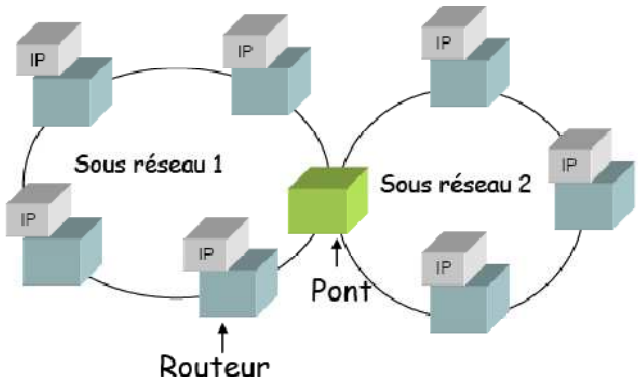


Figure 2.11 : Anneaux hiérarchiques

Il existe de nombreuses topologies différentes et chacune est adaptée à un cahier des charges spécifique. Les réseaux hybrides présentent généralement des performances intéressantes par rapport aux autres réseaux mais ils sont trop souvent dédiés à une application donnée.

3.5 Comparaison des différents types de réseau sur puce

Plusieurs études ont démontré les **limites en performance et en surface** des bus comparativement aux réseaux sur puce directs [11],[10],[9] .

Voici un exemple sur les limitations des bus, nous comparons un bus partagé avec un réseau direct. Considérons un bus idéal, capable d'envoyer un mot par cycle d'horloge et 10.000 mots à transmettre. 10.000 cycles d'horloge sont nécessaires pour transmettre ces données dans un bus idéal. Cependant d'après le Tableau 2-1, un réseau maillé composé de 25 routeurs (5x5) mettra seulement 3.000 cycles d'horloge pour transmettre le même nombre de données. Dans cette situation, ce réseau est 3 fois plus rapide que le bus idéal.

	Réseau Maillé 2 D
Latence Moyenne	195
Latence Minimale	51
Latence Maximale	757
Temps Total en Cycle d'Horloge	2988

Tableau 2.1 : Latences de 10 000 mots dans un réseau maillé à 2 dimensions

Une information pertinente à extraire de ce tableau concerne la latence. Si la latence moyenne est de l'ordre de 200 cycles d'horloge, certaines destinations sont atteintes avec une latence de l'ordre de 750 cycles d'horloge. Cet exemple illustre bien la nécessité d'offrir à ces réseaux une qualité de service. Cet effet peut aboutir à des disfonctionnements sérieux dans le cas d'applications temps réel.

Le Tableau 2-2 présente qualitativement les différentes catégories des réseaux sur puce.

Comme nous l'avons vu dans l'introduction, les systèmes sur puce de demain seront composés de nombreux cœurs. L'interconnexion de ces systèmes devra répondre à des critères de sélection : comme les **performances électriques**, la **surface**, la **puissance consommée** et la **flexibilité**.

A partir de cette première approche portant sur l'analyse des systèmes d'interconnexions utilisés dans les réseaux sur puce, il nous semble que les réseaux directs (maillés) est le type d'interconnexion offrant les meilleurs compromis entre « **scalabilité** » et « **performances** », et c'est ce que nous tacherons à démontrer.

Par la suite, une autre approche d'étude de ce type de réseaux qui consiste à les analyser sous formes de couches respectant le modèle OSI.

Nous ne développerons pas les couches « application » et « physique ». En revanche, les couches « transport », « réseau » et « donnée » sont pleinement détaillées, puisqu'elles établissent les protocoles étudiés dans ce manuscrit. La couche « transport » détaille la qualité de service et le contrôle de flux. La couche « réseau » signifie le type de commutation, l'algorithme de routage et l'emplacement des espaces mémoire. Enfin, la couche « Liaison de Donnée » décrit la synchronisation.

Réseau à médium partagé - bus		Av - Flexible facilement réutilisable
		Inc - Peu extensible, (faible « scalabilité ») - Consommation d'énergie élevée
Réseau direct Routage des données dans le réseau	**Point à point**	Av - Fort parallélisme - Consommation d'énergie optimisée
		Inc - Problèmes électriques - Pas flexible, conçu pour un système particulier
	Réseau sur puce avec routeur	Av **-** Flexible - Fort parallélisme - Bien adapté à un grand nombre de cœurs, extensible - supporte le GALS
		Inc - Manque de maturité (peu d'études de comportement, d'outils de conception)
Réseau indirect Routage des données en entrée	**Switch - Crossbar**	Av - Auto routable - parallélisme
		Inc - Conflits fréquents - Adaptés aux réseaux de petites tailles
Réseau hybride		Av - performances dédiées à l'application
		Inc - dédiée à une application spécifique, pas portable

Tableau 2.2 : Les avantages et les inconvénients des différentes catégories de réseaux sur puce

4. Couche transport

La couche transport [12] détermine les interfaces entre les ressources et le réseau. La couche transport a deux fonctions :

- elle gère l'optimisation des ressources du réseau en fournissant par exemple une qualité de service.
- elle régit les communications grâce à un contrôle de flux et un protocole de synchronisation adaptés au réseau.

4.1. Interfaces ressources – routeurs

Les interfaces entre la ressource et le routeur [13] améliorent le critère de flexibilité du réseau sur puce et plus particulièrement son degré d'adaptabilité à des systèmes différents.

Certaines interfaces limitées à des modèles de communication précis ne suffisent pas à rendre le système flexible. Ces interfaces adaptent le périphérique au réseau de communication lorsque les protocoles de cette unité matérielle sont incompatibles avec ceux du médium. Elles séparent matériellement la fonction « calcul » de la fonction « communication ».

La fonctionnalité des interfaces est simple, elles traduisent les messages des ressources en données compréhensibles pour l'interconnexion. Cette simplicité leur permet de consommer peu d'énergie. Dans la littérature des systèmes sur puce, nous trouvons plusieurs standards d'interfaces libres de droit dont : le VCI (« Virtual Component Interface »), l'OCP (« Open Core Protocole »), le wishbone et le AXI de ARM.

Le standard VCI a été normalisé par le consortium VSIA (Virtual Socket Interface Alliance). Il introduit un découplage complet entre la conception du réseau et le choix du composant système (microprocesseurs, mémoire, DSP…). Le contrôle de flux par VCI facilite la conception des interfaces.

L'OCP-IP (OCP International Partneship Association Inc) a été formé pour promouvoir et développer le standard OCP afin de faciliter la réutilisation et réduire le temps de conception des ressources (bloc IP).

4.2. Qualité de service

Les futurs systèmes convergent vers une augmentation de l'hétérogénéité. Ces systèmes deviennent de plus en plus dynamiques et les algorithmes de communication doivent répondre à des contraintes de plus en plus importantes (rapidité et efficacité). La question principale est donc de savoir comment concilier l'aspect dynamique du système d'interconnexion (imprévisible selon le cas) avec des contraintes temps réel liées.

Nous pouvons définir la qualité de service comme l'assurance de performances bornées sur un lien donné.

La gestion de qualité [5] de service peut être définie selon le type de commutation utilisé dans un réseau :

- la commutation de circuit : assure une qualité de service de façon implicite puisque la latence des paquets et la bande passante du circuit sont prévisibles et assurées.
- la commutation de paquet : un protocole doit être mis en place pour assurer cette qualité de service.

4.3. Contrôle de flux

Son rôle est de gérer la communication entre deux nœuds voisins [5], autorisant ou stoppant l'évolution des données dans le réseau. Lorsque le « buffer » d'un routeur est plein, le mécanisme de contrôle de flux stoppe la transmission des données. Lorsque le blocage se résorbe et que le « buffer » se libère, la transmission reprend son cours.

Le paquet ou le message sont généralement partitionnés en unité de taille fixe, appelée mot dont le transfert doit être synchronisé. Cette unité est définie comme la plus petite unité d'information pouvant transporter des requêtes ou des acquittements. Les signaux de requêtes et d'acquittement sont utilisés pour assurer le succès d'un transfert et la disponibilité des buffers du récepteur. Le contrôle de flux intervient dans la majorité des cas au niveau de cette unité d'information.

5. Couche réseau

La couche réseau est liée aux protocoles de contrôle du réseau. Nous présentons ci-dessous tous les aspects reliés à cette couche.

5.1. Conflits

Dans les réseaux directs, les messages traversent de nombreux nœuds intermédiaires avant d'atteindre leur cible. Il peut arriver que des données ne puissent pas atteindre leur destination même s'il n'y a aucune erreur d'interconnexion entre la source et la cible cela correspond à un conflit. Chaque message qui n'est pas arrivé à destination est alors susceptible de rencontrer un conflit. Trois classes de conflits classiques intervenant dans les réseaux sur puce : le « deadlock », le « livelock » et le « starvation ».

Ces conflits se produisent parce que le nombre de ressources est limité. En fonction des limitations de l'arbitre, de l'algorithme de routage ou des paramètres du réseau, les conflits sont différents. Le Tableau 2-3 regroupe la manière dont se manifestent les conflits et les ressources auxquelles ils sont liés.

	Effets	Causes
Livelock	Paquets circulent dans le réseau sans atteindre leurs	Algorithme de routage non déterministe
Deadlock	Paquets bloqués dans des buffers.	Paramètres du réseau (taille des paquets et des buffers, nombre de canaux)
Starvation	Paquets bloqués dans des buffers. Toujours les même paquets.	Mauvaise gestion des priorités de la part de l'arbitre

Tableau 2.3 : Les effets et les causes des conflits

La probabilité que ces conflits se produisent, augmente avec la charge du réseau et diminue avec l'augmentation de taille des buffers.

5.2. Commutation de circuit

Dans une commutation de circuit [5], un lien physique entre une source et une cible est réservé pour la transmission de données. L'en-tête du message à transmettre contient l'adresse de destination et des informations de contrôle. Lorsque cet en-tête est routé jusqu'à sa destination, un chemin complet est réservé et un acquittement est renvoyé à la source. Qu'il y ait des données ou non à transmettre sur la ligne, la communication est établie jusqu'à ce que l'un des utilisateurs raccroche. Dans les communications point à point, la commutation de circuit est prédominante. Un diagramme espace temps de l'acheminement d'un message à travers trois liens est montré Figure 2.12.

Tr: temps de routage
Tbloc: temps d'attente de la libération de la sortie
Tres: temps de réservation du chemin
Ttrans: Temps de transmission des données

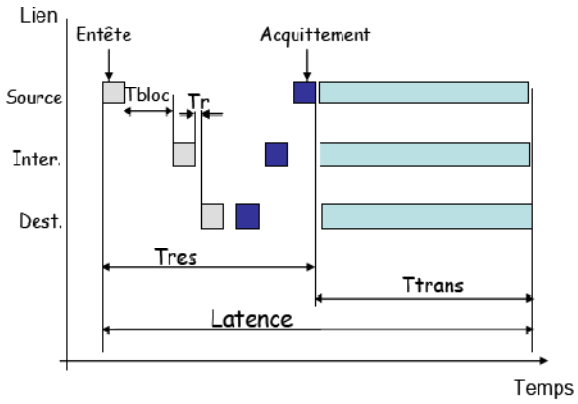


Figure 2.12 : Diagramme espace temps d'un paquet acheminé en commutation de circuit

La commutation de circuit est aussi appelée « bande passante garantie » (« guaranteed throughput » ou GT). Lorsque le message est acheminé, il utilise toute la bande passante du canal réservé et assure ainsi la qualité de service. La commutation de circuit est intéressante lorsque les messages sont de taille importante et peu fréquents. L'inconvénient de la commutation de circuit est qu'un message réservant un chemin physique empêche les autres messages d'emprunter ce chemin. Plusieurs autres messages peuvent ainsi rester bloqués.

5.3. Commutation de paquet

Avec la technique commutation de paquet, si le message à transmettre a une taille importante, il est alors partitionné et transmis en paquets de taille fixe. Chaque paquet est routé individuellement. L'entête du paquet contient des informations sur le routage du paquet et un contrôle des données suivantes. Les données utiles sont la charge du paquet, comme l'illustre la Figure 2.13.

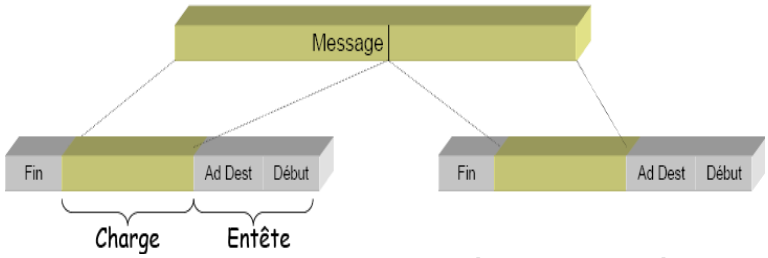


Figure 2.13 : Message partitionné et encapsulé en paquets

Contrairement à la commutation de circuit, la commutation de paquet [5] utilise le lien uniquement lorsqu'il y a des données à transmettre. Plusieurs paquets d'un même message peuvent circuler simultanément dans le réseau même si le premier paquet n'est pas arrivé à destination. Cette méthode est rapportée à la technique d'acheminement des données via le réseau internet.

L'inconvénient de cette méthode est qu'elle nécessite plus de temps que la commutation de circuit. La source et la cible doivent respectivement partitionner et reformer le paquet d'un message. De plus, chaque paquet est routé individuellement aux nœuds intermédiaires et les algorithmes de routage peuvent prendre du temps aussi. Nous décrivons ci-dessous différentes façons d'acheminer les paquets avec une commutation de paquet.

5.3.1. Les types d'acheminements possibles

Avec l'acheminement « stocke puis renvoie » (store and forward), le paquet est stocké entièrement dans une mémoire tampon (ou un buffer) avant d'être renvoyé vers un autre nœud en direction de sa cible. L'en-tête du paquet est extrait par le routeur intermédiaire et utilisé pour déterminer le port de sortie par lequel le paquet sera renvoyé. Sur le diagramme Figure. 2.14, nous voyons que la latence du paquet est proportionnelle au nombre de routeurs traversés entre la source et la cible, en effet l'algorithme de routage et la transmission du paquet prennent un certain temps qui augmente en cas de conflit. De plus, les buffers doivent avoir une taille suffisante pour contenir un paquet entier. En revanche, ce type d'acheminement est particulièrement peu touché par les « deadlocks » grâce à la taille des buffers.

Tr: temps de routage
Tbloc: temps d'attente de la libération de la sortie
Ttrans: Temps de transmission du paquet

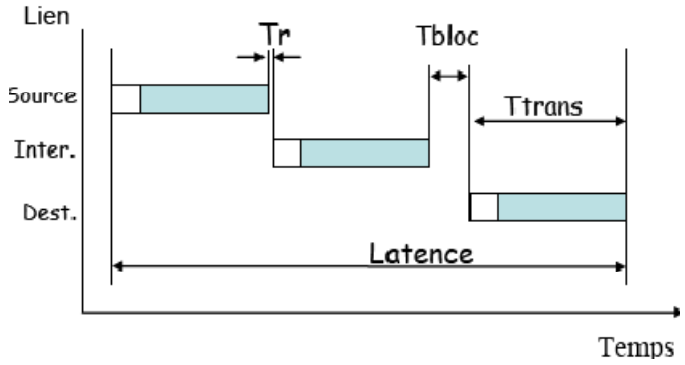


Figure 2.14 : Diagramme espace temps d'un paquet acheminé en « stock et renvoi »

Afin de limiter la latence du paquet, l'acheminement « Virtual cut through » (passe virtuellement à travers) dans lequel l'en-tête du paquet est traité dès qu'il est reçu. Le routeur renvoie l'en-tête et les données suivantes dès que le routage est déterminé et uniquement si la destination est disponible. Le paquet peut être acheminé avant la réception de toutes les données. En absence de conflit, la latence est déterminée par le temps d'exécution de l'algorithme de routage dans chaque nœud et le temps de transmission à travers le canal. Le message est alors « pipeliné » à travers les routeurs successifs. Si l'en-tête est bloqué dans un routeur dont le canal de sortie est occupé, le paquet complet est alors stocké dans le buffer du nœud. Les buffers doivent donc pouvoir conserver un paquet complet en cas de conflit.

Tr: temps de routage Tbloc
Tbloc: temps d'attente de la libération de la sortie

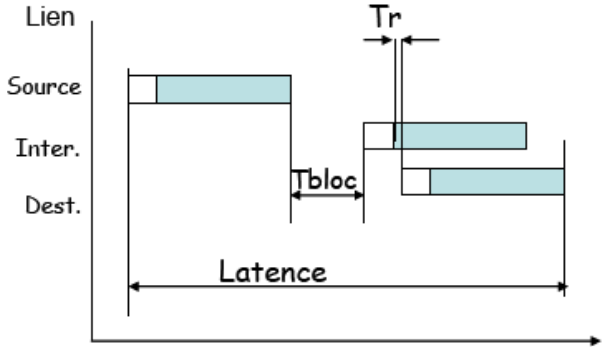
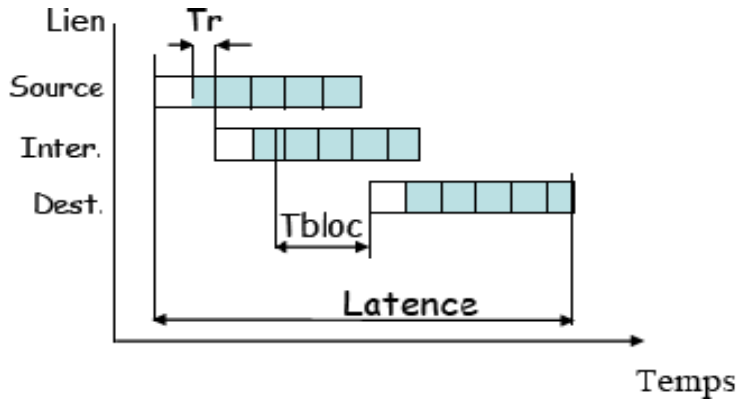


Figure 2.15 : Diagramme espace temps d'un paquet acheminé en Virtual cut through

Les buffers nécessaires pour stocker un paquet entier rendent difficile l'intégration du circuit. Avec un acheminement en « wormhole », les paquets sont découpés en mots de taille fixe, l'en-tête est routé puis les mots de données suivent le même chemin

indépendamment les uns des autres. C'est-à-dire que lorsque l'en-tête est stocké dans un buffer, les mots de données se bloquent dans les buffers des routeurs du chemin emprunté par l'entête. De plus, il est important de noter que les buffers utilisés peuvent stocker quelques mots mais pas le paquet en entier. En cas de conflit, le paquet est bloqué sur place et peut ainsi occuper des buffers de plusieurs routeurs. Ce qui favorise l'apparition des conflits « deadlock ».



Tr: temps de routage
Tbloc: temps d'attente de la libération de la sortie

Figure 2.16: Diagramme espace temps d'un paquet acheminé en "wormhole"

Le mécanisme du facteur fou (« Mad Postman ») est possible avec un routage déterministe dans un réseau à 2 dimensions. Le premier mot d'en-tête contient l'adresse de destination dans une dimension (X par exemple). Quand le paquet atteint ce nœud, le premier mot d'en-tête est écarté et le message est renvoyé suivant une autre dimension (Y) dont l'adresse est inscrite dans le deuxième mot.

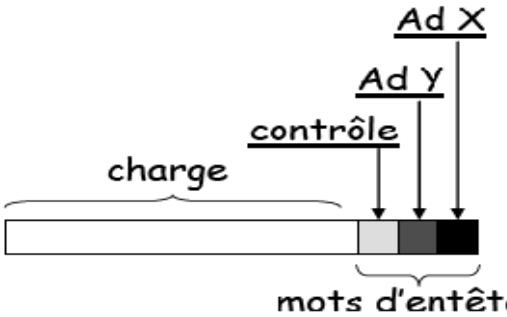


Figure 2.17: Format d'un paquet en postier fou

En « Virtual cut through » ou en « wormhole », l'adresse de destination est gardée jusqu'à ce que le paquet soit arrivé à destination. Le mode « facteur fou » est le type d'acheminement le plus rapide au niveau du routeur. C'est la méthode pour laquelle le paquet reste le moins longtemps dans un routeur. L'acheminement des données se fait bit par bit.

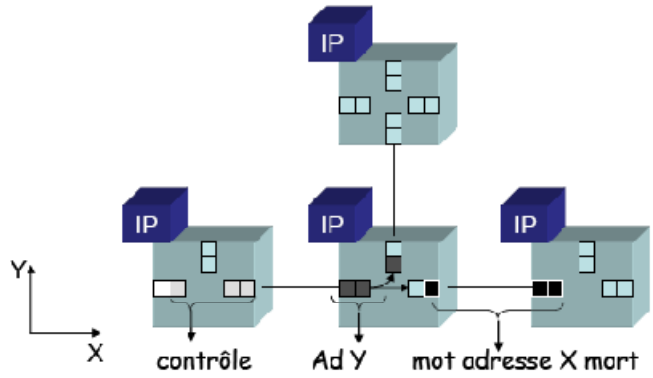


Figure 2.18: Acheminement d'un paquet en mode "postier fou" lors d'un changement de direction

5.4. Algorithmes de routage

Les algorithmes de routages établissant le chemin emprunté par chaque paquet ou message pour rejoindre sa destination. Nous présentons ici trois types de classes d'algorithmes de routage adaptables à différentes topologies :

5.4.1. Déterministe

Établissent toujours le même chemin des paquets entre chaque paire de nœuds en fonction de l'adresse de destination. Le paquet ne peut effectuer qu'un seul changement de direction (Figure. 2.19). Le routage déterministe est devenu très populaire lors de l'invention de l'acheminement en « wormhole » [9]. Il permet de garder le routage en matériel aussi compact et rapide que possible.

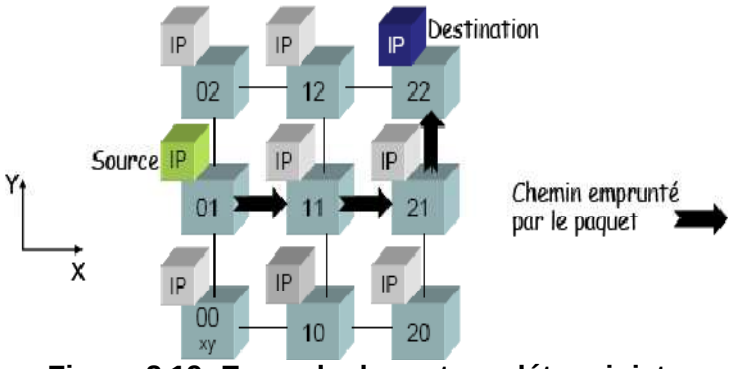


Figure 2.19: Exemple de routage déterministe

L'algorithme de routage déterministe pour les topologies régulières est simple. Une simple machine d'état est suffisante, permettant ainsi une intégration compacte et rapide. De plus, il évite les conflits de type « livelock ». En revanche, il n'est pas très performant lorsque le trafic n'est pas uniforme, spécialement lorsque les paires de nœud non voisins s'échangent des données.

5.4.2. Semi-adaptatif

Ce routage représente un compromis entre la flexibilité et le faible coût : il utilise la flexibilité du routage adaptatif et le coût modéré en complexité du routage déterministe. Contrairement au routage déterministe où le paquet est stocké dans les

buffers en attendant la libération d'un lien, le routage semi adaptatif permet au paquet de contourner un conflit. Le paquet ne peut pas circuler librement dans le réseau et l'algorithme l'empêche de faire demi-tour. Ceci permet d'éviter les « livelock».

Dans le premier cas illustré Figure 2.20, nous considérons la présence d'un conflit sur le port Ouest du nœud 21. Grâce à l'algorithme de routage semi adaptatif, le paquet est capable de contourner le conflit via le nœud12. En revanche, en considérant le deuxième cas Figure 2.21, avec également des conflits sur les ports sud des routeurs 12 et 21, le paquet ne peut pas contourner ces conflits en passant par le routeur 20, puisqu'il devrait alors faire demi-tour et ce routage l'en empêche. Le paquet attend la fin des conflits dans les espaces mémoires disponibles.

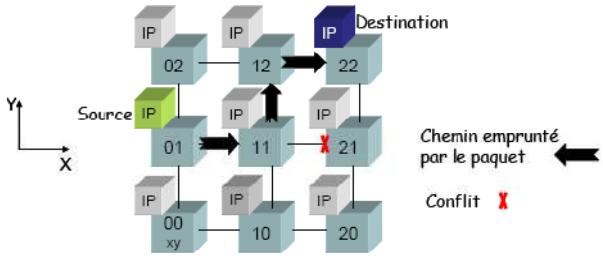


Figure 2.20: Premier cas de routage semi adaptatif

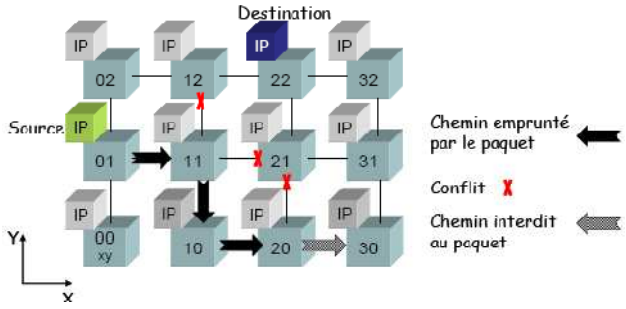


Figure 2.21: Deuxième cas de routage semi adaptatif

5.4.3. Adaptatif

Cet algorithme s'adapte facilement à toutes les topologies. Les paquets peuvent y circuler librement sans restriction. Un paquet peut contourner tous les conflits qu'il croise. L'inconvénient de cette technique est qu'elle est complexe à mettre en place et qu'elle engendre des conflits « livelock ». En effet, les paquets peuvent tourner dans le réseau en occupant les ressources sans trouver leur destination.

5.5. Emplacement des espaces mémoire

Les buffers sont des FIFOs («First In First Out ») stockant les messages ou les paquets en transit. Ils sont indispensables au bon routage des paquets. Différentes d'architectures sont présentées dans la section suivante.

5.5.1. FIFO en entrée

Le buffer en entrée est la plus simple des stratégies de mémorisation : les paquets attendent dans des FIFOs dans les ports d'entrées du routeur avant d'être routé. Il faut N buffers pour un routeur de N ports.

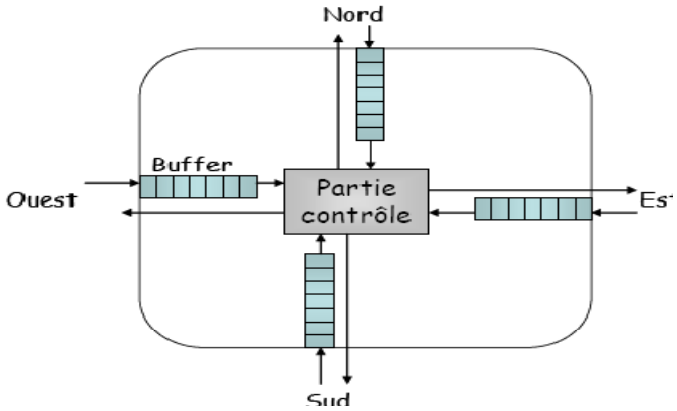


Figure 2.22: Routeur avec des buffers en entrée

5.5.1. FIFO en sortie

Avec une stratégie de mémorisation en sortie (« output queuing »), les paquets sont mémorisés après avoir été routés. Il y a un buffer sur chaque route entre une entrée et une sortie. Ils sont localisés sur les ports de sortie, juste avant un multiplexeur qui sert d'arbitre comme l'illustre la Figure 2.23. Les fils d'interconnexions deviennent importants et chaque port nécessite son propre aiguilleur (en entrée) et son arbitre (en sortie).

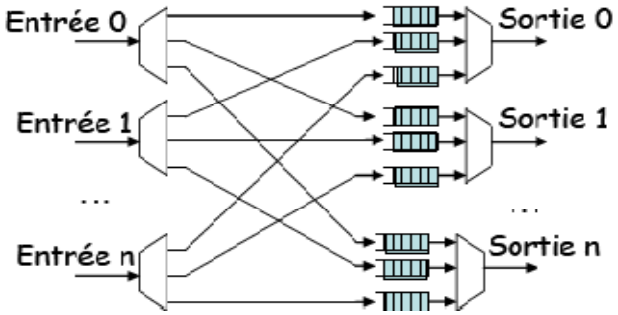


Figure 2.23: Routeur avec des buffers en sortie

Les buffers en entrée présentent un problème fondamental appelé « *head-of-line* » qui est le suivant: si le premier paquet d'une file d'attente est bloqué par un conflit de ressource, les paquets suivants sont aussi bloqués même s'ils pourraient être routés vers des sorties disponibles. Avec des buffers en sortie le temps d'attente est beaucoup moins long car le système ne rencontre pas ce problème. Les buffers en sortie présentent donc de meilleures performances que les buffers en entrée. Cette théorie a été quantifiée pour des réseaux non intégrés sur puce. Dans le but d'obtenir de meilleures performances, d'autres types de mémorisation ont été proposés tels que des buffers en entrée et en sortie ou plusieurs buffers en parallèle en entrée (« virtual output »).

6. Couche liaison de données

La couche liaison donnée permet de définir les protocoles des liens physiques entre les routeurs afin d'assurer le transfert des données entre deux routeurs. Les deux protocoles définis dans la couche donnée sont le code correcteur d'erreur et la gestion de la synchronisation.

6.1. Codes de détection et de correction d'erreur

Les protocoles de détection et de réparation d'erreur au niveau du paquet ont été développés pour des réseaux traditionnels et s'appliquent aux réseaux sur puce, sans négliger la consommation d'énergie supplémentaire qu'ils apportent [10]. Ils permettent de vérifier que les données reçues sont bien celles envoyées. Les deux techniques que sont le « code de correction d'erreur » et le « code de détection d'erreur avec retransmission » ajoutent de la redondance au transfert d'information. Le code de correction d'erreur permet de détecter une donnée erronée puis de retrouver sa bonne valeur sans avoir à retransmettre tout le paquet. Cette technique est généralement plus complexe et nécessite plus de redondance que le code de détection. En effet, le code de détection d'erreur permet juste de détecter une donnée erronée puis le protocole prévoit de demander à nouveau la transmission du paquet.

En fonction du réseau, cette technique peut représenter un surcoût en consommation d'énergie et peut obliger le réseau à émettre plusieurs fois les mêmes paquets.

7. Etat de l'art. Quelques exemples de réseaux sur puce

Il existe donc de nombreuses architectures de réseaux sur puce. Nous avons choisi de présenter plus en détail des réseaux sur puce caractéristiques et représentatifs d'origines [1], [2], [5], universitaires ou industriels avec des topologies très différentes.

7.1. Les réseaux intégrés industriels

Les solutions industrielles actuelles sont fondées sur l'utilisation de bus (réseau partage). Les différentes solutions proposées sont issues des grands fabricants précurseurs dans la réalisation de SoC. Elles sont le fruit d'un effort de standardisation, au sein de chacune de ces sociétés, vers l'élaboration d'un NoC (un bloc IP consacre à l'échange de données).

7.1.1. Le STBus

Le STBus est un réseau d'interconnexion pour SoC (Figure. 2.25) développé par la société ST Microelectronics.

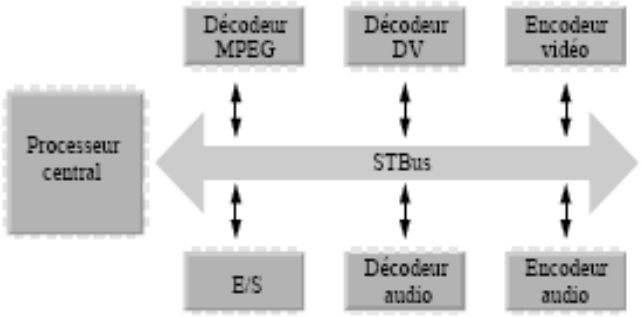


Figure 2.24 : Utilisation classique du STBus dans un système sur puce.

La brique de base est un nœud ("***node***") qui permet d'interconnecter localement des initiateurs et des cibles (une vingtaine). Plusieurs nœuds peuvent être associes pour réaliser un réseau hybride qui présente deux versions différentes : le STBus et le

ST Bus, interconnectes entre eux). Il permet donc de construire un système de communication hiérarchique.

ST Microelectronics a de plus développé un environnement de conception permettant de synthétiser le réseau de communication. Le protocole de communication OCP est compatible avec le STBus.

Les spécifications du STBus n'étant actuellement pas mises à disposition par la société ST Microelectronics, il sera difficile d'évaluer ses performances. De plus, le classement du STBus dans les interconnexions de type bus partage n'est pas immédiat, car il pourrait aussi être classé dans les réseaux d'interconnexion à cause de sa hiérarchie de bus.

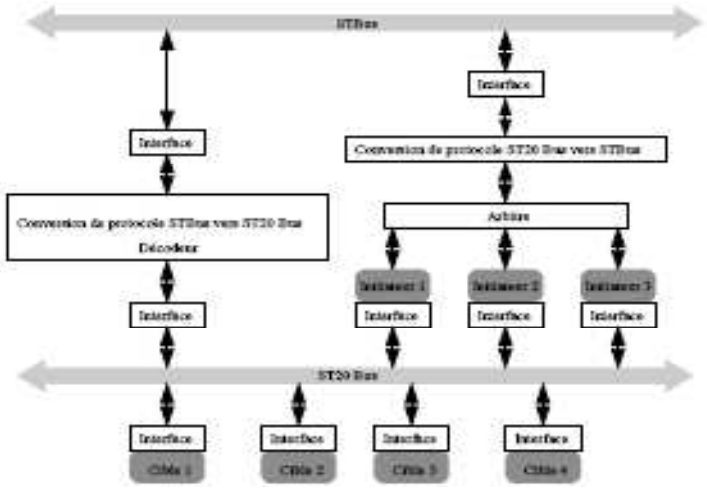


Figure 2.25 : Exemple d'utilisation hiérarchique du STBus.

7.1.2. Le bus AMBA

C'est un système d'interconnexion développé par la société ARM, il est devenu, lors de ces dernières années, un "standard" très répandu. En effet, il est doté de performances "raisonnables" en termes de débit global (de l'ordre de quelques Gbit/s).

A l'instar de nombreuses architectures, le bus AMBA est en réalité compose de deux bus:

- Le bus AHB (anciennement ASB) qui permet d'interconnecter des ressources en maximisant le débit.
- Le bus APB, utilise pour les périphériques qui nécessitent un débit plus faible. Une passerelle permet d'interconnecter ces deux bus.

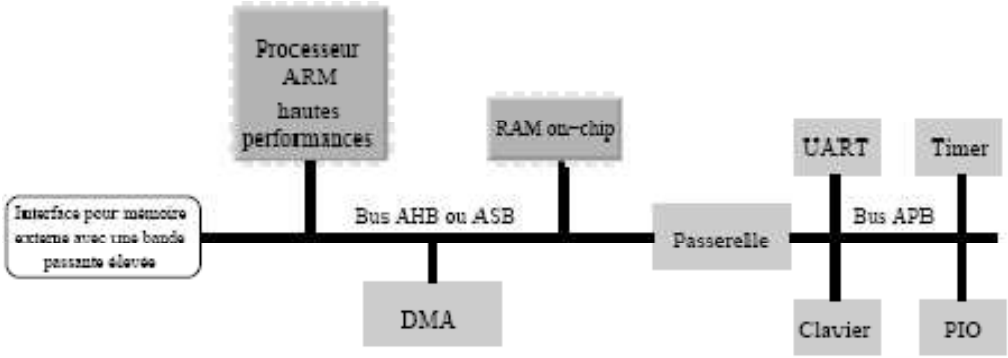


Figure 2.26: Architecture typique d'un bus AMBA.

Kit de conception : Un kit de développement, ADK, fournit un environnement spécifique facilitant la conception rapide d'un SoC construit autour du bus AMBA. Ce kit contient une bibliothèque de nombreux blocs IP (ARM développe en effet de nombreux blocs numériques comme des DSP, des mémoires et des blocs IP spécialises). Un environnement graphique permet d'associer les blocs et des méthodes automatiques de test qui permettent d'évaluer les performances du système.

Spécifications : Le bus AMBA possède son propre protocole de communication ; ce protocole étant public, il est possible a une équipe de concepteurs d'adapter un nouveau bloc IP au bus AMBA grâce a un *wrapper.*

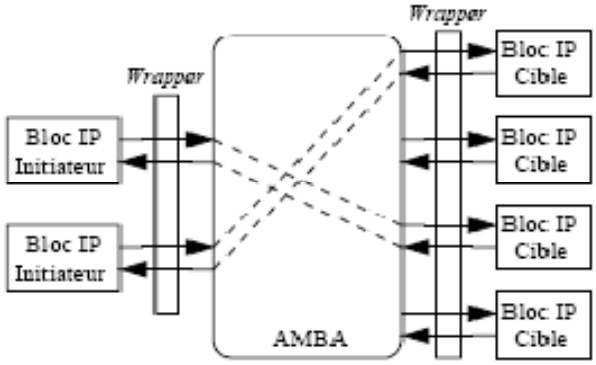


Figure 2.27 : Concept du réseau d'interconnexion "multicouches" de ARM.

Les principales spécifications du bus AMBA sont les suivantes :

- L'architecture est scalable jusqu'à une dizaine d'initiateurs.
- L'arbitrage est pris en charge par le bus.

Le bus AHB autorise la communication simultanée de plusieurs initiateurs (à condition que leurs cibles soient différentes, cf. Figure. 2.28). A titre d'exemple, la figure 2.29 présente une implémentation simple de l'interconnexion (appelé aussi "matrice d'interconnexion") dans un système possédant 2 initiateurs et 2 cibles.

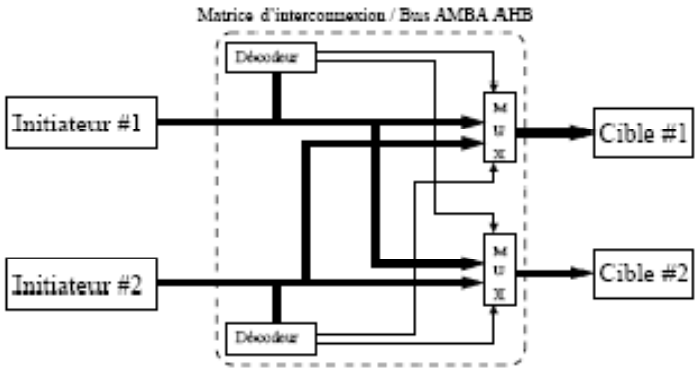


Figure 2.28 : Matrice d'interconnexion reliant 2 initiateurs et 2 cibles.

Le bus AHB autorise des opérations de type pipeline, des transferts de données en mode rafale, des transactions de type *split transaction*.

Le bus AHB associe les composants suivants :

- Des initiateurs AHB (ou "*bus master* ") capables d'émettre une requête (lecture et écriture) et de fournir l'adresse et le contrôle.
- Des cibles AHB (ou "*bus slave") rép*ondent aux requêtes (en écriture ou en lecture) dans un espace d'adressage donne. .

– L'arbitre AHB (ou "*bus arbiter* ") gère les requêtes en provenance des différents initiateurs.

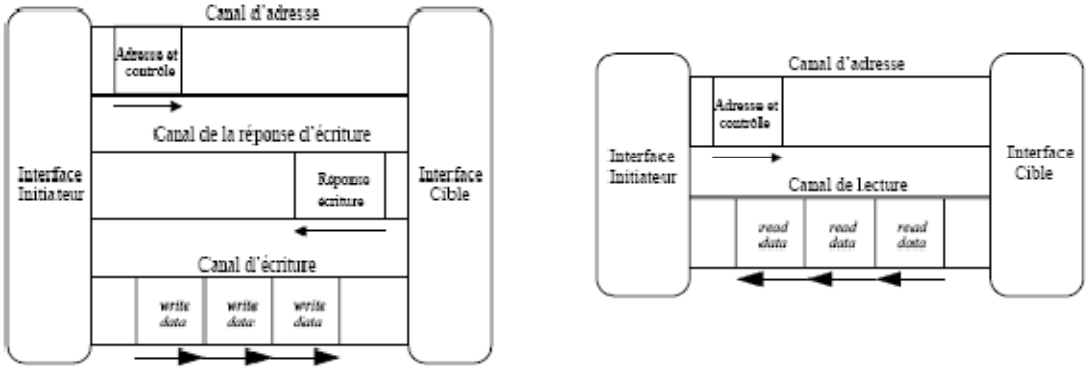
– Le décodeur AHB décode l'adresse de chaque requête et fournit un signal de sélection à la cible visée.

Contrairement au bus AHB qui est utilise pour des blocs nécessitant un fort débit (processeur, bloc mémoire intègre, bloc mémoire externe, périphérique DMA, etc.), le bus APB est recommande pour gérer des communications avec des registres, des périphériques a faibles débits ou pour grouper des périphériques à faible largeur de bus pour éviter une sur-occupation du bus AHB.

Protocole de communication : Le protocole de communication optimise les ressources matérielles du bus. ARM a récemment introduit le protocole AXI.

Les principales caractéristiques de ce protocole de nouvelle génération sont les suivantes:

- Les canaux de données et d'adresses/contrôles sont sépares.
- Les canaux d'écriture et de lecture sont sépares (DMA).
- Il est possible d'ajouter sur le bus des registres supplémentaires pour définir une zone asynchrone.
- Il est possible de gérer des transactions désordonnées (ou *out of order)*.



(a) Architecture des canaux d'écriture (b) Architecture des canaux de lecture

Figure 2.29 : Protocole AXI.

7.1.3. Le bus SiliconBackplane III

C'est un système d'interconnexion développé par la société Sonics, il est spécialement conçu pour les applications multimédia. Ce réseau d'interconnexion est le premier réseau à avoir été commercialise "clé en main" avec une plate-forme de développement rapide et efficace ce qui en fait un succès auprès de nombreux industriels (grâce a une bibliothèque complète de plusieurs blocs IP).

SiliconBackplane utilise le protocole de communication OCP, la Figure. 2.31 En montre une vue générale.

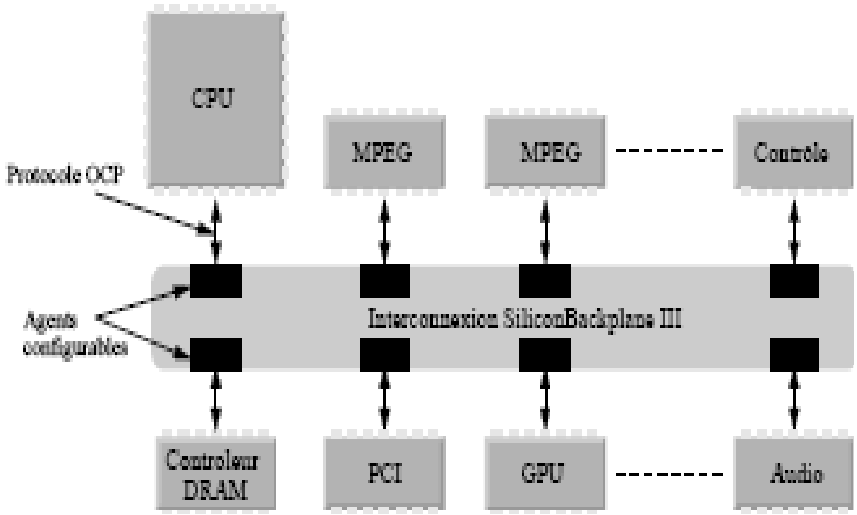


Figure 2.30 : Implémentation possible de l'interconnexion intégrée SiliconBackplane III.

D'après Sonics, le système SiliconBackplane possède les caractéristiques suivantes :

- Des estimateurs permettent d'analyser les performances globales (QoS) du réseau
- Le réseau est configurable et scalable ;
- Il est possible de connecter de la mémoire externe (DRAM).
- L'architecture de communication est distribuée ce qui facilite le "*floorplanning*" des blocs IP.
- Le "Multicast" et/ou "broadcast" sont possibles.
- Le "Multithread" est garanti (dans la définition établie par Sonics, il est possible que plusieurs initiateurs du système communiquent simultanément sur le réseau), mais les spécifications ne précisent pas si un arbitre est nécessaire pour ce fonctionnement.

Le réseau d'interconnexion SiliconBackplane est reconnu et est très utilise dans le monde industriel des lors que les applications visées concernent le multimédia et que les débits restent raisonnables (quelques centaines de Mo/s).

Sonics propose une version de ce bus, SonicsMX, pour des applications mobiles (PDA, téléphonie mobile 3G) caractérisées par une consommation réduite.

7.1.4. Le bus CoreConnect

Développé par la société IBM; son principe de fonctionnement est similaire a celui du bus AMBA. En effet, CoreConnect est constitue de deux "sous-bus" : le bus PLB a haut débit, et le bus OPB pour les ressources nécessitant un débit plus faible et permettant de désengorger le bus principal PLB. L'architecture de l'interconnexion CoreConnect contient donc une passerelle entre ces deux bus. Le bus DCR est un bus spécifique qui permet de synchroniser certaines fonctions (processeurs, mémoires, etc.) qui sont interconnectées au bus CoreConnect. Un kit de développement est disponible pour développer les applications utilisant CoreConnect.

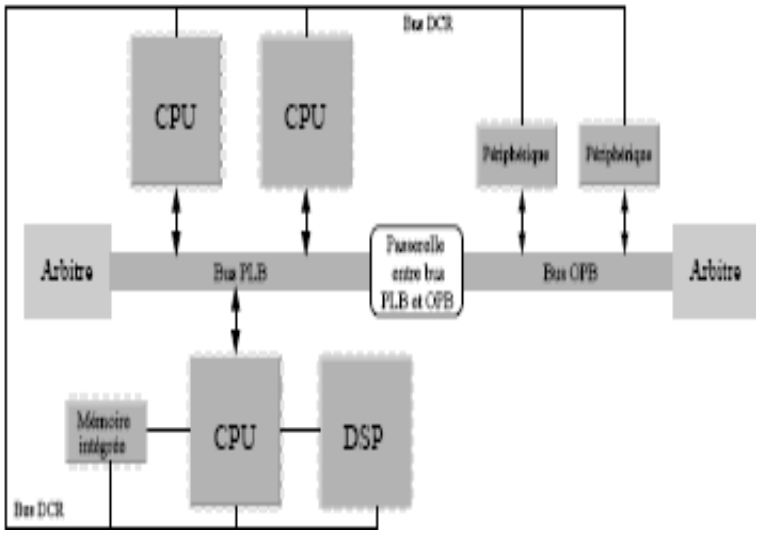


Figure 2.31 : Architecture possible d'un système sur puce intégrant un bus CoreConnect.

Le bus PLB est complètement synchrone et permet d'interconnecter jusqu'à 8 initiateurs. Les bus d'adresses de lecture et d'écriture sont sépares (ceci permet l'utilisation des "*split transactions*" ce qui augmente la bande passante disponible sur le réseau). Le mode rafale et le "pipelinage" sont supportes. L'arbitrage permet de gérer la bande passante disponible grâce à des algorithmes de gestion complexes.

Le bus OPB est totalement synchrone, il comprend un bus d'adresses et un bus de données. Le bus OPB dispose d'un arbitre qui peut gérer jusqu'à 4 initiateurs.

CoreConnect possède un troisième "sous-bus" : le bus DCR. Il permet de synchroniser et de transmettre des données entre les registres généraux (des ressources. En gérant la configuration des registres, le bus DCR réduit la "charge" du bus PLB, il est complètement synchrone.

Tableau 2.4 : Performances annoncées par IBM pour différentes architectures du bus CoreConnect.

Paramètres	Core connect 32	Core connect 64	Core connect 128
Grandeur du Bus PLB (bits)	32	64	128
Fréquence max (MHz)	66	133	189
Bande passante max (Mo/s)	264	800	$2{,}9x2^{10}$

7.1.4. Le bus Avalon

Conçu et développé par la société Altera. Avalon est une architecture de bus configurable synchrone (8, 16 ou 32 bits) en fonction des ressources qui y sont connectées. Une requête est traitée en un cycle d'horloge. La Figure 2.33 montre un exemple d'utilisation de ce bus.

Avalon est capable de gérer plusieurs initiateurs, l'arbitre est directement encapsulé dans le bus. Plusieurs initiateurs peuvent communiquer simultanément, s'ils ne font pas appel à la même cible au même temps de cycle du bus. Les adresses (codées sur 32 bits), les données et les signaux de contrôle ont leurs propres canaux.

Des transmissions de type "*streaming*" sont possibles avec le bus Avalon ; de plus, l'arbitre peut allouer à un couple initiateur/cible exigeant un surplus de bande passante pendant une période transitoire.

Le bus Avalon est souvent utilise dans les FPGA de Altera configures en processeur NIOS par exemple.

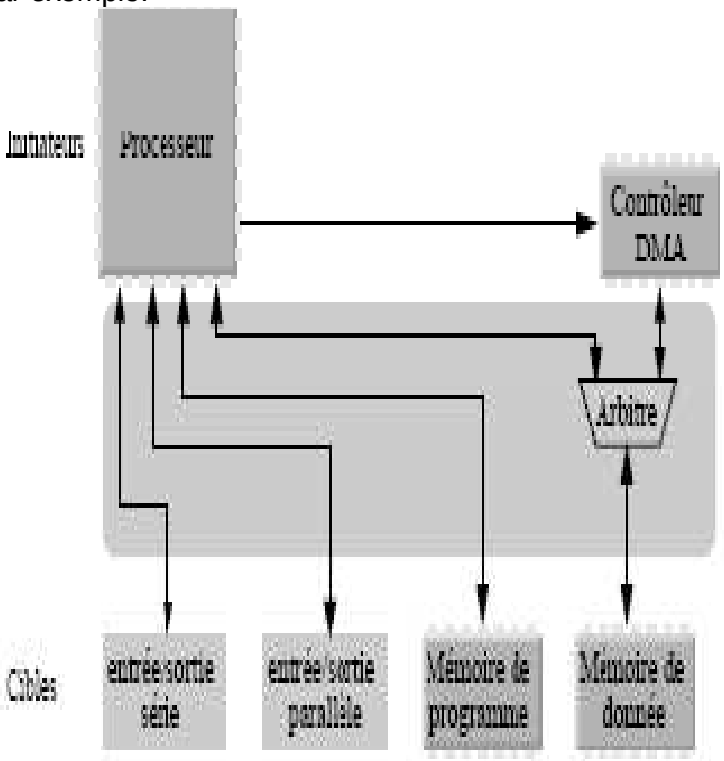


Figure 2.32 : Exemple d'architecture du bus Avalon "multi-initiateurs".

Un kit de développement (SOPC Builder) est disponible, il permet d'optimiser les communications et de vérifier le fonctionnement d'une transaction.

7.1.5. Résumé des spécifications des bus et des réseaux partagés

Les tables 2.5 et 2.6 résument les différents bus partagés présentés précédemment en indiquant les principales caractéristiques fournies par les sociétés qui les développent. L'abréviation "*NC*" signifie que la caractéristique concernée n'est pas communiquée.

Tableau 2.5 : Résumé des principales spécifications des bus partages - série 1 (_ : oui - X : non).

nom	AMBA AHB	AMBA ASB	AMBA APB	CoreConnect PLB	CoreConnect OPB
société	ARM	ARM	ARM	IBM	IBM
année de dévelloppement	1999	1999	1999	1999	1999
protocole	AXI	AXI	AXI	natif	natif
plate-forme	ADK	ADK	ADK	Bus Model Toolkits	Bus Model Toolkits
initiateur	Multiple	Multiple	1	8	4
largeur maximale (add.)	1024(&32)	256(&32)	32(&32)	256(&32)	32(&32)
débit maximal	*NC*	*NC*	*NC*	*2.9 Go/s (128bits)*	*NC*
bus pipeliné	_	_	x	_	x
mode rafale	_	_	x	_	_

Tableau 2.6 : Résumé des principales spécifications des bus partages - série 2 (_ : oui - X : non).

nom	STBus	SiliconBackplane III	SonicSMX	Avalon	PI-Bus	FP1	SuperHywy
société	STM	SonicS	SonicS	Altera	OMI	Infinoon	SuperH
année de dévelloppement	2004	2002	2004	2003	1996	2001	2003
protocole	natif	OCP	OCP	SPI	natif	natif	VCI
plate-forme	*NC*	SonicSStudio	SonicSStudio	SOPC Builder	*NC*	*NC*	AEDK
initiateur	Multiple	Multiple	Multiple	Multiple	Multiple	Multiple	Multiple
largeur maximale (add.)	*NC*	128	*NC*	32(&32)	64(&32)	32(&32)	1024
débit maximal	*NC*	*4Go/s*	*NC*	*NC*	*5 Go/s (64bits)*	*NC*	64 bits
bus pipeliné	*NC*	*NC*	*NC*	*NC*	_	_	_
mode rafale	*NC*	_	_	_	_	*NC*	*NC*

7.2. Les réseaux directs et indirects

Les limitations des réseaux d'interconnexion bases sur les bus ont conduit de nombreux laboratoires universitaires à étudier de véritables réseaux intègres (NoC). Dans cette partie, sans prétendre être exhaustif, nous décrirons les principales architectures proposées.

Les SoC du futur pourront compter quelques dizaines, voire quelques centaines, de ressources. Les architectures à base de bus sont inadaptées pour interconnecter ces ressources. Un NoC est constitue de nœuds relies avec quelques plus proches voisins (les interconnexions entre les nœuds du réseau sont appelées des bus).

Si chaque nœud est associe a une ressource, le réseau est dit direct ; dans le cas contraire, il est dit indirect. En général chaque nœud est associe a un nombre de ressources faible (inférieur a 10).

Un réseau sera en général conçu pour être scalable, lorsque le nombre de ressources à interconnecter augmente, les ressources matérielles de l'interconnexion (et sa complexité) augmentent, ainsi que le débit maximal du réseau.

En général, les études menées sur les architectures de NoC s'accompagnent du développement de plate-forme permettant d'estimer leurs performances.

7.2.1. Arteris NoC

Son architecture est développée par la société Arteris, première société à commercialiser un réseau intégré. La Figure 2.34 présente la vue simplifiée d'une application possible utilisant ce réseau d'interconnexion

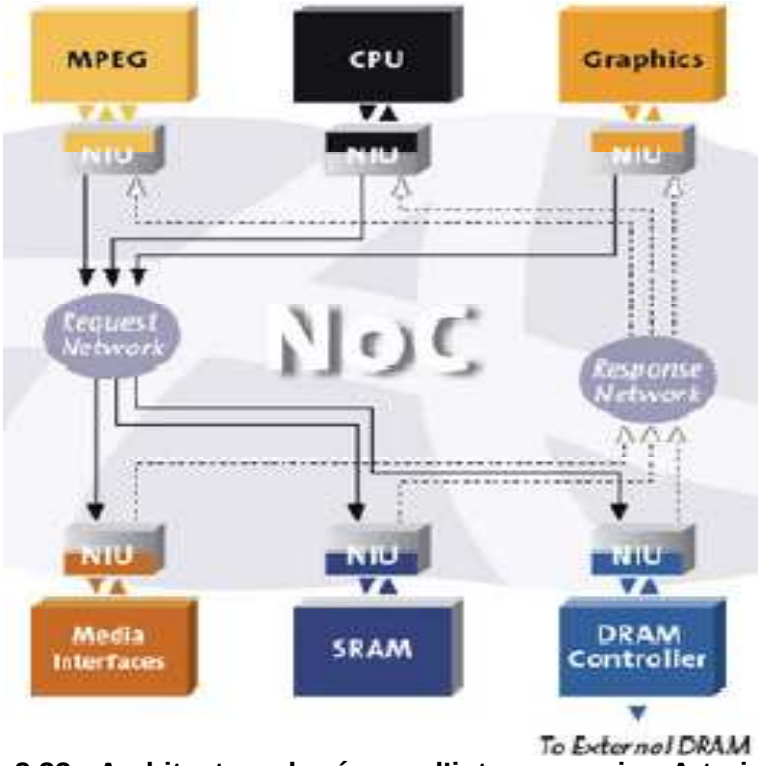


Figure 2.33 : Architecture du réseau d'interconnexion Arteris NoC.

Ce réseau utilise un transfert de données utilisant le routage par paquets entre des *Switchs* (réseaux indirects). La topologie du réseau n'est pas figée, un outil de développement permet de la choisir en fonction de l'application. Des *wrappers* (NIU) permettent d'assurer la compatibilité des blocs IP avec le réseau et son protocole de communication.

7.2.2. Le réseau RAW

L'architecture RAW est un exemple classique de réseau d'interconnexion utilisant un maillage de blocs élémentaires identiques relies par des routeurs, avec la possibilité de stockage local des données dans chaque bloc élémentaire.

Les routeurs sont interconnectés avec leurs proches voisins, chaque nœud est constitué d'un routeur et d'un processeur associe à sa mémoire locale.

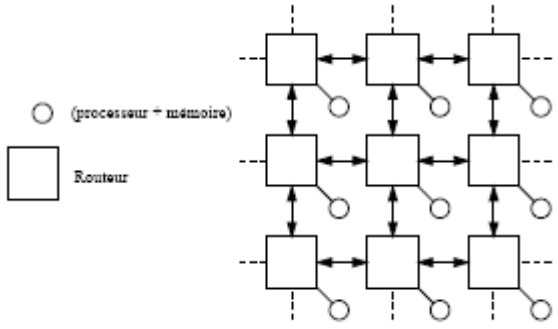


Figure 2.34 : Architecture du réseau d'interconnexion RAW.

La matrice d'interconnexion RAW est reconfiguration dynamiquement, c'est-à-dire que le chemin emprunte par le flux de données entre un émetteur et un récepteur peut changer en fonction de la disponibilité des routeurs.

Jusqu'a aujourd'hui l'architecture RAW n'a pas été intégrée, elle a été utilisée pour connecter, au niveau carte, des DSP de la société Xilinx.

7.2.3. Le réseau SPIN

Développé par le laboratoire LIP, est un réseau d'interconnexion basé sur la commutation de paquets, il utilise le protocole de communication VCI. L'architecture de SPIN est un arbre élargi. Les bus entre les nœuds sont bidirectionnels (32 bits de données et 4 bits qui permettent de définir la destination et le destinataire). Un réseau SPIN capable d'interconnecter 32 ressources (16 initiateurs et 16 cibles).

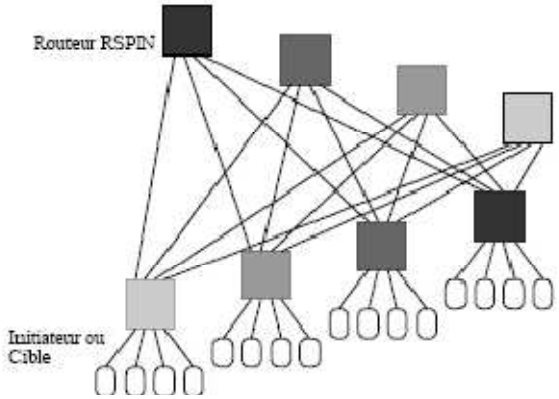


Figure 2.35 : Architecture du réseau d'interconnexion SPIN.

Chaque nœud du réseau est un routeur RSPIN. Un routeur RSPIN est un *crossbar* 10 × 10 qui permet de transmettre une entrée parmi 8, vers n'importe laquelle des 8 sorties. Deux entrées et deux sorties internes permettent de mémoriser un message lorsque le destinataire suivant n'est pas disponible. Cette méthode permet bien évidemment de lutter contre la contention du réseau.

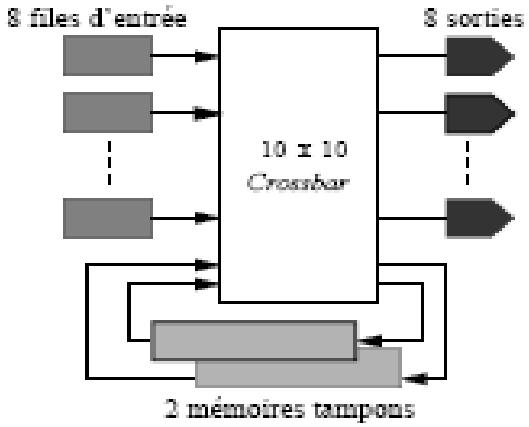


Figure 2.36 : Architecture du routeur RSPIN.

Les deux stratégies de commutation ont été étudiées, la commutation de paquets et la commutation de circuits. Il a été prouve que pour ce réseau, l'apport de la commutation de paquets n'est pas significatif (la réduction de la contention due a une commutation de paquets est largement compensée par le coût en surface de silicium). Une fois un chemin de routage ouvert entre un initiateur et sa cible, le transfert de données se fait à un débit de 6,4 Gbit/s.

7.2.4. Le réseau Nostrum

Développé par le laboratoire LECS à l'université KTH (Stockholm.), est un réseau direct ou chaque nœud est associe a une ressource.

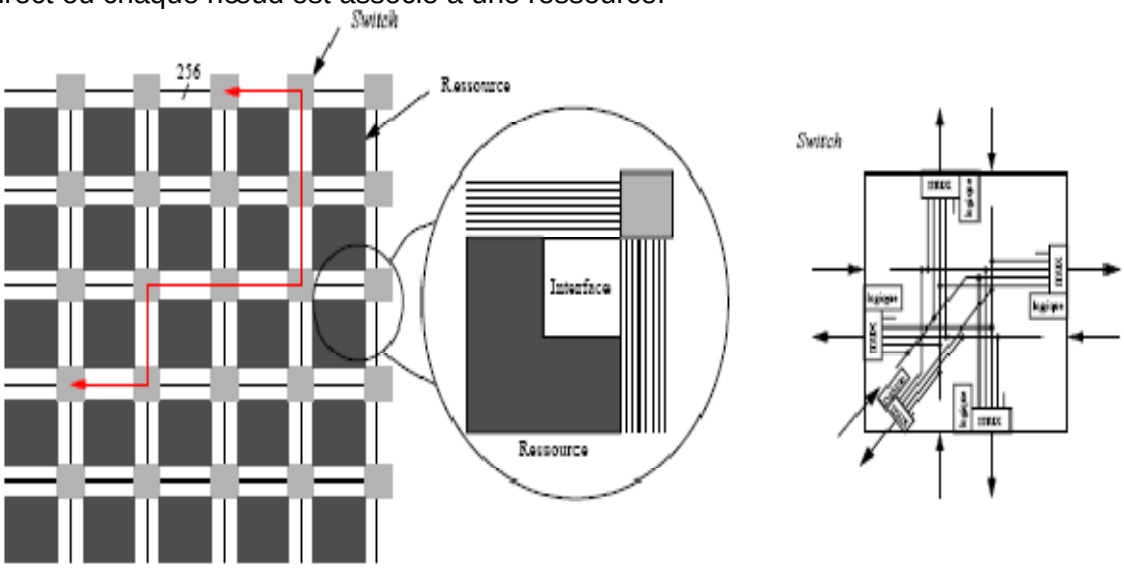


Figure 2.37 : Architecture de l'interconnexion Nostrum.

Le maillage de Nostrum est régulier : un *Switch* est connecte a 4 *Switchs* (proches voisins) et a une ressource locale, une ressource est connectée à un *Switch* (Figure. 2.38). Les communications possibles dans l'interconnexion sont donc de deux types : *Switch*-a-*Switch* et/ou *Switch*-a-ressource.

Les communications de données *Switch*-a-*Switch* sont bidirectionnelles de 256 bits.

Pour une architecture à 16 ressources de 2 mm × 2 mm chacune, la surface globale de la puce sera de 22 mm × 22 mm en technologie CMOS 60 nm. Chaque ressource a une adresse unique et communique avec le réseau par l'intermédiaire d'un *wrapper* (RNI).

7.2.5. Le réseau SoCBus

L'architecture SoCBus est un NoC direct, base sur la commutation de paquets, développée à l'université de Linköping. Son architecture et sa topologie sont identiques à celles du réseau Nostrum. Le débit maximal alloue a une transmission est environ 2,6 Gbit/s.

7.2.6. Le réseau Æthereal NoC

L'interconnexion Athereal NoC est développée par un laboratoire de recherche de la société Philips. Sa topologie générale correspond à un réseau indirect.

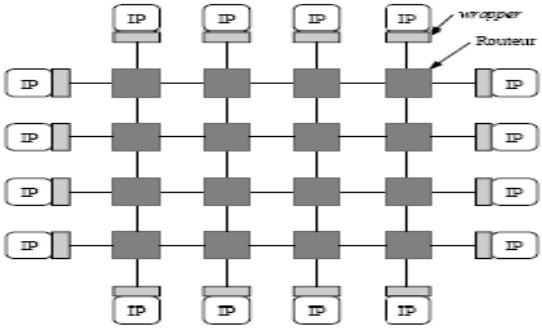


Figure 2.38 : Architecture de l'interconnexion Athereal NoC.

La largeur des bus (bidirectionnel) entre les routeurs est de 32 bits et autorise un débit maximal par lien de 16 Gbit/s, en technologie CMOS 0,13 µm. Le principe d'initiateur cible est présent avec des stratégies de routages spécifiques, dont la commutation de circuits (qui garantit une arrivée des paquets dans l'ordre) et la commutation de paquets (utilisation maximale du débit mis a disposition du réseau).

7.2.7. Le réseau Proteo

Le réseau Proteo est développé à l'université technologique de Tampere. La Figure 2.41 présente sa topologie générale. Proteo est un NoC qui permet d'interconnecter plusieurs sous-systèmes entre eux, dont leurs ressources sont elles-mêmes connectées entre elles par un sous-réseau. Il y a deux types de composants : des routeurs dans les sous-réseaux et des passerelles entre les sous-réseaux. Proteo est un NoC utilisant la méthode de commutation de paquets et est compatible avec le protocole VCI.

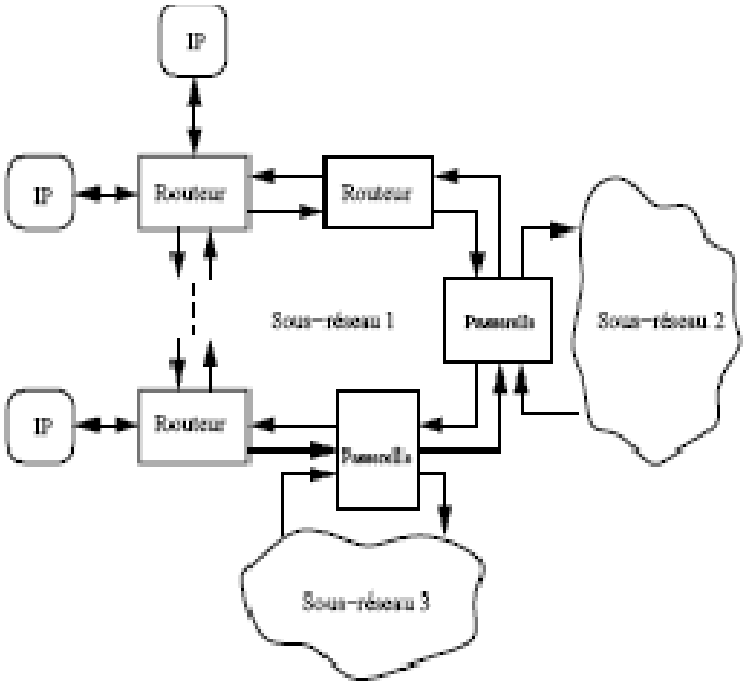


Figure 2.39 : Architecture de l'interconnexion Proteo.

7.2.8. ×pipes Lite

×pipes Lite est une librairie de *wrappers*, de *Switchs* et de liens pour réaliser un NoC utilisant la commutation de paquets. L'architecture du réseau n'est donc pas figée, et plusieurs topologies sont disponibles, comme un réseau direct ou indirect par exemple. ×pipes utilise le protocole OCP.

7.2.9. MANGO NoC

MANGO NoC est un réseau direct dont l'architecture est identique à celle de Nostrum. Chaque nœud est associé à une ressource et à ses 4 proches voisins. Ce qui différencie ces deux réseaux est l'implémentation du routeur et le protocole de communication.

7.3. Récapitulatif

Comme nous venons de le voir, les réseaux sur puce ont des origines variées (industrielles ou universitaires). Chaque équipe de recherche a mis l'accent sur un point particulier en convenance avec ses attentes. Certains réseaux sont dédiés à une application particulière, d'autres sont fondamentalement portables alors que d'autres réseaux sont plus orientés vers la qualité de service.

Le Tableau 2.7 suivant rassemble les réseaux sur puce existants, classés par ordre chronologiques. Nous avons choisi d'indiquer 3 paramètres architecturaux : la topologie, l'emplacement des buffers et le transfert des données utilisés par ces réseaux directs.

Réseau direct	**Topologie**	**Emplacement des éléments de bufferisation**	**Transfert des données**
SPIN	Fat – tree	En entré + 2 espaces en sortie partagés	Wormhole
aSoC	Maillé	Aucun	Wormhole
Dally	Torus	En entré	-
Octagon	Octogone	En entré et en sortie du cœur (et non du routeur)	
Sgroi	Maillé		Stock puis renvoi
Cliche	Maillé	En entré ou « virtual output »	-
Marescaux	Torus	« virtual output »	-
Eclipse	Maillé hiérarchique	En sortie	-
Proteo	Anneaux hiérarchiques	En entrée et en sortie	-
eathereal	Maillé	En entré	Stock puis renvoi
Xpipes	adaptable	En sortie	Wormhole
Hermes	Maillé	En entré (ou en sortie)	Wormhole
QNoC	Maillé irrégulier	En entrée et en sortie	Wormhole
Arteris	Maillé hiérarchique		Wormhole (synchrone) ou stock puis renvoi (asynchrone)
FAUST	Maillé	En entrée et en sortie	Wormhole
DyNoC	Maillé irrégulier		Stock puis renvoi
Becker	Nids d'abeilles		Stock puis renvoi

Tableau 2.7 : Etat de l'art des réseaux directs

La liste des topologies montre que la majorité des réseaux directs sont maillés (hiérarchiques ou non). En effet, cette topologie est la plus simple à mettre en place tant au niveau implantation sur silicium qu'au niveau des protocoles qui l'accompagnent.

Parmi la liste des réseaux, nous remarquons que la majorité a des buffers en entrée, malgré les contentions engendrées par cette architecture (tel que le « Head-of-line »). Ceci s'explique par le fait que cette architecture est simple à mettre en place : le matériel de contrôle est peu coûteux, le nombre de buffers est proportionnel au nombre de ports et la latence à vide n'est que de quelques cycles d'horloge.

Nous observons également que le transfert des données se fait majoritairement en « wormhole» mais également en « stock puis renvoi ». En effet, le « Virtual cut through » et le facteur fou sont absents de cette liste car ils sont à la fois plus complexes à mettre en place pour des performances pas forcements plus intéressants.

Si le « stock puis renvoi » présente des performances médiocres et des surfaces de buffers parmi les plus importantes, c'est aussi un protocole simple à programmer et dont la taille des buffers ne dépasse pas beaucoup celles du réseau en « wormhole » lorsque les paquets sont petits.

De manière générale, pour choisir les paramètres architecturaux d'un réseau direct une équipe privilégie souvent la simplicité. Ceci rappelle que les réseaux directs sont encore jeunes et qu'avant d'optimiser un système, on le conçoit de façon simple.

8. « Nostrum » et l'approche en couche de la communication dans un NoC

Après avoir présenté toutes les techniques et mécanismes associés au réseau sur puce d'une part et d'avoir donné un résumé de l'état de l'art des NoC; nous examinerons ci-après le réseau type Nostrum qui nous servira de modèle de référence [14], [15].

Notre intérêt à ce type d'architecture est justifié par le fait que c'est un modèle fondé sur le principe de communication en couche.

La force motrice derrière l'approche posée en couches est que l'interface et l'ensemble des services qu'une couche fournit aux couches au-dessus et au-dessous est statique alors que la mise en œuvre de ces services est dynamique. Cette approche en couches de la communication à grande ampleur, rehausse les possibilités d'exécuter des simulations et implémentations des couches différentes dans une multitude de design/modèle/langage de spécification, dû à la flexibilité de mise en œuvre. Elle allège aussi des changements dans le protocole empilé, c.-à-d. qu'il est facile de changer l'implémentation d'une ou de plusieurs couches, sans affecter les autres.

Dans la même couche communiquant avec leurs pairs utilisant un ou plusieurs protocoles, c.-à-d. Les règles pour l'interprétation d'informations communiquées dans cette couche. Ces unités d'information convenu utilisés pour implémenter le protocole sont appelées des Unités des Données du Protocole **(*Protocol Data Units* PDUs).**

Afin de fournir ces interfaces de services aux couches supérieures et inférieures, celles-ci doivent être défini. Ces interfaces sont connues comme Points d'accès du

Service **(*Service Access Points* SAP)**). Aussi longtemps qu'une couche s'accorde avec son SAP et implémente ses services, "toute" mise en œuvre peut être utilisée.

Ce qu'on vient de décrire jusqu'à maintenant dans ce paragraphe est valable pour tous les modèles de communication en couche tel que le modèle OSI ; mais le plus important c'est de décrire les particularités d'une telle approche adaptée au contexte NoC notamment pour le réseau Nostrum.

Le design des réseaux de communication a été traditionnellement découplé des spécifications finales des applications et est influencé fortement par la standardisation et les contraintes de compatibilité des infrastructures des réseaux. Dans les réseaux SoC, ces contraintes sont moins restrictives parce que les développeurs conçoivent le réseau de communication sur la structure de silicon. Donc, seule l'interface abstraite des nœuds du réseau exige la standardisation.

Les sept couches du modèle OSI est considérée seulement comme une aide conceptuelle dans le processus du design, une fois le design est mis en place les couches peuvent être désengrangées à un niveau logique afin de rehausser la possibilité de mise en œuvre de optimisations de l'implémentation matérielle.

La couche transport est la première couche qui offre des services de bout en bout, en cachant les détailles réseau, donc les services de la communication doivent être offerts au moins au niveau de cette couche.

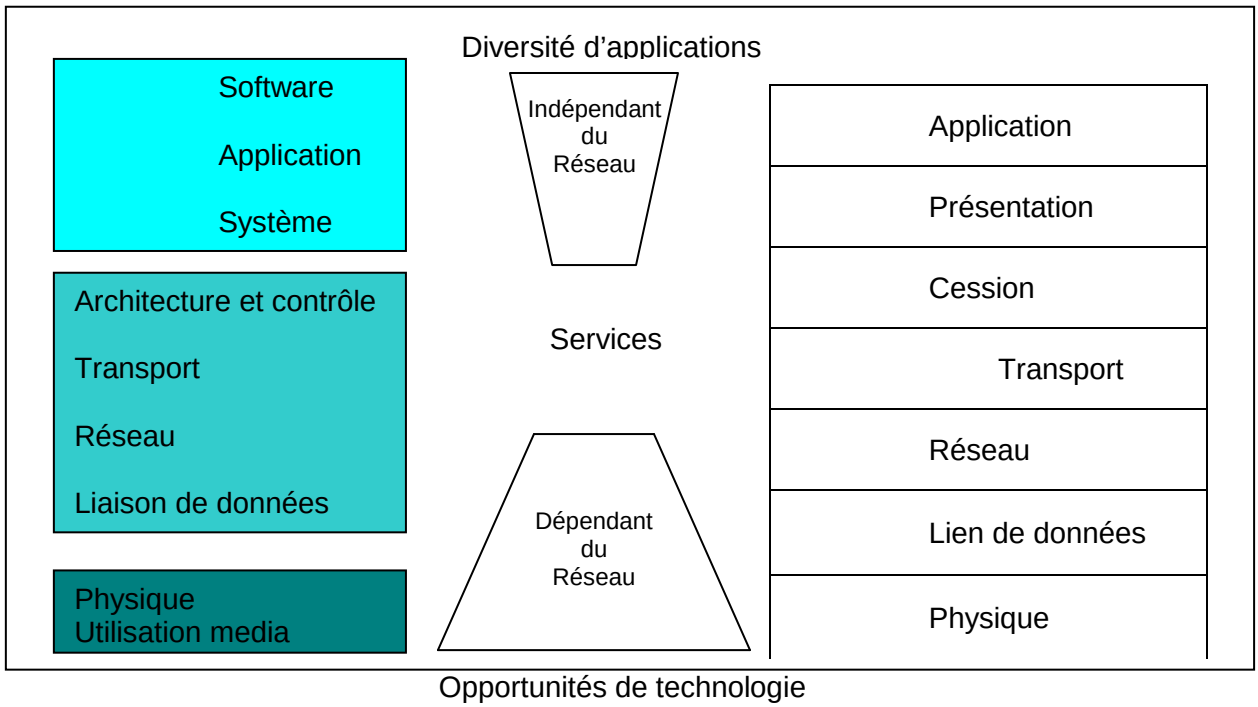


Figure 2.40 : Le modèle OSI du NOC cache les détails d'interconnexion et autorise l'emploie d'éléments réutilisables au-dessus

8.1. Description du protocole de communication du NoC retenu

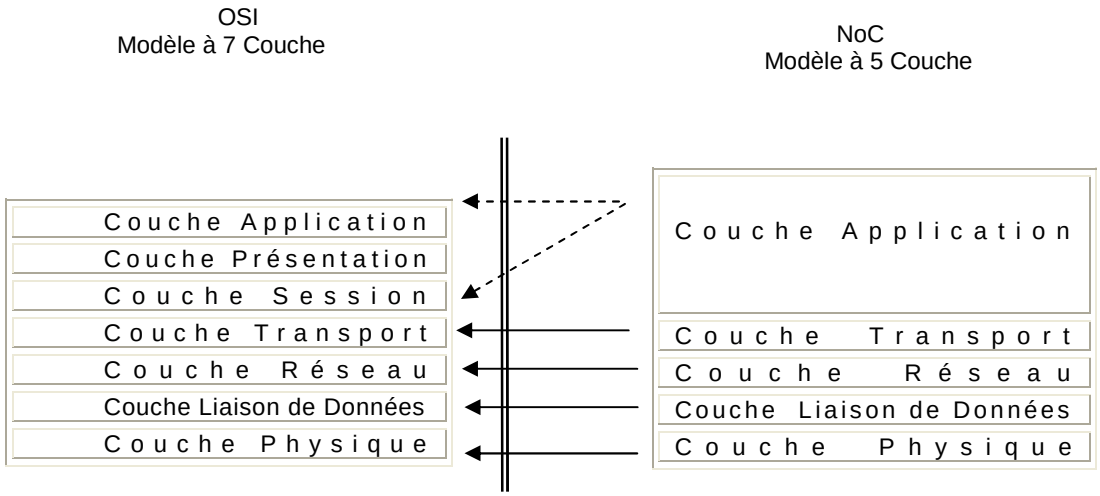


Figure 2.41 : Comparaison entre les modèles OSI à 7 niveaux et NOC à 5 niveaux.

8.1.1. Couche physique

Cette couche concerne les caractéristiques du média physique utilisé pour connecter les Switch et ressources entre eux. Dans le contexte SOC, il spécifie le niveau de voltage, la longueur et la largeur des liaisons, la cadence des signaux, le nombre des liaisons connectant deux unités, etc.

8.1.2. Couche liaison

Cette couche a la responsabilité de transférer de façon fiable les informations le long de la liaison physique. Elle fournit les fonctionnalités de transfert d'un mot d'information d'un nœud à l'autre sans erreurs. Comme les deux unités connectées peuvent travailler de façon asynchrone, la couche liaison prend en charge la synchronisation matérielle en plus de la détection et/ou de la correction des erreurs. La couche liaison inclut aussi l'encodage et la gestion du débit de données pour le contrôle de la consommation d'énergie.

8.1.3. Couche réseau

La couche réseau assure le service de transmission de paquets d'une ressource à une autre en utilisant le réseau de Switch. Ses fonctionnalités incluent la bufferisations des paquets et la prise de décisions de routage au niveau des Switch intermédiaires.

8.1.4. Couche transport

Cette couche est responsable de l'établissement des connections de bout en bout et la transmission des messages en utilisant les couches inférieures. Par conséquent, ses fonctionnalités incluent la paquétisation du message à la source et la dépaquetisation ou l'assemblage des paquets en message au nœud de destination.

8.1.5. Couche application

Au niveau de cette couche peuvent être fusionnées les fonctionnalités importantes des trois couches supérieurs du modèle de référence OSI. Les services inclus dans cette couche sont notamment, la gestion et la synchronisation des messages et la conversion des formats de données. Les fonctions de cette couche sont prises en charge par les ressources (IP, CPU) connectées au réseau.

8.2. Architecture retenue du Nœud NoC

Après avoir donnée la description de la pile protocolaire assurant une flexibilité de la communication d'une architecture NoC, on propose la description de cette architecture physique conforme à la description en couche de cette plateforme de communication

Dans le contexte des communications sur puce, cette plateforme ou le Backbone de communication correspond à l'infrastructure de communication prévue pour l'intégration des cores pour le développement des SOCs. Cette infrastructure est constituée des trois composants suivants :

1. Un réseau de switchs ;
2. Les trois premières couches du protocole de communication ;
3. Les services de communication.

Les services de communication correspondant aux trois premières couches du protocole devant être implémentés dans chaque Switch du réseau. Alors que les services correspondant aux couches supérieures du protocole devant être implémentés au niveau des ressources.

8.2.1. Réseau d'accès

Autres aspects importants de la communication sur puce sont les méthodes requises pour préparer l'intégration des ressources au système. Ceci inclus les méthodes permettant de contrôler si les ressources répondent aux critères d'intégrations notamment, physiques, électriques et de temps. Une interface doit être ajoutée à la ressource de telle sorte que son fonctionnement interne devienne transparent par rapport au réseau et aux autres ressources du réseau. Nous appellerons cette interface, Interface Ressource-Réseau RNI.

8.2.2. Ressources

Les ressources dans le réseau peuvent avoir différentes fonctionnalités. Par conséquent, une ressource peut être un processeur standard ou un core DSP, une mémoire, un bloc ASIC implémentant une fonction séquentielle ou parallèle ou tout simplement un bloc hardware reconfigurable. Il est requit que la ressource doit être implémentée en utilisant les mêmes technologies que celle du réseau de Switchs et son implémentation doit satisfaire aux contraintes des couches métalliques (matérielles) et des niveaux de tensions.

8.2.3. Interface Ressource-Réseau

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

Dans ce but, les différentes contributions dans le domaine des NoC introduisent généralement le concept d'interface réseau (NI). Cette interface fournit un ensemble de services à l'unité de traitement tout en masquant les mécanismes internes de fonctionnement du réseau [16]. Une ressource est donc constituée de l'association d'une interface réseau avec une unité de traitement (Figure 2.44).

On compte lors de la conception d'un NoC à faciliter l'intégration de blocs réutilisables (IP). Pour cela, il paraît intéressant de pouvoir concevoir une unité de traitement indépendamment du réseau d'interconnexion sur lequel elle sera connectée.

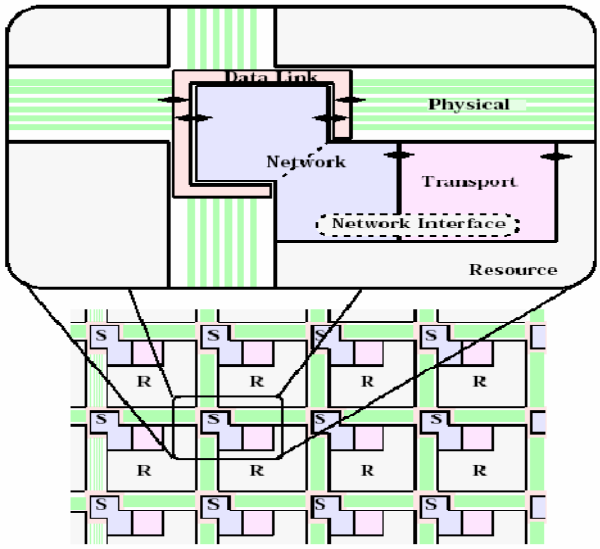


Figure 2.42 : Positionnement de l'interface réseau dans le contexte d'un NoC

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

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

A partir de cette approche qu'on viens de définir dans ce paragraphe et les avantages associés énumérés, nous avons retenu l'approche en couche pour pouvoir

décrire notre nœud Noc tenant comptes des contraintes et spécificités des réseaux sur puce d'une part et des standards utilisés d'autre part définis pour des applications adoptant cette approche de communication.

9. Conclusion

Les réseaux sur puce ont été présentés d'un point de vue architectural dans ce chapitre. L'architecture de ces réseaux est définie par leur topologie et leur fonctionnement physique. Les réseaux à médium partagés semblent avoir peu d'avenir dans le monde de l'interconnexion puisqu'ils sont limités à une dizaine de cœurs et ne sont pas du tout adaptés à la synchronisation GALS. Les réseaux indirects sont également limités en nombre de cœurs par des contraintes physiques. Les réseaux directs et hybrides seront probablement préférés dans le futur. En effet, ils sont particulièrement adaptés en performance et en surface aux systèmes avec beaucoup de cœurs et supportent souvent très bien la synchronisation GALS. Parmi les réseaux directs, la topologie maillée à deux dimensions est la topologie la plus simple à implanter, à configurer et où les protocoles d'arbitrage et de routage sont les plus simples.

Ce chapitre a également présenté les couches associées au réseau sur puce plus précisément la couche transport, réseau et liaison de données. Les protocoles essentiels au bon fonctionnement et les paramètres architecturaux du réseau sont définis dans ces trois couches.

A la fin de ce chapitre, on a pu exposer les particularités d'un modèle de référence dont on s'est inspiré pour la conception de notre modèle. Pour élaborer un tel modèle on est appelé à adopté une méthodologie de travail dont le chapitre suivant exposera.

Chapitre 3

Approche de conception et méthodologie de modélisation

1. Introduction

Après avoir présenté dans les deux premiers chapitres le contexte de notre mémoire; A partir de la problématique posée par les systèmes de communication dans les SoCs jusqu'à l'introduction du nouveau paradigme ''network on chip'' à travers lequel on propose une architecture de communication flexible adoptant les concepts de réseau d'ordinateurs en terme de topologie et de protocole de communication, dans lequel nous avons traité en détails tous les aspects réseau associés, pouvant être adaptés à une communication dans un SoC avec des contraintes bien spécifiques ; en donnant leurs différentes implémentations avec leurs avantages et inconvénients.

Ce chapitre se propose alors en vue de présenter dans un premier temps notre approche de conception retenue pour tels systèmes avec les outils associés et de présenter en second lieux l'environnement de modélisation adopté.

2. La modélisation

L'essor des nouvelles technologies et les retours d'expériences dans de nombreux domaines ont permis d'obtenir des améliorations marquées de la qualité des produits [17]. Il y a quelques centaines d'années, la construction de ponts était le fruit de calculs approximatifs et de solutions empiriques pour répondre au problème donné. De la même façon, la phase de validation du pont consistait à demander à un ensemble de personnes de traverser le pont au pas cadencé. L'objectif était de tester le phénomène de résonance du pont : si le pont résistait, le test était valide. Sinon, le pont, et incidemment les personnes sur ce même pont, finissaient au fond de la rivière ou de la vallée. Par la suite, l'évolution de notre civilisation, avec l'apparition du train et de l'automobile, a rendu cette campagne de tests de plus en plus complexe : il fallait gérer de nombreux paramètres qui entraient en ligne de compte. De même, à la vue du coût d'un ouvrage d'art qu'est un pont, les spécialistes se sont vite rendu compte **qu'il ne fallait pas attendre la phase de tests pour déceler une erreur de conception ou de calcul.**

La révolution en matière de réalisation architecturale est arrivée avec l'apparition des **techniques de modélisation** basées sur le développement des **outils mathématiques**. Aujourd'hui, un pont n'est plus construit de manière empirique. En effet, sa construction repose sur des fondements mathématiques et physiques qui permettent à ses concepteurs de travailler d'abord sur des ***modèles abstraits***. Ces modèles abstraits, de plus en plus performants et complets, prennent en compte de nombreux paramètres comme les conditions atmosphériques, la nature du terrain, le trafic estimé. Plus complets, ils permettent une **meilleure gestion des risques et proposent des solutions économiquement et physiquement valides**. Afin de gérer tous les paramètres et d'effectuer des simulations, l'outil informatique est employé. Les modèles sont exprimés dans un **langage manipulable** par l'ordinateur qui peut alors effectuer des calculs et automatiser une partie des traitements.

3. Définitions

3.1. Modèle

La Modélisation **[18]** est la représentation formelle d'un concept, d'un système, d'un sous-ensemble etc.

Un modèle peut cependant être mathématique, dans ce cas, il peut être vu comme un ensemble d'assertions concernant les propriétés d'un système telles que sa fonctionnalité, sa dimension physique etc.

La modélisation mathématique [17] a de nombreux avantages dont les principaux sont:

- Une précision dans la spécification formelle supérieure à celle fournie par une description informelle en langage naturel qui est souvent ambiguë. C'est la force de la modélisation : *lever les ambiguïtés*,
- des fondements permettant d'énoncer des propriétés et d'étudier le comportement du système dans son ensemble.

Un modèle peut également être constructif, dans ce cas, il définit un procédé informatique qui simule un ensemble de propriétés d'un système. Les modèles constructifs appelés également « modèles exécutables » sont souvent utilisés pour décrire le comportement d'un système répondant à une stimulation qui lui est extérieure.

Des modèles exécutables s'appellent parfois des simulations (quand le modèle exécutable est clairement distinct du système qu'il modélise). Cependant, dans beaucoup de systèmes électroniques et particulièrement pour les logiciels embarqués, un modèle qui commence comme une simulation évolue vers une implémentation logicielle du système. Dans ce cas, la distinction entre le modèle et le système devient floue.

3.2. Conception

La conception est par contre la définition d'une implémentation, c'est-à-dire d'un modèle exécutable sur une plate-forme cible qui impose des contraintes sur le fonctionnement du système. La conception implique généralement la réalisation de plusieurs modèles, chacun étant un raffinement du précédent. Le premier modèle peut être considéré comme la spécification formelle du système, et le dernier comme son implémentation.

En somme, le but de la modélisation est l'exploration des modèles pour une conception finale alors que le but de la conception demeure l'implémentation.

La conception et la modélisation sont évidemment étroitement liées. Dans certaines circonstances, les modèles peuvent être immuables, c'est-à-dire, qu'ils décrivent des sous-ensembles, des contraintes ou des comportements qui sont extérieurement imposés à une conception. Par exemple, ils peuvent décrire un système hydraulique qui n'est pas dans la

conception, mais qui doit être commandé par un système électronique lequel par contre se trouve dans la conception.

3.3. Validation d'un modèle

Quelle que soit la forme des modèles, ils partagent tous une même préoccupation qui est la capture du comportement d'un système et la description de ses propriétés pertinentes. Un modèle servira donc à *valider le cahier de charges*, à *détecter les éventuelles erreurs* de conception avant la réalisation du système et; éventuellement contribuera à déterminer certains paramètres du système tels que les paramètres de contrôle/commande pour un système embarqué. L'analyse et la validation d'un modèle est donc une étape primordiale, notamment dans le cadre des systèmes embarqués. L'analyse et la validation doivent garantir que le système, une fois réalisé, se comporte correctement d'un point de vue fonctionnel et temporel.

La validation d'un modèle peut s'effectuer de deux manières à savoir, par preuve formelle ou par **simulation**. La validation par simulation consiste à exécuter un modèle et observer son comportement. La simulation fait appel à des environnements logiciels, appelés environnement de simulation, qui sont capables de reproduire un contexte d'exécution proche de l'environnement du système à valider, et d'exprimer les différentes conditions dans lequel peut se retrouver le système. matlab et ptolemy ii par exemple, sont des environnements de simulation. Pour ce faire, un environnement de simulation adopte généralement un modèle de temps logique, appelé temps de simulation. Ce modèle de temps fournit une référence temporelle avec des unités pouvant être élastiques afin de simuler certaines situations ou conditions réelles du système.

3.4. Types de modèle

La modélisation peut prendre diverses formes [17], en fonction de la nature des mathématiques retenues pour exprimer et formaliser le système. Dans cette perspective, nous distinguons différents types de modèles comme les modèles mettant en œuvre des machines d'états abstraits, ceux basés sur des automates ou bien encore les modèles fonctionnels.

3.4.1. Les modèles basés sur les machines d'états abstraits

Dans ce type de modèle, le système est représenté par son état et un ensemble d'opérations qui modifient et interviennent sur cet état. Chaque opération permet d'effectuer une transition entre les états que peut prendre le système. Plusieurs méthodes formelles utilisent ce formalisme.

1. Les **ASMs (Abstract State Machines)** sont l'une d'entre elles. L'avantage des ASMs est de pouvoir être utilisé à la fois comme langage de spécification et comme langage de programmation. En effet, elles peuvent être exécutées si les ensembles et les fonctions exprimées sont calculables.

2. La **méthode VDM**. Cette méthode permet de concevoir et de spécifier des systèmes logiciels.
3. La **méthode Z**, quant à elle, a été conçue comme une notation permettant de décrire et de développer des spécifications formelles

Les méthodes formelles décrites dans cette section couvrent une importante partie du cycle de développement d'un logiciel. Elles ont comme avantage de se baser sur des mathématiques simples à appréhender. Cependant, ce ne sont pas des modélisations simples à mettre en œuvre. Si elles sont simples à comprendre, elles le sont moins à utiliser, nécessitant alors de l'expérience et de la pratique

3.4.2. Les modèles basés sur les automates

Ce type de modélisation s'appuie sur une **description comportementale** du système. La description donne les actions et les réactions du système face à différents stimuli. Un automate est un système de transitions d'états finis. Il est constitué d'un ensemble d'états et d'un ensemble de transitions décrivant le flot de contrôle du programme.

L'ISO a développé **LOTOS** pour vérifier les protocoles de communication spécifiés sous l'OSI. LOTOS est alors devenu un standard international dans le domaine des protocoles. Bien que développé pour spécifier un système formel de protocoles de communications, il peut servir dans d'autres domaines. LOTOS permet la modélisation de comportements séquentiels, de choix, de comportements concurrents et non déterministes. Ce langage permet de travailler sur des spécifications de très haut niveau d'abstraction avec :

- des données sous forme de types abstraits algébriques,
- la récursivité,
- l'interruption de communication entre deux processus,

Les spécifications en LOTOS décrivent les comportements observables de systèmes. Un système est alors décrit par une architecture de processus. Ces processus vont communiquer au moyen d'interactions qui peuvent soit être une communication, soit une synchronisation entre plusieurs processus. Les spécifications dans ce langage sont à la fois exécutables et prouvables. Du point de vue industriel, des applications industrielles de quelques milliers de lignes de spécifications ont été écrites avec LOTOS. La vérification automatisée de ces spécifications n'a toutefois pas été encore réalisée.

Le **langage SDL** est issu des recommandations de l'ITU. Cette technique de formalisation formelle s'appuie sur des concepts simples, les machines d'états finis communicantes, et une représentation graphique permettant de rendre son utilisation intuitive et présenter visuellement les relations entre les éléments du système. Grâce à sa prise en charge du temps réel, SDL est bien adapté au monde des télécommunications. Il est à la fois utilisé pour développer des standards et des produits. Dans une spécification SDL, il est nécessaire de disposer de plusieurs éléments pour représenter :

- la spécification de la structure du système,
- la spécification des communications à l'intérieur du système, mais aussi avec l'environnement,

- la spécification du comportement, les propriétés dynamiques, de chaque partie du système,
- les spécifications des informations internes modifiées ou modifiant le comportement du système.

SDL peut être appliqué à différents niveaux : de la simple utilisation des diagrammes illustrant la description d'un protocole, à la modélisation qui simule le comportement du système. Chaque niveau d'utilisation de SDL apporte de la clarté à la spécification. Pour utiliser toute la force du langage et assurer ainsi une description comportementale non ambiguë du système, les modèles doivent respecter toutes les règles du langage. Les outils fournis autour de SDL aident à construire de tels modèles en mettant en évidence les erreurs.

FDR permet de vérifier de nombreuses propriétés sur des systèmes d'états finis. Ce formalisme s'appuie sur la théorie de CSP développée à l'Université d'Oxford par Hoare. CSP a d'ailleurs déjà été utilisé avec succès en industrie. La méthode pour établir qu'une propriété est vérifiée revient à réaliser un raffinement du système de transitions représentant la propriété. Avec FDR, il y a aussi la possibilité de vérifier le déterminisme d'une machine d'états. Cela est principalement utilisé pour vérifier des propriétés de sécurité.

Les *automates* sont très utiles dans le domaine de *la modélisation des protocoles*. La représentation simplifiée sous la forme états-transitions aide à l'utilisation de ces formalismes dans ce genre d'applications. Quand il s'agit de modéliser un logiciel, les automates montrent leurs limites, principalement à cause de l'exploration des différentes possibilités, des différents états à considérer.

3.4.3. Les modèles fonctionnels

Les modèles fonctionnels se basent sur la notion de fonction dans le sens mathématique du terme. Dans d'autres approches adoptant le point de vue impératif, le système est représenté par un ensemble d'états et d'actions qui modifient ces états. Dans la succession d'actions manipulant l'état du système, nous disposons par exemple des mécanismes comme les séquences ou les répétitions. Dans l'approche fonctionnelle, seules les expressions manipulant des fonctions sans effet de bord sont considérées. Le système est alors représenté par une série de définitions de fonctions.

La programmation fonctionnelle est un style de programmation qui met l'accent sur l'évaluation des expressions plutôt que dans l'exécution des commandes. Ainsi, une modélisation fonctionnelle peut directement, et naturellement, être implémentée dans ces langages fonctionnels, réduisant ainsi l'écart entre spécification et implémentation. Ces langages sont généralement considérés comme de bons outils pour la conception et le prototypage des programmes sûrs et corrects. Plus encore, les modèles fonctionnels, utilisés avec la théorie des types, représentent un cadre général de développement pour quelques ateliers dédiés au développement de preuves formelles comme Coq ou Isabelle.

4. Flot de conception général

Il est important de pouvoir modéliser un système entier à plusieurs niveaux d'abstraction, allant de la spécification à l'implémentation. Partant d'un niveau fonctionnel

abstrait, le modèle est affiné en vue d'être partitionné suivant une spécification architecturale. Cet affinement se poursuit progressivement en vue de tenir compte des détails de l'implémentation et des contraintes de synthèse. Il est ainsi possible de surveiller la cohérence du système tout au long du processus du design.

Les simulations peuvent être réalisées avec le même ensemble de test et le design peut ainsi être vérifié à chaque niveau, ce qui évite de lourdes modifications en phase finale. La fiabilité du design est améliorée en créant une spécification détaillée du système et en permettant de tester le hardware et le software par rapport à cette spécification tout au long du cycle de conception. Par ailleurs, le fait de garder le même langage pour tous les niveaux d'abstraction permet d'éviter des traductions propices aux erreurs.

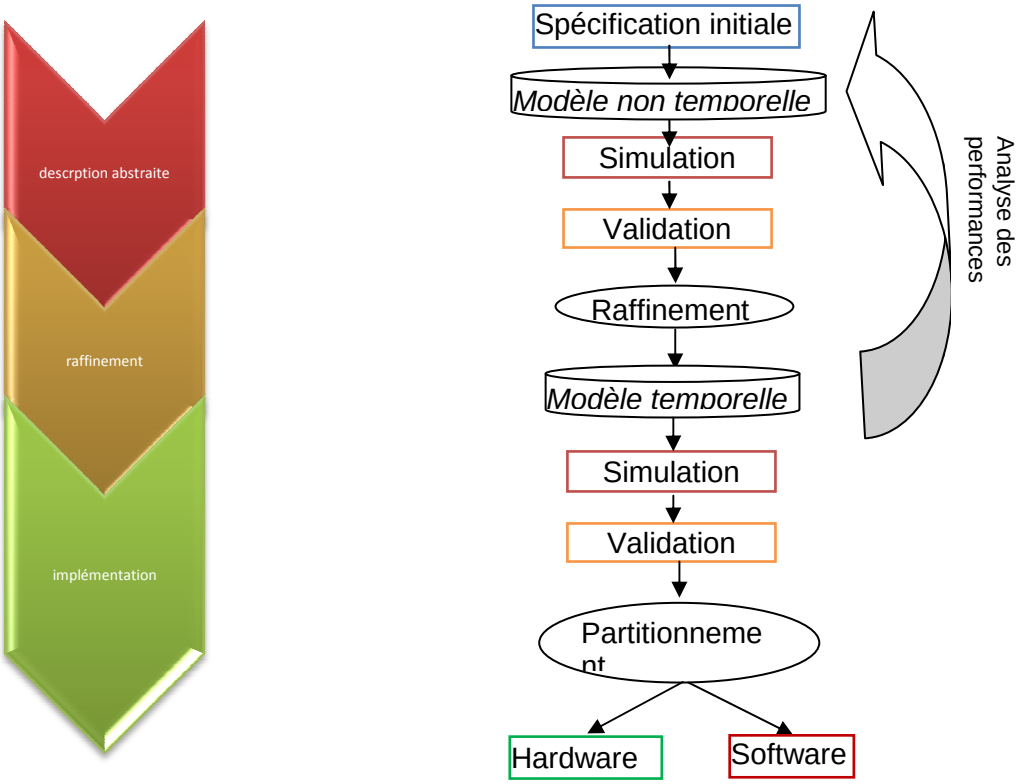


Figure 3.1 : Niveaux d'abstraction

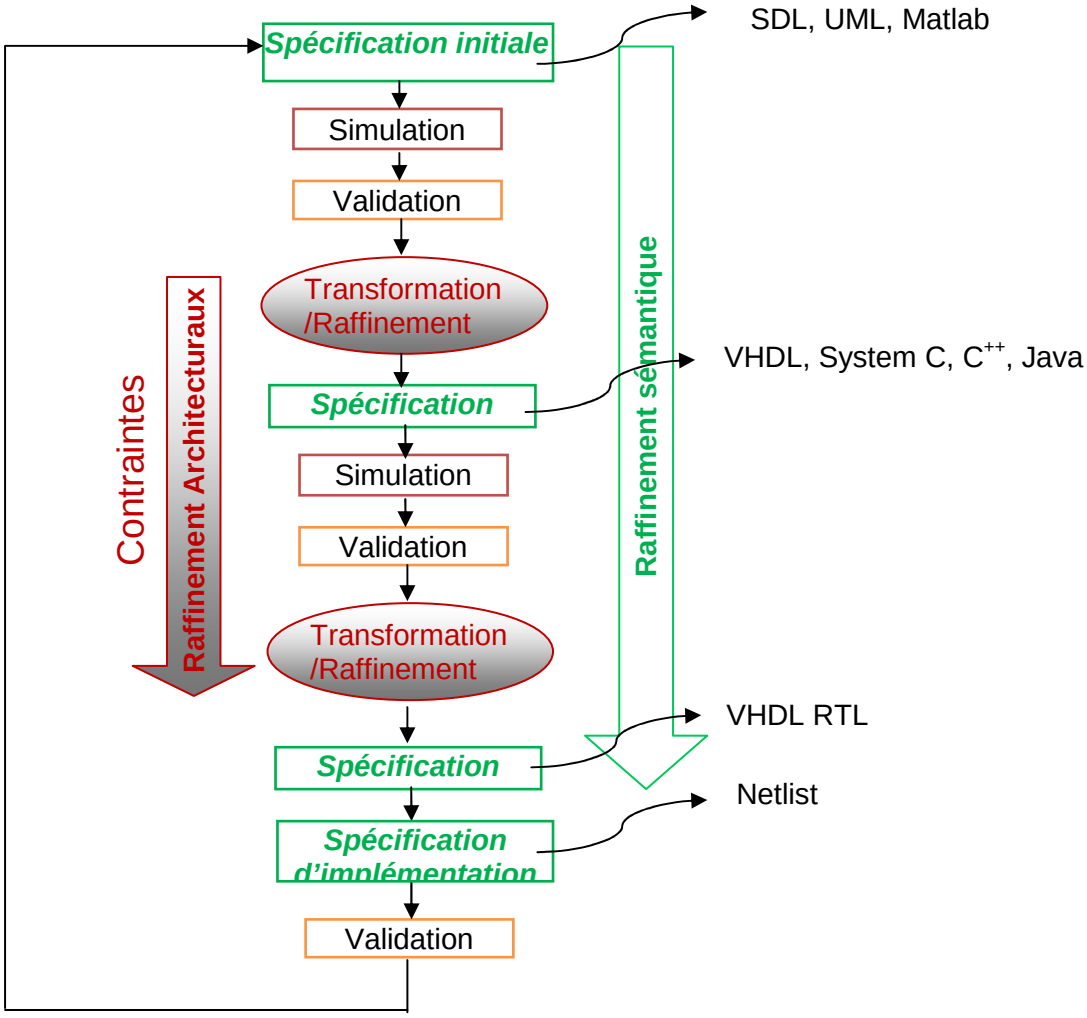


Figure 3.2 : Conception de SoC : Approche Top Down Classique

L'adoption d'un langage système nécessite la définition d'une méthodologie de design adaptée [19]. Il est envisageable de recourir à des outils formels en première phase de spécification tel que SDL ou UML ou d'employer Matlab. La méthodologie doit spécifier clairement le processus d'affinement progressif des modèles et la façon dont la transition se fait d'un niveau d'abstraction à un autre. Par ailleurs il est aussi essentiel de définir les procédures de simulation, de vérification à suivre ainsi que les techniques de partitionnement.

Un autre point important dans le design système est la réutilisation et la flexibilité des modèles. En ce sens, il faut différencier l'aspect fonction de l'aspect communication des modules. La communication peut être modélisée à divers niveaux d'abstraction offrant, par conséquent, plus de flexibilité au design.

Cependant, actuellement un outil de synthèse totalement automatisé pour générer systématiquement le design à partir de la spécification abstraite n'existe pas. Pour aller de la

description au niveau système à l'implémentation, les différents niveaux d'abstraction doivent être bien définis et les méthodes de raffinement bien précises et claires. Les modèles peuvent ainsi être dérivés de leurs prédécesseurs hiérarchiques par des transformations vérifiables.

4.1. Cas des NoCs

Aujourd'hui le développement du NOC est concentré sur le développement d'une configuration de réseau appropriée. Les spécifications établies doivent être examinées. Des outils pour l'évaluation d'une conception de NOC, doivent être considérés en conséquence. Si une conception d'un NOC peut réclamer avoir un certain degré d'efficacité, ceci devrait être soutenue par des simulations d'exécution. Il y a plusieurs simulateurs de réseau disponibles, par exemple NS-2 a été employé à cette fin. NS-2 était cependant, non conçu pour être employé pour le NOC et les possibilités de configuration semblent incapables de répondre aux exigences d'un modèle de NOC. Une autre idée est d'employer un langage de programmation à niveau élevé ordinaire, comme C++ pour construire un simulateur. Ici il y a, naturellement, une possibilité pour rendre lui aussi précis que le réalisateur veut, mais à leur tour lui prendra beaucoup de temps pour se développer.

L'idée de ce projet est de concevoir avec une approche descendante (top down) dans laquelle on commence par une description très abstraite, et comme sue citer dans le paragraphe 3.4.2 ; Les modèles basés sur les automates sont plus adaptés à la description de protocole de communication et comme nous avons décrit dans le chapitre 2, modéliser un réseau de communication nous ramènera à modéliser une pile protocolaire associée à chaque élément du réseau. Un type de modèle a été retenu il est question maintenant de procéder au choix du langage de spécification système, afin d'établir un modèle qui répondra aux exigences d'un simulateur de NOC, qu'on enrichira de détails afin de rendre les résultats valides pour une conception spécifique.

5. Les langages de spécification

Avec le travail sur la normalisation pour l'interconnexion des systèmes ouverts (*OSI*), des groupes travaillant sur les techniques de description formelles ont été établis avec *ISO* et *ITU* au début des années quatre vingt dans le but d'étudier la possibilité d'utiliser des spécifications formelles pour la définition des protocoles et services de l'*OSI [20]*. Leurs travaux ont abouti à la proposition de langages tels que *SDL* ou *Estelle*. Ces langages sont appelés *techniques de description formelles* ou *TDFs* puisqu'ils possèdent non seulement une syntaxe formelle mais aussi une sémantique formelle qui définit le sens, d'une manière formelle, d'une spécification valide. La sémantique formelle est essentielle pour construire les outils qui vont aider à valider les spécifications et à développer des implantations.

Les *TDF*s ont été développées pour assurer:

• des spécifications non ambiguës, claires et concises,
• des spécifications complètes,
• des spécifications consistantes (en isolation et par rapport les unes aux autres),
• des spécifications traçables,
• des implantations conformes aux spécifications.

Nous présentons dans ce qui suit, le langage *SDL* qui est le langage le plus utilisé pour spécifier les protocoles de communication.

6. Présentation du langage SDL

SDL est le langage de spécification et description recommandé par l'ITU pour spécifier des systèmes de télécommunication d'une manière non ambiguë. Sa force réside, entre autres, dans le fait que c'est un standard international très reconnu et dans le fait qu'il offre une indépendance des intérêts de certaines compagnies ou vendeurs. SDL peut être utilisé pour spécifier d'une manière fonctionnelle le comportement d'objets si ceux-ci peuvent être spécifiés à l'aide d'un modèle discret; i.e., l'objet communique avec son environnement par des messages discrets. Une spécification SDL définit le comportement d'un système par le biais de stimuli et réponses. Le modèle de spécification est basé sur le concept de machines à états finis étendus communicantes (CEFSM) [19].

Un système décrit en SDL est composé d'un ensemble de processus concurrents communicant à travers des signaux. SDL supporte différents concepts permettant la description de systèmes tels que la structure, le comportement et la communication. Il existe deux formats SDL, un graphique et l'autre textuel, bien que différente, ces deux manière de décrire un système SDL ont une même base sémantique commun.

7. Approche de conception

Le système dans le cas de notre travail consiste à un Nœud type pour un réseau de communication sur puce qui sera valider en testant son bon fonctionnement ; ce test alors ne pourra être évalué que si la communication avec d'autre nœud de même type (même description) est parfaitement assurée (suivant le cahier des charges).

Ceci est dit, on distinguera deux niveaux de modélisation dont les objectifs sont complémentaires. Le premier niveau consiste à modéliser le nœud Noc. Le second prend en charge le trafic réseau généré.

7.1. Modélisation du nœud

Il s'agit de modéliser le fonctionnement de l'élément de base d'un réseau de communication sur puce indépendamment du flux de données (applications) généré par les IP Core. Pour ce faire nous avons opté à Utiliser un environnement de modélisation haut-niveau, afin de développer un modèle fonctionnel sans avoir le niveau de détail nécessaire à une implémentation physique. Ces environnements fournissent un ensemble de primitives permettant de construire un système et de le simuler et valider (dans notre cas nous avons choisi SDL, un code cible associé est généré pour pouvoir l'exploiter dans 7.2).

7.2. Modélisation d'un réseau sur puce (NoC)

Pour ce faire on doit être en mesure d'utiliser ou concevoir un outil de modélisation de réseau, trois façons de procéder vous est présentées ci-après :

1. Reprendre un outil standard développé pour la modélisation et simulation des réseaux (NS-2….) ; le but dans ce cas est de l'adapter aux contrainte des NoCs, un passage de SDL est possible à l'aide du code C généré adapté par l'outil « targeting ».

2. Développer un modèle spécifique de l'architecture que l'on vise à partir de langages de description matériels standards dans le but d'avoir un modèle précis. Dans cette optique, il préférable de le faire avec le langage VHDL qui a l'avantage de pouvoir être aisément raffinés pour devenir synthétisables.

3. Exploité le code cible de SDL du nœud pour créer une nouvelle librairie ; afin de construire tout un réseau ; et ce en multipliant d'abord le nombre de Nœuds ; et les organiser ensuite suivant une topologie justifiée

Le flot de modélisation retenu est explicité par la figure 3.3 appliqué sur l'environnement de développement TauSDL élaboré par la société Telelogic, permettant de modéliser, simuler et surtout valider des systèmes dans notre cas, il s'agit d'un premier temps de modéliser le Nœud NOC et de générer en second le réseau associé sur NoC.

Cette figure expose clairement la suite logique de la conception assistée par TauSDL. Ainsi, l'approche de création de tout projet se résume aux étapes suivantes :

- description du programme en utilisant l'éditeur SDL,
- Ensuite, avec le compilateur SDL, on prendra connaissance des erreurs de conception,
- Puis, il y a le processus de validation simulée, en d'autres termes, le programme va simuler tous les cas possibles et va détecter les erreurs liées à l'utilisation du système,
- L'avant dernière étape, est le ***targeting*** (ciblage en français). Elle consiste à cibler l'environnement dans le quel, on veut que notre programme tourne.
- Enfin, lorsque toutes les étapes précédentes se sont bien déroulées, le code est alors généré, l'application est ainsi créée, un fichier exécutable est prêt à être utilisé

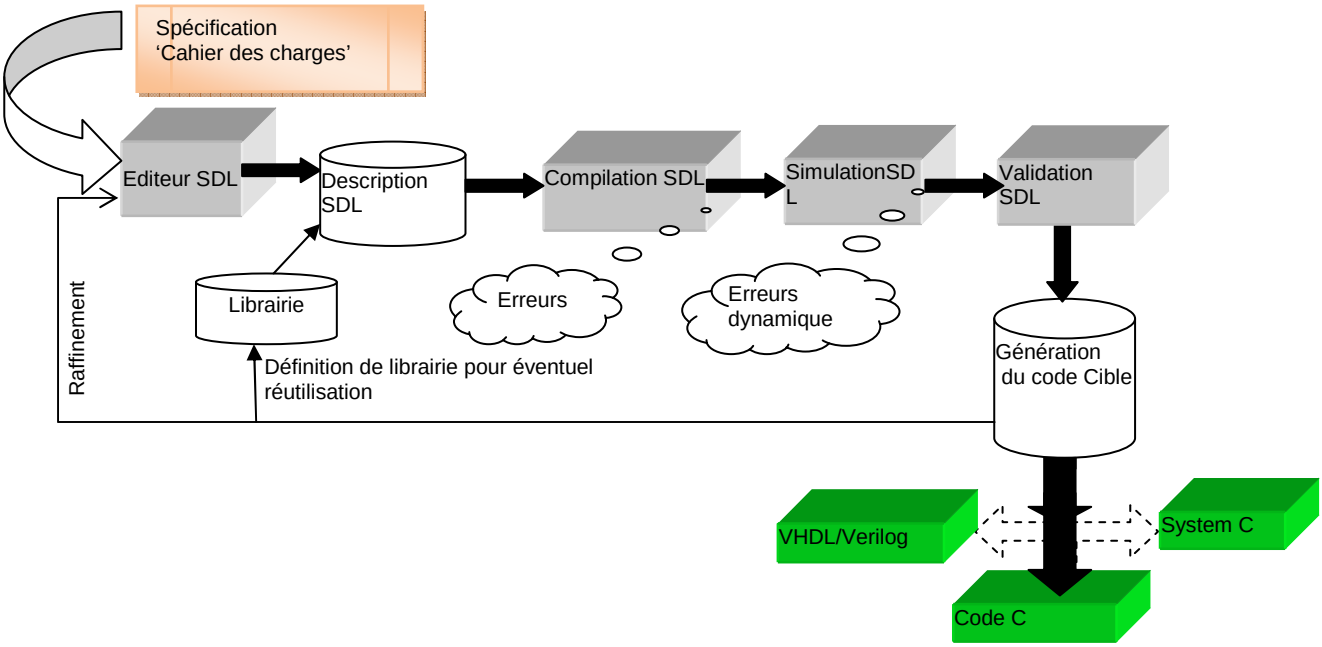


Figure 3.3 : Le flot de conception dans TauSDL

8. Conclusion

Nous avons tenté dans ce chapitre de montrer que le langage formel SDL, se trouve être tout à fait approprié pour la traduction et la validation des spécifications de notre système décrite essentiellement par la pile protocolaire à associer à un nœud NOC donné.

Il ne faut pas perdre de vue que l'objectif final est l'implémentation bas niveau du réseau d'interconnexion. Il devient évident que l'approche de modélisation à travers le langage cité n'est qu'une étape, à notre avis essentielle, pour arriver à la cible choisie. Ceci reste possible à travers des passerelles (traducteurs) déjà existantes ou à développer. On peut citer comme exemple le passage de SDL vers les langages C ou VHDL. Ceci rend envisageable dans la phase de la conception le recours à des environnements spécialisés pour l'analyse et l'évaluation de performances des systèmes ainsi décrits.

Le choix opéré s'inscrit donc dans une approche méthodologique de développement de NOC.

En exploitant ce langage et son environnement, Nous décrirons dans le chapitre suivant la description détaillée de la modélisation de notre réseau sur puce.

Chapitre 4

Modélisation du NoC retenu

1. INTRODUCTION

Dans le chapitre II, les réseaux sur puce ont été présentés d'un point de vue architectural. L'architecture de ces réseaux est définie par leur topologie et leurs fonctionnements physique, ils étaient aussi présentés d'un point de vue protocolaire

A partir de cette étude, nous proposons dans ce chapitre la description de notre modèle associé à un nœud type NoC. La prise de décisions de conception sera justifiée au fur et à mesure dans notre exposé après avoir rappelé la définition du mécanisme utilisé.

Nous donnerons par la suite notre modélisation en utilisant le langage SDL déjà introduit dans le chapitre 3.

2. Cahier des charges de la Conception de Nœud NoC

2. 1. Topologie du réseau sur puce retenue

La topologie spécifie l'organisation physique du réseau. Elle définit donc comment les nœuds sont connectés entre eux. De nombreuses topologies ont été envisagées (chapitre 2). Pour comparer les topologies, différents critères physiques sont utilisés. Les plus courants sont :

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

La plupart des propositions d'architecture NoC sont basées sur une topologie de maillage simple à deux dimensions [24]. Les trois raisons principales qui expliquent ce choix sont [16] :

1. La facilité d'implémentation. En effet, avec les technologies silicium, il est plus simple de construire des structures planes. Les topologies à dimensions supérieures (tels que les cubes ou hypercubes) qui sont utilisées pour des réseaux non-intégrées ne sont donc pas reprises dans le cas des NoC. De plus les nœuds ont la même architecture et les liens sont de même longueur ce qui facilite les étapes de placement et de routage.

2. Une connectivité suffisante. Les architectures de NoC sont étudiées dans le but de réaliser des circuits intégrés complexes qui répondent à des besoins d'application. Dans la plupart de ces applications, les schémas de communication sont locaux. Une ressource du réseau n'a pas besoin de communiquer avec toutes les autres. La topologie peut donc rester simple. Contrairement aux architectures dédiées aux calculs massivement parallèles pour lesquelles la topologie est dessinée pour des communications arbitraires avec l'ensemble des ressources.

3. Des propriétés intrinsèques intéressantes. La complexité des nœuds est limitée à au plus cinq ports d'entrée/sortie ce qui simplifie l'élaboration d'algorithme de routage

L'architecture NOC retenue est basée sur une **topologie maillée à 2-dimensions**, permettant la communication entre des ressources hétérogènes. La figure montre une architecture NOC avec 16 ressources. Chaque Switch est connecté à quatre Switchs voisins et à une seule ressource. Nous supposons que les Switchs du NOC possèdent des tampons pour gérer le trafic de données.

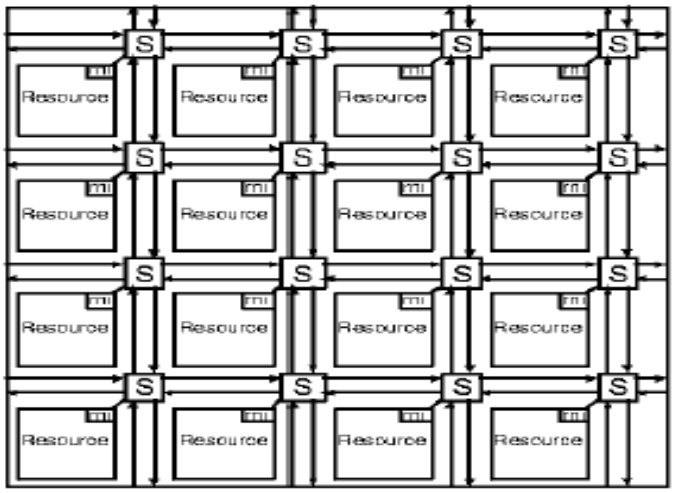


Figure 4.1: Architecture NOC de dimension 4 x 4

2.2. Mécanismes de communication

Après avoir sélectionné une topologie, il faut spécifier comment les informations vont transiter sur le réseau : c'est le mécanisme de **commutation de paquet** qui a été retenue dans laquelle la structure d'échange au niveau du réseau est le paquet. Un paquet peut contenir une fraction, un ou plusieurs messages. Les paquets sont transmis sans qu'il y ait de **connexion établie préalablement (mode non connecté)**. L'avantage est de pouvoir accéder au réseau plus rapidement et de partager les ressources..

Les paquets doivent contenir des informations sur leurs destinations : les informations de contrôle sont envoyées en même temps que les données. Les nœuds sont souvent plus complexes, ils ont parfois à modifier le contenu des paquets pour mettre à jour des informations de routage par exemple, la latence est plus grande. Il faut introduire des systèmes de gestion des blocages et des priorités. Les paquets peuvent arriver de façon désordonnée et il y a moins de garantie temporelle sur le transfert.

2.3. Modes de commutation

Dans le cas de la commutation par paquet, il faut définir aussi le mode de commutation autrement dit la façon dont les paquets vont transiter d'un nœud du réseau au suivant. Ainsi le mode retenu pour notre modélisation est **stocke puis renvoie (Store-and-forward)** : Ce mode de commutation consiste à faire transiter le paquet dans son ensemble d'un nœud à l'autre. Cela implique que chaque nœud doit être en mesure de stocker la totalité d'un paquet. Il existe donc une taille de paquet maximale. De plus la latence des échanges est importante puisqu'il faut multiplier le temps de transfert d'un paquet entre deux nœuds par le nombre de sauts.

2.4. Résumé des spécifications retenues

Le schéma ci-après permet de résumer l'ensemble des décisions de modélisation prises, le but étant de pouvoir les décrire sur SDL, c'est le but donc des paragraphes suivants.

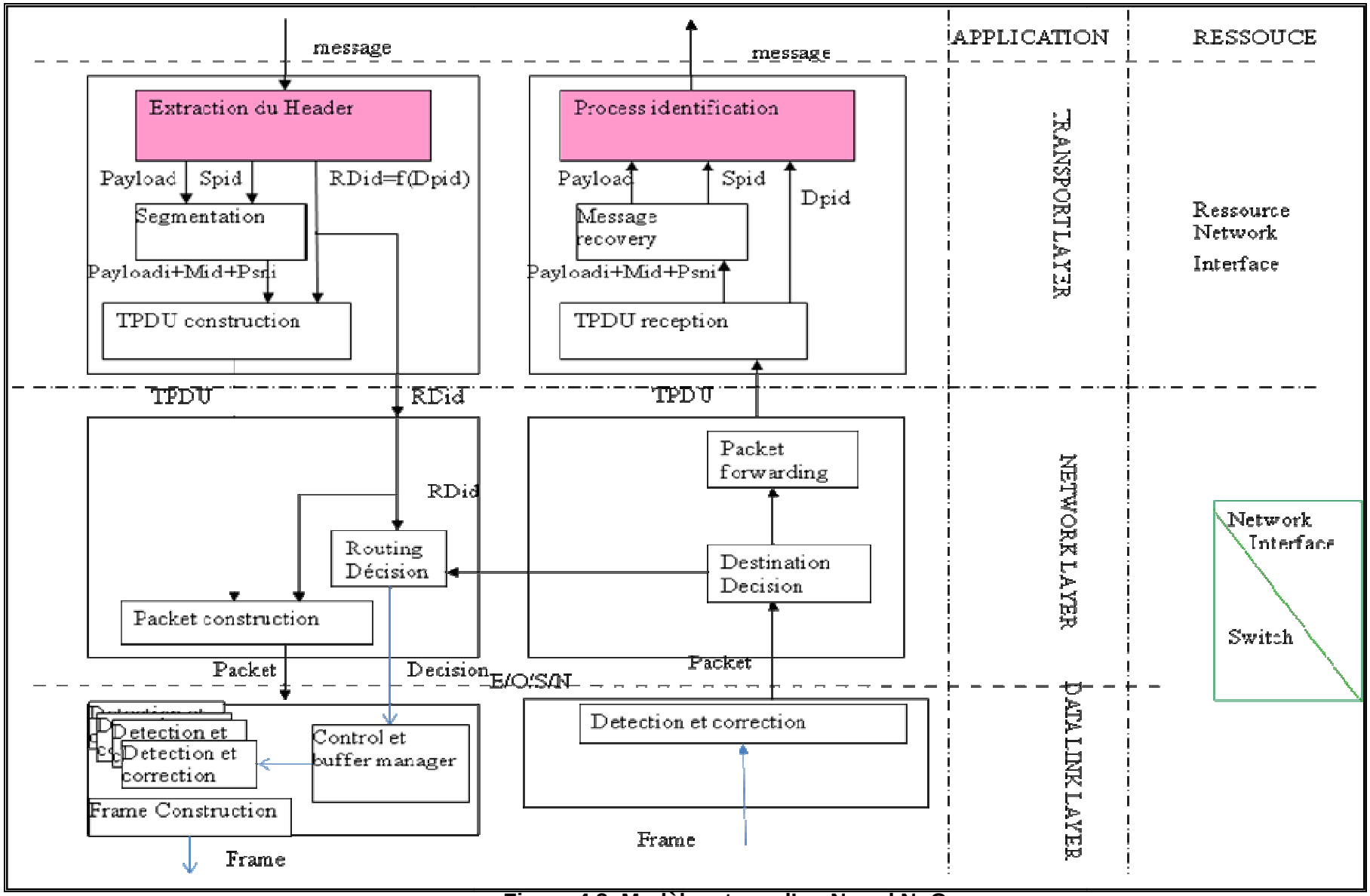


Figure 4.2: Modèle retenu d'un Nœud NoC

3. Description d'un nœud NoC

3.1. Vue d'ensemble

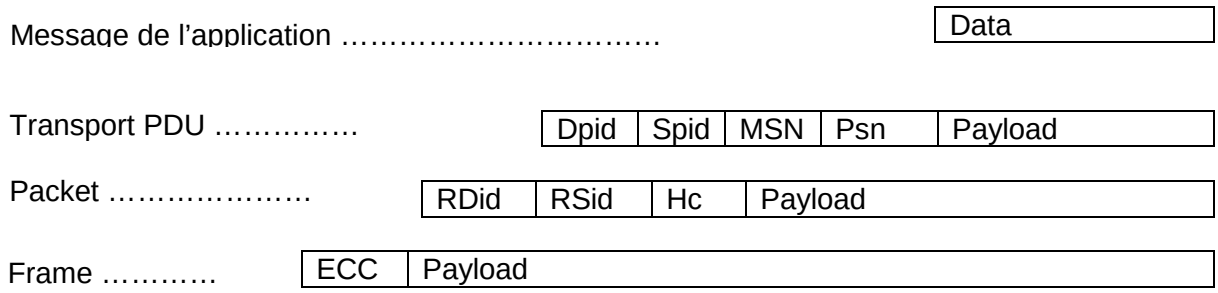
Notre modélisation NoC consiste à en proposer un simple modèle d'un nœud dont les spécificités peuvent être bien résumées dans le schéma de la figure 4.2

Comme a été déjà introduite notre approche de conception est **l'Approche en Couche** ceci veut dire que le nœud Noc est décrit sur notre environnement sous format de 3 couches présentées dans le chapitre 2 (paragraphe 8.1).

La figure 4.3 est la description système du nœud. Comme présenté dans cette figure, notre système comporte trois (03) blocs, chacun assure des fonctions et offre des services bien déterminés et dont deux (02) communiquant avec l'environnement extérieur, cet environnement extérieur est représenté par deux interfaces, la première est associée à la couche application associée au IP core, la deuxième représente les interfaces avec les 4 nœuds voisins.

Nous avons également défini des signaux (figure 4.4) qui peuvent être émis de et vers leurs blocs respectifs. Ces signaux représentent un protocole de communication entres les trois blocs.

La structure des données manipulée entre les différentes couches est schématisée ci-dessous :



PSN (4bits) : packet Sequence number: un champ de 4 bit permettant d'adresser 16 différents paquets provenant du même message.

MSN (8bits)= message séquence number : permet d'identifier le message source et le nombre de paquet associés et afin reconstruire le message à l'arrivée.

Dpid (4bits) : destination process identification
Spid (4bits) : source process identification

RSid (4bits): ressource source identification
RDid (4bits): ressource destination identification
Les identificateurs de ressources sont codés sur 4bit permettant d'adresser un réseau de 16 ressources
Hc (4bits) : indicateur de nombre de saut

ECC (16bits) : error control, calculé en utilisant l'algorithme du CRC-16

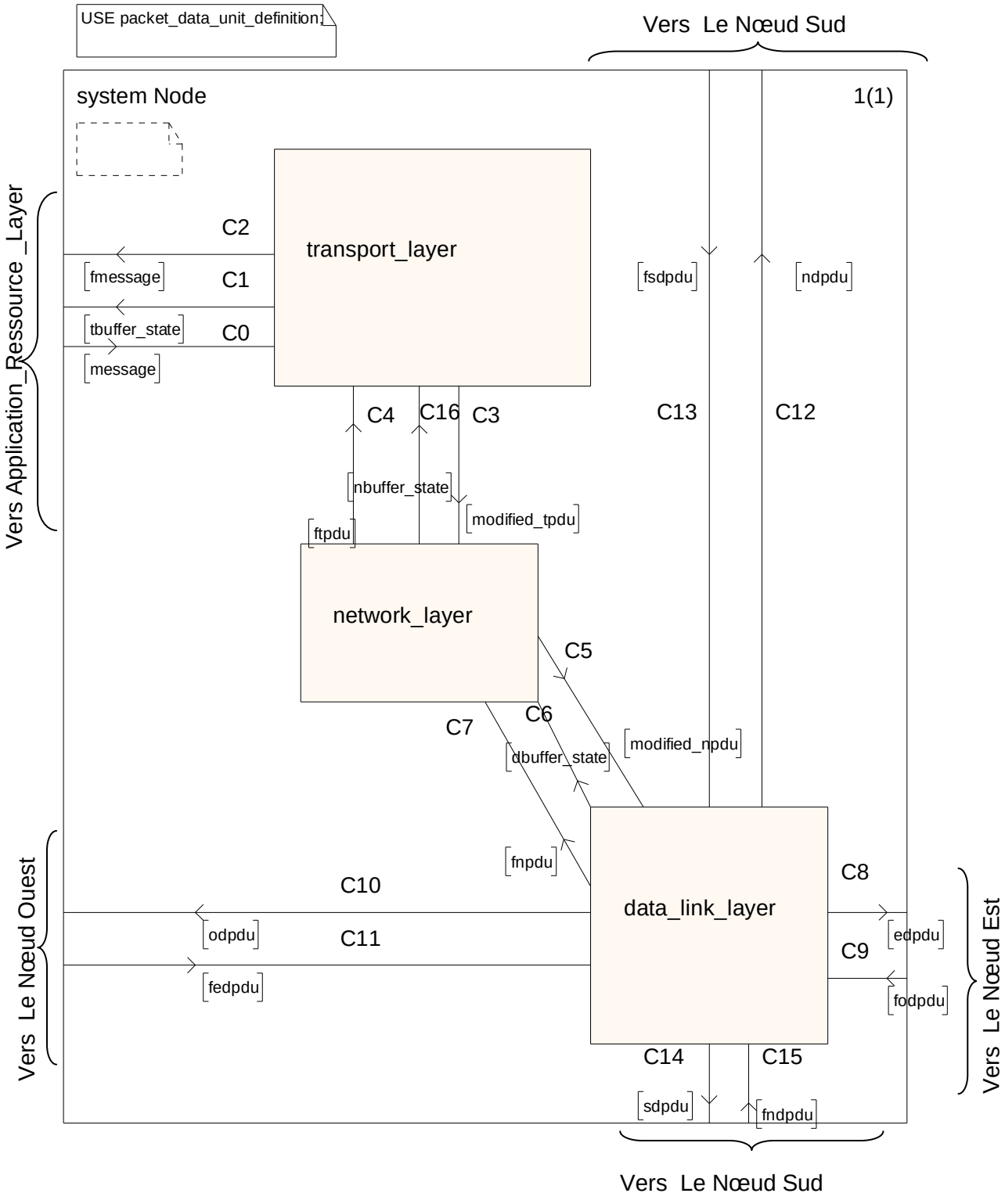


Figure 4.3 : Modélisation système du Nœud NoC

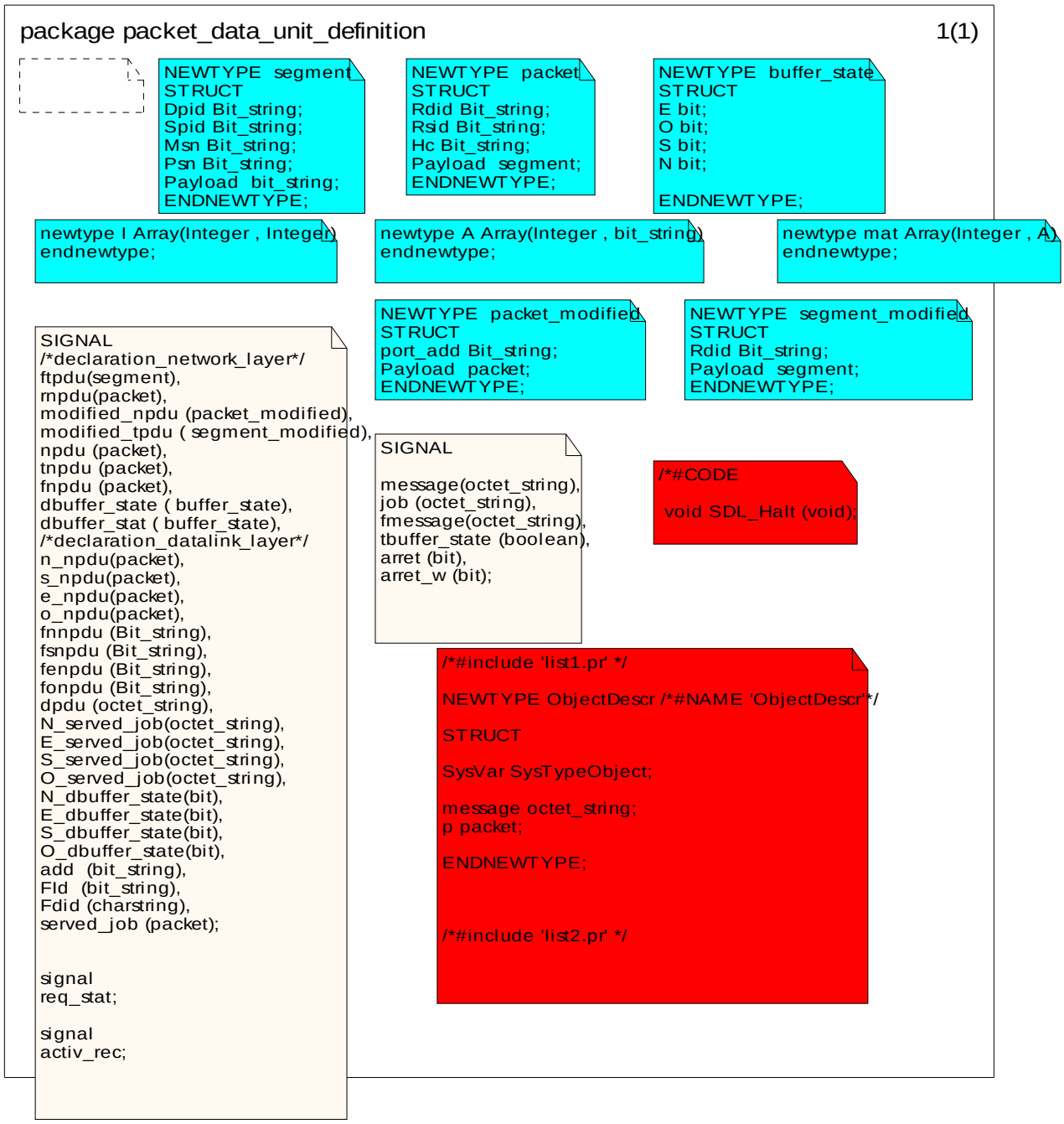


Figure 4.4 : Le Package PDU associé au Nœud NoC

3.2. La couche liaison de données

Le bloc SDL responsable des fonctionnalités de la couche liaison de données est composé de deux sous Blocs *d_encapsulation* et *d_desencapsulation*, présentant quatre interfaces reliant le nœud en question à ces 4 nœuds voisins (Est/Ouest/Nord/Sud).

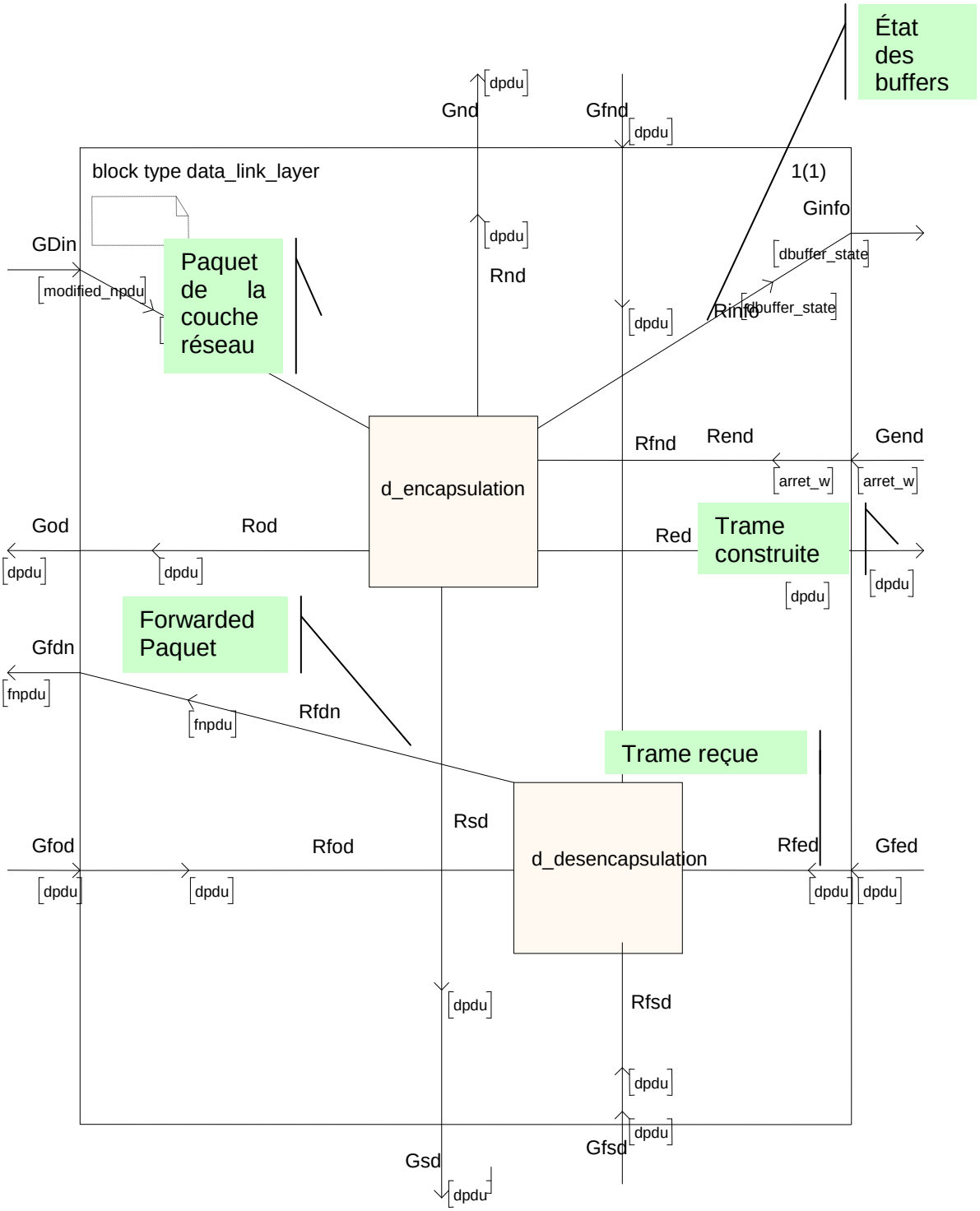


Figure 4.5 : Bloc de la couche liaison de données

3 .2.1. Block encapsulation

Ce bloc permet de recevoir les PDUs de la couche réseau, d'examiner leurs contenus afin de lire les informations de routage permettant de les faire transiter vers les autres nœuds via un port bien défini (Est ; Ouest ; Sud ; Nord)

La transition de tels PDU (paquet) est assurée après avoir constitué les data link PDU (trame) contenants les informations de calcul d'erreur.

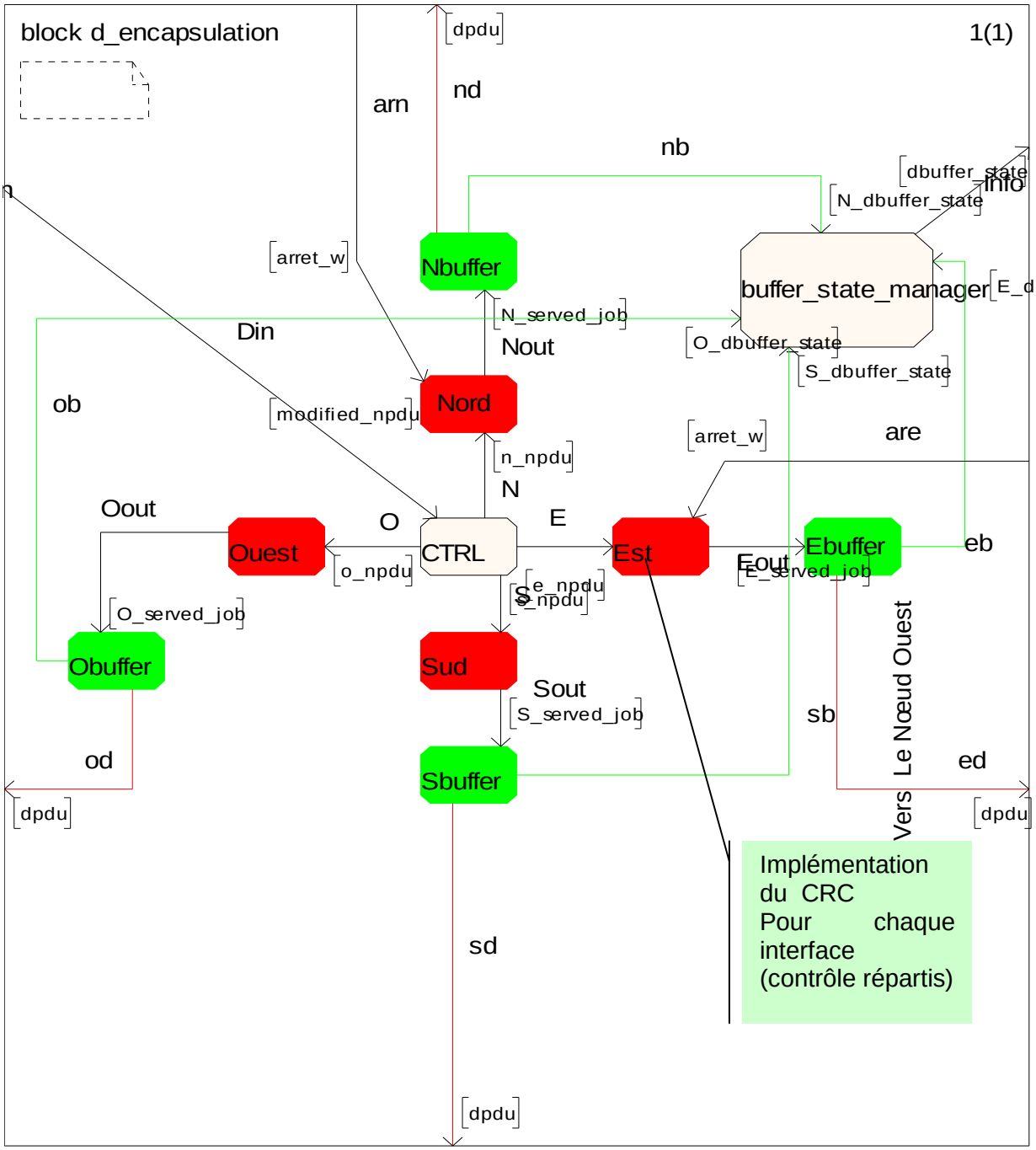


Figure 4.6 : Les process du bloc n-encapsulation

Les principales fonctions donc à modéliser pour cette couche consiste en :

1/ contrôle d'erreur :

Les trames manipulées à ce niveau sont de 64bits, contenant un champ de 16 bits associé à l'information de contrôle d'erreur et ce en implémentant l'algorithme CRC (correspondant à un des 4 ***process*** Sud, Nord, Est, Ouest mentionnées sur la figure 4.6 en rouge).

Le contrôle d`erreur été choisi de tel sorte qu'il soit exécuté en mode répartis dans le but d'assurer un traitement parallèle sur chacun des (04) ports.

a. Description de l'algorithme de calcul de CRC

Le **contrôle de redondance cyclique** (noté **CRC**, ou en anglais *Cyclic Redundancy Check*) est un moyen de contrôle d'intégrité des données puissant et facile à mettre en œuvre. Il représente la principale méthode de détection d'erreurs utilisée dans les télécommunications.

Le **contrôle de redondance cyclique** consiste à protéger des blocs de données, appelés *trames* (*frames* en anglais). A chaque trame est associé un bloc de données, appelé *code de contrôle* (parfois *CRC* par abus de langage ou *FCS* pour *Frame Check Sequence* dans le cas d'un code de 32 bits). Le *code CRC* contient des éléments redondants vis-à-vis de la trame, permettant de détecter les erreurs, mais aussi de les réparer.

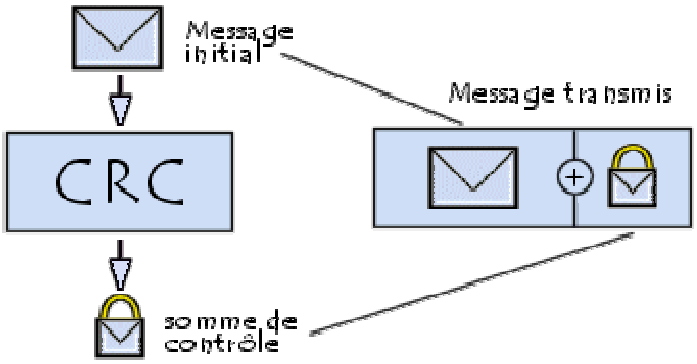


Figure 4.7:Schématisation du calcul CRC

Le principe du *CRC* consiste à traiter les séquences binaires comme des polynômes binaires, c'est-à-dire des polynômes dont les coefficients correspondent à la séquence binaire. Ainsi la séquence binaire *0110101001* peut être représentée sous la forme polynomiale suivante :
$0*X^9 + 1*X^8 + 1*X^7 + 0*X^6 + 1*X^5 + 0*X^4 + 1*X^3 + 0*X^2 + 0*X^1 + 1*X^0$
Soit
$X^8 + X^7 + X^5 + X^3 + X^0$
Ou encore
$X^8 + X^7 + X^5 + X^3 + 1$

De cette façon, le bit de poids faible de la séquence (le bit le plus à droite) représente le degré 0 du polynôme ($X^0 = 1$), le $4^{ème}$ bit en partant de la droite

représente le degré 3 du polynôme (X^3)... Une séquence de *n* bits constitue donc un polynôme de degré maximal *n-1*. Toutes les expressions polynomiales sont manipulées par la suite avec un arithmétique modulo 2.

Dans ce mécanisme de détection d'erreur, un polynôme prédéfini (appelé *polynôme générateur* et noté G(X)) est connu de l'émetteur et du récepteur. La détection d'erreur consiste pour l'émetteur à effectuer un algorithme sur les bits de la trame afin de générer un CRC, et de transmettre ces deux éléments au récepteur. Il suffit alors au récepteur d'effectuer le même calcul afin de vérifier que le CRC est valide.

Polynômes générateurs

Les polynômes générateurs les plus couramment employés sont :

- **CRC-12** : $X^{12} + X^{11} + X^3 + X^2 + X + 1$
- **CRC-16** : $X^{16} + X^{15} + X^2 + 1$
- **CRC CCITT V41** : $X^{16} + X^{12} + X^5 + 1$
  (Ce code est notamment utilisé dans la procédure *HDLC*.)
- **CRC-32** (**Ethernet**) : = $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
- **CRC ARPA** : $X^{24} + X^{23} + X^{17} + X^{16} + X^{15} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^5 + X^3 + 1$

2/ la gestion des buffers

Les Buffers donnent la possibilité de stocker momentanément les paquets au lieu de les perdre dans le cas ou il est impossible de les transmettre immédiatement. La couche data Link permet la gestion de ces buffer afin de synchroniser la communication d`une part et la gestion d`autre part des liens entre les nœuds communicants en mode non connecté.

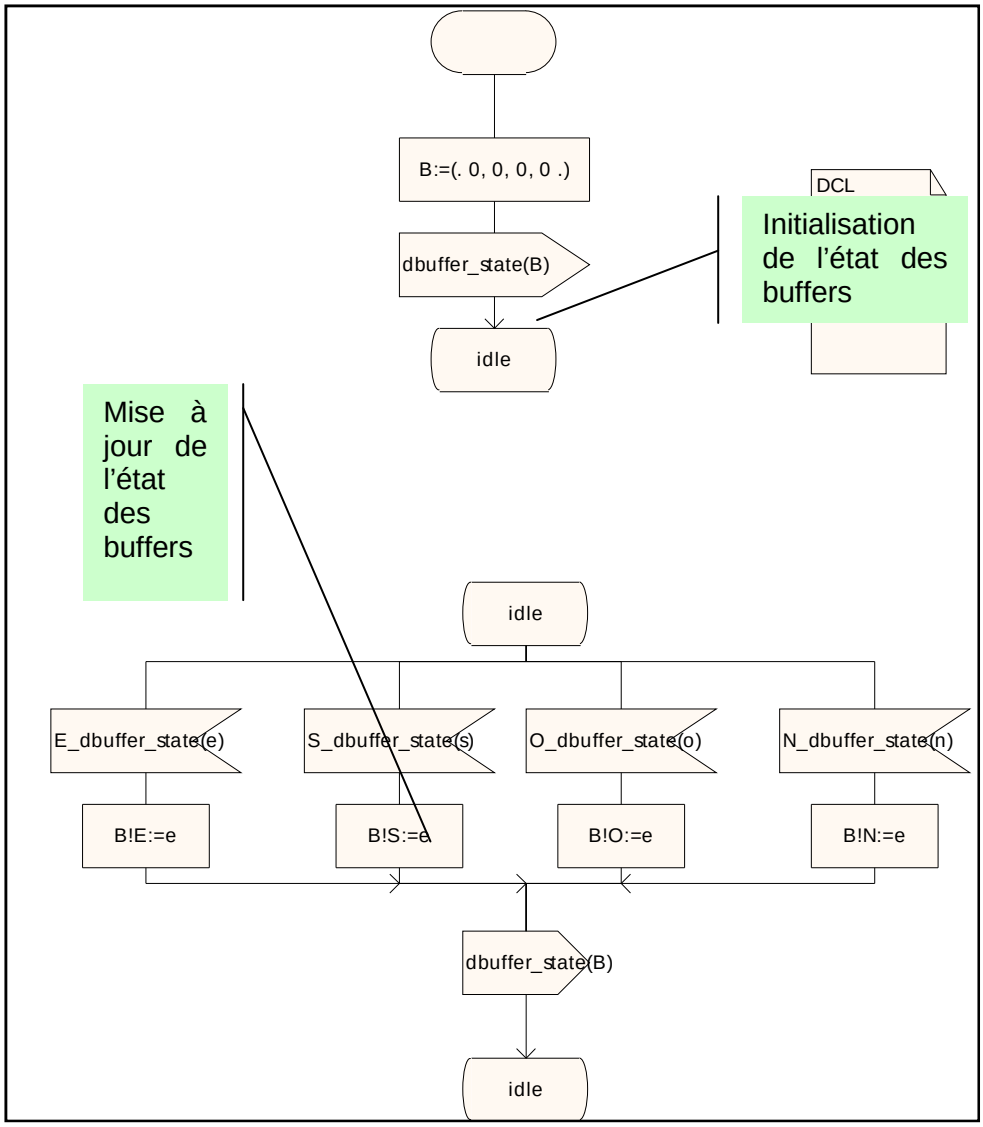


Figure 4.8 : Gestion des buffers

Sachant que les liens dans SDL sont associés à des buffers de taille infinie ; des buffers de taille modulable ont été introduits dans notre description, et ce à l'aide de la notion de Queue dans SDL, la figure 4.9 montrant la description d'un buffer communicant son état empty ou full au process de gestion de buffer afin de mettre à jour les informations nécessaires au routage (disponibilité des liens).

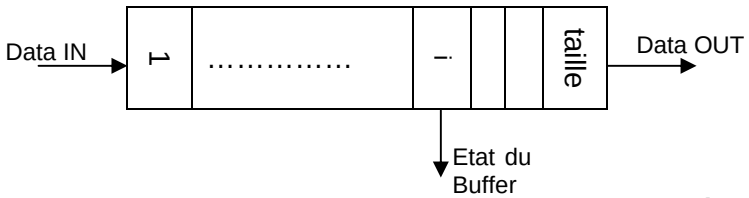


Figure 4.9:Interfaces du buffer modélisé

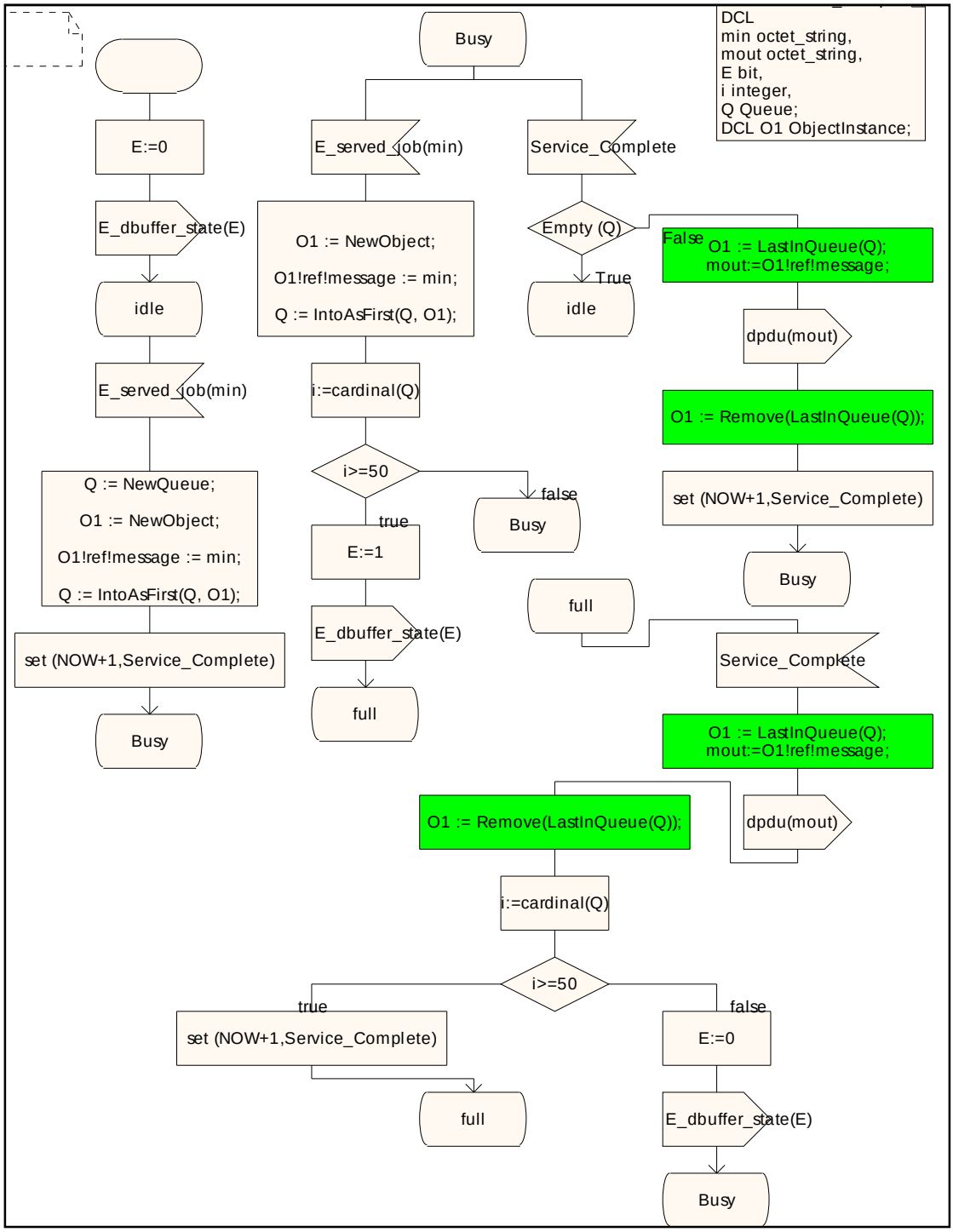


Figure 4.10 : Description SDL d'un buffer de taille modulable

3/ la communication avec la couche réseau

Cette communication consiste d'abord à recevoir le Network PDU, pour l'examiner et extraire l'information de routage permettant de faire passer la trame construite via le bon port.

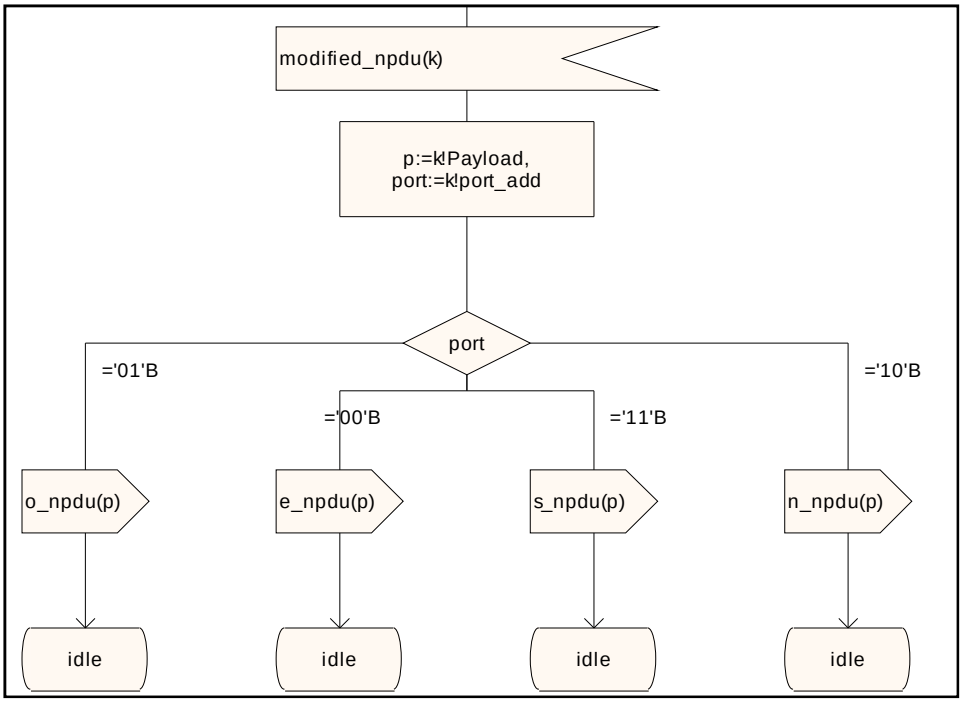


Figure 4.11 : Réception des PDU réseau et aiguillage des paquets vers les ports .

3 .2.2. Block désencapsulation

Dans le cas de notre modélisation une machine émettrice ou réceptrice est un Nœud NoC composé de 3 couches, donc de 3 processus d'encapsulation et de 3 autres pour la désencapsulation.

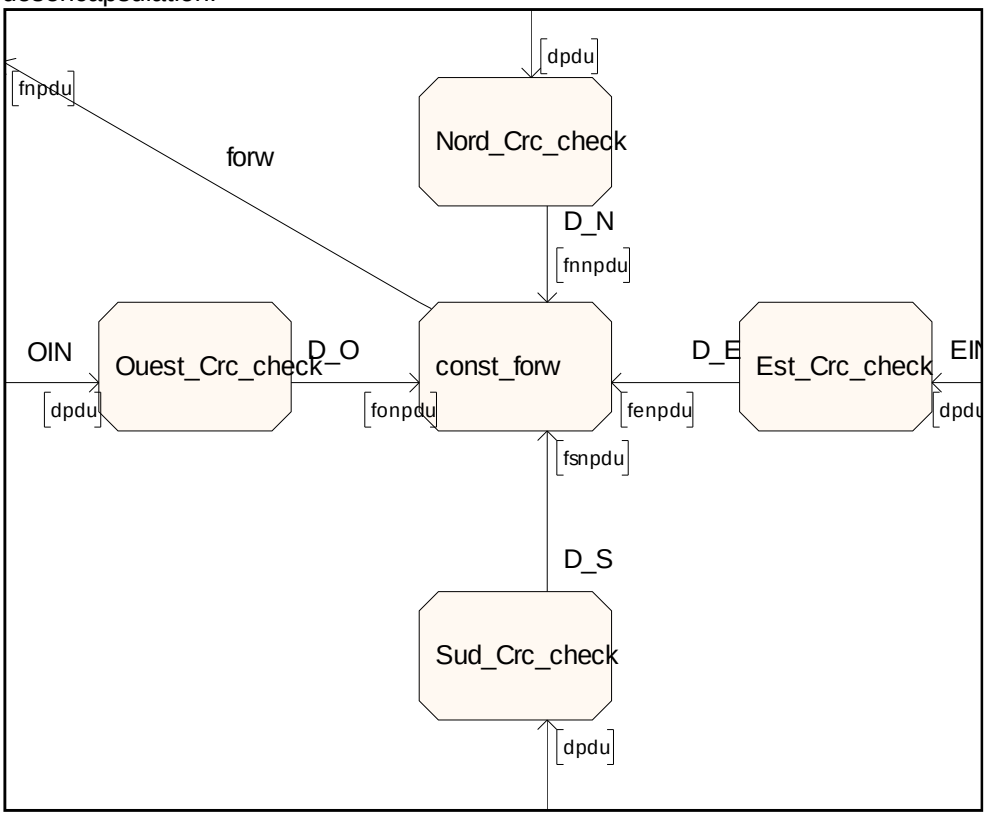


Figure 4.12 : La désencapsulation au niveau de la couche liaison de données

La couche Data Link de notre Nœud présente 4 interfaces de communication avec les 4 nœuds voisins, donc on a mis en ouvre 4 process (*_CRC_check) pour la réception des trames entrantes, afin de tester l'intégrité de leurs contenus ; pour les faire passer ensuite à un process central permettant de désencapsuler les data Link PDU afin de retrouver une structure de données adéquate à la couche réseau en d`autre terme elle construit le format PDU approprié a la couche réseau (network PDU) et de les transmettre à la couche réseau.

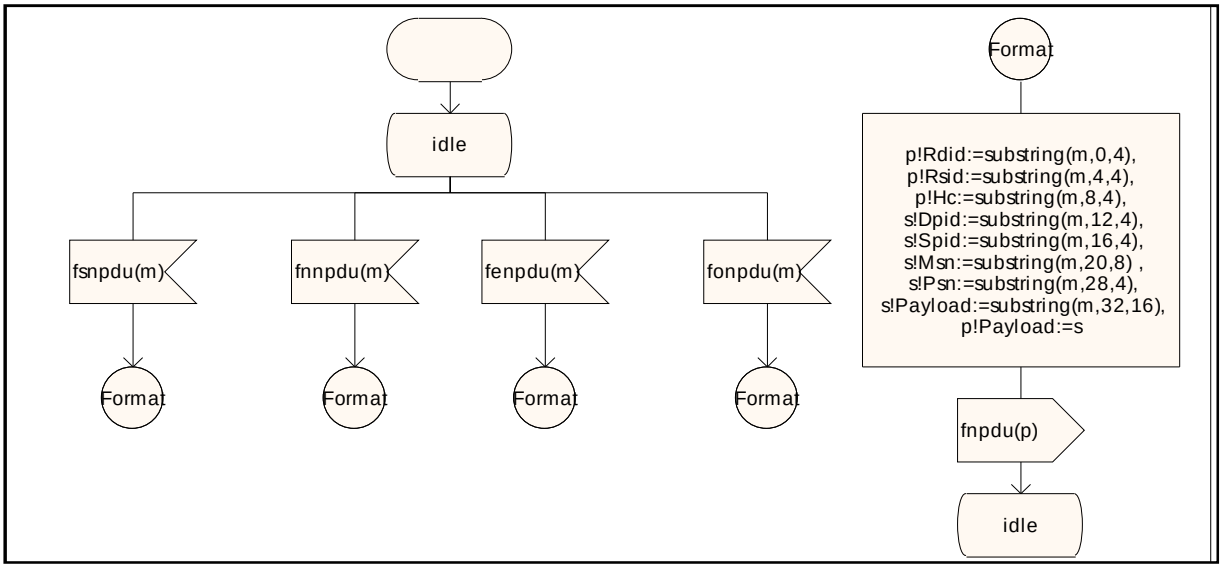


Figure 4.13 : Constitution et transmission des Network PDU

3.3. La couche réseau

Chaque nœud est identifié par son adresse (ligne, colonne) qui doit être affectée au démarrage du NOC, le nombre de bit accordé dépend de la taille de la matrice associé au réseau. Dans le cas de notre description, on suppose que le réseau est 4X4 donc 4 bits suffisent pour adresser correctement les nœuds de notre réseau.

De même que la couche data Link, la couche réseau est constituée de deux bloc *n_encapsulation* et *n_désencapsulation*

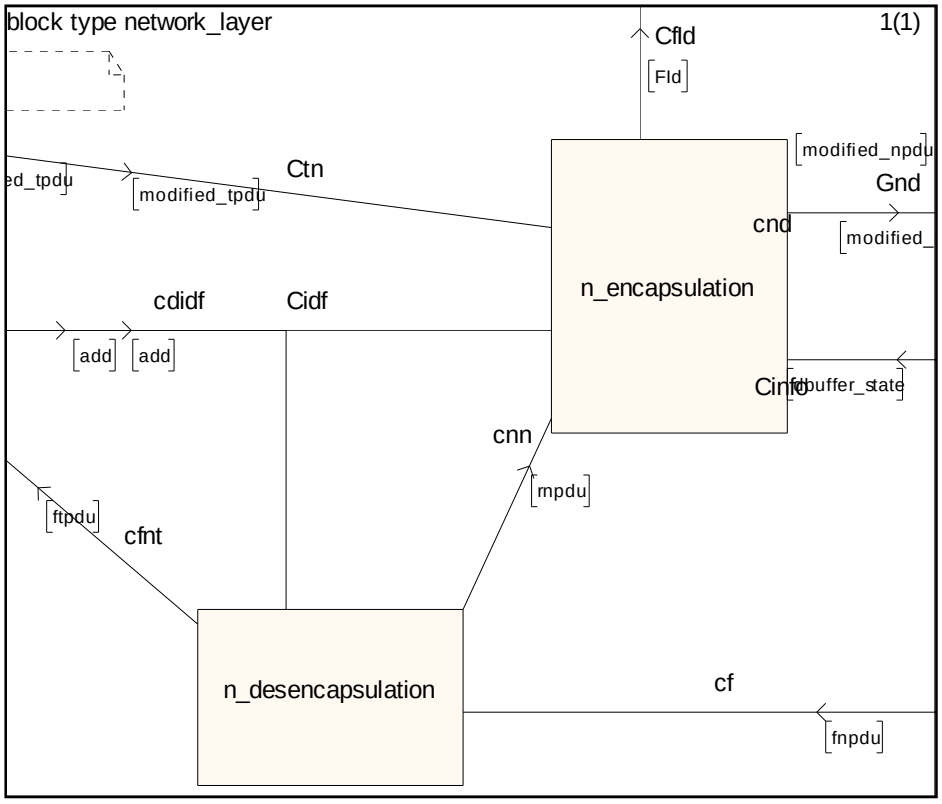


Figure 4.14 : Bloc de la couche Réseau

3 .3.1. Block encapsulation

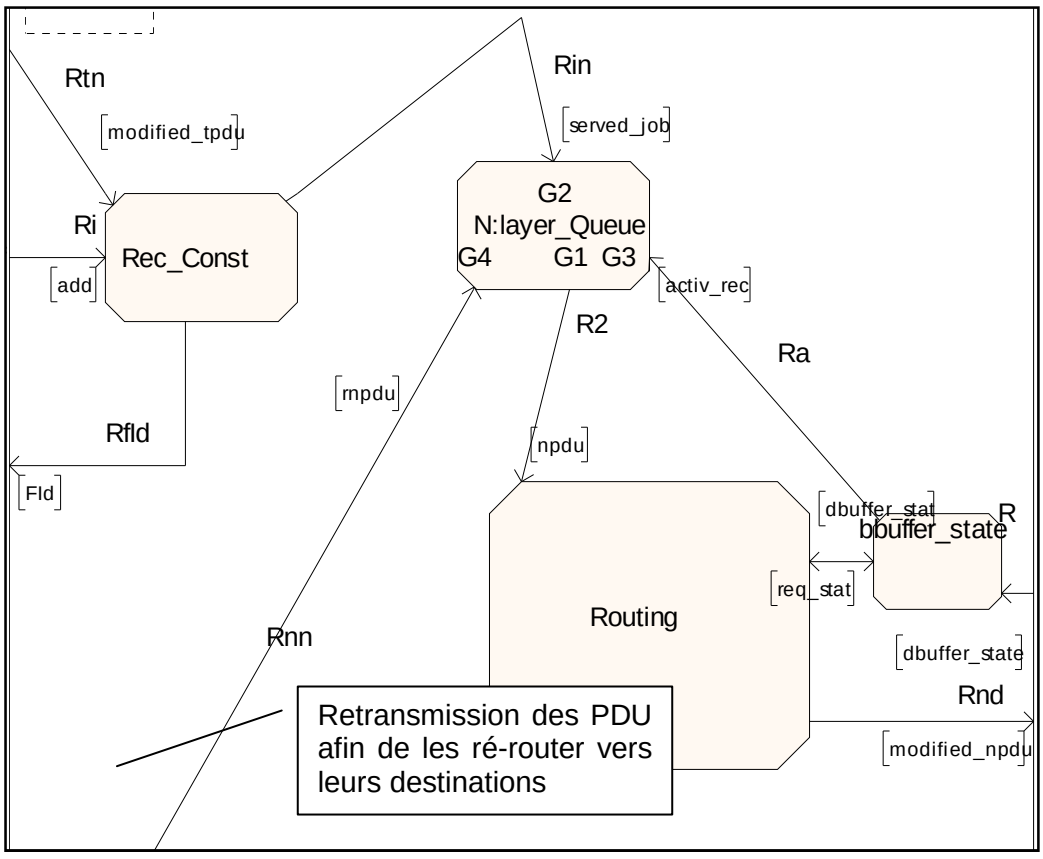


Figure 4.15 : Les process du bloc n_encpasulation

Les fonctions que nous avons implémentées sur une couche réseau sont :

1/le routage

A l`aide d`un algorithme de routage permettant de déterminer le chemin à emprunter entre la ressource émettrice et la ressource réceptrice.

C'est un point déterminant dans la modélisation d'un NoC. Il faut réussir le meilleur compromis entre une utilisation optimale des canaux de communication du réseau et un algorithme suffisamment simple à implémenter ne nécessitant pas des ressources matérielles trop importantes.

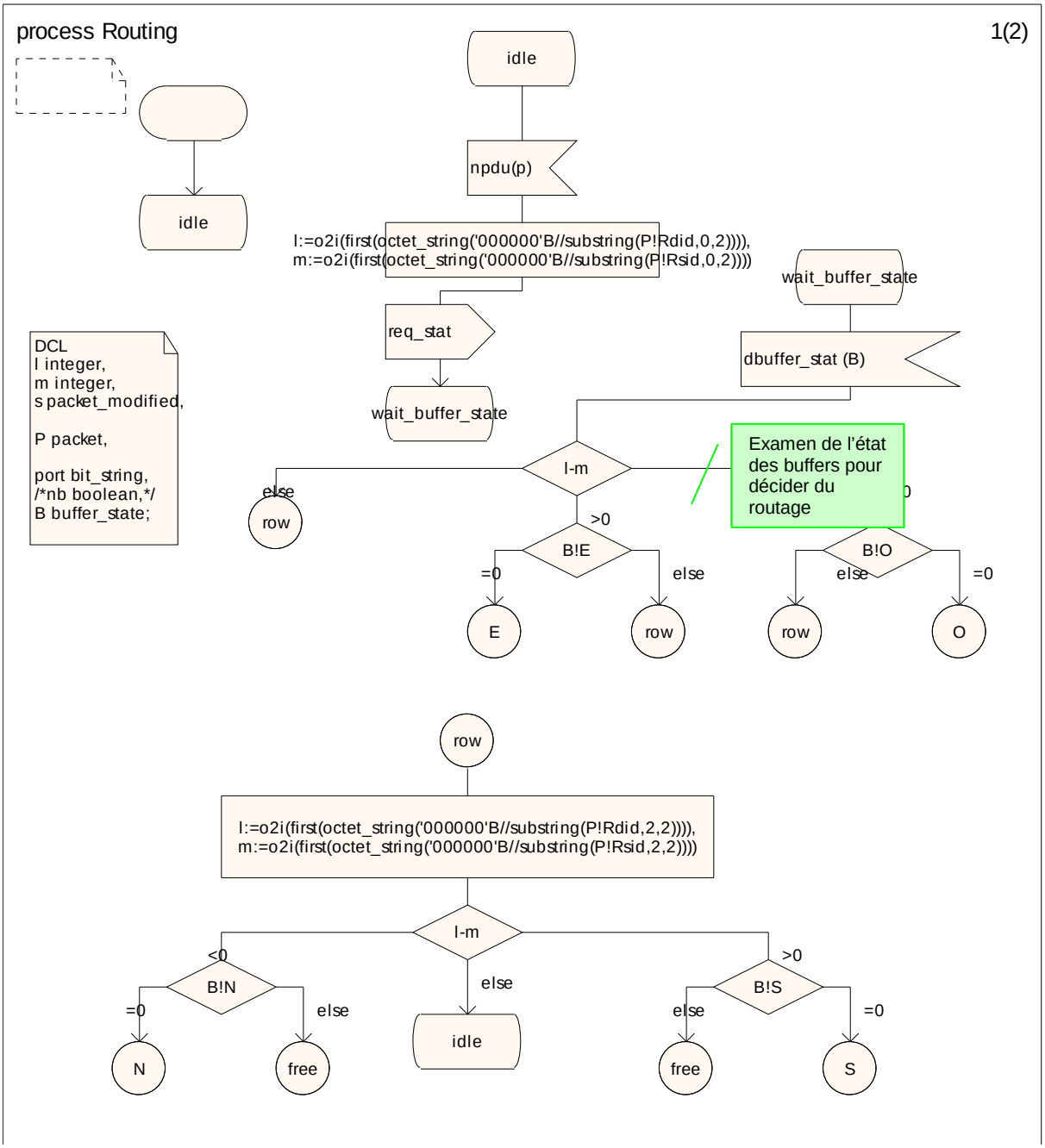


Figure 4.16 : Implémentation de l'algorithme de routage

Dans le cas de notre modélisation un simple algorithme de routage dynamique est retenu permettant de prendre des décisions sur la route à emprunter. Il consiste à comparer les adresses ressources, destination et source, suivant les colonnes afin d'émettre les paquets associés à droite ou à gauche suivant le résultat de comparaison, si par contre les deux ressources se trouvent sur la même colonne ou les buffers associés sont pleins, une seconde comparaison est effectuée sur les mêmes adresses mais cette fois suivant les lignes, si par défaut les buffers associés sont pleins, deux manières peuvent être retenue:

- 1/ Le paquet est transmis suivant la direction possédant un buffer associé libre
- 2/ Le paquet est abandonné.

La décision du routage est fortement liée à l'état des buffers; un process Buffer_state a été décrit afin de communiquer l'état des 4 buffers de la couche Data Link à la couche réseau afin de décider du port par lequel les trames vont transiter.

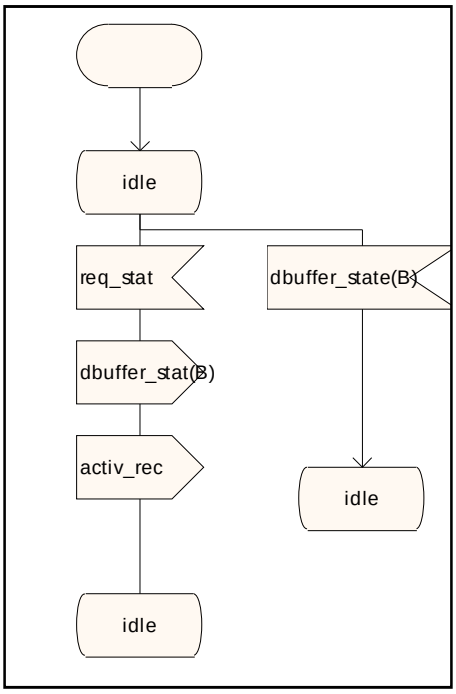


Figure 4.17 : Gestion de l'état de data Link buffers

Le choix d'un tel algorithme est justifié du fait que:

Il n'est pas nécessaire de déclarer de larges tables de routage exploitant beaucoup de ressources d'une part, la connaissance au préalable de la structure du réseau d'autre part. Cependant ce type d'algorithme présente un inconvénient relatif à la garantie de transport en temps réel mais ceci peut être admissible vu la surface des NoC permettant un transfert rapide.

2/la gestion de la validité des paquets

Un compteur temps est associe a chaque paquet transmis afin d`éviter qu'un paquet tourne en rond dans le réseau.

3/ la communication avec la couche Transport

4 / la communication avec la couche data Link

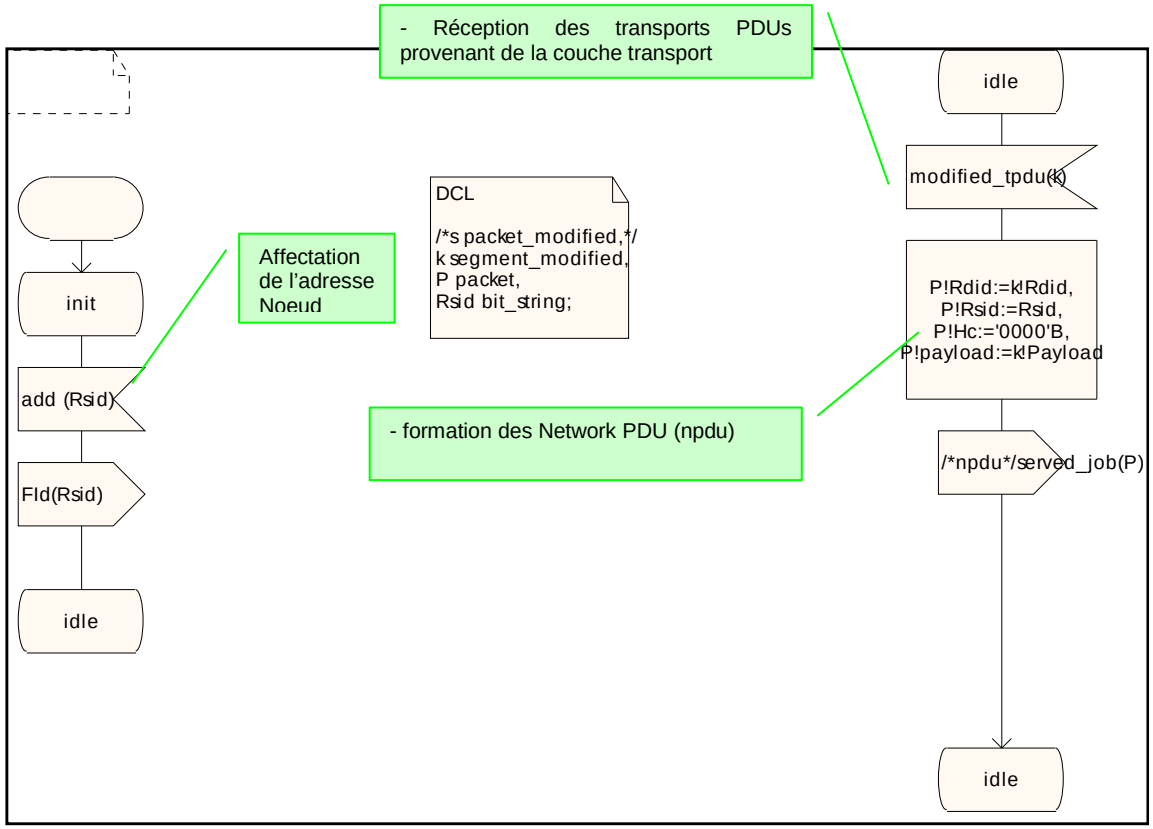


Figure 4.18 : Initialisation et formation des Network PDU

3 .3.2. Block désencapsulation

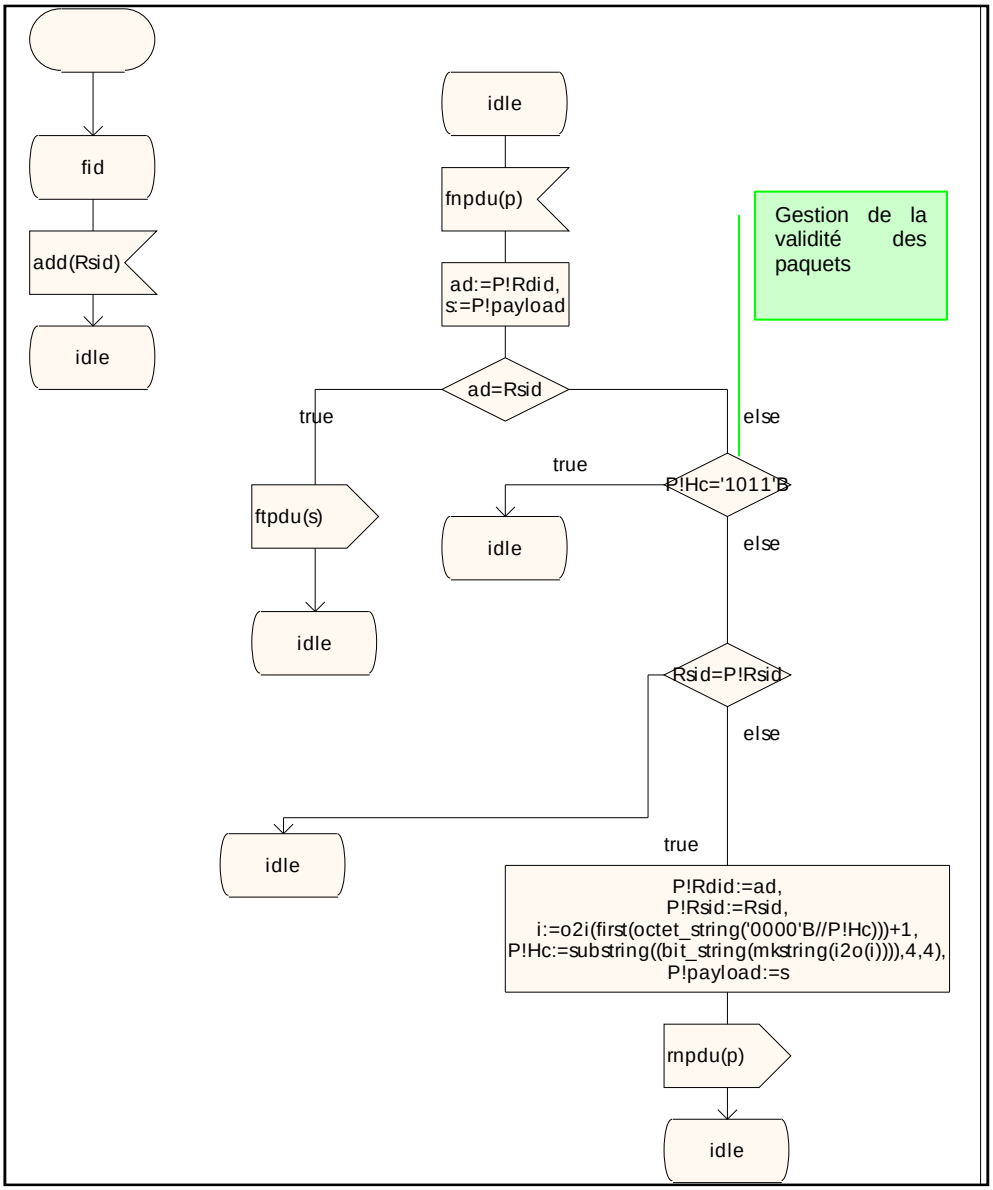


Figure 4.19 : Description de la désencapsulation au niveau de la couche réseau

3.4. La couche transport

Cette couche a pour but la segmentation et le réassemblage des paquets respectivement transmis ou reçu. On considère qu'au niveau d'un nœud, on peut avoir N processus locaux. Chaque processus peut transmettre à un nœud cible un message qui sera fragmenté en plusieurs segment.

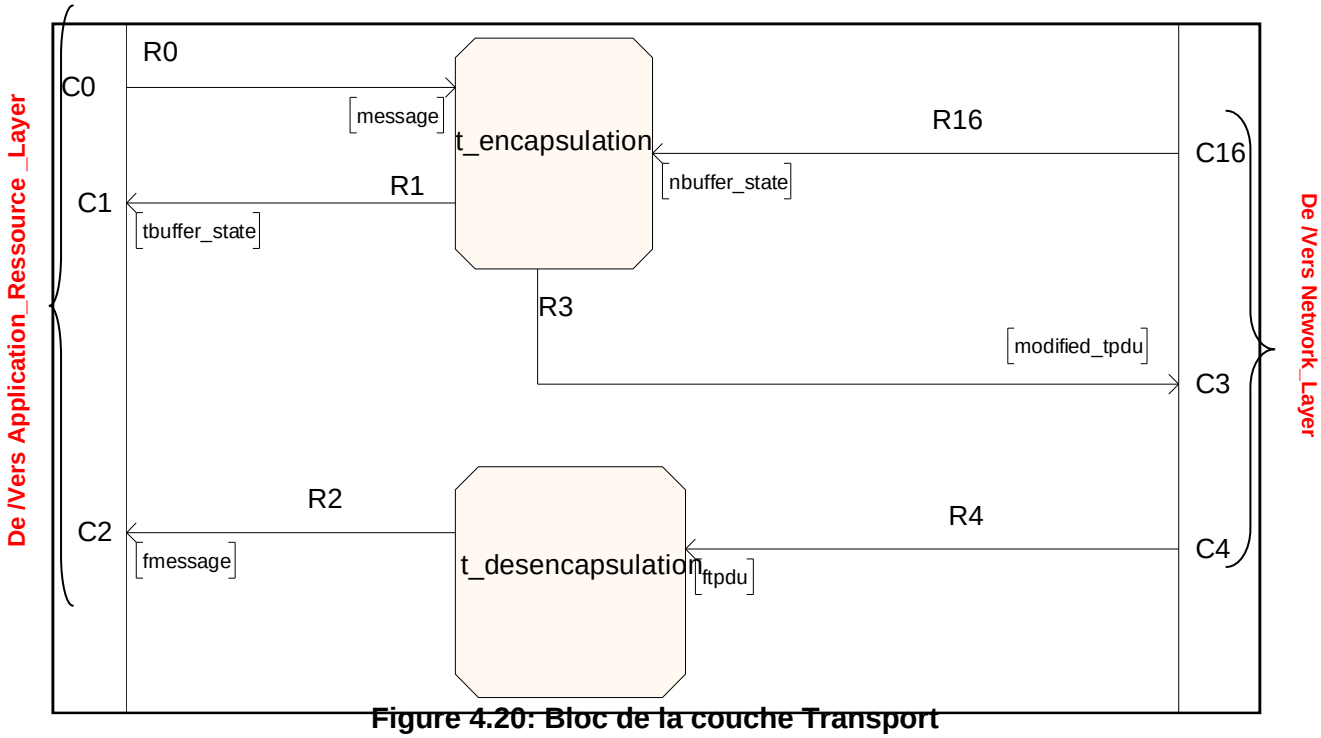


Figure 4.20: Bloc de la couche Transport

On déduit les fonctions à réaliser :

1/l'émission des segments en introduisant :

MSN (Message Séquence Number): permettant d`identifier le message et de définir le nombre de paquets associes a ce message

PSN (Packet Sequence Number) permet d`indiquer l`ordre du paquet traité afin d`assurer la reconstruction du message.

2/réception et reconstruction des messages à partir de MSN et PSN.

3/ la communication avec la ressource

Consiste à transmettre le message ainsi reconstruit

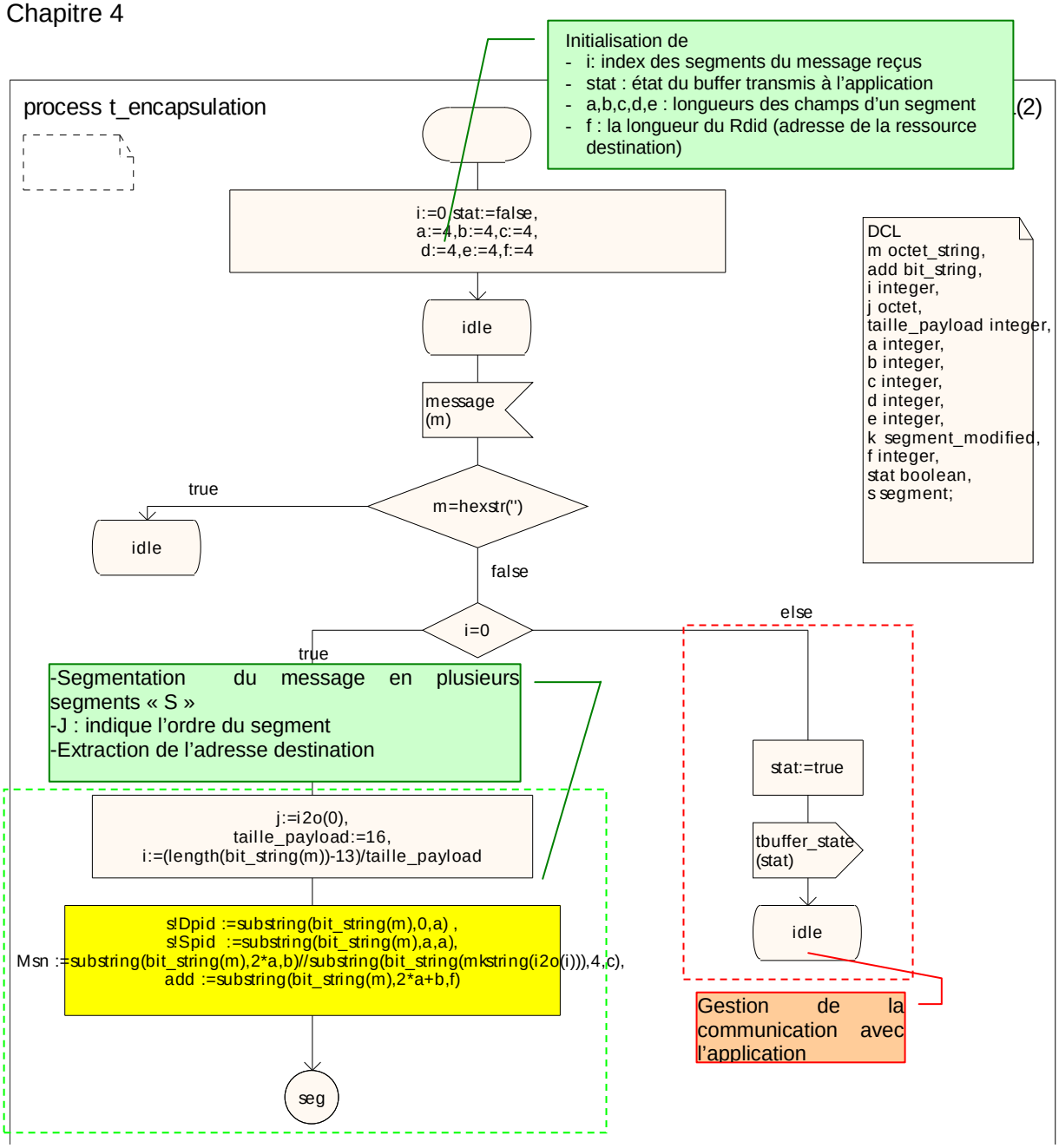


Figure 4.21: Description des fonctionnalités de la couche transport

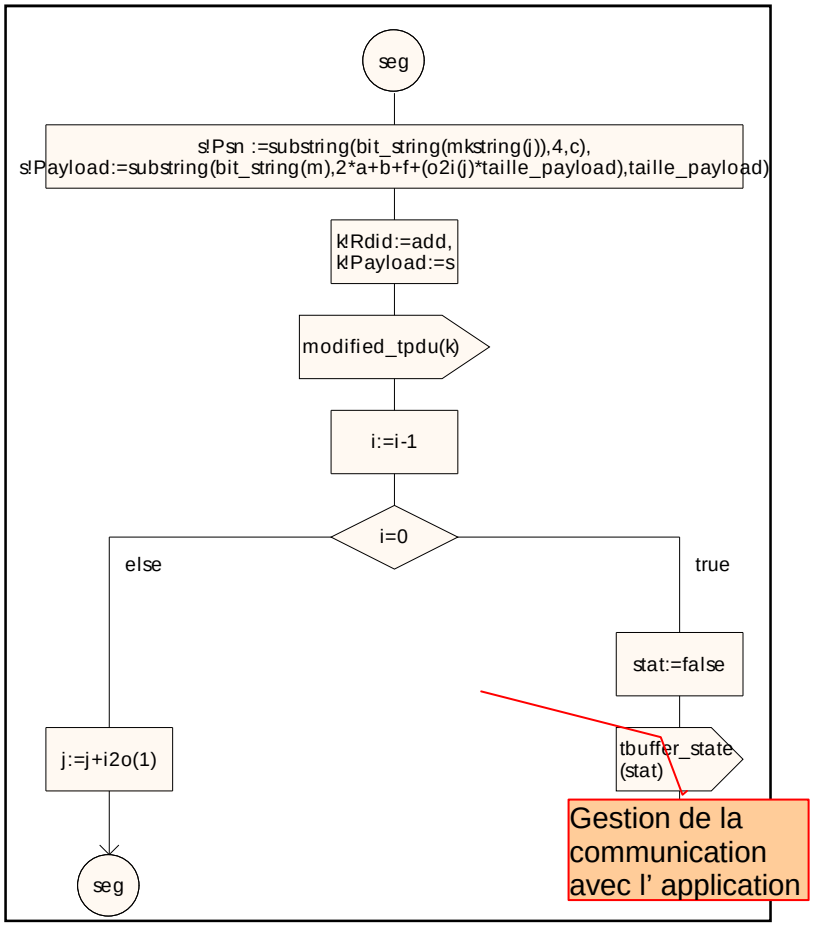


Figure 4.22: Description des fonctionnalités de la couche transport (suite)

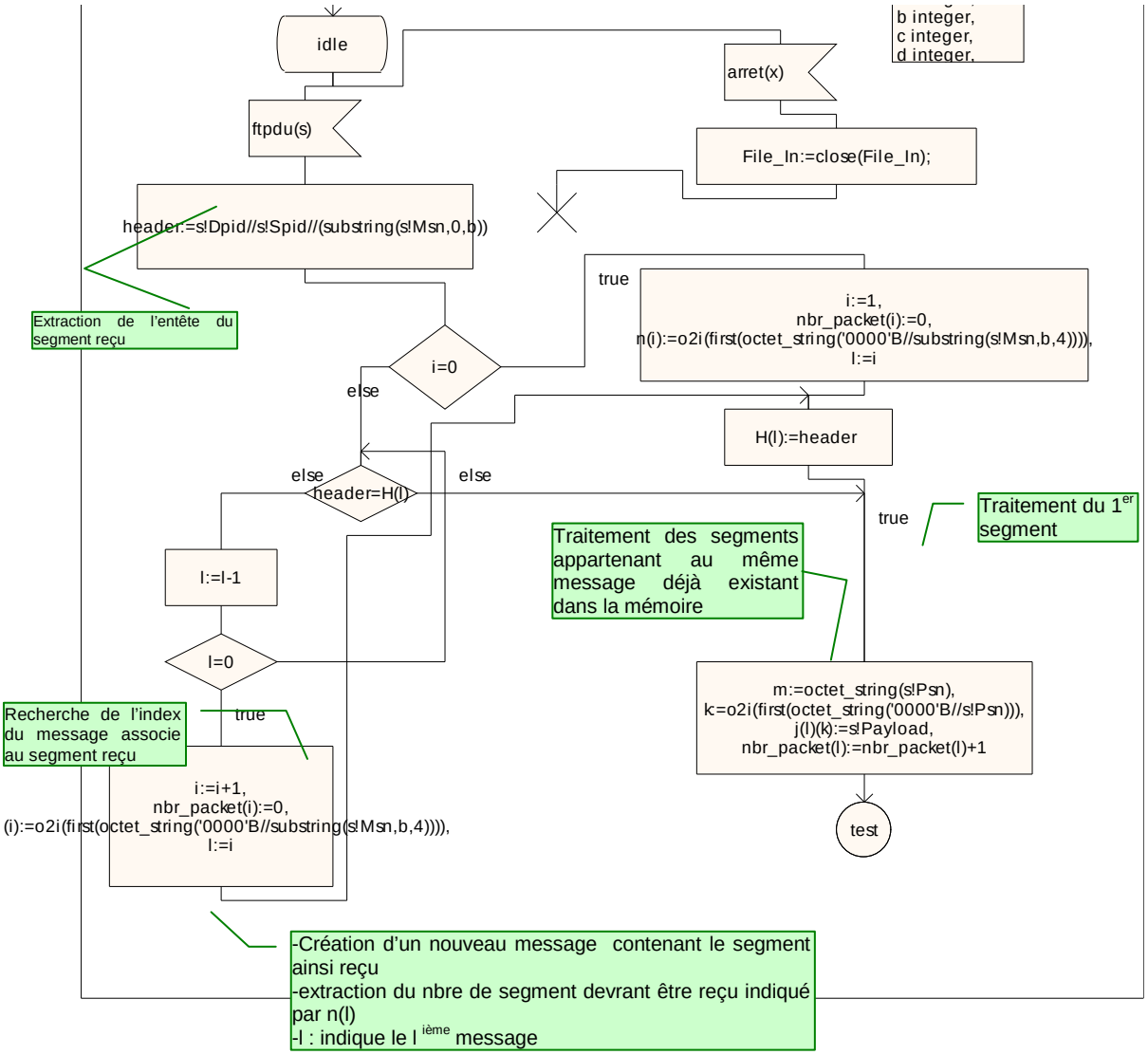


Figure 4.23: Description du process de désencapsulation au niveau de la couche transport

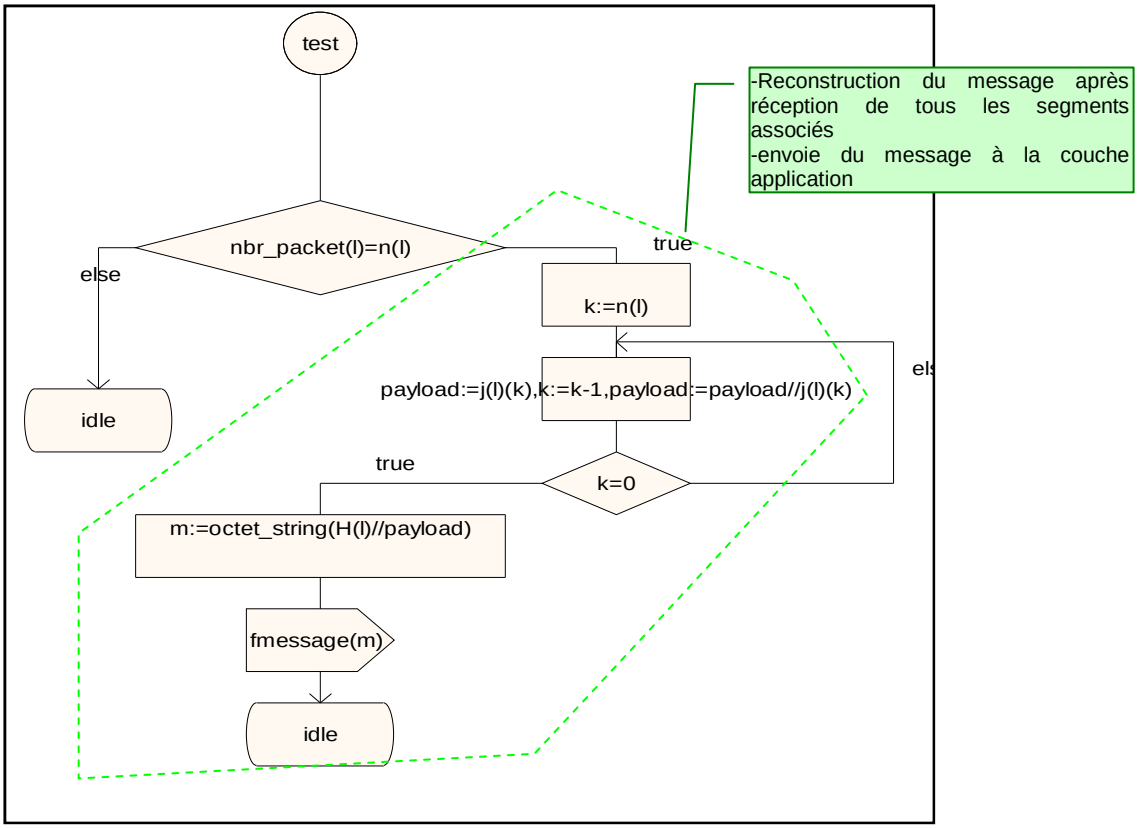


Figure 4.24: Description du process de désencapsulation au niveau de La couche transport (suite)

3.5. Simulation

Dans cette section nous allons décrire les étapes à suivre pour préparer notre simulation. En effet, comme la montre la figure 4.25, nous devons tout d'abord générer du code C à partir de notre modèle SDL.

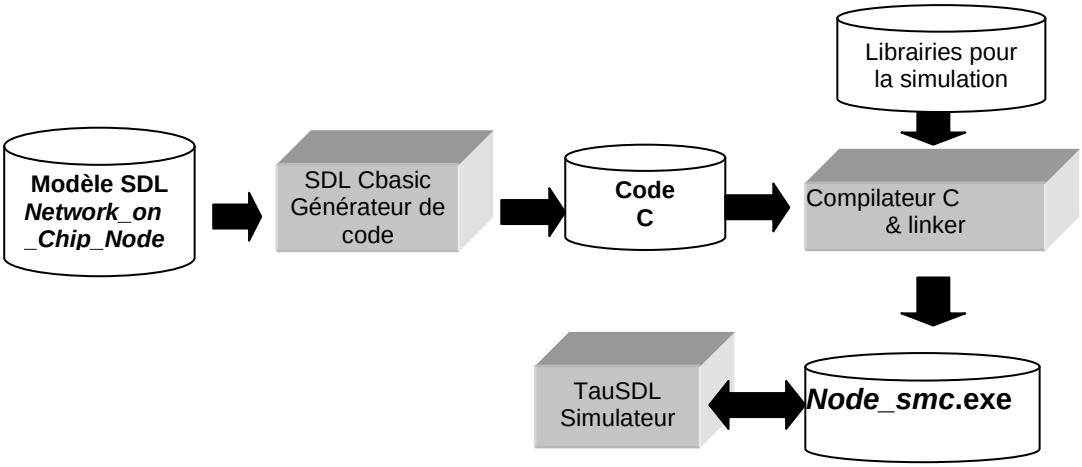


Figure 4.25 : Schéma de préparation de la simulation

Ensuite, le code C généré devra être compilé pour créer un exécutable avec lequel le simulateur peut interagir. Ces opérations se font à travers de l'instruction ***make*** de TauSDL.

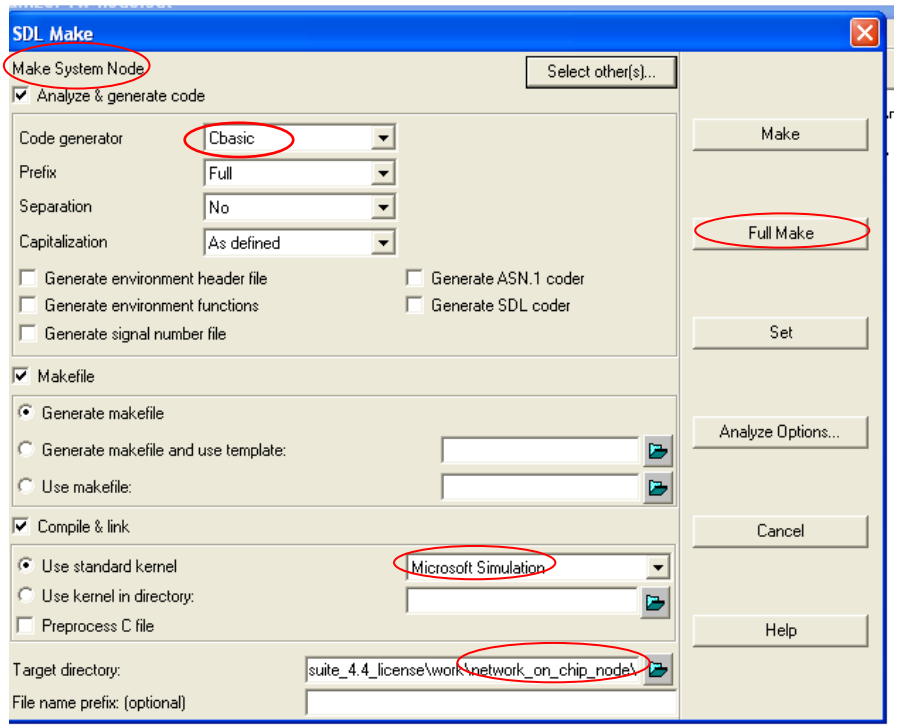


Figure 4.26 : La capture d'écran *make* de TauSDL

3.5.1. Lancement du simulateur

Après la génération du code C, nous pourrons maintenant procéder la simulation. Le simulateur de l'environnement de développement TauSDL nous permet de voir d'une manière graphique et très intuitive le comportement de notre système en utilisant l'éditeur MSC (message sequence chart). La simulation consiste à envoyer des messages applications ou trames de nœuds voisins.

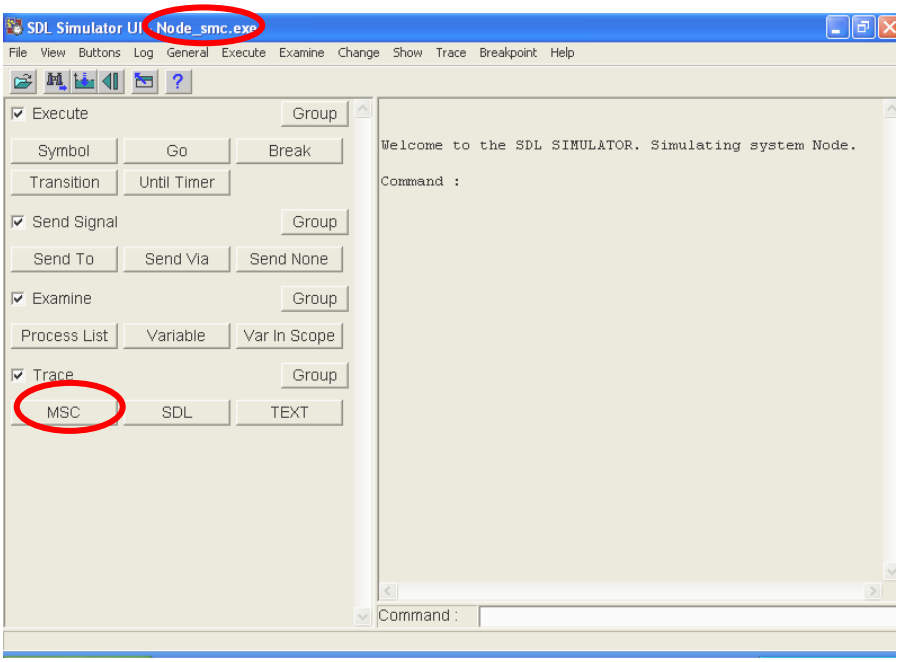


Figure 4.27: Capture d'écran de l'interface graphique du simulateur

3.5.2. Simulation MSC

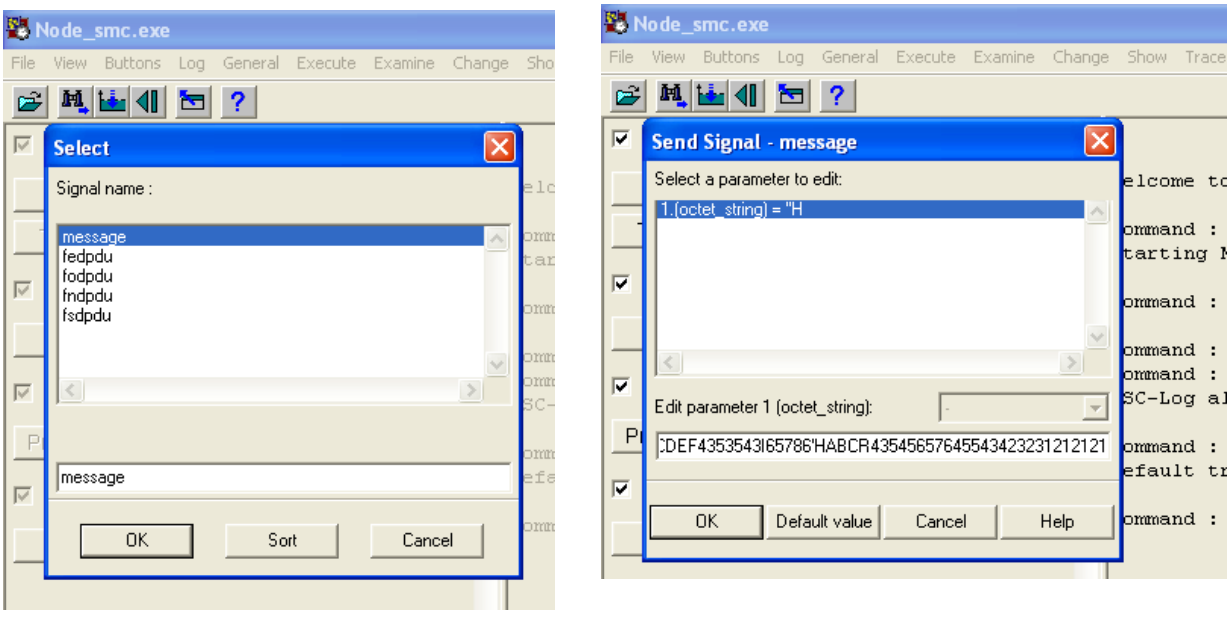


Figure 4.28: Capture d'écran de l'interface graphique du simulateur

La figure 4.29 donne une partie des séquences de fonctionnement dérivée du diagramme MSC.

Ce digramme a été déroulé en envoyant le message arbitraire suivant :

```
'ABCDEF435261617171129202026752423231212ABCD0'H
```

Celui-ci sera pris par les couches transport et réseau à travers les processus qui y sont actifs.

Tableau 4.1 : Transport PDU

(.'1001'B	,(.'1010'B	,'1011'B	,'100001010'B	'0000'B,	'1011110111101000'B.).)
Address de destination	Indicateurs de process		Le message a été segmenté en 10 segments	Premier segment envoyé	Première partie de la charge utile

Le Tableau 4.1 donne un transport PDU avec l'information de l'adresse destination

Tableau 4.2 : Network PDU

'00'B,	1001'B	'0001'B	'0000'B,	(.'1010'B, '1011'B, '100001010'B, '0000'B, '1011110111101000'B .) .) .)
Information sur le port de sortie de la trame après avoir comparé les colonnes sources et Destination 10-00> 0 => port E	Addresses Resources		Indicateur du nombre de Sauts	Transport PDU- Payload

Le Tableau 4.2 donne un Network PDU avec les informations de routage

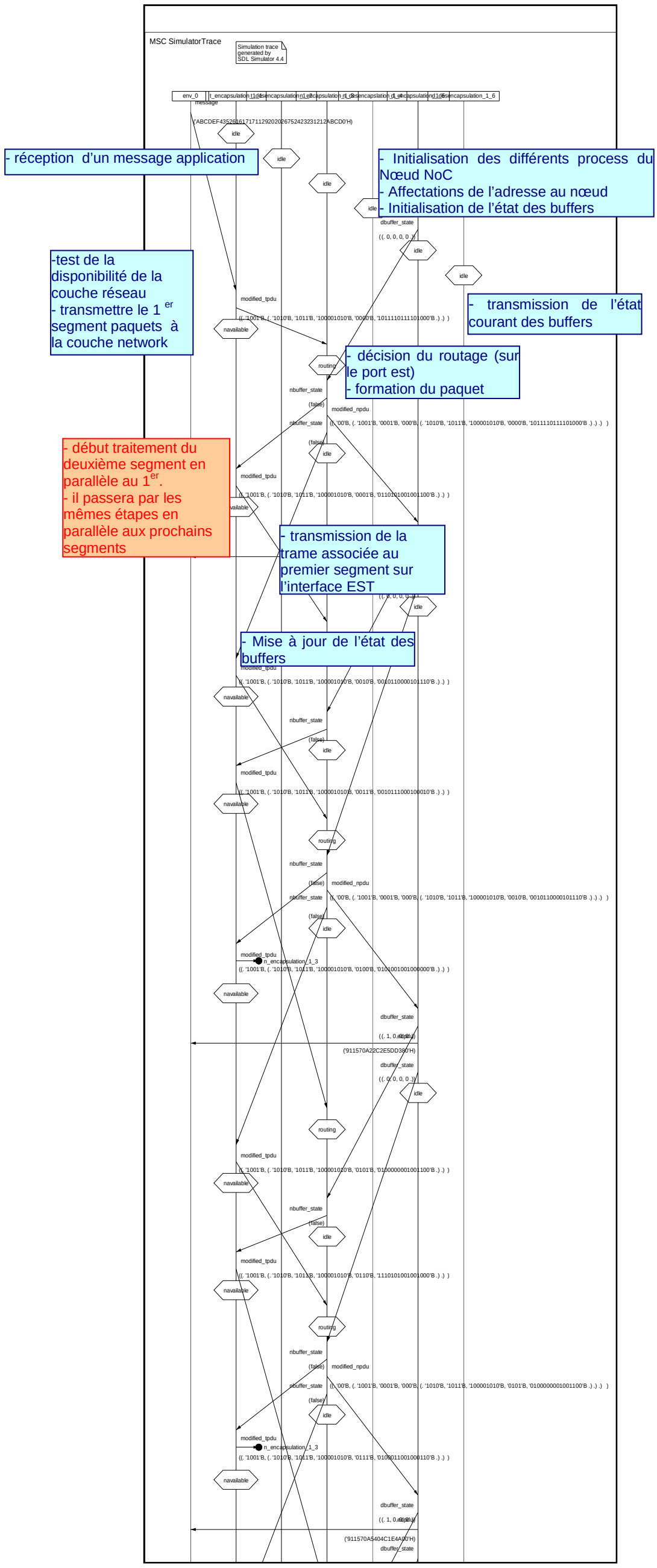


Figure 4.29 : Exemple de Trace de fonctionnement à travers le diagramme MSC

4. Modélisation et validation d'un Réseau sur Puce (4X4)

Le but de notre travail consiste à construire un Nœud Network On Chip avec les fonctionnalités nécessaires à un réseau de communication adaptées aux contraintes de communication sur Puce.

Ceci étant dit, la description ci-dessus présentée d'un Nœud doit nous permettre de réaliser un réseau de communication sur puce à partir de la multiplication du nombre des nœuds et de les organiser suivant le schéma type de la topologie régulière précédemment introduite spécifiée dans le cahier des charges (mesh 4X4).

Trois manières (voir chapitre 3) peuvent être envisagées afin de valider la modélisation du nœud présenté et de vérifier le fonctionnement du réseau dérivé. Dans le cas de notre développement, nous avons opté pour l'utilisation de SDL de bout en bout, autrement dit de la modélisation du Nœud jusqu'à la génération du réseau de communication.

Pour ce faire une notion très importante dans SDL a été utilisée, c'est celle de la programmation ou la conception dite Orientée Objet permettant de générer des objets type assurant ainsi leurs réutilisation en les déclarant dans des packages créés.

4.1. SDL, langage orienté OBJET

Pour une réutilisation facile de définition de type de données et de structures dans différents systèmes, celles ci peuvent êtres mises dans des *packages* comme c'est montré dans la figure 4.4. Ces packages, peuvent être importés de n'importe quel système (figure 4.30).

Aussi, SDL permet la réutilisation de structure (systèmes, blocks, processus, services).En effet, chaque structure peut être typée, on peut définir des types de systèmes, des types de blocs, des types de processus et des types de services. La figure 4.30 donne un exemple d'utilisation de process type.

Le package ***Untitled***, sera de même utilisé pour générer le réseau de nœud dont la structure est définie dans le bloc type : « NoC_Node »

En résumé, nous avons créé dans le cas de notre modélisation (04) librairies :

1. Une librairie des structures de données utilisées « packet_data_unit_definition ».
2. Une librairie de procédures « Procedure ».
3. Une librairie des process type « Process_Type_package ».
4. Une librairie des blocs type « Untitled ».

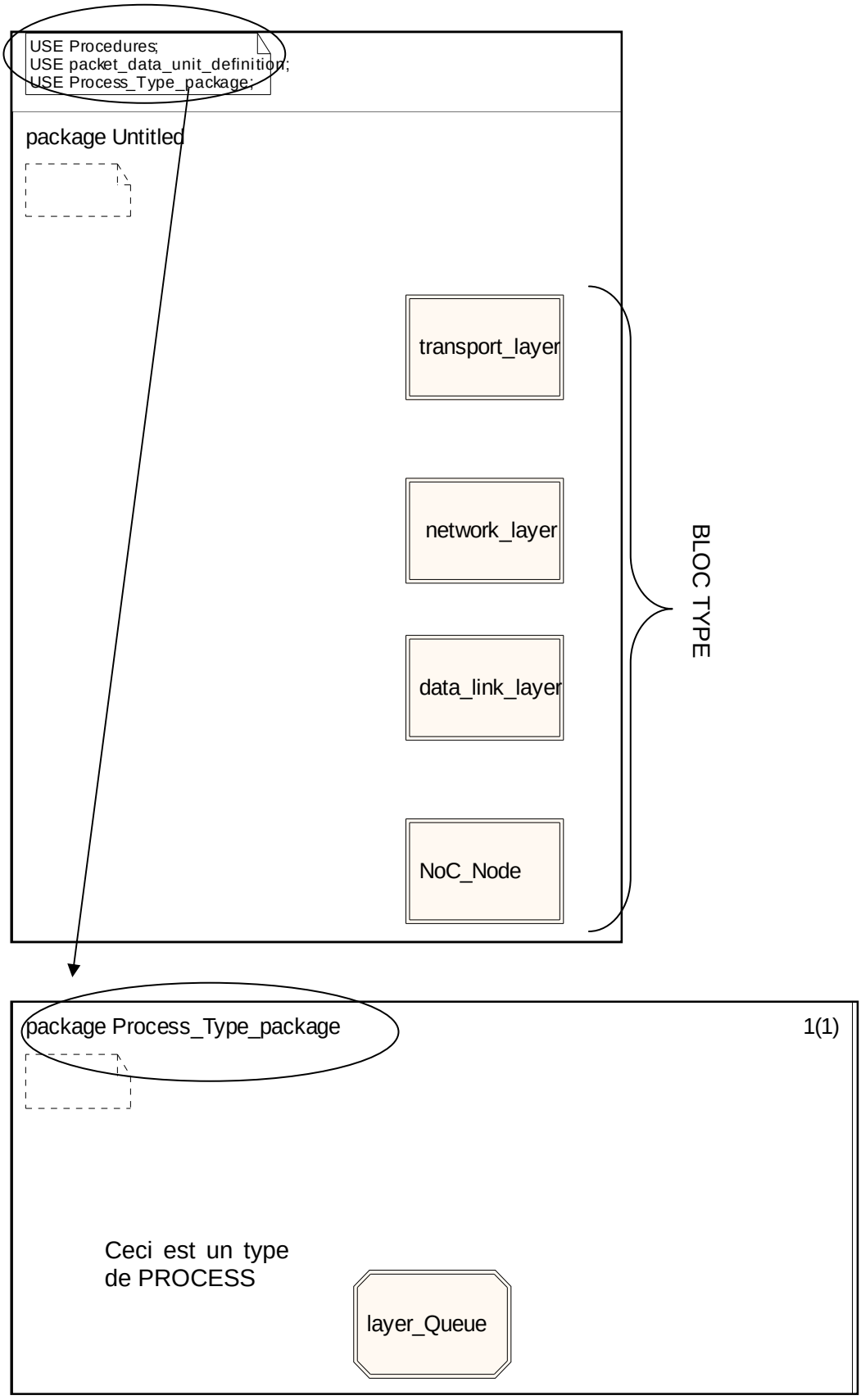


Figure 4.30: Exemple d'utilisation de package, bloc et process type

4.2. Architecture du Network On Chip Node Type

Comme il a été précédemment mentionné, un nœud est constitué de trois (03) couches (layer) ; la figure 4.31 présente un *nœud TYPE* qui en plus des fonctions décrites dans les paragraphes précédents ; il a été complété par d'autres process et procédures relatifs à la partie évaluation de performances du système modélisé.

Un objet type doit présenter des portes (Gates), interfaces qui permettront de raccorder cet objet type avec les autres blocs sans être obliger de reproduire la totalité de la description système de cet objet.

On peut remarquer, que ce nœud type ne présente aucune entrée de messages de l'application associée à ce nœud. Ceci est dû à l'utilisation de la fonction de lecture Fichier, permettant de lire un contenu généré et sauvegardé dans un fichier jouant le rôle d'une application.

Tous nœud avant son démarrage doit être activé et initialisé ceci est possible par l'entrée de l'adresse qui en plus de l'activation du nœud, elle permet son adressage.

Un nœud communique avec ses quatre voisins via des ports d'entrée et de ports de sortie faisant transiter des data Link PDU ***dpdu.***

Les 3 couches constituantes le nœud type sont eux même définies sous formes de blocs type. Ils sont déclarés dans le package Untitled présenté dans la figure 4.30.

La figure 4.32 propose le code texte de la description SDL associée à la description présentée dans la figure 4.31.

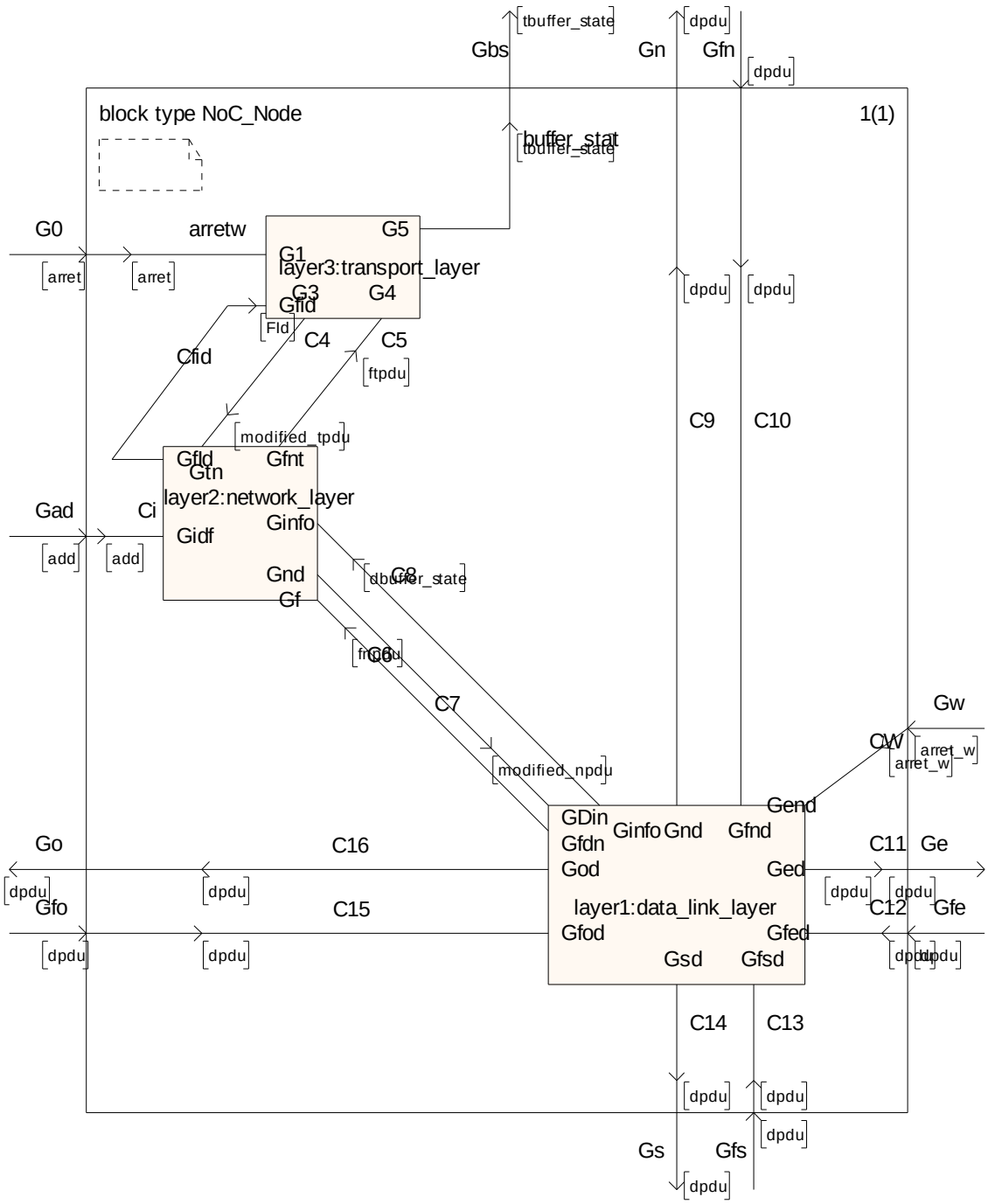


Figure 4.31:Architecture d'un Nœud Type avec les différentes interfaces

```
block type NoC_Node;
gate Gbs out with tbuffer_state;
gate Gn out with dpdu;
gate Gfn in with dpdu;
gate Go out with dpdu;
gate Ge out with dpdu;
gate Gs out with dpdu;
gate G0 in with arret;
gate Gad in with add;
gate Gw in with arret_w;
gate Gfo in with dpdu;
gate Gfe in with dpdu;
gate Gfs in with dpdu;
substructure NoC_Node;
channel C10 from env via Gfn to layer1 via Gfnd with dpdu;
endchannel C10;
channel buffer_stat from layer3 via G5 to env via Gbs with tbuffer_state;
endchannel buffer_stat;
channel arretw from env via G0 to layer3 via G1 with arret;
endchannel arretw;
channel C4 from layer3 via G3 to layer2 via Gtn with modified_tpdu;
endchannel C4;
channel C5 from layer2 via Gfnt to layer3 via G4 with ftpdu;
endchannel C5;
channel Cfid from layer2 via GfId to layer3 via Gfid with FId;
endchannel Cfid;
channel Ci from env via Gad to layer2 via Gidf with add;
endchannel Ci;
channel C7 from layer2 via Gnd to layer1 via GDin with modified_npdu;
endchannel C7;
channel CW from env via Gw to layer1 via Gend with arret_w;
endchannel CW;
channel C8 from layer1 via Ginfo to layer2 via Ginfo with dbuffer_state;
endchannel C8;
channel C9 from layer1 via Gnd to env via Gn with dpdu;
endchannel C9;
channel C6 from layer1 via Gfdn to layer2 via Gf with fnpdu;
endchannel C6;
channel C16 from layer1 via God to env via Go with dpdu;
endchannel C16;
channel C11 from layer1 via Ged to env via Ge with dpdu;
endchannel C11;
channel C15 from env via Gfo to layer1 via Gfod with dpdu;
endchannel C15;
channel C12 from env via Gfe to layer1 via Gfed with dpdu;
endchannel C12;
channel C14 from layer1 via Gsd to env via Gs with dpdu;
endchannel C14;
channel C13 from env via Gfs to layer1 via Gfsd with dpdu;
endchannel C13;
block layer3:transport_layer;
block layer2:network_layer;
block layer1:data_link_layer;
endsubstructure NoC_Node;
endblock type NoC_Node;
```

Figure 4.32: Code Texte SDL du Nœud type

4.3. Listing des packages utilisés

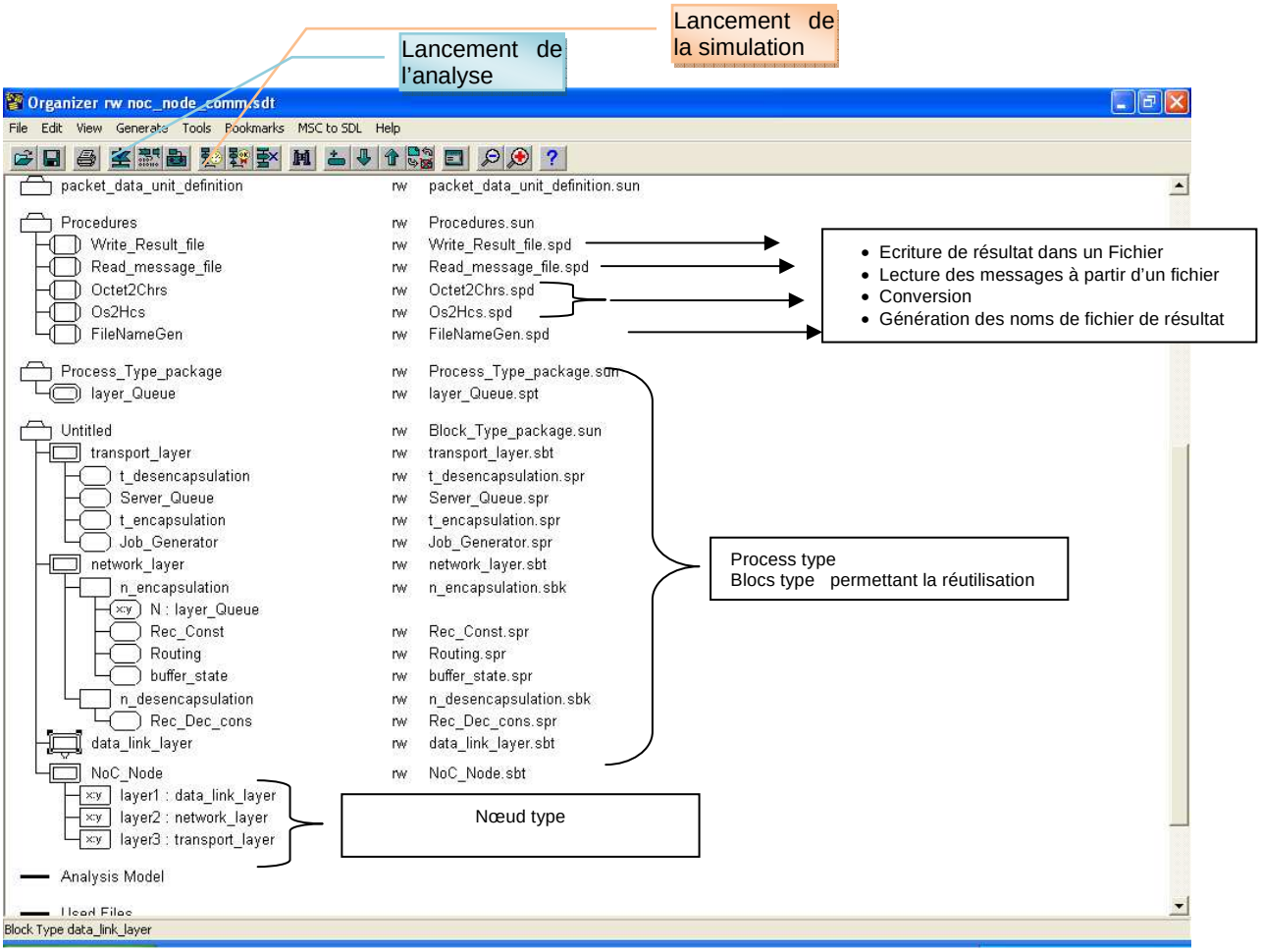


Figure 4.33 : Capture d'écran TauSDL

4.4. Évaluation de performance

Le but de cette partie est de décrire le modèle d'évaluation des performances qu'on a réalisé sous SDL.

Il faut noter qu'une description du comportement fonctionnel du système et le modèle de performance du même système sont différents, même si les deux sont décrits sous SDL. Ces deux modèles décrivent deux différents aspects du même système. C'est pour cette raison qu'une phase de modélisation dans le développement de la simulation de performances est nécessaire.

Modèle de queue :

La plupart des modèles de performance peuvent être considérés comme des modèles de queue ou un réseau de queue. Un modèle de queue type est donné dans la figure ci-après :

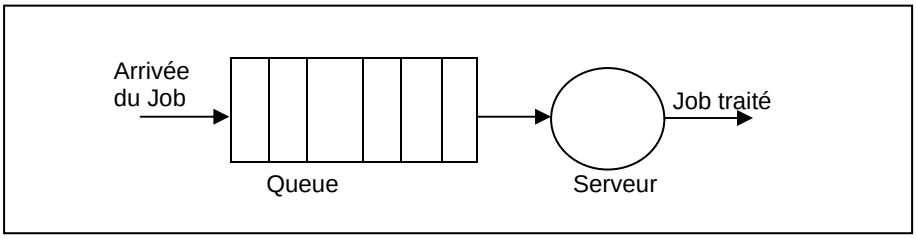


Figure 4.34 : Modèle de queue

Le modèle contient du job qui nécessite un service du serveur, ce job pourra attendre dans une queue pour avoir le service voulu. Les propriétés de base de ce modèle sont :

1. La durée du service pour un job,
2. La discipline de la queue; i.e. ordre d'insertion du job dans la queue et suppression de la queue pour avoir le service. Les disciplines typiques sont

    - FCFS (First Come- First served) premier arrivé premier servi;
    - Ordre de priorité.

3. Temps entre l'arrivé du job

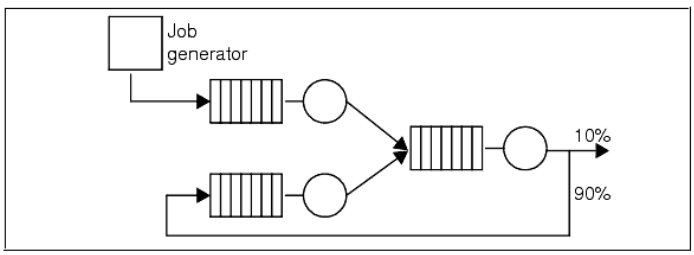


Figure 4.35 : Simple modèle de réseau de queue

Dans un modèle de queue or réseau de queue, il est intéressant d'estimer par exemple :

- Pour le Job: temps total dans un system, et temps total d'attente
- Pour le serveur : charge
- Pour queue : la taille moyenne et maximum de la queue

De telles estimations peuvent s'obtenir par 2 voies ; en utilisant les théories mathématiques de la théorie des queues ou par des mesures durant les simulations.

Dans le cas de notre modélisation des process ont été ajoutés à la modélisation présentée précédemment et des process ont été modifiés en définissant d'autres procédures permettant :

- la lecture des fichiers contenant les messages à traiter ;
- le traçage par l'écriture des résultats et informations utiles dans des fichiers.

Les principaux process qui ont été ajoutés à notre description sont :

- générateur de trafic (Job Generator)
- queue du serveur (Serveur Queue)

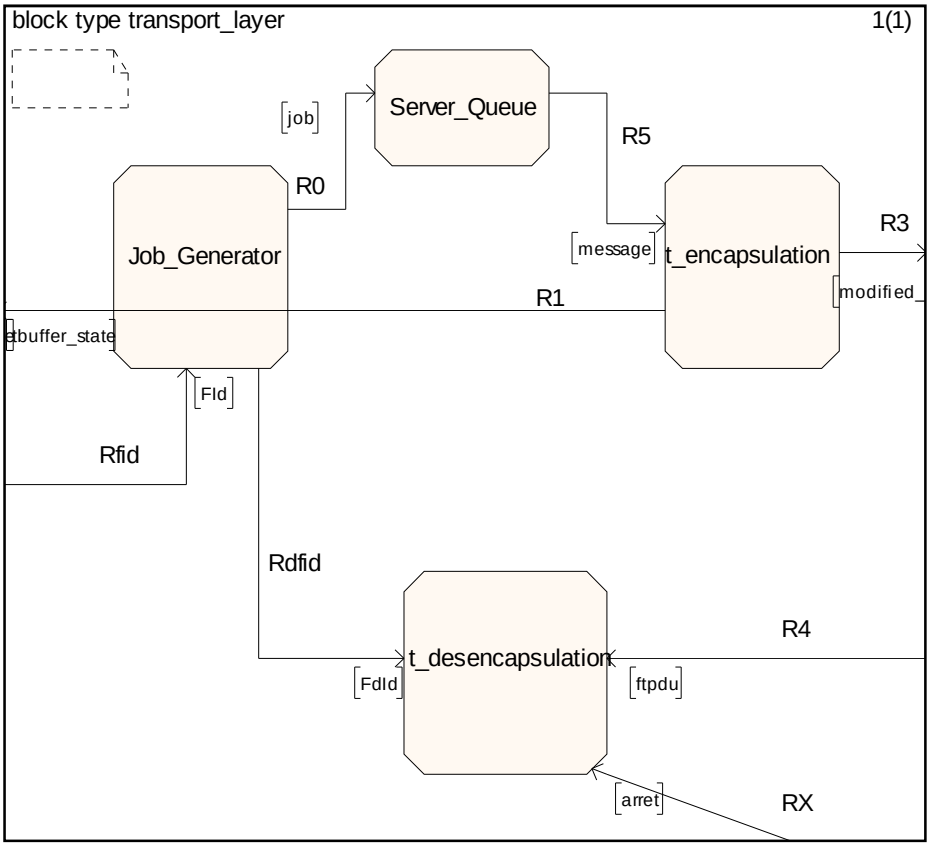


Figure 4.36 : Modèle de système d'évaluation de performance pour la couche transport

Implémentation du Générateur de trafic

Le générateur de trafic initialise un compteur temps, et quand ce compteur expire ; un nouveau job est généré et le transmis au système queue, le compteur se met de nouveau à zéro. La valeur du temps est égale à la T=valeur courante+valeur aléatoire

Dans le but de contrôler et analyser les données traitées par notre nœud, il a été convenu de lire ces données à partir d'un fichier dont le contenu est créé manuellement.

Une autre manière de procéder pour pouvoir générer ce trafic est possible, elle consistera à créer un contenu à partir d'un générateur de trafic aléatoire respectant une loi mathématique donnée, pour ce faire on procédera de la même façon qu'on a fait pour générer une variable aléatoire du temps définie pour délivrer le job au serveur. Il suffira dans ce cas de remplacer la procédure de lecture fichier par une commande de générateur aléatoire.

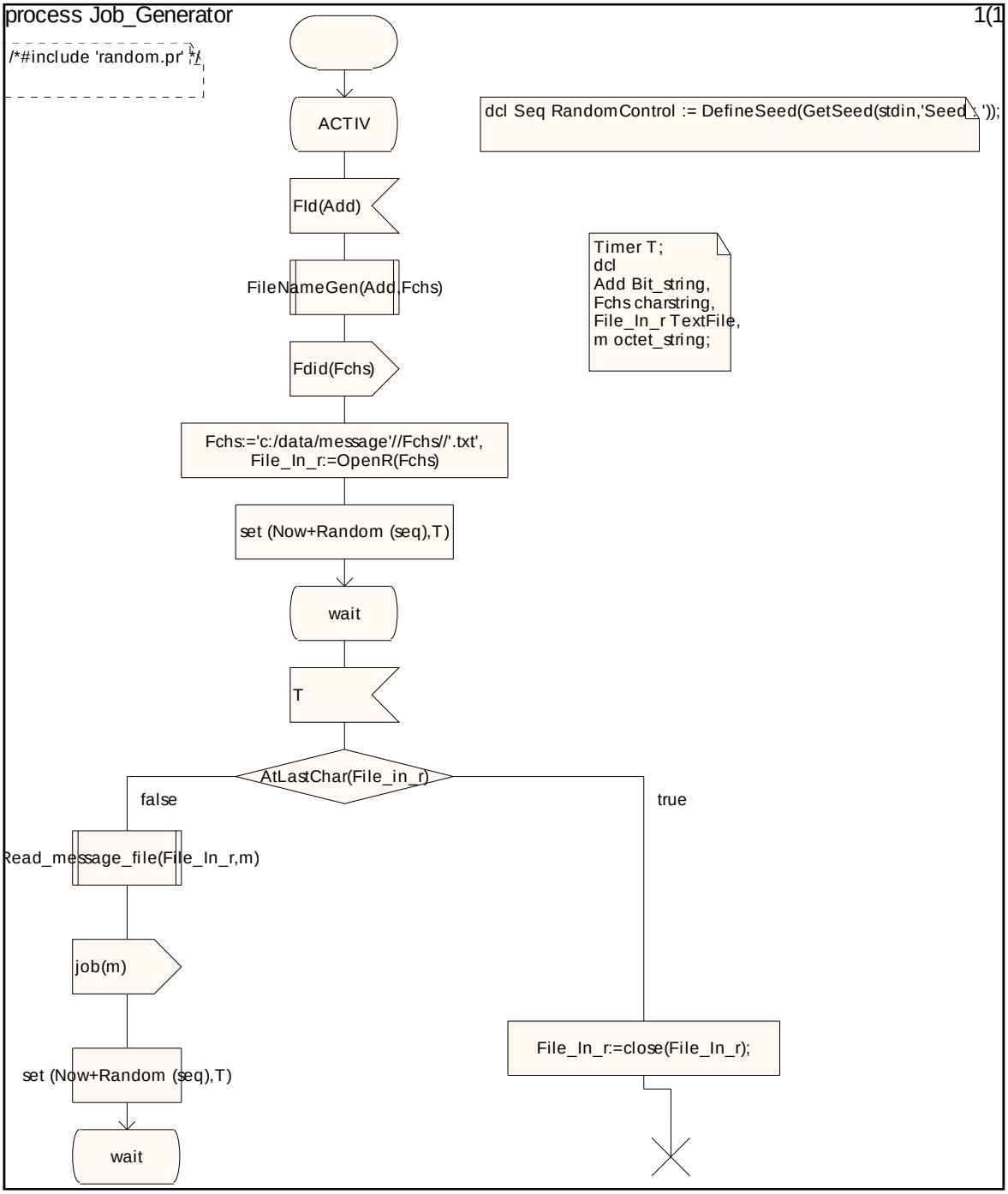


Figure 4.37 : Modélisation du générateur de trafic

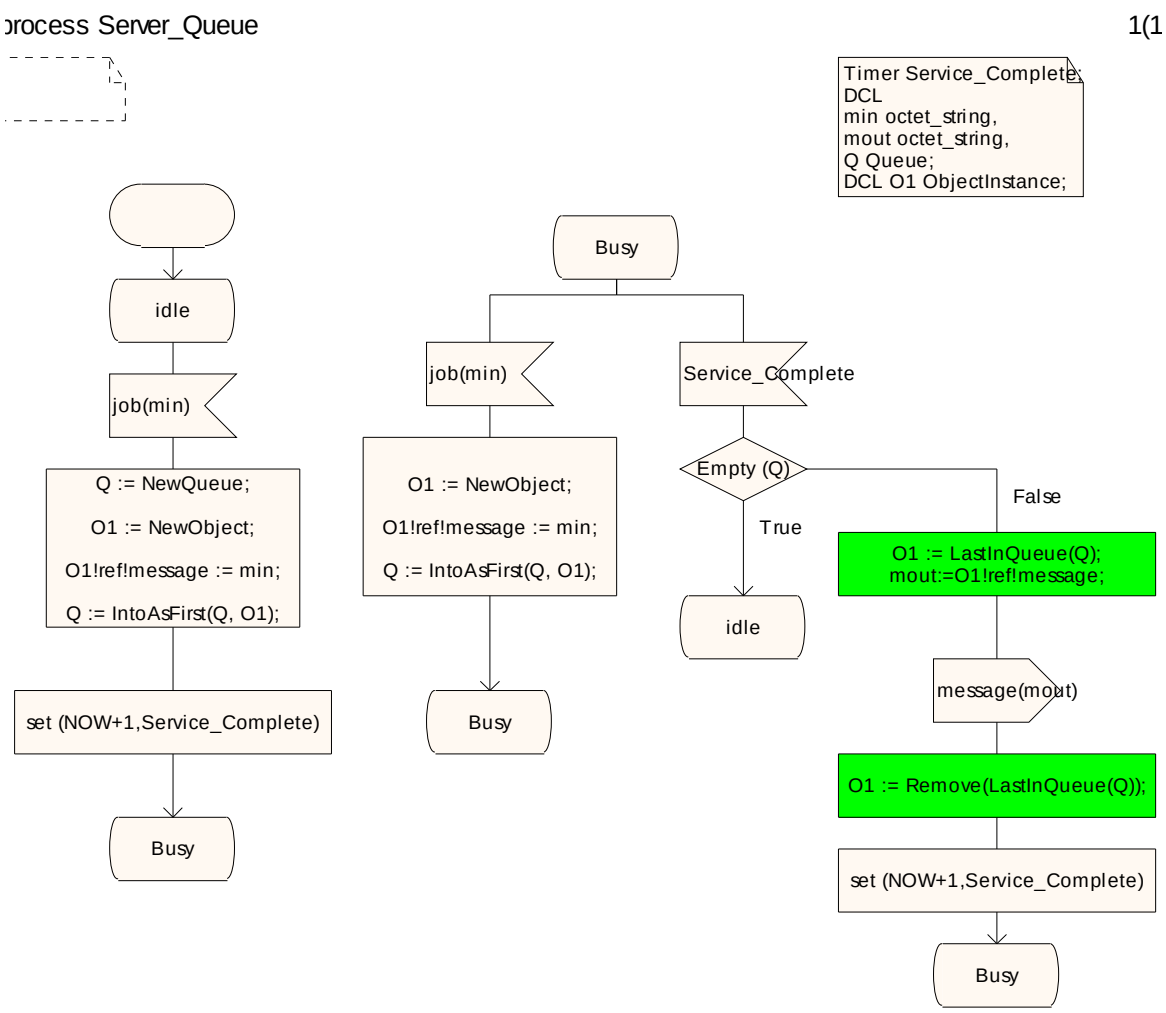


Figure 4. 38 : Process associé à la queue de la couche transport

La figure 4.38 donne le procces décrivant un exemple de queue que nous avons ajouté à la description de notre nœud. Au démarrage du process, une nouvelle queue Q est créée; toute queue doit contenir des objets type (Object instance), dans notre cas ca sera des messages de type chaine d'Octet. La discipline que nous avons retenu lors de notre modélisation; est basée sur le principe de premier arrivé, premier servi.

4.5. Architecture du réseau

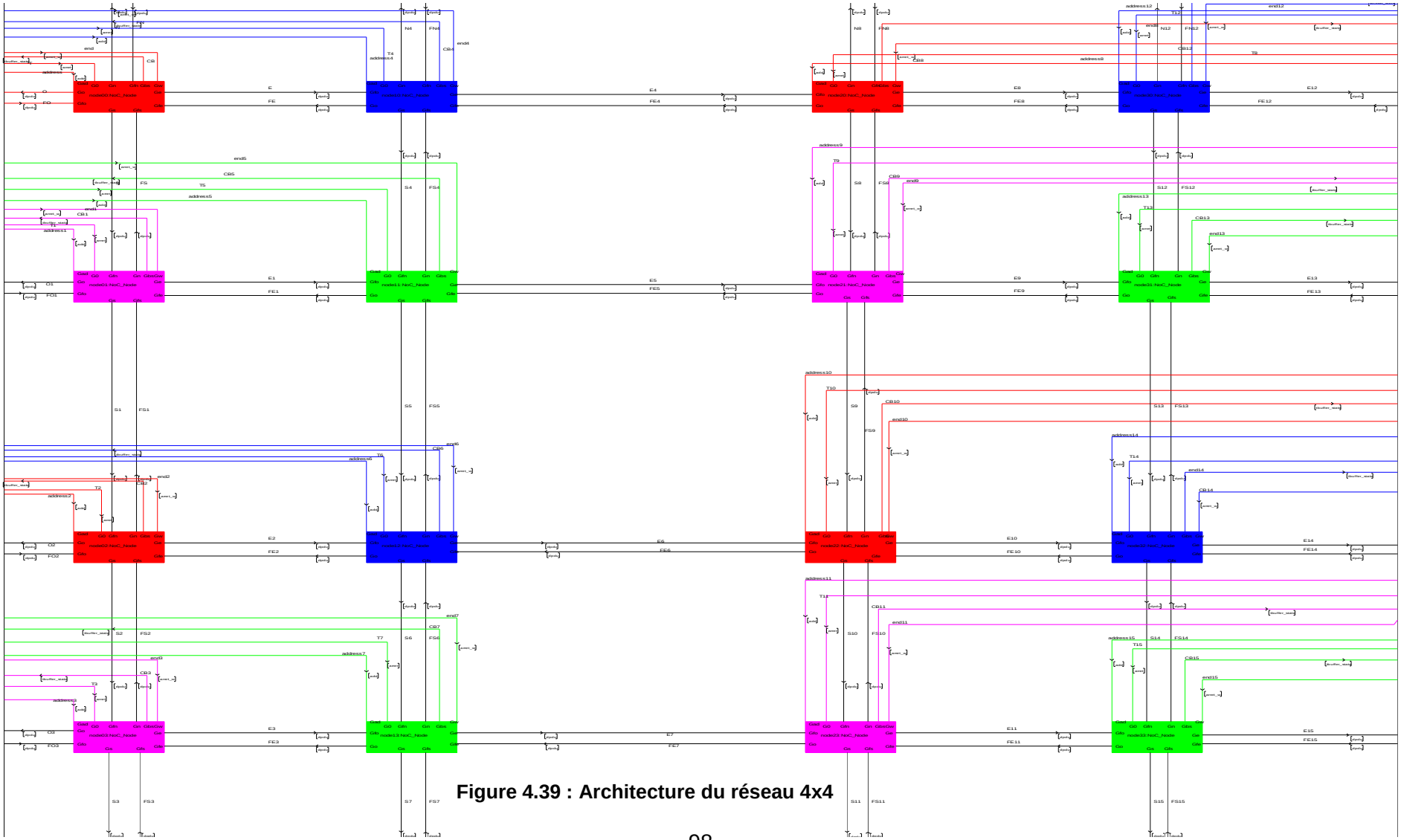


Figure 4.39 : Architecture du réseau 4x4

4.6. Simulation du réseau

Une fois le réseau est généré, il nous faut maintenant passer à l'autre étape qui nous permettra de valider le modèle décrit sous SDL du nœud, elle consiste en la simulation du réseau de Nœuds.

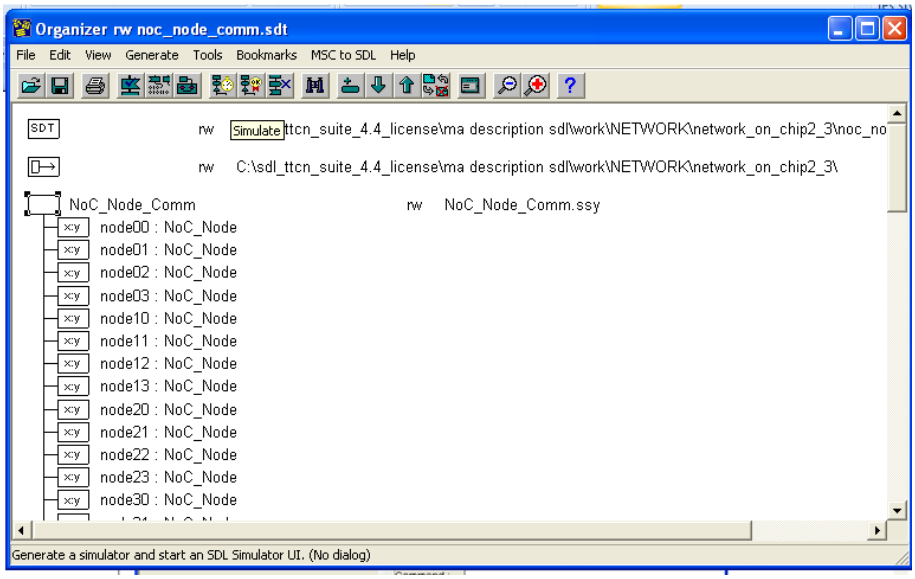


Figure 4.40 : Capture d'écran de l'organisateur de notre système

Pour ce faire il faut alors lancer le simulateur, la figure ci-dessous montre une capture d'écran du simulateur lancé contenant les fonctions de base suivante :

- Exécution de la description globale, pas à pas, jusqu'à une transition, jusqu'à l'expiration d'un compteur, ou encore arrêter l'exécution ;
- Transmission de signaux de l'environnement
- Examiner des process, variables ;
- Traçage de graphe.

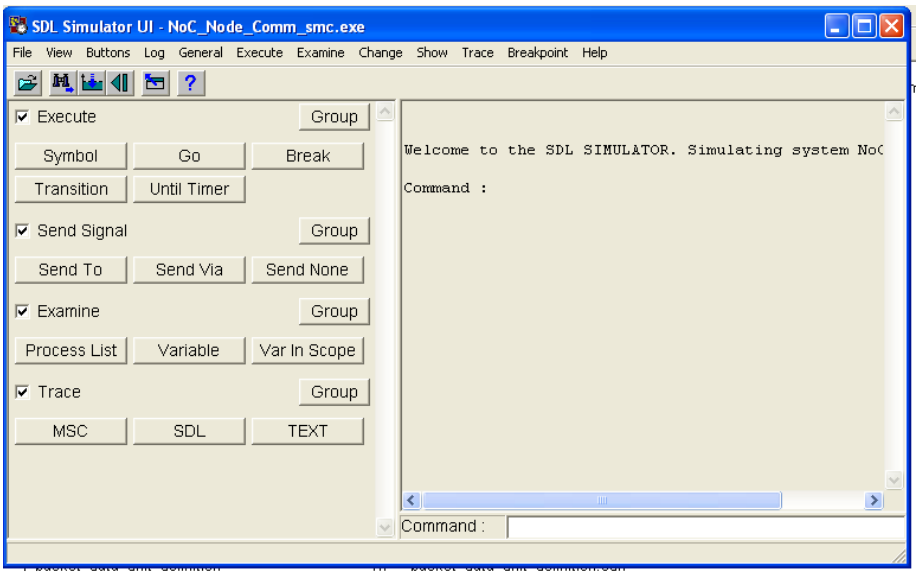


Figure 4.41 : Interface graphique du simulateur SDL

Afin de lancer la simulation à plusieurs reprises avec des données d'initialisation de réseau (adressage) ; on fait appel à la notion de ***command script*** qu'on crée une fois pour toute et auxquels on fait appel chaque fois qu'on veut exécuter le simulateur.

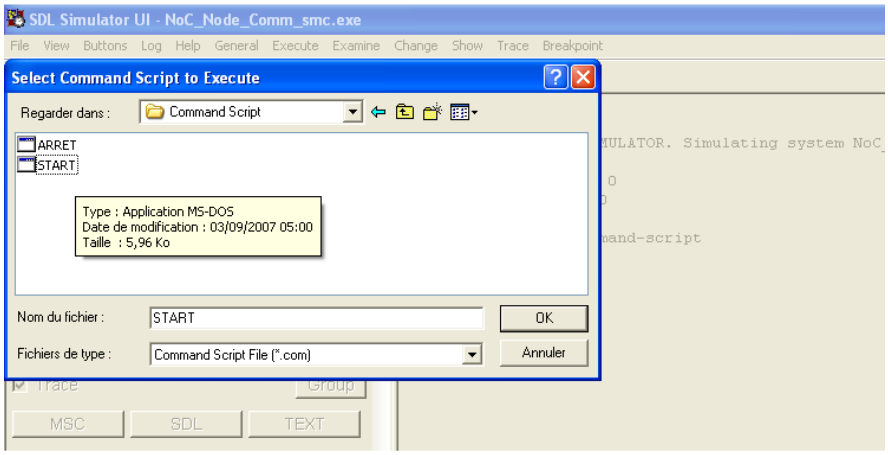


Figure 4.42 : Choix des fichiers de Command Script à exécuter

Une fois le simulateur lancé, une interface pour initialiser les valeurs de compteur temps définie dans les générateur de trafic est affichée afin d'y introduire les valeurs à initialiser.

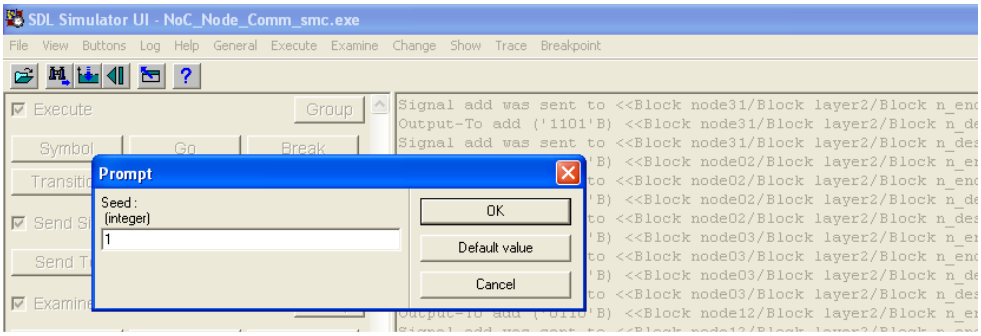


Figure 4.43 : initialisation des valeurs aléatoires (Random Values)

L'exécution de la description est en cours ; un graphe MSC peut être tracé dans le cas de simulation du réseau, ce graphe s'étale sur plus de 300 pages, il nous sera impossible de pouvoir l'imprimer de façon à le visualiser correctement.

L'exécution terminée ; on doit lancer une procédure pour la fermeture des fichiers servants à l'écriture des résultats; un autre command script est alors exécuté

La figure 4.44 montre l'ensemble des16 fichiers lecture dont les noms est généré relativement au nœud associé **[«message » « numeroligne » « numerocolonne ».txt]** sont créés manuellement contenants les messages à transmettre, ils jouent alors le rôle de générateurs de trafic simulants ainsi les flux des applications.

Un autre type de fichier est généré au cours de l'exécution ; il consiste en les fichiers d'écriture de résultat ; on a préféré produire des fichiers de type **Excel** afin de procéder à la

fin de la simulation, à des calculs ou encore de traçage de graphe pour la partie évaluation de performances.

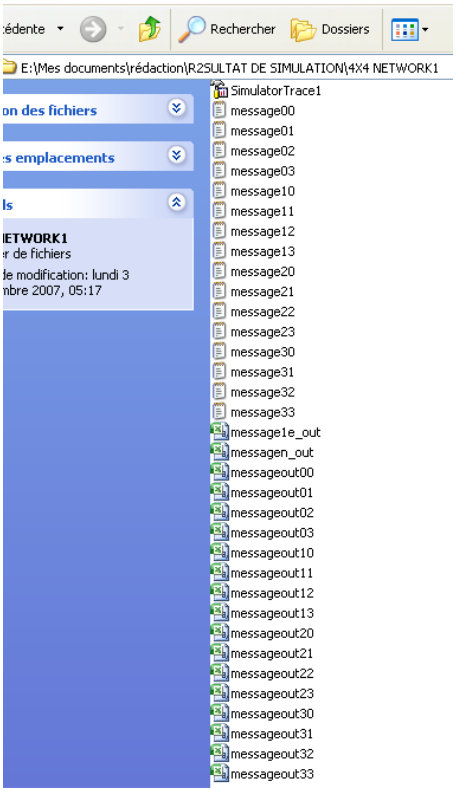


Figure 4.44 : Manipulation des Fichier

Dans une première simulation, nous voulons envoyer un message du premier Nœud (première ligne, première colonne) vers le dernier Nœud de la maille (dernière ligne, dernière colonne), afin de contrôler que la communication dans le réseau est vérifiée de bout en bout.

Le message introduit manuellement dans le fichier associé au premier Nœud (message00.txt) est :

Tableau 4.3 : message source

125 **Information utile utilisée par la communication entre application des nœuds communicants**	**F** **Adresse destination** **F⇔1111 ⇔33** **Le nœud destinataire est le nœud de la 4ème ligne et 4ème colonne**	**00060005000400030002000 1** **Payload**

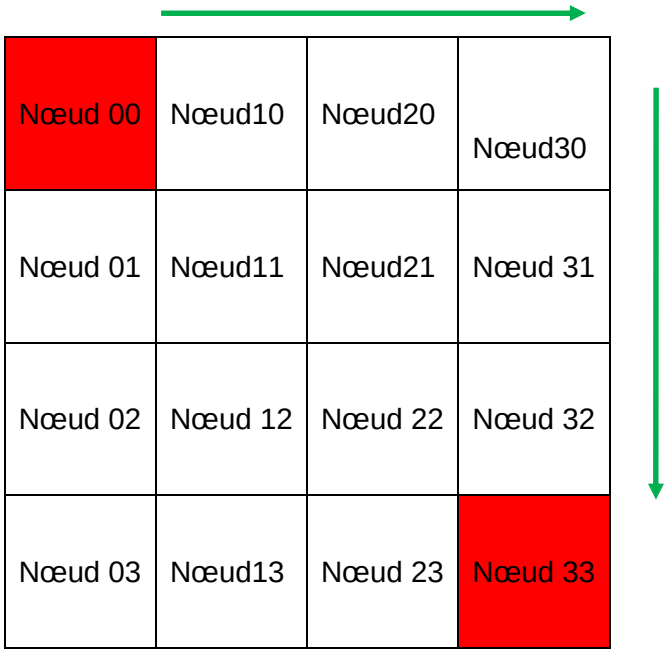


Figure 4.45 : La route des trames

Le tableau 4.4 est extrait du fichier d'écriture généré par le port Est du Nœud20, il donne les six trames associées au message source (Tableau 4.3) ; la deuxième colonne donne les instants ou ces trame sont envoyées.

NB :

Dans la modélisation SDL une transition ou un process s'exécute indépendamment du temps, autrement dit le temps d'exécution dans SDL est supposé égal à zéro. Pour être capable de modéliser le temps de traitement et ajouter des contraintes de temps ; une notion de Timer 'réveil' peut être sous SDL associée à un compteur temps avec une durée estimée.

Donc les résultats temps générés sont modélisés en estimant par exemple le temps de génération du trafic et le temps de traitement.

Tableau 4.4 : Data Link PDUs

Trame (dpdu)	Temps de livraison au nœud voisin
F821256000064EBD	4.1251
F8212561000549EE	5.1251
F82125620004009F	6.1251
F821256300034748	7.1251
F82125640002D2F9	8.1251
F82125650001D5AA	9.1251

Tableau 4.5 : Structure du premier DPDU associé au message source

F Adresse destination F⇔1111 ⇔33	8 Adresse du nœud transitaire 8⇔1000 ⇔20	2 Nombre de saut	125 Information utile utilisée par la communication entre application des nœuds communicants	6 Nombre des dpdu associés au message source	0 L'ordre du dpdu généré	0006 La charge du dpdu	4EBD Calcul CRC

De la même façon, ces 6 dpdu transitent via le réseau jusqu'à ce qu'ils arrivent au Nœud destinataire du massage « Nœud 33 » dont l'adresse est 1111=F.

En suivant la simulation de cette communication; les dpdu passent comme prévu par l'algorithme de routage par les nœuds montrés dans la figure 4.45, dans chaque saut deux parties des trames changent de valeurs :

- l'adresse du Nœud intermédiaire,
- le nombre de saut.

A la fin de cette simulation on peut aussi récupérer le résultat de cette communication, en vérifiant le contenu du fichier des résultats associé au nœud destinataire.

125000060005000400030002000 1 Message source transmis	13.1251 L'instant où le message est reconstruit à partir des différents segments reçus

La deuxième partie de la simulation consiste en la création de plusieurs messages, de les inclure dans les fichiers de telle sorte à créer un trafic dans le réseau dans différents chemins.

Tableau 4.6 : les messages à communiquer entre les Nœuds du réseau construit

Le contenu du fichier message00	Le contenu du fichier message21	Le contenu du fichier message32
• 125F00060005000400 0300020001 • ABCA1111222333344 44455556666 • 123C1000200030004 00050006000	• 1230006000500040 00300020001	• CFA114525376547845534 76524654345874

Le tableau 4.6 montre les messages transmis par les Nœuds en vue de les communiquer à des Nœuds dont les adresses sont concaténées aux messages à transmettre :

Le nœud 00 : transmet des données aux nœuds : « 33 », « 22 », « 30 ».
Le nœud 21 : transmet une donnée au nœud: « 00 ».
Le nœud 32 : transmet une donnée au nœud: « 01 ».

A partir de ceci, on déduit que 5 Nœuds sont destinataires de message. La figure 4.46 montre les fichiers associés au à ces nœuds (entouré d'un cercle) ; contrairement aux 11

autres fichiers résultats, ceux la sont de taille differente de zéro, ce qui nous informe qu'un contenu a été introduit qui ne pourra être que les messages envoyés.

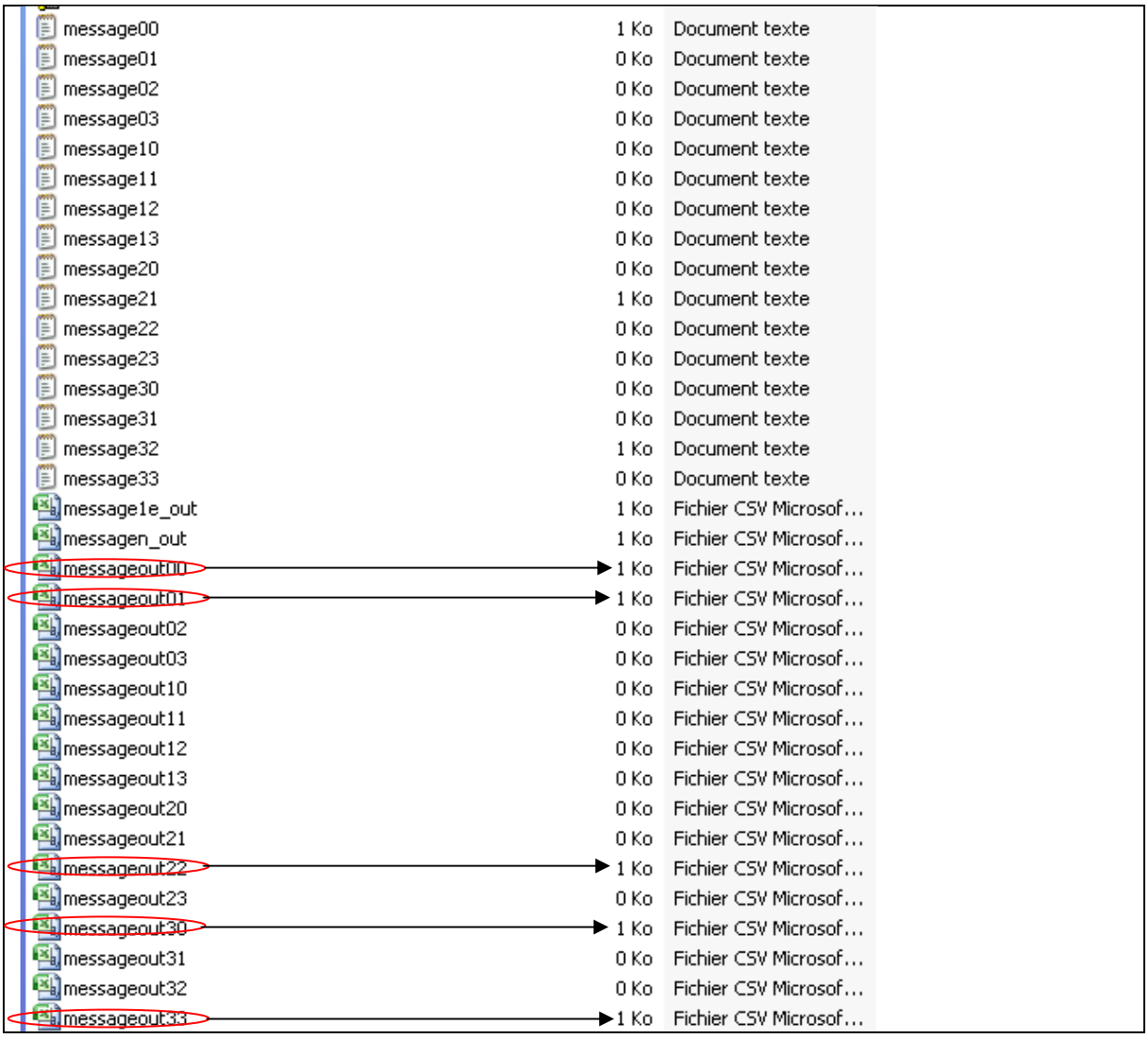


Figure 4.46 : Les fichiers résultats créés

Tableau 4.7 : Le contenu des fichiers résultats

Nœud N°	Message reçu	Instant de reconstruction
00	123000600050004000300020001O	12.1251
01	CFA014525376547845534765246543458740	16.1251
22	ABC011112222333344445555666 6	20.1251
30	123010002000300040005000 6000	24.1251
30	125000060005000400030002 0001	18.1251

4. Conclusion

Nous avons tenté dans ce chapitre de présenter le travail expérimental que nous avons fait, ca consiste d'élaborer en premier lieu, un cahier des charges décrivant les spécifications du Nœud NoC que nous avons arrêté après l'analyse des mécanismes réseau présentés dans le chapitre 2.

Nous avons ensuite présenté l'architecture de ce Nœud modélisé sous SDL ; le fonctionnement de ce Nœud est bien vérifié en exécutant les simulations associées.

Une fois le modèle nœud est vérifié ; il est nécessaire de pouvoir valider son fonctionnement, et ce en réalisant des communications dans un réseau dont les éléments de base respectent les spécifications du modèle retenu ; pour ce faire le langage objet est alors exploité.

La partie ajoutée à la description système Nœud, relative à l'évaluation des performances permettra d'évaluer notre protocole modélisé, de la raffiner et l'optimiser, de comparer l'implémentation des différents mécanismes réseau envisageables ; pour enfin arriver à une modélisation conforme synthétisable.

Additivement à la vérification du fonctionnement de l'architecture ainsi modélisée, nous avons procéder à la définition d'un système d'évaluation des performances correspondant à cette architecture.

Le système Nœud que nous avons décrit sous SDL peut être exploité après sa validation, afin de réaliser un nœud NoC Matériel en utilisant VHDL en faisant appel à des passerelles permettant le passage du code SDL vers le langage VHDL, donc on pourra parler de IP core matériel que nous implémentera dans un FPGA ou un ASIC.

Du fait que nous avons pu générer le code SDL associé à une architecture de communication pour les SOCs basée sur l'approche NoC; on pourra de même exploiter ce code afin d'implémenter cette architecture NoC sur une puce (ASIC, FPGA) capable d'assurer la communication des éléments de n'importe quels systèmes implémentés sur cette même puce.

Conclusion Générale

L'évolution technologique dans le domaine de la conception et la réalisation des circuits logiques a permis un taux d'intégration de plus en plus élevé au point ou on parle de plus d'intégration d'un système sur puce en place et lieu de fonction. Ceci a nécessité et nécessite encore des efforts pour combler les manques aussi bien sous l'angle des outils que des paradigmes. La nécessité d'introduire ces derniers est due à l'obsolescence ou à l'inadéquation des approches déjà connues et utilisées.

C'est ainsi que le paradigme des réseaux sur Puce (NoC) est apparu et est en cours de développement, un paradigme qui permettra de mieux prendre en compte la communication des systèmes sur chip qui jusqu'à maintenant présente des limites dans la conception de ces systèmes en terme de scalabilité et de performance.

Il nous a été demandé à travers ce présent mémoire de développer un système d'interconnexion sur puce basé sur l'approche NOC. Pour ce faire nous avons procédé d'abord à établir une rétrospective et l'état de l'art des NoC, afin de retenir un modèle de référence basé sur ce qui a été fait jusqu'à maintenant dans ce domaine. Le Modèle Nostrum alors présentait pour nous, une très bonne plateforme pour la conception des NoC, vue son architecture d'une part utilisant l'approche en couche, et de la disponibilité relative de la documentation y afférente.

Au cours de ce projet nous avons procédé à l'élaboration d`un modèle de réseau adapté à un nœud NoC, ce modèle consiste en un allégement du modèle de référence OSI en un modèle en couche mieux adapté aux particularités d`un réseau sur puce.

Cette simplification du modèle consiste en premier lieu à rassembler les trois couches supérieures du modèle OSI (application, session et présentation) en une seule couche application indépendante du réseau, réservée à accueillir les IP cores réutilisables. En deuxième lieu la couche transport assurera le service fin a fin et offre a l`application une interface d`accès au réseau, ceci donc garantira une abstraction aux concepteurs d'application. En fin les couches réseau et liaison de données vont assurer le transport des paquets entre les ressources suivant des mécanismes de routage et de gestion de liaison.

Le modèle ainsi décrit a été traduit à l'aide du langage SDL. L'environnement TauSDL a permis de simuler son comportement, de le valider et d'obtenir également un exécutable grâce auquel nous avons pu tester notre Nœud NoC.

Additivement à l'exécutable, SDL nous permet de construire des Librairies, on parle alors dans ce cas de programmation dite Objet, un outil que nous considérons très important

du fait qu'il nous a permis de réutiliser nos descriptions pour pouvoir générer le réseau associé à notre modèle, de simuler le système ainsi décrit pour en fin valider son fonctionnement.

Notre travail consistait à proposer une architecture de communication sur puce, et de la valider. On a doté notre modélisation d'une extension destinée à offrir un système d'évaluation des performances d'une telle architecture et de faciliter la modification des paramètres définissant un Nœud (algorithme de routage, gestion des buffers et contrôle de l'intégrité des données de bout en bout).

La méthodologie de modélisation suivie pour arriver au Nœud NoC, à savoir, la modélisation du protocole avec un langage de description formel, et enfin l'utilisation du code cible pour construire des librairies et valider le système pour générer en fin un code C.
Un tel code pourrait être exploitable dans la chaine de développement des NoCs de différentes manières à savoir:

1. L'exécuter sur un environnement matériel-processeur- comme application logiciel,
2. Utiliser des passerelles pour arriver à une spécification système synthétisable par un langage de description matérielle tel que VHDL, Verilog… ainsi on pourra concevoir des systèmes, de les optimiser avec une telle procédure rapide et simple, qui peut être appliquée à tout système de communication. on parle dans ce cas des outils de prototypage rapide

Ce mémoire nous a permis de nous initier au nouveau paradigme de conception des futures systèmes électroniques implémentés sur une même surface de silicium, ces systèmes sur puce de plus en plus complexes exigent une méthodologie de conception flexible permettant d'assurer un meilleur temps de mise sur le marché consistant à l'élaboration du modèle, le simuler et le raffiner en introduisant les différentes contraintes du système jusqu'à atteindre les objectifs voulus.

Additivement à l'importance de cette méthodologie dans le cycle de vie des systèmes, un facteur très important à prendre en compte à savoir la communication à l'intérieur de ces systèmes qui était l'objet de notre mémoire. Ceci dit ce travail nous permettra de voir des axes de recherches prometteurs avec des potentialités immenses et de larges perspectives.

Glossaire

ADK : AMBA development kit

ASIC : Application Specific Integrated Circuit, (circuits spécifiques). Circuit intégré développé sur la base de spécifications d'utilisateur.

ASM : Abstract State Machines, des Notions abstraites d'états et d'opérations où les transitions vues comme des mises à jour gardées.

Broadcast : l'émission multiple, c'est-à-dire qu'un cœur émet des informations à destination de plusieurs récepteurs. Le « broadcast » est très avantageux pour certaines applications, comme par exemple en traitement d'image.

Buffer : Un espace mémoire FIFO (« First In First Out ») utilisés pour stocker des paquets en transit.

CEFSM : automates à état fini étendus communicants via les files

CMOS : Complementary metal–oxide–semiconductor, une classe importante des circuits integrés. La technologie CMOS est utilisée dans les chips.

CPU : Central Process Unit, Unité centrale de calcul

Débit : C'est la quantité maximale d'information transitant dans une interconnexion pendant une durée déterminée.

DRAM : Dynamique RAM

DSP : Digital Signal Processor

FPGA : « Field Programmable Gate Array » Puce de semi-conducteur programmable ou circuit intégré. Le FPGA est programmé par l'utilisateur après la production et représente une alternative flexible aux ASIC) de base.

GALS : Globalement Asynchrone Localement Synchrone.

GT : Guaranteed throughput, bande passante garantie

Head-of-line : Ce problème se rencontre dans les architectures à buffers en entrée. Il traduit un blocage intervenant lorsque le premier paquet d'une file d'attente est bloqué par un conflit de ressource, les paquets suivants sont alors bloqués même s'ils pourraient être routés vers des sorties disponibles.

IP : Intellectual Property est un bloc fonctionnel également appelé coeur. Chaque IP dédié ou reconfigurable est composé de processeurs, de mémoires et/ou d'interfaces.

ITU : International Telecommunication Union**,** Union internationale des télécommunications,

Latence : C'est le temps écoulé entre le moment où le message transmis est initialisé jusqu'au moment où le message est acquitté.

Mot : Unité d'information utile appartenant au message à transmettre.

NOC : Network On Chip, réseau sur puce.

MPEG 4 : (ISO/CEI 14496), introduit en 1998, est une norme de codage d'objets audiovisuels spécifiée par le Moving Picture Experts Group.

NI : Network Interface

NIU : Network Interface Unit

Nœud : Un nœud peut contenir un ou plusieurs éléments processeur, mémoire et entrée/sortie connectés à un routeur.

NS-2 : Network Simulator ou plus communément NS est un logiciel libre de simulation par événements discrets très largement utilisé dans la recherche académique et dans l'industrie. Il est considéré par beaucoup de spécialistes des télécommunications comme le meilleur logiciel de simulation par événements discrets, en raison de son modèle libre, permettant l'ajout très rapide de modèles correspondant à des technologies émergentes. Il est basé sur l'utilisation de langages de scripts pour la commande des simulations (tcl/tk) alors que le cœur des simulations est implémenté avec le langage C++.

OSI : Modèle OSI (Open Systems Interconnection) a été crée par l'ISO Organisation Internationale de Normalisation norme 7498, dans le but d'offrir une base commune à la description de tout réseau informatique. Dans ce modèle, l'ensemble des protocoles d'un réseau est décomposé en 7 parties appelées couches OSI numérotées de 1 à 7.

PCB : Printed Circuit Board, synonyme de Circuit imprimé

PDA : Personal Digital Assistant, Un assistant personnel ou ordinateur de poche est un appareil numérique portable

PDU : Protocol Data Units, des unités de données utilisées pour implémenter le protocole

PLL : Phase-Locked Loop, Une Boucle à verrouillage de phase est un montage électronique permettant d'asservir la phase instantanée de sortie sur la phase instantanée d'entrée, mais elle permet aussi d'asservir une fréquence de sortie sur un multiple de la fréquence d'entrée.

QoS : Quality of Service, Qualité de Service. La qualité de service est définie comme l'assurance de performances bornées sur un lien donné. Son but est d'offrir aux utilisateurs un système aux comportements prévisibles.

RAM : Random Access Memory, mémoire à accès aléatoire

RFC : Request For Comments, ou propositions de spécifications sont un ensemble de documents
qui font référence auprès de la Communauté Internet et qui décrivent, spécifient, aident à l'implémentation, standardisent et débattent de la majorité des normes, standards, technologies et protocoles liés à Internet et aux réseaux en général. Ils se présentent sous forme de fichiers textes dont le nom est "rfcxxxx.txt" dont xxxx est un nombre incrémenté pour chaque nouveau RFC. Il en existe actuellement plus de 2000.

RNI : Ressource Network Interface, une interface implémentée entre la ressource et le réseau de communication.

ROM : Read Only Memory, désigne une mémoire morte, c'est-à-dire accessible uniquement en lecture

RTL : Register Transfer Level est une méthode de description des architectures microélectroniques. Dans la conception RTL, le comportement d'un circuit est défini en termes d'envois de signaux ou de transferts de données entre registres, et les opérations logiques effectuées sur ces signaux.

SAP : Service Access Point, Dans l'architecture réseau OSI, point d'appel au sein d'une couche de l'architecture aux services de couches inférieures.

Scalabilité : La « scalabilité » d'un médium est sa capacité à évoluer. Si l'ajout de coeurs fait évoluer les performances et le coût de l'interconnexion de façon proportionnel, le médium est alors « scalable ».

SDL : Specification and Description Language, Langage de description et de spécification utilisé dans les protocoles de télécommunications

SOC : System On Chip (système sur puce).

TAUSDL : environnement de développement

TDF : Les Techniques de Description Formelle (TDF) pour la spécification de systèmes communicants sont utilisées pour valider et tester des implémentations de ces systèmes

Topologie : Il définit la façon dont sont agencés les noeuds les uns par rapports aux autres.

Trafic : C'est la quantité de mot entrant dans le réseau par cycle d'horloge et par coeur émetteur.

VHDL : Very High Speed Integrated Circuit Hardware Description Language, Langage de description du matériel pour circuits intégrés de très haute vitesse en anglais) est un langage de description du matériel destiné à décrire le comportement et/ou l'architecture d'un système électronique numérique

Wormhole : Avec un acheminement en « wormhole », les paquets sont découpés en mots de taille fixe, l'en-tête est routé puis les mots de données suivent le même chemin indépendamment les uns des autres.

Références

Bibliographie

[1]. Fréderic Hunsinger, « Méthode de validation globale pour les sytèmes monopuces», Laboratoire TIMA, Institut National Polytechnique de Grenoble, mars 2006.

[2]. Arnaud Grasset, « synthèse des interfaces de communication des systèmes monopuces: de la spécification à la génération automatique», Institut national de Grenoble, janvier 2005.

[4]. Ahmed Amine Jerraya, «de l'ASIC au SoC, puis au réseau de composants sur puce», Veille Technologique, le Magazine d'ASPROM, Déc2001/Jan 2002.

[5]. Séverine Riso, «Evaluation des paramètres architecturaux des réseaux sur puce», Université Montpellier II / LIRMM, Novembre 2005.

[6]. R. Leiserson, «Creating a world of Smart Re-configurable Devices », France, septembre 2002.

[8]. Axel Jantsch, Hannu Tenhunen, « Network on Chip », Kluwer Acadimic Publishers, New York, Boston, Dordrecht, London, Moscow, 2003.

[9]. W.J. Dally, B. Towels « Route Packets, Not Wires : On-Chip Interconnection Networks », Design Automation Conference (DAC), Juin 2001, Las Vegas, USA, pp 684-689.

[10]. Luca Benini, Giovanni De Micheli, «Networks on Chips: A New SoC Paradigm», IEEE Computer Society, 2002.

[12]. Antoine Scherrer, Antoine Fraboulet, «Etude de la couche transport des réseaux sur puces», Laboratoire CITI, INSA Lyon, 2003

[14]. Yi-Ran Sun, «Simulation and Performance Evaluation for Networks on Chip», KTH, Suede

[15]. Mikael Millberg, « The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone» Draft v0.1.48 - November 2002 ISRN KTH/IMIT/LECS/R-02/01--SE TRITA-IMIT-LECSR02:01, Institute of Microelectronics and Information Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, November 2002.

[16]. M. Millberg, et. al, "The Nostrum backbone-a communication protocol stack for Network on Chip", Proc. 17th International Conference on VLSI Design, 2004, p. 693-696.

[17]. Ludovic Casset , « Développement formel d'un vérifieur embarqué de byte code Java Card à l'aide de la méthode B », Universite De La Mediterranee, Octobre 2002

[18]. Mokhoo MBOBI, « Modélisation Hétérogène Non Hiérarchique », Université Paris Xi Orsay, décembre 2004

[19]. Damien Hubaux, Lotfi Guedria, Luc Vandendorpe, Michel Verleysen et Jean- Didier Legat, « Nouvelles méthodes de conception de systèmes électroniques intégrés », Hamamet, Tunisie, 10- 13 avril 2002

[20]. http://www.iro.umontreal.ca/~aboulham/pdfs_sources/Bourhfirthese.pdf

[21]. Z. Mammeri « Introduction au langage de description et de spécification SDL », Université de Paul Sabatier- Toulouse, 2001.

[22]. Z. Mammeri, « SDL, modèleisation de protocole et système réactifs », Paris, Hermes Sciences Publication, 2000

[23]. Recommandation Z.100, « Langage De Description Et De Spécification », Recommandations UIT-T Z.100, 03/93

[24]. G. Sassatelli, "A mesh based Network on Chip characterization : A GALS approch", Proc. 20th International Conference on design of Circuits and integrated Systems, 2005.

Webgraphie

[3]. www.st.com

[7]. http://public.itrs.net, «International Technology Roadmap for Semiconductors»

[11]. http://www.arteris.net/noc_whitepaper.pdf, « A comparison of Network-on-Chip and Busses », 2005.

[13]. http://4more.av.it.pt, Y.Durand, «Description of final SoC architecture and subsystems interfaces specification », Oct2004.