THÈSE DE DOCTORAT DE DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ INFORMATIQUE

Présentée par Adrijean ANDRIAHANTENAINA
Pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ PARIS VI

IMPLÉMENTATION MATÉRIELLE D'UN MICRO-RÉSEAU SPIN À 32 PORTS

Soutenue le 31 janvier 2006 devant le jury composé de

Alain GREINER	Directeur de thèse
Ian O'CONNOR	Rapporteur
Frédéric PÉTROT	Rapporteur
Jean-Pierre SCHOELLKOPF	Examinateur
François CHAROT	Examinateur
Nathalie DRACH-TEMAM	Examinatrice

A ma maman chérie, reçois ce fruit de tes efforts et de tes sacrifices.

A ma merveilleuse femme, merci pour ta patience, ton courage et ton soutien quotidien. Tu es la meilleure partie de chacune de mes journées.

Résumé

Les interconnexions à bus central représentent le principal goulot d'étranglement au niveau architectural dans la gestion des communications entre les processeurs, les mémoires et les périphériques. Le micro-réseau *SPIN* est une réponse à ce défi car sa bande passante croît linéairement avec le nombre d'abonnés. Il est constitué de routeurs *RSPIN* et permet le routage de trafic unidirectionnel ou de type *requête/réponse*. A vide, le temps de traversée moyen d'un routeur *RSPIN* est de 2.5 cycles et sa surface est de 0.29 mm^2.
La conception d'une *puce d'évaluation*, composée d'un micro-réseau *SPIN* à 32 ports et de toute la logique nécessaire à sa caractérisation, en collaboration avec *STMicroelectronics* a permis de déterminer les performances d'un micro-réseau *SPIN* à 32 ports. Il est composé de 1411136 transistors, tient sur une surface de 4.64 mm^2 pour un procédé de fabrication 0.13 µm, et possède une bande passante cumulée pratique d'environ 109 Gbit/s.

Mots clés

Micro-réseau sur puce (NoC) – Système sur puce (SoC) – Bande passante – Latence – Seuil de saturation – commutation de paquet

Abstract

The central bus interconnections represent the main bottleneck at the architectural level in the communication management between the processors, the memories and the peripherals. The *SPIN* micro-network is an answer to this challenge because its bandwidth grows linearly with the component number. It is made up of *RSPIN* routers and allows the routing of either unidirectionnal or *request/response* type traffic. The average latency of an empty *RSPIN* router is 2.5 clock cycles and its area is about 0.29 mm².
The design of an evaluation chip, made up of a 32-ports *SPIN* micro-network with all the necessary logic for its characterization, in collaboration with *STMicroelectronics* allows us to determine the performances of a 32-ports *SPIN* micro-network. It is made up of 1411136 transistors, fits on 4.64 mm² for a 0.13 µm process, and has a usefull overall bandwidth of about 109 Gbit/s.

Keywords

Network On Chip (NoC) – System On Chip (SoC) – Bandwidth – Latency – Saturation threshold – packet switching

Remerciements

Je tiens d'abord à exprimer ma profonde reconnaissance à mon directeur de thèse, le Professeur Alain Greiner, pour toute la confiance qu'il m'a témoignée et toute sa disponibilité à diriger cette thèse. Je le remercie également pour les longues discussions techniques, pour les précieux conseils qu'il m'a donnés, et pour tout ce qu'il m'a appris pendant cette thèse et lors de la rédaction de ce manuscrit. Je le remercie enfin de m'avoir permis d'intégrer son équipe : le département *ASIM* du *LIP6*.

Je remercie Monsieur Ian O'connor, maître de conférences au *LEOM* (*Laboratoire d'Electronique Optoélectronique et Microsystème*) à l'école centrale de Lyon, et Monsieur Frédéric Pétrot, professeur au laboratoire *TIMA* (*Techniques of Informatics and Microelectronics for computer Architecture*) de Grenoble, qui ont accepté d'être les rapporteurs de ma thèse de doctorat. Je remercie également Monsieur François Charot, chargé de recherche *INRIA* à l'*IRISA* de Rennes, Monsieur Jean-Pierre Schoellkopf, directeur de l'équipe *Advanced Digital Design* de *STMicroelectronics* de Crolles et Madame Nathalie Drach-Temam, professeur au département *ASIM* du *LIP6*, qui ont bien voulu en être les examinateurs.

J'associe à ces remerciements Messieurs Bertrand Borot et Jean-Pierre Schoellkopf de *STMicroelectronics* de Crolles, ainsi que Monsieur Karim Dioury d'*Avertec* qui n'ont pas hésité à engager la compétence de leurs équipes respectives pour permettre la fabrication de cette *puce d'évaluation*.

Je remercie, bien évidement, Pierre Guerrier, car l'architecture *SPIN* que nous avons implémentée est issue de ses recherches. Un grand merci aux concepteurs d'outils de l'équipe *ASIM* du *LIP6* (Frédéric Pétrot, Franck Wajsbürt, Jean-Paul Chaput,…) qui n'ont pas hésité à modifier leurs outils afin de s'adapter aux spécificités du projet *SPIN*. Je remercie également le reste de l'équipe *ASIM* du *LIP6* avec qui j'ai partagé tant d'années et avec qui ce fut un plaisir de travailler.

Enfin, tous mes remerciements à Danièle Dromard et Patricia Rasoanaivo pour l'aide essentielle qu'elles m'ont apportée dans la correction de ce manuscrit.

Table des matières

Table des figures

Chapitre 1 *Introduction*

Les procédés de fabrication des circuits intégrés sur silicium ne cessent de progresser. La densité d'intégration augmente donc au fur et à mesure que la technologie progresse, et avec elle, le nombre de transistors que l'on peut mettre sur une seule puce, se compte désormais en dizaines de millions.

Avec cette capacité d'intégration, les composants qui étaient auparavant des périphériques au niveau carte peuvent être désormais intégrés sur la même puce que le ou les processeurs qui les utilisent. Les circuits intégrés deviennent donc des systèmes multi-processeurs intégrés sur puce (ou *MP-SoC*). Par souci de rapidité de mise sur le marché (*time to market*), les composants sont, en général, des composants virtuels re-utilisables, qui sont conçus une fois pour toute et réutilisés ensuite dans plusieurs systèmes.

Lorsque l'on intègre plusieurs processeurs sur la même puce, le bus partagé devient très vite un goulot d'étranglement de l'architecture car ses performances ne peuvent tout simplement pas satisfaire celles de tous les composants présents sur la puce.
De nombreuses recherches dans le domaine des communications sur puce se sont développées ces dernières années. La tendance s'oriente vers le remplacement des bus partagés par des réseaux sur puce (*Network On Chip*). Le micro-réseau *SPIN* (*Scalable Programmable Integrated Network*), développé au *LIP6*, et publié en 2000 a été un précurseur dans ce domaine.

La thèse de Pierre Guerrier [1] détaille les grandes lignes et les choix architecturaux du micro-réseau *SPIN*. Par contre dans cette thèse, nous présentons une réalisation matérielle permettant une évaluation précise des performances, nous proposons une stratégie détaillée pour le test de fabrication, ainsi que différentes évolutions architecturales.

Pour déterminer les performances du micro-réseau *SPIN*, nous avons conçu une *puce d'évaluation* en collaboration avec *STMicroelectronics*. Il est composé d'un micro-réseau *SPIN* à 32 ports et de toute la logique nécessaire à sa caractérisation.

Ce manuscrit est organisé de la façon suivante :

- dans le deuxième chapitre, nous exposons la problématique de nos recherches et les différents objectifs que nous nous sommes fixés ;
- le troisième chapitre est un tour d'horizon des réseaux sur puce (*NoC*) actuellement disponibles ;
- le quatrième chapitre analyse les caractéristiques du micro-réseau *SPIN*. Nous décrivons, en particulier, la topologie, le type de commutation, le format des paquets, la stratégie de routage et le contrôle de flux ;
- le cinquième chapitre décrit la micro-architecture du routeur *RSPIN* (*router for SPIN*), en insistant sur les évolutions architecturales que nous avons introduites ;
- le sixième chapitre détaille les mécanismes de test implémentés dans le micro-réseau *SPIN* ;

- dans le septième chapitre, nous présentons en détail la méthode de réalisation de la macro-cellule *VLSI* constituant le routeur *RSPIN*. Nous y décrivons tout le processus de placement/routage utilisant la bibliothèque de cellules symbolique de la chaîne *ALLIANCE* ;
- le huitième chapitre décrit la réalisation d'un micro-réseau *SPIN* à 32 ports, ainsi que la conception d'une *puce d'évaluation* contenant la macro-cellule *SPIN32* entourée de toute la logique nécessaire à son instrumentation ;
- le neuvième chapitre donne le résultat des mesures que nous avons effectuées ;
- enfin, le dixième chapitre est la conclusion. Nous y reprenons les enseignements que nous avons tirées de ce travail et les évolutions futures qui sont déjà en cours au laboratoire *LIP6*.

Chapitre 2 *Problématique*

1 *Avant-propos*

Alors que la technologie 0.13 µm est largement utilisée actuellement, que les technologies 90 nm et 65nm sont opérationnelles et que la technologie 45 nm est en cours de validation, la capacité d'intégration double tous les 2 ans. Elle permet aujourd'hui d'intégrer plusieurs dizaines de millions de transistors sur une seule puce [15]. Cette tendance va se poursuivre, et il est clair que l'enjeu de la prochaine décennie dans le secteur des semi-conducteurs est d'intégrer sur une même puce des systèmes multi-processeurs hétérogènes (*Multi-Processors System-On-Chip*).

Dans ce cadre, le principal goulot d'étranglement au niveau architectural réside dans la gestion des communications entre les processeurs, les mémoires, et les périphériques. Les interconnexions à bus central (Figure 1) montrent très vite leurs limites. En effet, un bus central est une ressource partagée. Son extensibilité est donc très mauvaise. Si le nombre d'abonnés sur le bus augmente, la bande passante disponible pour chacun est inversement proportionnelle à leur nombre car la bande passante d'un bus central reste constante quel que soit le nombre d'abonnés. De plus, les capacités parasites vont également augmenter et la fréquence d'utilisation du bus sera limitée. Enfin, le *BCU* (*Bus Control Unit*) perd en performances s'il a beaucoup d'abonnés à satisfaire car le temps alloué à chacun diminue alors que le temps d'arbitrage augmente.

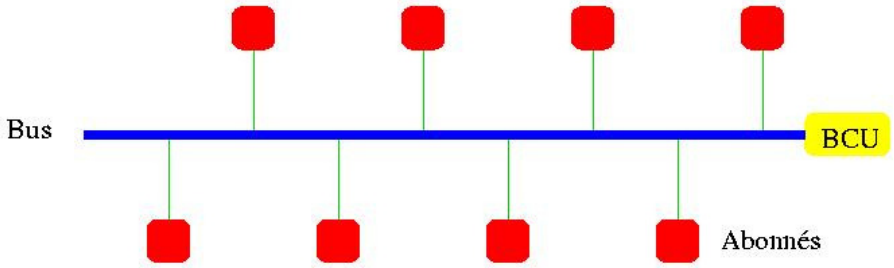

Figure 1 : Systèmes à bus central

Le micro-réseau d'interconnexion *SPIN* est une réponse à ce défi. L'idée directrice du projet *SPIN* est de remplacer le bus partagé par un micro-réseau [2] [3] d'interconnexion multi-étages utilisant des liaisons bidirectionnelles point à point. Il utilise la technique de commutation de paquets. Si cette technique est bien connue et a fait largement ses preuves dans le domaine des télécommunications et des machines parallèles, elle n'avait jamais été utilisée dans le domaine des systèmes intégrés sur puce.

Les principes généraux de l'architecture du micro-réseau *SPIN* ont été définis dans la thèse de Pierre Guerrier [1]. Cette architecture s'appuie largement sur l'expérience accumulée par le laboratoire *LIP6* [4] dans le domaine des réseaux hautes performances pour machines parallèles. Le micro-réseau *SPIN* possède une topologie en arbre quaternaire élargie [5], utilise la technique de commutation de paquet [6] [7] et l'algorithme de routage dit *wormhole* [8]. Il adopte une stratégie hybride de mémorisation des paquets, à la fois en entrée [5] [14] [9] et en sortie [12] [9] [10] des routeurs constituant le micro-réseau d'interconnexion. Le micro-réseau *SPIN* utilise des liens bidirectionnels, sans canaux virtuels [11] et un routage distribué qui peut être déterministe ou adaptatif. Il possède un contrôle de flux à base de crédits [12] [13] [14].

L'architecture *SPIN* a été publiée en 2000, et depuis, de nombreuses autres architectures de micro-réseaux ont été publiées. Cependant, une comparaison sérieuse de ces différentes architectures, en termes de performances, consommation, coût en surface de silicium nécessite d'aller jusqu'à l'implémentation matérielle de l'architecture sur silicium. Le but principal de notre travail est donc l'implémentation effective – jusqu'au dessin des masques de fabrication – du micro-réseau *SPIN* de façon à fournir des réponses quantitatives en termes de débit, latence, consommation et surface de silicium. De plus, nous avons été amenés à améliorer cette architecture, pour prendre en compte de nouveaux besoins tels que la garantie d'acheminement des paquets dans l'ordre chronologique, le traitement des interblocages dans le cas d'un trafic de type *requêtes/réponses*, ou la prise en compte du test de production.

2 Améliorations fonctionnelles du micro-réseau SPIN

2.1 Acheminement dans l'ordre

L'évaluation des problèmes posés par le remplacement du bus système par un micro-réseau tel que *SPIN* a mis en évidence que l'absence de garantie d'acheminement des paquets dans l'ordre chronologique (*in order delivery*) liée au caractère adaptatif du micro-réseau *SPIN* est une source de difficultés pour les protocoles de plus haut niveau. En effet, les protocoles de reprise sur erreur et de consistance mémoire sont beaucoup plus faciles à implémenter si les paquets sont livrés dans l'ordre chronologique. De plus, si un message est divisé en plusieurs paquets, la recomposition du message est plus complexe si les paquets arrivent dans le désordre car il faut reconstruire l'information concernant l'ordre des paquets au niveau de la destination.

Il était donc nécessaire de modifier l'algorithme de routage implémenté dans le routeur *RSPIN* pour rendre possible un acheminement des paquets dans l'ordre chronologique, lorsque l'application le nécessite. Si le routeur *RSPIN* fonctionne en mode déterministe, les paquets arrivent dans l'ordre chronologique. Par contre, si l'algorithme de routage est adaptatif, les paquets peuvent arriver dans le désordre car tous les paquets d'un même message ne prendront pas obligatoirement le même chemin et peuvent donc se doubler.

2.2 Traitement de l'interblocage

L'interblocage est un problème fatal dans tout système de communication. La thèse de Pierre Guerrier [1] a démontré que le micro-réseau *SPIN* garantit l'absence d'interblocage (*deadlock*) lorsque le trafic est unidirectionnel (un composant émetteur envoie un message sans attendre une réponse et un composant récepteur accepte toujours les messages qui lui arrivent sans envoyer aucun acquittement). Mais l'utilisation du micro-réseau *SPIN* dans une architecture multi-processeurs à mémoire partagée a mis en évidence une possibilité d'interblocage dans le cas d'un trafic de type *requête/réponse*, tel que l'on trouve dans les systèmes à mémoire partagée. Ce type de trafic introduit un cycle de dépendance évidente entre la *requête* et la *réponse* (le composant qui a envoyé la *requête* attend une *réponse*).
Nous avons enrichi l'algorithme de routage pour que le micro-réseau *SPIN* puisse garantir l'absence d'interblocage, quel que soit le type de trafic.

2.3 Testabilité

Un des enjeux le plus important dans le domaine des systèmes intégrés sur puce est le test après fabrication. Les systèmes sur puce devenant de plus en plus complexes, il est absolument nécessaire de pouvoir tester le comportement du micro-réseau du point de vue logique, mais également du point de vue des contraintes électriques et temporelles. En effet, le grand nombre de fils d'interconnexion augmente les risques de diaphonie entre signaux et donc la sensibilité à des fautes temporelles. Nous avons donc proposé une stratégie complète de test du micro-réseau *SPIN*, s'appuyant sur 3 modes de test, dont 2 s'exécutent à la fréquence de fonctionnement nominale (*at speed*) pour pouvoir prendre en compte les défauts dus aux problèmes de diaphonie.

3 Caractéristiques physiques

3.1 Le coût en surface de silicium

La surface est une préoccupation essentielle pour tous les concepteurs de systèmes intégrés sur puce. Malgré leur complexité croissante, ils doivent rester très compacts, pour des raisons de rendement de fabrication. Un micro-réseau d'interconnexion multi-étages à commutation de paquets tel que *SPIN* utilise un grand nombre de routeurs qu'il faut interconnecter par des nappes de fils comportant plusieurs milliers de fils. Le problème topologique n'est pas simple à résoudre. De plus, tous les routeurs du micro-réseau *SPIN* sont cadencés par la même horloge. Ceci pousse à implémenter tous les routeurs *RSPIN* constituant le micro-réseau *SPIN* de façon centralisée. Tous les routeurs *RSPIN* du micro-réseau *SPIN* sont intégrés dans une macro-cellule synchrone située au centre de la puce.
La faisabilité topologique d'une telle implémentation centralisée mérite d'être étudiée de façon approfondie. Il faudra en particulier déterminer si les niveaux de métallisation disponibles permettent effectivement de router les nombreuses nappes de fils d'interconnexion sans agrandir de façon significative la surface de silicium utilisée par le micro-réseau *SPIN*.

D'autre part, le coût de l'extensibilité en terme de surface doit être quantifié. C'est-à-dire que lorsque le nombre de composants interconnectés augmente, la surface de silicium occupée par le micro-réseau *SPIN* doit être évaluée.

3.2 Consommation

Selon la feuille de route de la *VSIA* [15], la fréquence de fonctionnement et le nombre de composants intégrés sur une même puce vont aller en augmentant. La consommation deviendra rapidement le principal facteur limitant dans les systèmes intégrés sur puce.
Un système intégré sur puce doit donc avoir une très faible consommation, surtout s'il est dédié à une application sans fil ou à une plate-forme de calcul mobile hautes performances, car malgré les avancées dans ce domaine, la durée de vie d'une batterie reste limitée. Il faut absolument faire des économies d'énergie partout où cela est possible.
La consommation est donc un critère fondamental pour évaluer les qualités d'une architecture d'interconnexion. Celle-ci dépendant fortement des caractéristiques physiques, il est indispensable d'aller jusqu'au silicium.

3.3 Performances

L'implémentation physique du micro-réseau *SPIN* est la seule solution permettant d'évaluer sérieusement la fréquence de fonctionnement, et donc les performances réelles en termes de latence ou de débit. C'est également la seule façon de mesurer la robustesse du micro-réseau *SPIN* en termes de taux d'erreur (*Bit Error Rate*), en prenant en compte les paramètres environnementaux (température, tension d'alimentation, …). Pour cela, il faut concevoir un environnement complet d'instrumentation pour faire les mesures de performances du micro-réseau *SPIN* en termes de bande passante, de fiabilité, de latence et de seuil de saturation.

4 Conclusion

Nous avons présenté dans ce chapitre les objectifs principaux de notre travail. Nous souhaitons répondre aux questions suivantes, concernant le micro-réseau *SPIN* :

- comment peut-on modifier l'architecture du micro-réseau *SPIN* pour fournir une garantie (optionnelle) d'acheminement dans l'ordre chronologique ?
- comment peut-on modifier l'algorithme de routage pour garantir l'absence d'interblocage, même dans le cas d'un trafic de type *requête/réponse* ?
- quel matériel faut-il introduire dans le micro-réseau *SPIN* pour garantir un taux de couverture suffisant pour le test de fabrication, en présence de fautes temporelles ?
- quelle est la surface occupée par le micro-réseau *SPIN*, lorsque l'on prend en compte les nappes de fils entre les routeurs *RSPIN* ?
- quelles sont les performances réelles du micro-réseau *SPIN*, en termes de temps de cycle, consommation, et taux d'erreur ?

Chapitre 3 *Etat de l'art*

Il existe actuellement de nombreux projets de recherche dans le domaine des architectures de micro-réseaux intégrés sur puce. Puisque notre travail porte essentiellement sur les problèmes posés par la réalisation effective (jusqu'au dessin des masques), et par l'industrialisation (testabilité) d'un tel micro-réseau, nous nous sommes limités à présenter les architectures de communication génériques ayant atteint un niveau de maturité suffisant pour qu'il existe une telle implémentation matérielle.

1 SiliconBackplane™

Pour satisfaire les besoins de ses clients, *Sonics* [16] a développé *SMART™* (*Sonics Methodology and Architecture for Rapid Time-to market*) [17], une collection complète de composants *IP* respectant la norme *OCP* (*Open Core Protocol*) [18].
Le cœur de *SMART™* est composé du micro-réseau *SiliconBackplan™* et du réseau hiérarchique *MultiChip Backplane™*. La Figure 2 représente un système utilisant les composants de *SMART™*.

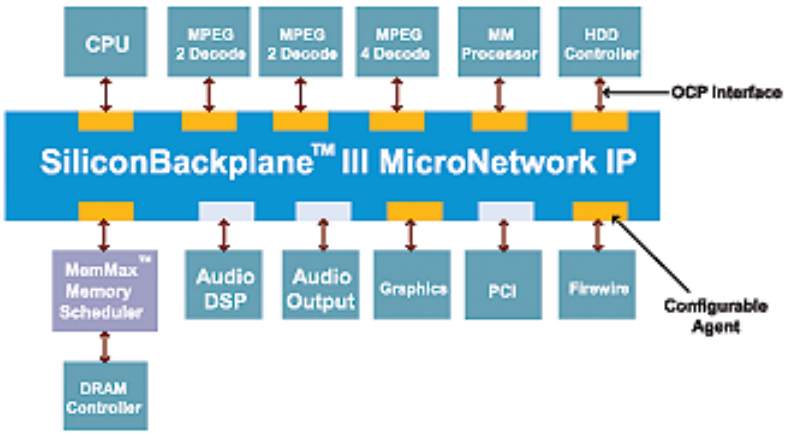

Figure 2: Système utilisant le micro-réseau SiliconBackplane™

Le *MultiChip Backplane™* permet la décomposition hiérarchique d'un système sur puce en plusieurs sous-systèmes, découplés entre eux en terme de fréquence de fonctionnement et de largeur du bus de donnée. Chaque sous-système possède un micro-réseau *SiliconBackplane™*. Il permet également l'envoi multiple (*multi-cast*) : une information envoyée par un composant peut être destinée à plusieurs destinations à la fois, à travers un ou plusieurs micro-réseaux.

Le micro-réseau *SiliconBackplane*™ se présente comme un micro-réseau, mais il s'agit en réalité d'un bus à multiplexage temporel, fortement pipeliné. Chaque composant se voit alloué une certaine fraction de la bande passante globale (en pratique, un certain nombre de *time slot*) lui permettant d'opérer à sa propre fréquence. Le micro-réseau *SiliconBackplane*™ offre ainsi une garantie de bande passante et de latence qui permet d'assurer que les échéances temps réel et le débit des différents types de flots de données soient satisfaits.
Le micro-réseau *SiliconBackplane*™ [19] utilise un agent pour chaque composant *IP*. Celui-ci joue le rôle d'un contrôleur réseau prenant en charge le protocole de communication. Grâce aux agents, tous les composants *IP* sont découplés les uns des autres, ainsi que du micro-réseau *SiliconBackplane*™ lui-même. Cela permet au concepteur de systèmes de choisir indépendamment les uns des autres les paramètres de fonctionnement des différents composants *IP* utilisés ainsi que ceux du micro-réseau *SiliconBackplane*™, à l'aide de l'environnement de développement *Sonics Studio*™.

La technique de multiplexage temporel est la principale limite du micro-réseau *SiliconBackplane*™ : cette architecture étant fondamentalement un bus, reste une ressource partagée. Par conséquent, il n'est pas réellement extensible ; et la bande passante totale ne peut dépasser 4.8 Go/s à une fréquence de fonctionnement de 300 MHz.

2 CoreConnect™

CoreConnect™ [20] est un produit *IBM* [21] qui fait partie de *Blue Logic*™ [22], l'environnement de développement de systèmes sur puce complexes. La réutilisation des composants *IP* est rendue possible car *Coreconnect*™ fournit deux sortes de bus utilisant la même interface pour les communications entre composants : le bus *PLB* (*Processor Local Bus*) et le bus *OPB* (*On-Chip Peripheral Bus*). Il fournit également des arbitres pour chaque bus, ainsi que des passerelles entre les bus de différents types ou de même type. La Figure 3 donne un exemple d'un système utilisant *CoreConnect*™.

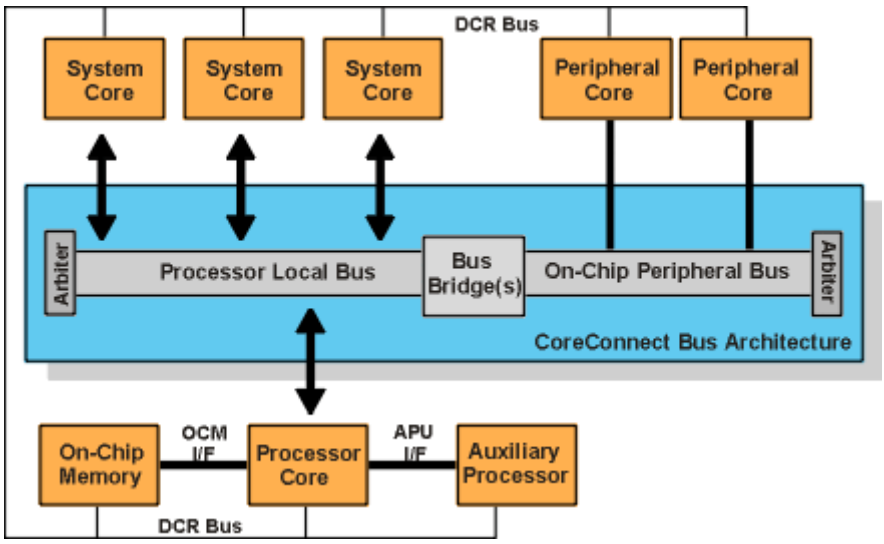

Figure 3: Système utilisant CoreConnect™

Les composants les plus performants, tels que les processeurs ou les contrôleurs *DMA*, nécessitant de grandes bandes passantes et de faibles latences sont mis sur le bus *PLB*. Ce bus supporte jusqu'à 16 initiateurs et permet plusieurs transactions simultanées. Cependant, comme tous les bus, sa fréquence de fonctionnement dépend du nombre d'abonnés. Le bus *PLB* est entièrement synchrone et son architecture permet l'extension du bus de données jusqu'à 256 bits de large. Il offre un système de priorité qui permet l'interruption de transactions en cours par d'autres transactions plus prioritaires.

Les périphériques ayant de faibles débits sont connectés au bus *OPB*, ce qui permet d'alléger le trafic sur le bus *PLB* pour lui garantir des performances élevées. Ce bus, entièrement synchrone, possède un bus de donnée de 32 bits de large. Il supporte plusieurs initiateurs ainsi que plusieurs cibles.

De même que pour le micro-réseau *SiliconBackplane™*, *CoreConnect™* est une ressource partagée dont la bande passante n'est pas extensible.

3 AMBA™

AMBA™ (*Advanced Microcontroller Bus Architecture*) [23] [24] est, à l'origine, un standard de communication pour les microcontrôleurs hautes performances embarqués sur puce utilisant un processeur *ARM* [25]. *AMBA™* est une architecture assez fermée, qui permet la réutilisation des composants *IP*, pourvu que ces composants soient compatibles avec le standard *AMBA™*.

La spécification *AMBA™* définit le protocole de communication qui permet le transfert de données entre les différents composants du système, sans définir l'architecture de l'interconnexion. Cela donne aux concepteurs la liberté de définir eux-mêmes une architecture qui peut aller de la simple liaison bidirectionnelle point à point à une structure complexe. La Figure 4 montre un exemple de système utilisant *AMBA™*. Un transfert de données est constitué d'une transaction d'écriture ou de lecture de l'initiateur et d'une réponse de la cible.

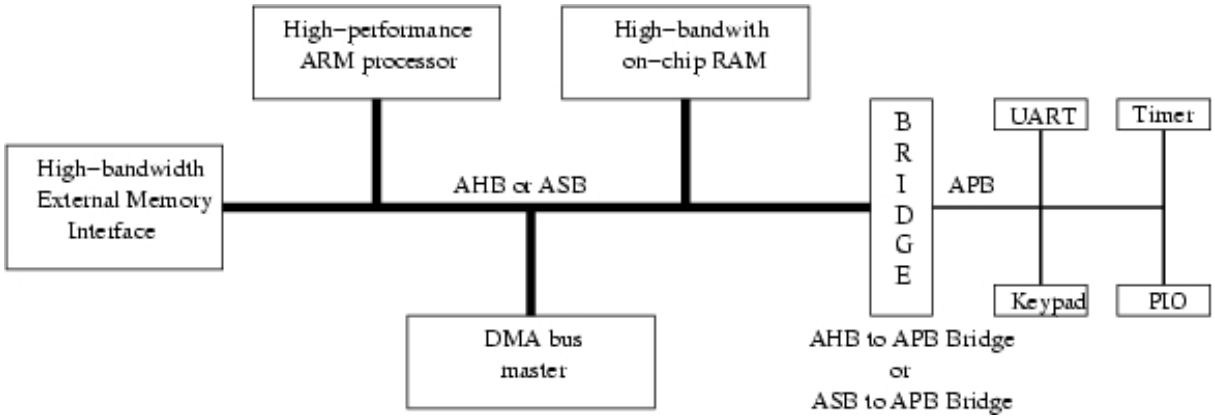

Figure 4 : Exemple d'un système utilisant AMBA™

AMBA™, spécifie trois bus : le bus *AHB* (*Advanced High-performance Bus*), le bus *ASB* (*Advanced System Bus*) et le bus *APB* (*Advanced Peripheral Bus*).

Le bus *AHB* et le bus *ASB* sont des bus multi-initiateurs, multi-cibles destinés aux composants du système, tel que les processeurs, les *DSP* (*Digital Signal Processor*), les mémoires intégrées et les contrôleurs de mémoires externes (le bus *AHB* est plus performant du point de vue de la fréquence et de la bande passante).
Le bus *APB*, quant à lui, est un bus à initiateur unique destiné aux périphériques, de préférence à basse consommation. Ce bus peut être connecté au bus *ASB* ou au bus *AHB*, au même titre qu'un abonné à travers une passerelle, pour rendre les périphériques connectés accessibles aux composants du système sur puce. Du point de vue du bus *APB*, le seul initiateur est la passerelle vers le bus *ASB* ou le bus *AHB*. Tous les périphériques connectés au bus *APB* sont des cibles.

Comme dans tous les bus, *AMBA™* possède un arbitre de bus assurant qu'un seul initiateur est actif à un instant donné. *AMBA™* est logiquement un bus mais a l'avantage d'être implémenté physiquement comme des multiplexeurs. L'utilisation de multiplexeurs pour la réalisation physique permet d'éviter les problèmes électriques liés aux portes trois-états (*tristate*). Cela permet également d'éviter que les abonnés non actifs ne constituent des capacités parasites qui limitent la fréquence de fonctionnement du bus. Malheureusement, le nombre d'initiateurs est limité à 16 car le nombre d'entrée du multiplexeur ne peut augmenter indéfiniment. De plus, l'arbitrage central demeure un goulot d'étranglement. L'extensibilité du bus *AMBA™* est donc très limitée.
Enfin, tous les composants *IP* utilisés doivent être compatibles avec le standard *AMBA™*.

4 Æthereal™

Æthereal™ [26] [27] [28] [29] est un réseau sur puce développé par *Philips* [30].A la différence des trois architectures précédentes, cette architecture est un véritable micro-réseau intégré, capable de fournir une bande passante qui augmente avec le nombre d'abonnés.
Le micro-réseau sur puce *Æthereal™* fournit deux types de service : un service d'acheminement de paquets de type *Best Effort*, sans garantie de service, et un service d'acheminement de paquets de type *Guaranteed-throughput,* avec garantie de latence et garantie de débit pour les applications soumises à des contraintes temps réel. *Æthereal™* supporte différentes interfaces standards (*VCI*, *OCP*, *AMBA*, *AXI* ou *DTL*), et permet l'envoi multiple (*multicast*).

Il est composé de deux types de modules : les routeurs et les modules d'interface. Ces modules sont reliés entre eux par des nappes de fils sans pipeline. Conceptuellement, le routeur *Æthereal™* est équivalent à deux routeurs indépendants [31] :
Le routeur *Best Effort Services* (*BES*), inspiré de la technologie *SPIN*, est un routeur à commutation de paquets de type *wormhole* avec des mémoires tampons en entrée. Il offre une communication sans perte ni corruption des paquets, avec un contrôle de flux à base de crédits, un mécanisme de priorité de type *round-robin* et un algorithme de routage déterministe qui permet aux paquets d'être acheminés dans l'ordre chronologique.
Le routeur *Guaranteed-throughput Services* (*GS*), quant à lui, apporte une garantie de débit et de latence par une technique de multiplexage temporel basé sur la commutation de circuits, tout en *pipelinant* les données : chaque module d'interface contient une table d'allocation

statique définissant pour chaque intervalle de temps donné, quelle tranche de temps est allouée à quelle communication de bout en bout (entre un initiateur *i* et une cible *j*). Le routeur *GS* apporte donc la garantie de bande passante mais ceci suppose un fonctionnement totalement synchrone du routeur *GS*.

Le module d'interface (*Network Interface*) [32] du micro-réseau *Æthereal*™ joue le rôle de contrôleur réseau en prenant en charge le protocole de communication. En général, plusieurs composants peuvent être connectés à un module d'interface et inversement. Le micro-réseau sur puce *Æthereal*™ est reconfigurable. La reconfiguration se fait à travers le micro-réseau *Æthereal*™ lui-même, sans aucun autre signal de contrôle additionnel.

La Figure 5 donne un exemple de système sur puce utilisant un micro-réseau *Æthereal*™. Le système contient des mémoires (M*i*), des processeurs (P*i*) et des interfaces mémoires externes (MI*i*). Ces composants communiquent entre eux par l'intermédiaire du micro-réseau sur puce *Æthereal*™, constitué de routeurs (R*i*) et de modules d'interface (NI*i*).

L'utilisation de routeurs et de liaisons point à point fait de l'architecture *Æthereal*™ un véritable micro-réseau intégré dont la philosophie générale est très voisine de celle de l'architecture *SPIN* à laquelle nous nous intéressons dans cette thèse. Cependant la richesse des fonctionnalités offertes, en fait une architecture très coûteuse en termes de surface ou de consommation. En effet, l'implémentation montre que la surface d'un routeur *Æthereal*™ (*GS et BES*) va de 0.13 mm² à 0.24 mm² selon la complexité [28]; mais le micro-réseau *Æthereal*™ nécessite plus de routeur que le micro-réseau *SPIN* pour un même nombre de ports. De plus, la taille du module d'interface nécessaire entre chacun des abonnés et le micro-réseau *Æthereal*™ est de 0.25 mm² [32].

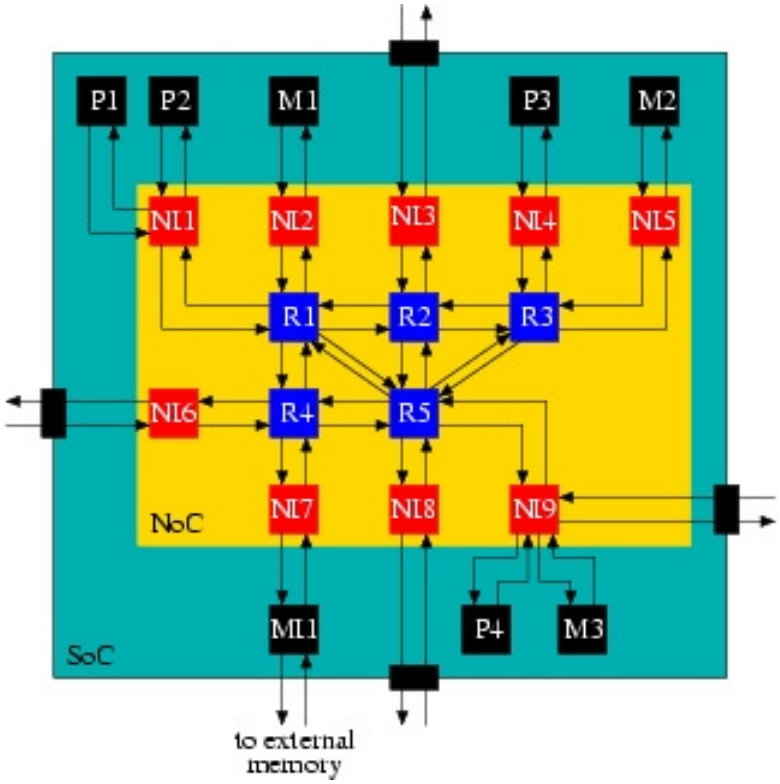

Figure 5 : Système sur puce utilisant Æthereal™

5 Conclusion

Dans ce chapitre, nous avons décrit différentes architectures d'interconnexions génériques ayant atteint un niveau de maturité suffisant pour être exploitées industriellement. Les architectures *CoreConnect*™, *SiliconBackplane*™ et *AMBA*™ restent fondamentalement des bus, possédant une bande passante limitée. Le partage de la bande passante entre les différentes composantes du système est un goulot d'étranglement qui limite l'extensibilité du système intégré sur puce. En revanche, le micro-réseau sur puce *Æthereal*™ est un vrai micro-réseau sur puce permettant plusieurs communications simultanées ; mais il est très complexe et fourni une solution très riche qui coûte énormément en terme de surface de silicium.

Chapitre 4 *Architecture du micro-réseau SPIN*

Les principes généraux du micro-réseau *SPIN* ont été définis dans la thèse de Pierre Guerrier [1]. Nous rappelons dans ce chapitre les caractéristiques principales de cette architecture.

1 Architecture générale

L'idée directrice du projet *SPIN* est de remplacer le bus partagé par un micro-réseau d'interconnexions à commutation de paquets [33] multi-étages utilisant des liaisons point à point. Le micro-réseau *SPIN* respecte l'interface standard *VCI* (*Virtual Component Interface*) [15], une variante de la norme *OCP* [18], qui se prête particulièrement bien à la réalisation d'architectures multi-processeurs à mémoire partagée. Le micro-réseau *SPIN* est réellement extensible : il offre une bande passante cumulée qui croît linéairement avec le nombre d'abonnés connectés.

Le micro-réseau *SPIN* repose sur trois macro-cellules *VLSI* : les routeurs *RSPIN* (*Router for SPIN*), le *wrapper Initiateur* et le *wrapper Cible* qui jouent le rôle de contrôleurs réseau. La Figure 6 montre l'architecture générale.

Dans cette thèse, nous nous sommes concentrés exclusivement sur le micro-réseau de routeurs *RSPIN*. Les *wrappers Initiateur* et *cible* sont le sujet d'une autre thèse [34].

2 La topologie du micro-réseau SPIN

Le micro-réseau *SPIN* possède une topologie en arbre quaternaire élargi. Cette topologie se caractérise par le fait que chaque nœud possède 4 fils et que les pères sont répliqués 4 fois à chaque niveau de l'arbre (Figure 7). Ainsi, chacun des 4 pères offre un plus court chemin équivalent entre 2 feuilles de l'arbre. Dans cette topologie, le plus court chemin d'une feuille à l'autre est celui passant par un plus proche ancêtre commun.

La topologie en arbre quaternaire élargi possède plusieurs avantages : son diamètre (nombre maximal de noeuds entre deux abonnés) reste raisonnable ($2 * \log_4 n$, où n est le nombre d'abonnés du micro-réseau *SPIN*). Elle est non bloquante, extensible, et possède une structure naturellement hiérarchique dont on peut tirer partie dans les systèmes intégrés.

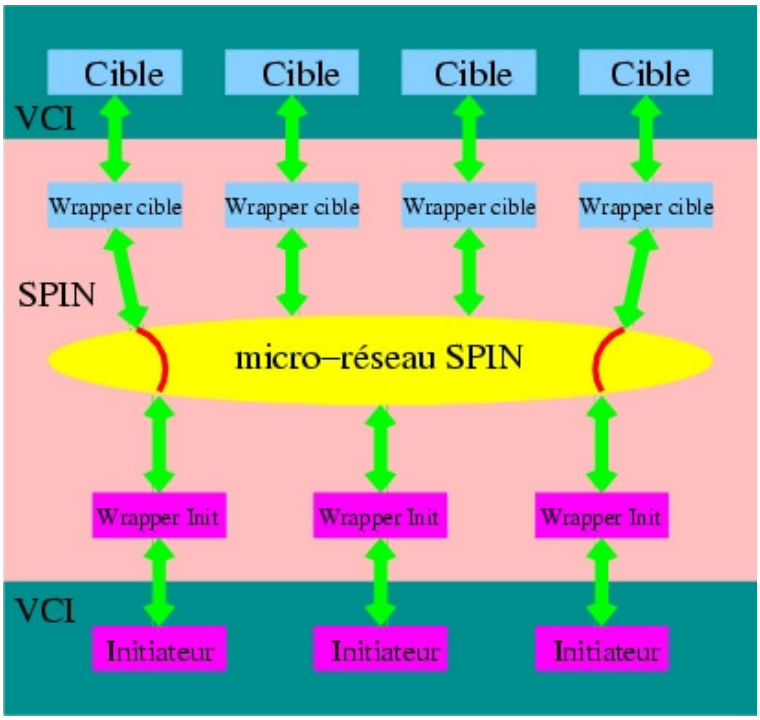

Figure 6 : Micro-réseau SPIN fournissant une interface VCI

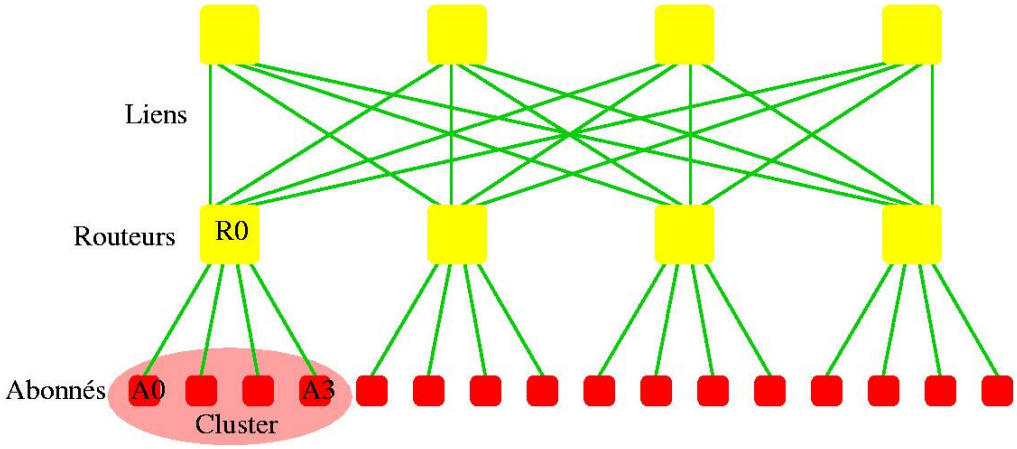

Figure 7 : Exemple de micro-réseau SPIN

La topologie en arbre quaternaire élargi fait du micro-réseau *SPIN* un micro-réseau indirect, car tous les routeurs *RSPIN* ne sont pas reliés à des abonnés.
On appelle *cluster* un groupe d'abonnés reliés au même routeur *RSPIN* de premier niveau. Comme dans les systèmes d'interconnexion utilisant une hiérarchie de bus, une manière très simple d'exploiter la localité spatiale est de mettre dans le même *cluster* les interlocuteurs fréquents (Figure 7) qui, logiquement, appartiennent à un même sous-système. Les communications internes à un même sous-système resteront donc locales, ce qui permet de réduire à la fois la latence et la consommation : pour aller de l'abonné A0 vers l'abonné A3, par exemple, un paquet ne doit remonter que d'un étage jusqu'au routeur R0 pour descendre ensuite vers l'abonné A3.

3 Le format du paquet SPIN

Le micro-réseau *SPIN* est un micro-réseau multi-étages à commutation de paquets. Les abonnés au micro-réseau communiquent entre eux par l'envoi dans le micro-réseau de paquets de taille finie. Du point de vue du routeur *RSPIN*, un paquet est composé d'un nombre quelconque de mots de 36 bits. Un paquet a une source et une destination ; il est ainsi envoyé de routeur *RSPIN* en routeur *RSPIN* à travers les liens.

Le micro-réseau *SPIN* considère un paquet comme une information atomique. Il existe un marqueur de début de paquet et un marqueur de fin de paquet. Une fois que le début du paquet est émis sur un lien entre deux routeurs *RSPIN*, le lien est occupé et restera occupé jusqu'à l'émission du marqueur de fin de paquet. La Figure 8 montre qu'un paquet *SPIN* est composé de deux parties distinctes : l'en-tête et le corps du paquet.

Figure 8 : Composition d'un paquet SPIN

3.1 L'en-tête du paquet

L'en-tête du paquet est un mot de 36 bits qui contient toutes les informations de routage. Il se décompose comme suit (Figure 9) :

- 1 bit de début de paquet *bp* (*begin of packet*), utilisé par le routeur *RSPIN* pour spécifier que ce mot est le début d'un paquet ;
- 1 bit de fin de paquet *ep* (*end of packet*), utilisé par le routeur *RSPIN* pour spécifier qu'il s'agit d'une fin de paquet ;
- 8 bits de destination (*dest*), analysés par le routeur *RSPIN*. Ces bits spécifient l'adresse réseau du composant de destination du paquet. Ceci limite le nombre d'abonnés à 256;
- 3 bits de typage du paquet utilisés par le routeur *RSPIN* pour contrôler l'algorithme de routage, paquet par paquet. Le bit *queue* indique si le paquet peut utiliser la queue centrale en cas de contention. Le bit *req* définit si le paquet est une *requête* ou une *réponse* (lorsque le micro-réseau *SPIN* est utilisé pour acheminer des transactions de type *VCI*). Le bit *in_order* indique si les paquets doivent être acheminés dans l'ordre chronologique ou non ; autrement dit, si le routage est adaptatif ou déterministe ;
- 23 bits de protocole qui ne sont pas utilisés par le routeur *RSPIN* mais par les protocoles, de niveaux plus élevés, implémentés dans les *wrappers*. Parmi ceux-ci, nous avons le champ *prot* (21 bits), le bit *er* (*error*) qui est utilisé pour spécifier qu'une erreur est survenue dans le transfert de ce mot, et le bit *par* (*parity*) qui est un bit de parité portant sur les 36 bits du mot.

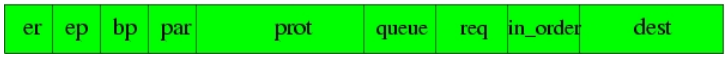

Figure 9 : En-tête d'un paquet

3.2 Le corps du paquet

Le corps du paquet est composé d'un nombre quelconque (qui peut être nul) de mots de 36 bits. Toutefois, la profondeur des files d'attente internes au routeur *RSPIN* a été optimisée pour des paquets ne dépassant pas 16 mots.

Chaque mot de 36 bits du corps du paquet se décompose comme ci-dessous (Figure 10):

- 1 bit de début de paquet *bp* (*begin of packet*), utilisé par le routeur *RSPIN* pour spécifier que ce mot est le début d'un paquet ;
- 1 bit de fin de paquet *ep* (*end of packet*), utilisé par le routeur *RSPIN* pour spécifier qu'il s'agit d'une fin de paquet ;
- 34 bits de données qui ne sont pas analysés par le routeur *RSPIN*. Parmi ceux-ci, nous avons le champ *data* (32 bits) qui contient les données utiles, ainsi que les deux bits *er* et *par*.

Figure 10 : Corps d'un paquet

4 Stratégie de routage adaptative

Le micro-réseau *SPIN* possède une stratégie de routage distribuée. Chaque routeur *RSPIN* est indépendant. Il prend les décisions d'aiguillage et/ou de mémorisation du paquet sans l'intervention d'une synchronisation centrale.
Le routage des paquets est du type *wormhole* [35] pour éviter que les tampons de mémorisation prennent trop de place sur le silicium et pour minimiser la latence.
Chaque routeur décode l'en-tête du paquet qui se présente à son entrée et ouvre le chemin adéquat pour l'envoyer vers une de ses sorties, sans attendre de recevoir le paquet en entier.
Cela permet d'avoir des *FIFOs* d'entrée de petite taille (4 mots). L'inconvénient est qu'un paquet peut s'étaler sur plusieurs routeurs. Il peut créer ainsi une contention en rendant indisponibles plusieurs liens si sa longueur est trop importante. Par exemple, sur la Figure 11, le paquet violet (----), qui s'étale sur 3 routeurs *RSPIN* (routeur 1, routeur 2, routeur 3), bloque le paquet bleu (—) si ce dernier veut aussi remonter vers le routeur 2.

L'algorithme de routage dépend directement de deux facteurs :

- De la topologie du micro-réseau *SPIN*, puisque la décision d'aiguillage d'un paquet doit être cohérente avec la position du routeur *RSPIN* dans le micro-réseau *SPIN*,
- De l'adresse de destination qui est placée dans l'en-tête du paquet par le producteur.

Si l'abonné consommateur se trouve sous le routeur *RSPIN* dans l'arbre quaternaire élargi que représente le micro-réseau *SPIN*, le paquet est envoyé vers le bas. Dans le cas contraire, il est envoyé vers le haut. Si le paquet est envoyé vers le haut, le port de sortie est choisi selon une priorité tournante (*round robin*) car les 4 ports de sortie sont complètement équivalents. Si le paquet est envoyé vers le bas, il est envoyé vers le sous-arbre qui contient l'abonné consommateur. Chaque abonné producteur peut donc communiquer avec n'importe quel abonné consommateur, simplement en changeant l'adresse de destination.

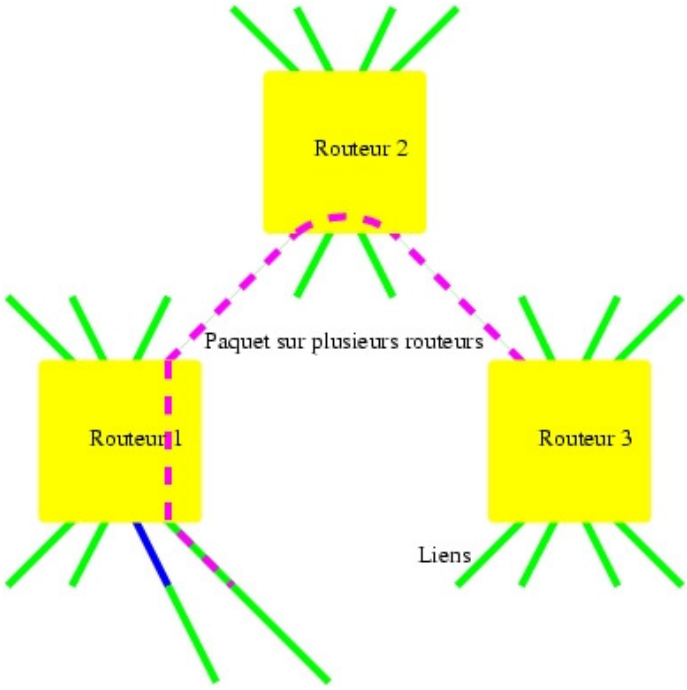

Figure 11 : Cas d'un paquet s'étalant sur plusieurs routeurs RSPIN

Cela signifie que le schéma de communication dans le micro-réseau *SPIN* n'est pas figé. Les canaux de communication sont ouverts automatiquement, selon les besoins des abonnés, sans aucune reconfiguration globale du système. Le micro-réseau *SPIN* est donc un micro-réseau à routage dynamique.

5 Les liens bidirectionnels

Les liens du micro-réseau *SPIN* sont des liens bidirectionnels bipoints. Chaque routeur *RSPIN* du micro-réseau *SPIN* possède 4 ports vers les liens montants (*u0*, *u1*, *u2*, *u3*) et 4 ports vers les liens descendants (*d0*, *d1*, *d2*, *d3*) (Figure 12).
Dans certains cas, les routeurs de plus haut niveau sont des demi-routeurs, c'est-à-dire que l'on n'utilise pas les 4 liens montants (*u0*, *u1*, *u2*, *u3*). Il n'y a donc aucune raison de mobiliser de la place sur le silicium pour ces liens. On utilisera donc en quelque sorte un demi-routeur dénommé *RSPIN2*, comme le montre la Figure 13.

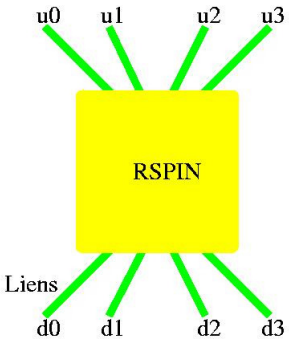

Figure 12 : Liens à l'interface du routeur RSPIN

Chaque lien bipoint dans le micro-réseau *SPIN* est constitué de 2 nappes de 38 bits. Ils se décomposent comme illustré par la Figure 14 :

- 36 bits qui constituent le mot d'un paquet,
- 2 bits, pour le contrôle de flux à crédits : *data_valid* (*dv*) et *credit_return* (*cr*).

Les deux directions du lien bidirectionnel peuvent être utilisées en parallèle, ce qui permet un fonctionnement *full-duplex*.

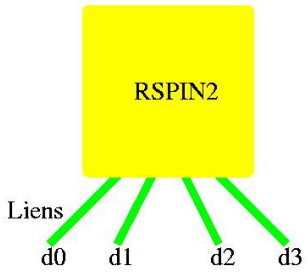

Figure 13 : Liens à l'interface du demi-routeur RSPIN2

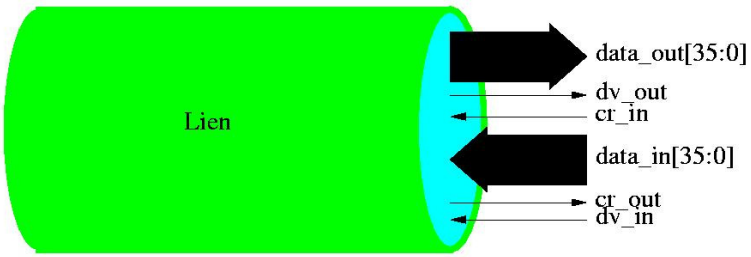

Figure 14 : Lien du micro-réseau SPIN

6 Le contrôle de flux

Pour chacune des deux directions, il existe au niveau de chaque lien physique un mécanisme de contrôle de flux de bas niveau. Le contrôle de flux est assuré par un mécanisme de crédit d'émission de mots. Le rôle de ce mécanisme est de garantir que si un tampon d'entrée est saturé, il ne recevra plus de données. Cette précaution assure un fonctionnement sans perte, puisque la contention, une fois détectée, se rétro-propagera vers l'émetteur primaire.

Le contrôle de flux à crédit consiste à avoir, dans le producteur, une estimation pessimiste de la place disponible dans le tampon d'entrée du consommateur. L'évaluation est évidemment initialisée à une valeur égale à la profondeur du tampon de réception. Chaque fois qu'un mot est envoyé à travers le lien par le producteur, l'estimateur est décrémenté. De même, lorsqu'une place se libère dans le tampon d'entrée du consommateur, celui-ci envoie un crédit pour permettre au producteur d'incrémenter son estimation (Figure 15).

Le contrôle de flux à crédits permet au producteur d'interrompre les envois de données dès qu'il n'y a plus de place disponible dans le tampon d'entrée du consommateur. Le contrôle de

flux se fait donc au niveau du producteur, avant de provoquer la saturation du tampon d'entrée du consommateur.

Le principal avantage du mécanisme de crédit est son caractère asynchrone : la durée effective du transfert d'un mot peut être supérieure à un cycle. Ceci permet d'insérer des étages de *pipeline* sur le lien physique, lorsque le temps de transmission entre deux routeurs *RSPIN* devient le facteur limitant la fréquence de fonctionnement du micro-réseau d'interconnexion *SPIN*. Cette propriété fait du micro-réseau *SPIN* un micro-réseau robuste du point de vue des problèmes de temps de propagation, tolérant une grande latence sur les liens entre routeurs *RSPIN*.

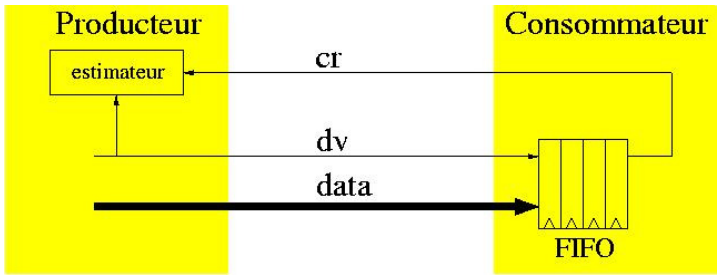

Figure 15 : Protocole de contrôle de flux à crédit

7 La bande passante

Lorsque plusieurs sous-systèmes sont intégrés sur une même puce, le système d'interconnexion doit pouvoir fournir à chaque sous-système la bande passante qui lui est nécessaire. Dans les domaines tels que la télévision haute définition ou les télécommunications, chaque sous-système peut émettre ou recevoir un débit de données de l'ordre de plusieurs Gbit/s. Il est certain que ces exigences en bande passante ne cesseront d'augmenter dans un futur proche.

Hormis les bits de contrôle de flux (*data_valid* et *credit_return*) et les bits de signalisation (*bp*, *ep*), le micro-réseau *SPIN* offre sur chacun de ses liens 34 bits de données utiles dans chaque direction. Avec une fréquence de fonctionnement de 200 MHz, la bande passante de chaque lien se calcule de la façon suivante :

2 * 34 bits * 200 MHz = 2 * 6,8 Gbit/s = 13,6 Gbit/s

Pour évaluer le débit utile, il faut cependant tenir compte du surcoût associé au mot d'en-tête du paquet *SPIN* et multiplier cette valeur par le rapport *(L-1) / 4*, où *L* est la longueur moyenne d'un paquet.

Il faut cependant noter qu'à cause des contentions qui peuvent se créer au niveau des tampons d'entrée des routeurs *RSPIN*, le débit réel est évidemment plus faible que le débit maximal. C'est une conséquence directe de la technique de commutation de paquets.
Des simulations seront effectuées pour mettre en évidence le pourcentage de la charge maximale auquel intervient la saturation du micro-réseau *SPIN*. Ainsi, nous pourrons quantifier le débit réel dans chaque direction pour chaque abonné.

8 Extensibilité

La topologie en arbre quaternaire élargi est naturellement extensible. Il existe deux types de micro-réseau *SPIN* : les micro-réseaux de type A et ceux de type B. Les micro-réseaux de type A sont des micro-réseaux dont les routeurs du niveau le plus élevé dans l'arbre élargi sont des demi-routeurs *RSPIN2* (un micro-réseau *SPIN* à 16 ports en est un exemple). Un micro-réseau de type B est construit en mettant tête-bêche deux micro-réseaux de type A d'ordre inférieur. Dans les micro-réseaux de type B, tous les routeurs sont des routeurs *RSPIN* ; un micro-réseau *SPIN* à 32 ports en est un exemple. Le Tableau 1 résume le nombre de routeurs *RSPIN* nécessaires en fonction du nombre d'abonnés ; il montre la limite de la topologie en arbre quaternaire élargi. En effet, au dessus de 128 abonnés, le nombre de routeurs *RSPIN* nécessaires est supérieur ou égal au nombre de ports. C'est pourquoi nous avons limité le champ d'adresse de destination dans les paquets à 8 bits, suffisant pour adresser 256 abonnés.

Type de micro-	Nombre de ports	Nombre de routeurs
A	4	1
B	8	2
A	16	8
B	32	16
A	64	48
B	128	96
A	256	256
B	512	512
A	1024	1380
B	2048	2760

Tableau 1 : Nombre de routeurs RSPIN en fonction du nombre de ports

9 Conclusion

Nous avons rappelé dans ce chapitre les principales caractéristiques architecturales du micro-réseau *SPIN*. Il s'agit d'un micro-réseau à commutation de paquets, extensible, ayant une topologie en arbre quaternaire élargi. Le micro-réseau *SPIN* offre une interface *VCI* et peut offrir une bande passante cumulée qui croît linéairement avec le nombre d'abonnés connectés. Sa stratégie de routage est adaptative, de type *wormhole* et utilise un contrôle de flux à base de crédits.

Le micro-réseau *SPIN* est constitué de routeurs *RSPIN*, connectés entre eux par des liens bipoints *full-duplex* ainsi que de *wrappers* fournissant une interface *VCI*.
Dans cette thèse, nous nous sommes concentrés exclusivement sur la réalisation matérielle d'un micro-réseau *SPIN* nu ; c'est-à-dire le réseau de routeur *RSPIN*, sans les *wrappers*.

Chapitre 5 *Micro-architecture du routeur RSPIN*

Le routeur, dénommé *RSPIN* (*Router for SPIN*), est l'élément de base du micro-réseau *SPIN*. Ce sont les routeurs *RSPIN* qui se chargent d'acheminer les paquets à leurs destinataires, et la performance globale du micro-réseau *SPIN* dépend directement du comportement individuel de chaque routeur *RSPIN*.
Ce chapitre présente la micro-architecture interne du routeur *RSPIN*. Avant de détailler celle-ci, nous introduisons deux évolutions importantes de la spécification fonctionnelle, relatives au traitement de l'interblocage, et au contrôle de l'adaptativité.

1 *Interblocage*

Le micro-réseau *SPIN* a été initialement conçu pour traiter du trafic unidirectionnel : n'importe quel composant peut communiquer avec n'importe quel composant et il n'y a pas de corrélation entre deux paquets. Par exemple dans les applications multimédia ou vidéo, un composant A envoie un message de longueur quelconque à un composant B sans acquittement.
Dans ce contexte, le problème de l'interblocage (*deadlock*) ne se pose pas. En effet, le trafic unidirectionnel a été étudié dans la thèse de Pierre Guerrier [1] et nous savons que la topologie en arbre quaternaire élargi, avec ou sans adaptativité, ne garantit pas l'absence de contention. En revanche, elle garantit l'absence d'interblocage pour ce type de trafic.

Nous voulons également utiliser le micro-réseau *SPIN* dans un contexte de systèmes utilisant un espace d'adressage partagé. Dans ce cas, une transaction est composée d'une paire de paquets (un paquet *requête* et un paquet *réponse*) corrélés entre eux :

- il faut recevoir un paquet *requête* avant de pouvoir envoyer un paquet *réponse*,
- le paquet *réponse* est renvoyé à l'émetteur du paquet *requête* correspondant.

La Figure 16 montre un exemple d'interblocage pouvant survenir dans un micro-réseau *SPIN* dans le cas d'un trafic de type *requête*/*réponse* :
Soit un routeur *RSPIN* ayant une cible *C* sur le port *d2* et des initiateurs *I0* sur le port *d0* et *I1* sur le port *d1* ; nous supposons que les ports *u0*, *u1*, *u2* et *u3* sont connectés au reste du micro-réseau *SPIN*. Admettons que l'initiateur *I0* du port *d0* envoie un long paquet *requête* vers la cible *C* du port *d2* (paquet violet --- sur la Figure 16) et qu'une cible se trouvant dans le reste du micro-réseau *SPIN* envoie un paquet *réponse* vers l'initiateur *I0* du port *d0* (paquet rouge — - — sur la Figure 16). Par ailleurs, supposons que l'initiateur *I1* du port *d1* envoie un paquet *requête* vers la cible *C* du port *d2* (paquet en noir — - - — sur la Figure 16) et que la cible *C* du port *d2* envoie un paquet *réponse* vers l'initiateur *I0* du port *d0* (paquet en bleu — sur la Figure 16). Le paquet noir (— - - —) veut aller vers le port *d2* mais ce dernier est

occupé par le paquet violet (---) ; le paquet noir (— - - —) va donc dans la queue centrale en attendant que le port se libère. Le paquet bleu (—) veut aller vers le port *d0* mais ce dernier est occupé par le paquet rouge (— - —) ; le paquet bleu (—) va également dans la queue centrale, mais se trouve derrière le paquet noir (— - - —). Le paquet bleu (—) est le paquet *réponse* de la cible *C* du port *d2* correspondant au paquet *requête* violet (---). Puisque le paquet bleu (—) est bloqué dans la queue centrale, la *FIFO réponse* de la cible *C* en *d2* va se remplir rapidement ; ce qui a pour conséquence que cette dernière ne pourra plus continuer à traiter le paquet *requête* représenté par le paquet violet (---). En résumé, le paquet violet (---) bloque le paquet noir (— - - —), le paquet noir (— - - —) bloque le paquet bleu (—) et le paquet bleu (—) bloque le paquet violet (---). Il y a donc interblocage.

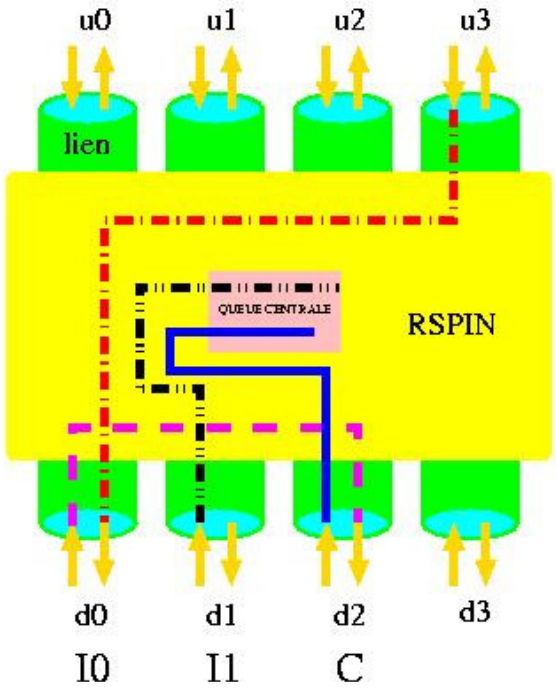

Figure 16 : Interblocage dans un micro-réseau SPIN

Pour éviter un interblocage, la solution classique est de séparer les ressources utilisées par les deux types de paquets (*requête* et *réponse*).
La topologie en arbre quaternaire élargi du micro-réseau *SPIN* est très avantageuse dans cette optique car l'ensemble des routeurs *RSPIN*, sauf les routeurs de premier niveau, constitue 4 sous-réseaux complètement indépendants. La Figure 17 montre un micro-réseau *SPIN* à 16 ports sur lequel sont représentés les 4 sous-réseaux : le sous-réseau noir (— - - —), rouge (— - —), bleu (—) et violet (---) (les routeurs de premier niveau sont mis en commun). Il suffit donc d'allouer un sous-réseau à un type particulier de paquet pour séparer les ressources utilisées par les paquets *requêtes* et les paquets *réponses*. Pour les routeurs de premier niveau, il suffit d'interdire l'utilisation des queues centrales. De la sorte, nous obtenons 4 sous-réseaux complètement indépendants.
Prendre en compte le type de paquet est une option du routeur *RSPIN*. Le wrapper *VCI/SPIN* signale au micro-réseau *SPIN* le type du paquet *VCI* en mettant le bit numéro 9 à la valeur logique '1' si le paquet est une *requête* et à la valeur logique '0' si le paquet est une *réponse*.
C'est le bloc *icb* (*Input Control Bloc*) qui aiguille les requêtes d'allocation:

- pour les paquets *requête*, les requêtes d'allocation ne sont envoyées qu'aux sorties 0 et 1,

- pour les paquets *réponse*, les requêtes d'allocation ne sont envoyées qu'aux sorties 2 et 3.

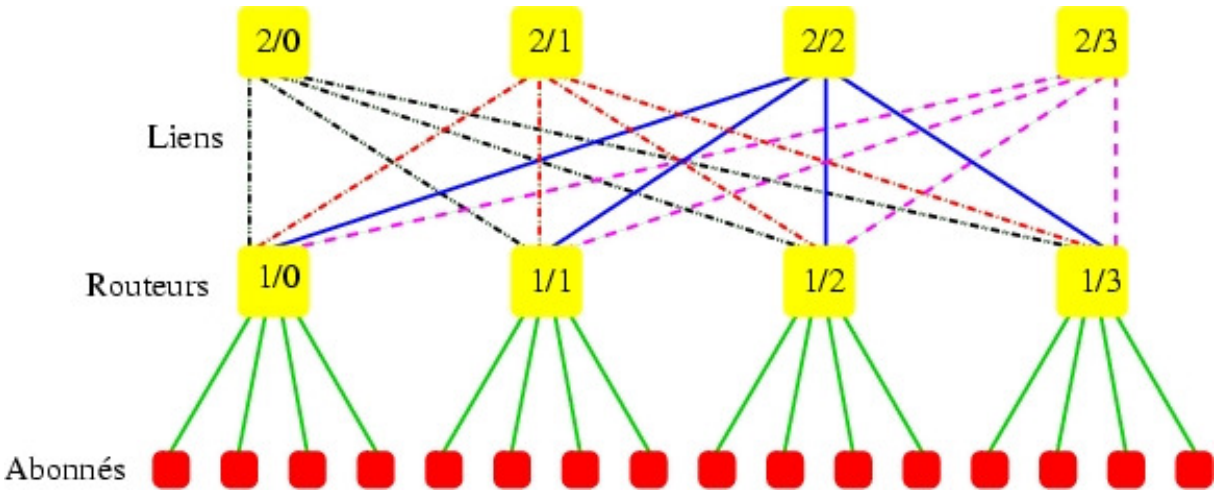

Figure 17 : Sous-réseaux dans un micro-réseau SPIN à 16 ports

2 Acheminement dans l'ordre chronologique

Nous allons montrer dans les sections suivantes que le routeur *RSPIN* propose deux mécanismes permettant d'augmenter la bande passante effective du micro-réseau *SPIN*, mais ayant pour conséquence la perte de la garantie d'acheminement des paquets dans l'ordre chronologique :

- le routage non déterministe des paquets montants,
- la présence des queues centrales.

Or l'utilisation a mis en évidence que la propriété d'acheminement des paquets dans l'ordre chronologique est parfois indispensable pour faciliter la mise en œuvre de certains protocoles de plus haut niveau, tels que les protocoles de reprise sur erreur ou les protocoles de consistance mémoire. En effet, une succession d'écritures et de lectures doit se faire dans l'ordre pour faciliter la consistance mémoire. De même, lorsqu'un paquet est corrompu ou se perd dans le micro-réseau *SPIN*, il doit être réémis. Si la *réponse* à une *requête* n'arrive pas à l'initiateur au bout d'un certain temps, l'initiateur doit émettre la *requête* à nouveau ; mais il doit d'abord s'assurer que le paquet *requête* ou le paquet *réponse* concerné s'est bien perdu ou a bien été corrompu. Une façon de faire est d'envoyer un paquet *balais* qui va suivre exactement le même chemin. Si le paquet *balais* revient à l'initiateur avant le paquet *réponse*, le paquet *requête* ou le paquet *réponse* a bien été corrompu ou perdu dans le micro-réseau *SPIN*. Evidemment, cette façon de faire n'est possible que si les paquets sont acheminés dans l'ordre chronologique.

Par conséquent, le routeur *RSPIN* propose une option garantissant l'acheminement dans l'ordre chronologique sur une base paquet par paquet : si le bit numéro 8 du paquet est à la valeur logique '1', la partie adaptative du routage est désactivée : lorsqu'un paquet veut monter, il n'a plus le choix entre les 4 chemins équivalents du haut. Au contraire, le bloc *icb*

va le forcer à aller vers le chemin montant qui lui correspond (Figure 18). De plus, toutes les queues centrales sont désactivées. Dans cette configuration, tous les paquets qui viennent d'un composant A et vont vers un composant B prennent exactement le même chemin et ne peuvent pas se dépasser. Ils vont donc être acheminés dans l'ordre chronologique.

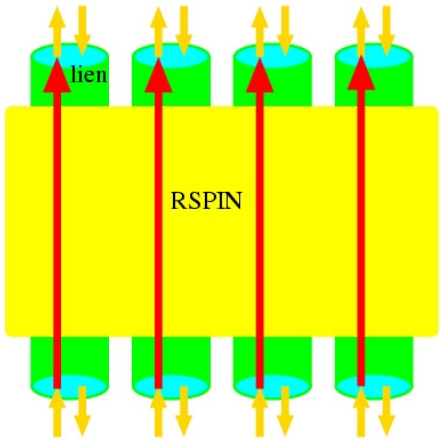

Figure 18 : Configuration du routage pour acheminer les paquets dans l'ordre chronologique

3 L'architecture interne du routeur RSPIN

Le routeur *RSPIN* est un *crossbar* possédant 8 ports bidirectionnels (Figure 19). Chaque port peut être désactivé s'il n'est pas utilisé, grâce à 8 registres de configuration. Le routeur *RSPIN* est composé d'une partie régulière de chemin de données contenant des files d'attente de type *FIFO* et d'un *crossbar*. Il possède également une partie contrôle qui contient les automates responsables du routage. Compte tenu du très grand nombre de nappes de fils de 36 bits, les considérations topologiques (c'est-à-dire les contraintes de placement relatif des différents modules composant le routeur *RSPIN*) sont très importantes. C'est pourquoi la micro-architecture décrite ici fait référence explicitement à la topologie.

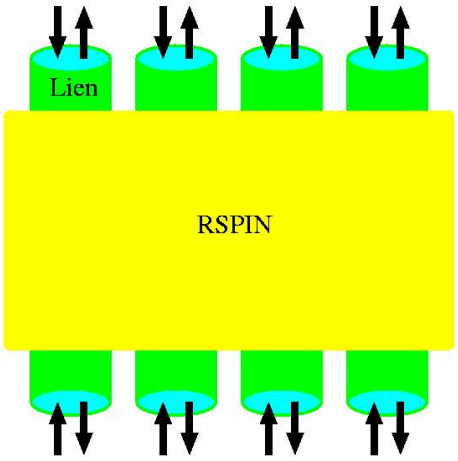

Figure 19 : Vue externe du routeur RSPIN

Comme le montre la Figure 22, l'architecture se décompose en 8 modules correspondant aux 8 ports d'entrée/sortie (*u0*, *u1*, *u2*, *u3*, *d0*, *d1*, *d2*, *d3*) et de deux modules correspondant aux queues centrales (*qu* et *qd*).

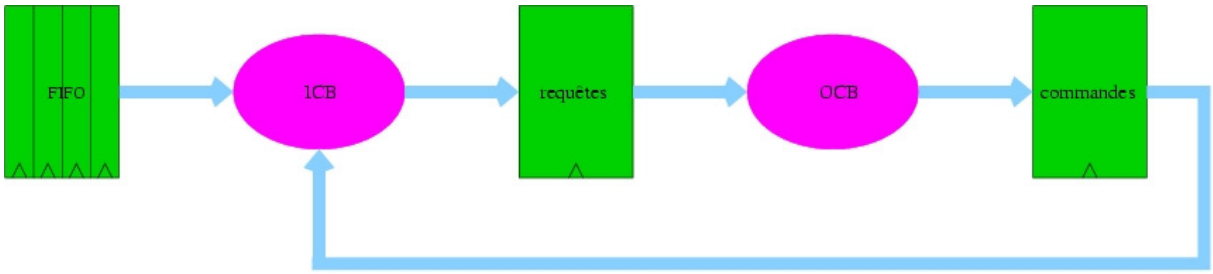

Figure 20 : Port d'entrée/sortie

Comme le montre la Figure 20, chaque port d'entrée/sortie contient :

- une *FIFO* d'entrée d'une profondeur 4 mots,
- un bloc de contrôle d'entrée *icb* (*Input Control Bloc*),
- un bloc de contrôle de sortie *ocb* (*Output Control Bloc*),
- des multiplexeurs, inverseurs, portes *NANDs* et *tristates* constituants le *crossbar* de données et de contrôle.

3.1 Les tampons d'entrées

Les tampons d'entrées sont des *FIFO* de 4 mots de 36 bits.
Ces *FIFO* ont une double fonction : elles sont utilisées par le mécanisme de contrôle de flux à crédits et ont, par ailleurs, une fonction de stockage en cas de contention. En effet, un paquet qui se trouve bloqué à l'entrée d'un routeur *RSPIN* (parce que le port de sortie qu'il souhaite emprunter est occupé) peut être stocké provisoirement dans la *FIFO* et ainsi libérer le lien d'entrée et le routeur *RSPIN* en amont.

Toutefois, si un paquet est bloqué à l'entrée d'un routeur *RSPIN*, les paquets qui sont situés derrière lui le sont également, même si les sorties qu'ils souhaitent emprunter sont libres (phénomène appelé *Head of Line Blocking*). Ce problème est résolu par la présence des deux queues centrales.

La profondeur de 4 mots représente un compromis : si l'on augmente la profondeur des *FIFOs*, le micro-réseau *SPIN* résiste mieux aux phénomènes de contention, mais la surface occupée par les *FIFOs* est une part importante de la surface totale. C'est la profondeur optimale pour les longueurs de paquet qui ne dépassent pas 16 mots.

3.2 Les queues centrales

Les queues centrales, qui constituent le *output buffering*, sont des *FIFO* de 18 mots de 36 bits. Elles servent à limiter les effets des contentions dans le micro-réseau *SPIN*.
Les contentions concernent exclusivement les paquets descendants. En effet, dans un *arbre quaternaire élargi*, un paquet montant dispose de 4 ports de sortie équivalents pour atteindre son but. Il n'y a donc jamais de contention pour les paquets montants. En revanche, un paquet descendant ne dispose que d'un seul port de sortie pour atteindre sa destination. Par conséquent, les queues centrales ne sont utilisées que pour les paquets descendants : on affecte une queue centrale pour les paquets descendants qui viennent du haut de l'*arbre quaternaire élargi* et une queue centrale pour les paquets descendants qui viennent du bas. La profondeur de 18 mots a été choisie afin de pouvoir contenir un paquet en entier. Nous verrons que la présence de ces queues centrales n'augmente en aucune manière la surface du routeur *RSPIN,* puisque celle-ci est définie par l'encombrement des nappes de fils du *crossbar*. Des simulations seront effectuées pour mettre en évidence l'effet de la présence de ces queues centrales sur le pourcentage de la charge maximale auquel intervient la saturation du micro-réseau *SPIN*.

3.3 Le crossbar

Pour prendre en compte les 2 queues centrales (*qu* et *qd*), en plus des 8 *FIFO* correspondant aux ports d'entrée (*u0*, *u1*, *u2*, *u3, d0*, *d1*, *d2* et *d3*), le routeur *RSPIN* contient un *crossbar* 10 x 10 . Les entrées des queues centrales sont équivalentes à des ports de sortie du routeur *RSPIN* ; les sorties des queues centrales sont équivalentes à des entrées du routeur *RSPIN*.
Il s'agit cependant d'un *crossbar* incomplet, car la stratégie de routage dans le micro-réseau *SPIN* introduit quelques restrictions sur les chemins que peuvent emprunter les paquets :

- les paquets qui viennent du haut ne peuvent pas aller dans la queue du bas (*qd*),
- les paquets qui viennent du bas ne peuvent pas aller dans la queue du haut (*qu*),
- les paquets qui viennent des queues centrales (*qd* et *qu*) ne peuvent que descendre,
- les paquets qui viennent du haut ne peuvent que descendre ou aller dans la queue du haut (*qu*) ; ils ne peuvent pas aller vers le haut.

Ce qui nous donne le *crossbar* partiel de la Figure 21.

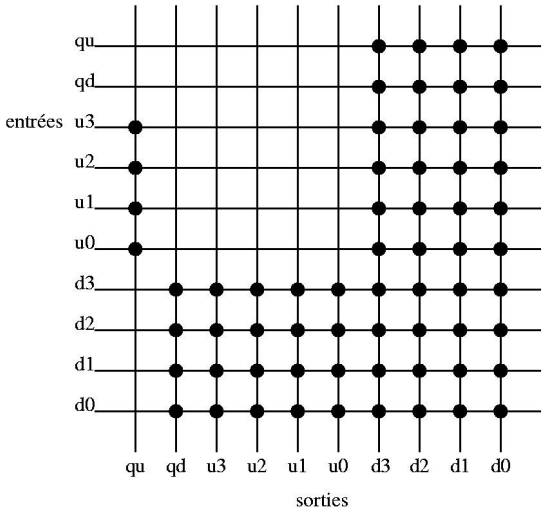

Figure 21 : Les chemins possibles dans le crossbar

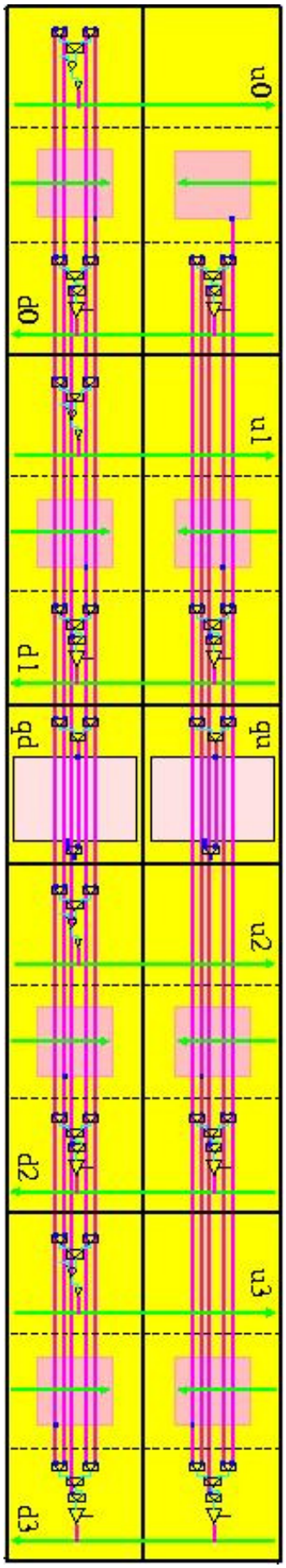

Figure 22 : Chemin de données du routeur RSPIN

Le *crossbar* est réalisé par un ensemble de multiplexeurs et de portes trois-états (*tristate*). Il est conçu de telle sorte que s'il n'y a aucune donnée valable sur les sorties, ces dernières seront mises par défaut à la valeur logique '0' par l'intermédiaire de portes *AND*, ce qui limite la consommation. Le *crossbar* pour les liens montants est donné à la Figure 23, celui pour les liens descendants à la Figure 25 et celui pour les queues centrales à la Figure 24.

Appelé aussi *crossbar* inverse, le *crossbar* de contrôle est un chemin qui s'ouvre en même temps que le chemin de données, et qui relie la même entrée et la même sortie que le *crossbar* de données mais dans le sens inverse. Le *crossbar* de contrôle véhicule les deux bits de contrôle de flux, c'est-à-dire le bit *r_ok* de la *FIFO* d'entrée et le bit de lecture *move*. Ce *crossbar* est constitué principalement de portes trois-états (*tristate*).

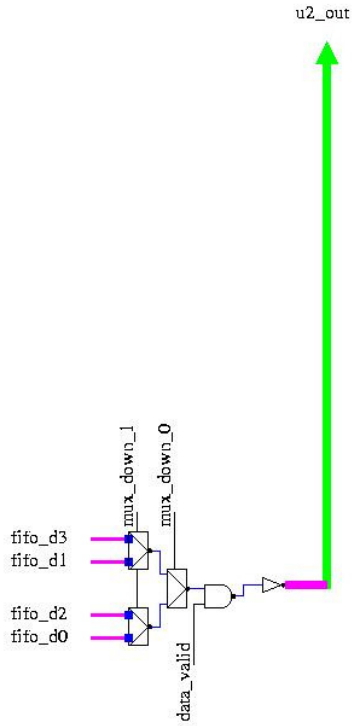

Figure 23 : Lien montant

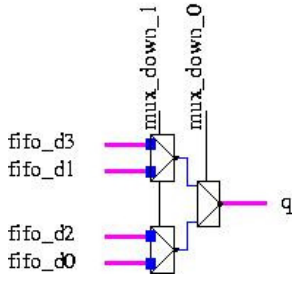

Figure 24 : Lien dans les queues centrales

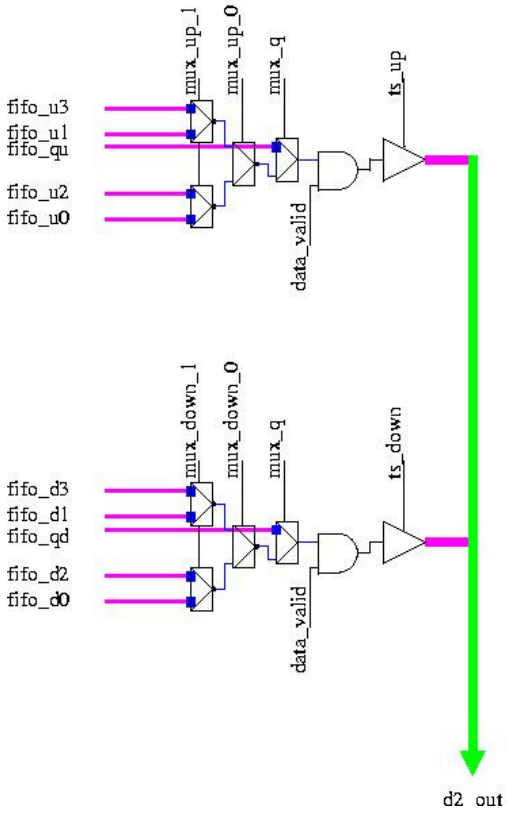

Figure 25 : Lien descendant

3.4 Bloc de contrôle d'entrée : génération des requêtes

Le bloc de contrôle d'entrée (Figure 20), dénommé également *icb* (*input control block*), décode l'en-tête du paquet pour en extraire l'adresse de destination. En fonction de cette adresse et de la position du routeur *RSPIN* dans le micro-réseau *SPIN*, le bloc *icb* envoie une requête vers le port de sortie sélectionné. Les requêtes sont mémorisées dans les registres de requêtes.

Puisqu'il existe plusieurs ports de sortie équivalents (dans le cas d'un paquet montant), le bloc *icb* prend en compte l'état des différents ports de sortie, qui peuvent être désactivés ou alloués à un autre port d'entrée. Lorsqu'un paquet doit descendre dans le micro-réseau *SPIN* et emprunter les sorties descendantes du routeur *RSPIN*, il n'a le choix que d'emprunter la sortie qui le rapproche le plus de sa destination finale car le routage des paquets descendants dans un *arbre quaternaire élargi* est déterministe. Si cette sortie est occupée, le paquet peut faire un tour dans la queue centrale afin de ne pas encombrer le micro-réseau *SPIN* et créer des contentions en amont. Sur la Figure 26 le paquet bleu (—) veut emprunter le même chemin descendant que le paquet violet (----). En attendant que le chemin se libère, le routeur *RSPIN* va stocker le paquet bleu (—) dans la queue centrale. Un paquet ne peut pas passer deux fois dans une queue centrale car un paquet sortant d'une queue centrale envoie une requête vers la sortie descendante désirée, que cette sortie soit libre ou non.

Ce mécanisme a pour conséquence la perte de la propriété d'acheminement dans l'ordre chronologique.

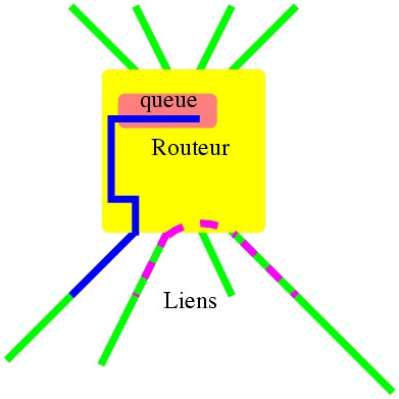

Figure 26 : Paquet devant patienter dans la queue centrale

Seuls les paquets venant du bas peuvent emprunter les sorties montantes du routeur *RSPIN*. Lorsque le bloc *icb* doit envoyer une requête vers une sortie du haut, elle l'envoi vers l'une des 4 sorties équivalentes. Le choix est pseudo-aléatoire indépendamment de l'état (alloué ou non) des ports de sortie. Il est renouvelé à chaque cycle de requête (tant que la requête n'est pas satisfaite) pour éviter d'attendre la libération éventuelle de la sortie choisie, alors que d'autres sorties se sont libérées. Le routage des paquets montants dans le micro-réseau *SPIN* est donc un routage adaptatif, qui permet de limiter les contentions ainsi que la latence.
Comme nous l'avons vu au paragraphe 2, il est possible d'interdire ce comportement adaptatif et l'utilisation de la queue centrale afin de garantir l'acheminement des paquets dans l'ordre chronologique.

Le pseudo-code ci-dessous résume la stratégie de génération des requêtes :

```
LOOP :      Attendre un début de paquet
            Extraire l'adresse de destination
            Si le paquet doit aller vers le haut
CHOICE :            Choisir un port de sortie montant
                    Envoyer une requête vers le port choisi
                    Si requête satisfaite
                            Retour à LOOP
                    Sinon
                            Retour à CHOICE
            Si le paquet doit aller vers le bas
                    Si la sortie est occupée
                            Envoyer une requête vers la queue centrale
                    Sinon
                            Envoyer une requête vers le port de sortie adéquat
                    Retour à LOOP
```

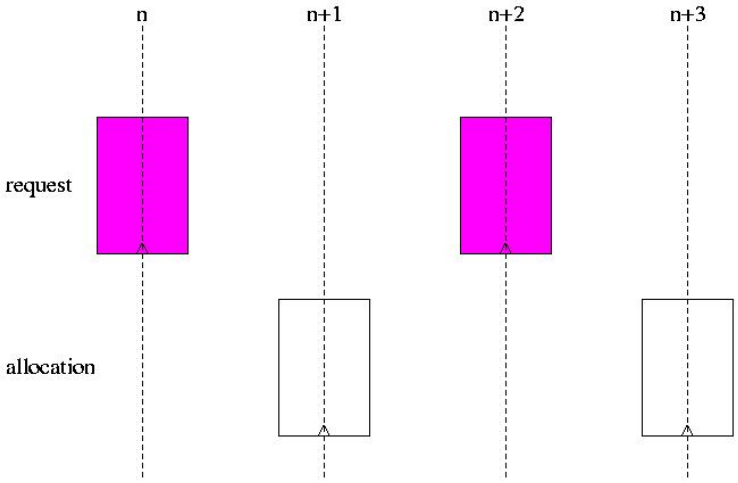

Figure 27 : Chronogramme d'échantillonnage des registres

3.5 Le bloc de contrôle de sortie : allocation

Le bloc de contrôle de sortie (Figure 20), dénommé également *ocb* (*Output Control Block*), d'un port *i* contient principalement l'automate d'état qui contrôle l'allocation du port de sortie *i* à l'un des ports d'entrée en fonction des requêtes qui lui sont envoyées (c'est-à-dire la configuration dynamique du *crossbar*). S'il n'y a pas de requêtes, il émet la valeur logique '0' sur le port de sortie; s'il y en a plusieurs, il en choisi une selon un ordre de priorité décrit ci-dessous.

Dans le cas où plusieurs requêtes venant de plusieurs blocs de contrôle d'entrée sont présentes, le choix s'effectue en respectant la priorité suivante : les requêtes qui viennent de la queue centrale du haut, les requêtes qui viennent de la queue centrale du bas, les requêtes qui viennent du haut et enfin les requêtes qui viennent du bas. S'il y a plusieurs requêtes qui viennent du bas ou plusieurs requêtes qui viennent du haut, le choix entre les requêtes se fait par une priorité tournante (*round robin*) entre les requêtes du bas, ou entre les requêtes du haut. Cette méthode garantit l'absence de famine.
Le chemin reste ouvert jusqu'à ce que le bloc *ocb* détecte le passage du témoin de fin de paquet (*ep*).
Le pseudo-code ci-dessous résume la stratégie d'allocation des chemins :

```
LOOP : Si requête
            Si plusieurs requêtes
                  Choisir une requête à satisfaire
            Satisfaire la requête
            Envoyer les commandes pour l'ouverture du chemin
            Attendre la fin du paquet
            Retour à LOOP
       Sinon
            Envoyer les commandes pour fermer le chemin
            Retour à LOOP
```

3.6 Allocation pipelinée en deux cycles et latence

Les requêtes calculées par les blocs *icb* sont enregistrées dans les registres de requêtes (Figure 20). L'état d'allocation des différentes sorties (état du *crossbar*) est enregistré dans les registres d'allocation (Figure 20) qui se trouvent dans les blocs *ocb*.
Pour éviter qu'une même requête soit envoyée à deux sorties différentes pour un même paquet, les registres de requête sont échantillonnés sur les cycles impairs et les registres d'allocation sur les cycles pairs, ce qui donne le chronogramme de la Figure 27 (où *n* est un nombre impair) :
Grâce à la technique de routage de type *wormhole* et à la stratégie d'allocation pipelinée en deux cycles décrite ci-dessus, le temps de traversée moyen d'un routeur *RSPIN* est de 2,5 cycles. En effet, si le paquet se présente dans le tampon d'entrée du routeur *RSPIN* à un cycle pair, le temps de traversée est de deux cycles (Figure 28). Si le paquet se présente dans le tampon d'entrée du routeur *RSPIN* à un cycle impair, alors le temps de traversée est de 3 cycles (Figure 29).

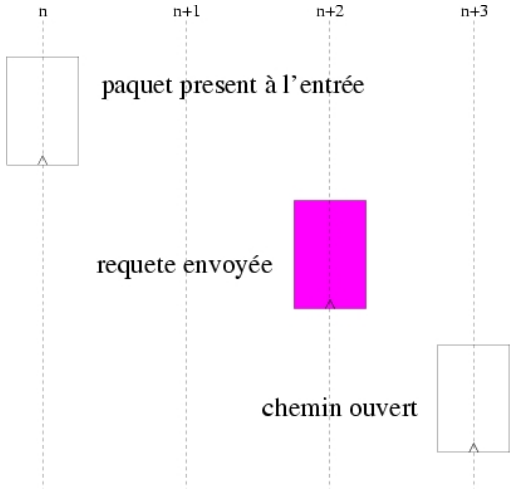

Figure 28 : Latence de deux cycles (n est un nombre impair)

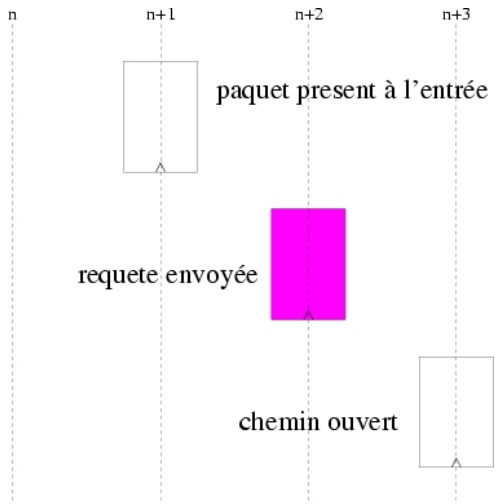

Figure 29 : Latence de 3 cycles (n est un nombre impair)

4 Conclusion

Ce chapitre détaille la micro-architecture du routeur *RSPIN*. C'est un *crossbar* constitué d'un chemin de données et d'une partie contrôle. Il possède 8 ports bidirectionnels. Nous avons mis des *FIFO*s à chaque entrée et deux queues centrales en alternative aux sorties pour diminuer la contention. Les *FIFO*s d'entrée sont également utilisées par le contrôle de flux à crédits pour assurer qu'aucun paquet ne se perde dans le micro-réseau *SPIN*. Le chemin de données du routeur *RSPIN* est un *crossbar* 10 x 10 composé de multiplexeurs et portes trois-états (*tristate*). La partie contrôle est constituée d'un bloc de contrôle d'entrée (*icb*) et d'un bloc de contrôle de sortie (*ocb*). Le bloc *icb* analyse l'en-tête du paquet et envoie une requête vers la sortie adéquate. Le routage des paquets montants est adaptatif, alors que celui des paquets descendants est déterministe. Le bloc *ocb* choisit une requête et envoie les commandes nécessaires vers les *crossbars* de données et de contrôles pour ouvrir un chemin entre l'entrée dont la requête est choisie et la sortie où il se trouve. Le choix de la requête à satisfaire est faite selon un ordre de priorité de type *round robin*. A vide, le temps de traversée moyen d'un routeur *RSPIN* est de 2,5 cycles. Lorsque l'on se trouve dans un espace d'adressage partagé, le router *RSPIN* peut séparer les ressources utilisées par les paquets de type *requête* et les paquets de type *réponse*. Enfin, le routeur *RSPIN* permet l'acheminement des paquets dans l'ordre chronologique.

Chapitre 6 *Test intégré*

Puisque l'objectif de notre travail est d'étudier la faisabilité matérielle d'un micro-réseau *SPIN*, nous devions prendre en compte le test de fabrication. Nous avons donc défini un mécanisme d'auto-test pour le routeur *RSPIN*.

1 Cahier des charges du test

Le mécanisme d'auto-test interne du micro-réseau *SPIN* doit tenir compte des spécificités de ce dernier. Il a deux objectifs principaux :

- le *crossbar* de données qui est la principale source de complexité du routeur *RSPIN*. En effet, ce *crossbar* offre des dizaines de milliers de configurations de connexions possibles ;
- les nappes de fils d'interconnexion entre les routeurs *RSPIN* qui sont sensibles au phénomène de diaphonie dans les procédés fortement submicroniques.

Pour atteindre le second objectif, il est nécessaire de mettre en œuvre une technique d'auto-test à la fréquence nominale (*test at speed*), seule technique qui peut permettre de détecter les fautes temporelles liées à la diaphonie. Nous avons implémenté dans le micro-réseau *SPIN* deux mécanismes d'auto-test : un auto-test interne du routeur *RSPIN* (dit test *bist*) et un auto-test externe du micro-réseau *SPIN* (dit test *eulérien*) dont le principe avait été proposé dans la thèse de Pierre Guerrier [[1]]. Le micro-réseau *SPIN* possède donc 4 modes de fonctionnement. Outre le mode *fonctionnel*, il y a 3 modes de tests : le mode *scan*, le mode *eulérien* et le mode *bist*.
Le mode *scan* est exécuté à la fréquence du matériel de test.

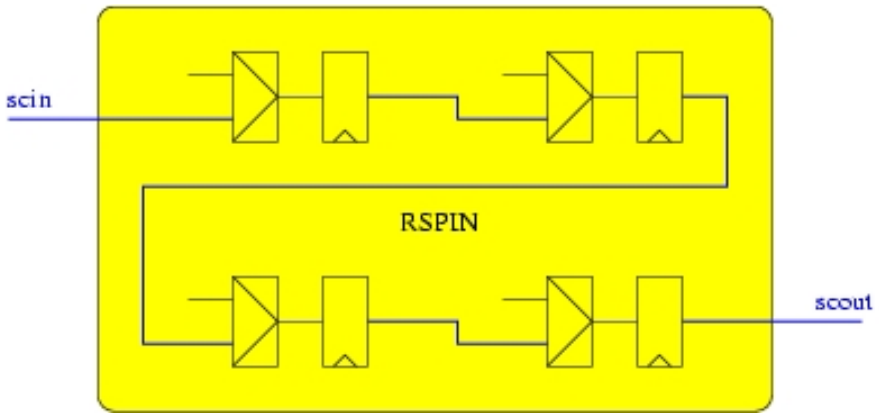

Figure 30 : Scan-path dans le routeur RSPIN

2 Le scan-path

Le mode *scan* est la méthode classique consistant à chaîner l'ensemble des registres de chaque routeur *RSPIN* par ajout d'un multiplexeur devant chaque registre appartenant à la partie contrôle. Le chemin de test, habituellement appelé *scan-path*, est utilisé pour initialiser les valeurs contenues dans les registres avant de lancer une séquence de *test at speed*, puis de lire les valeurs résultantes [36] (Figure 30).
Le matériel dédié au mode *scan* coûte 1,4 % de la surface totale du routeur *RSPIN*.

3 L'auto-test interne

L'auto-test interne (mode *bist*) doit couvrir le test du *crossbar* de données, du *crossbar* de contrôle et de la logique de contrôle.
Il consiste à faire circuler des paquets contenant un seul mot dans le routeur *RSPIN*. Chaque paquet va circuler dans un routeur *RSPIN* en passant par tous les chemins possibles du *crossbar*, c'est-à-dire que le paquet emprunte chaque chemin pouvant mener d'une entrée à une sortie du routeur *RSPIN*. Pour cela, nous avons rebouclé chaque sortie vers l'entrée du même port, de sorte que tous les chemins possibles soient mis en série (Figure 31).
Dans le mode *bist*, un routeur *RSPIN* prend donc en entrée la sortie du même port.

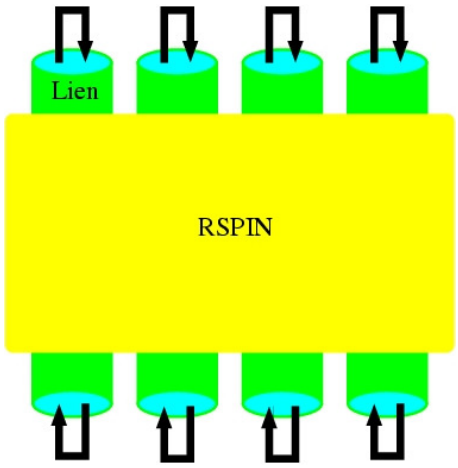

Figure 31 : Entrée/sortie du routeur RSPIN en mode bist

La génération de paquets se fait à l'aide d'un *LFSR* (*Linear Feedback Shift Register*) [37] [38] [39] (Figure 32). Dans chaque paquet, les 2 bits *bp* et *ep*, qui représentent le début et la fin du paquet, sont mis à la valeur logique '1' car la longueur du paquet est de un mot. Les 34 autres bits sont générés par le *LFSR*.

La récupération des paquets ayant emprunté tous les chemins possibles dans le routeur *RSPIN* se fait à l'aide d'un *MISR* (*Multiple Input Signature Register*) [40] [41] (Figure 32). Le *MISR* récupère tous les paquets et fournit une signature que l'on pourra comparer à une signature de référence en la récupérant à l'aide du chemin de tests.

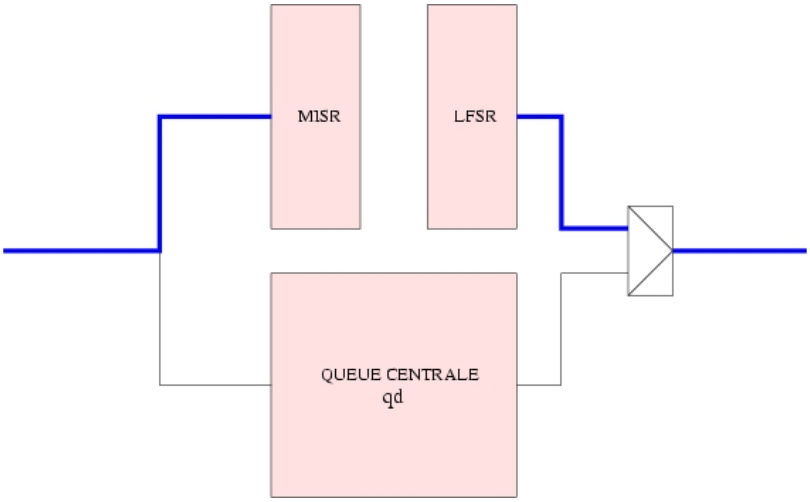

Figure 32 : Configuration en mode bist

En mode *bist*, chaque paquet doit parcourir tous les chemins possibles dans le routeur *RSPIN* avant d'être capturé par le *MISR* de données afin de générer une signature. Ceci permet de valider le fonctionnement du *crossbar* de données et de contrôle.

Les blocs de contrôle d'entrée (*icb*) et de sortie (*ocb*) sont court-circuités par une logique de contrôle centrale dédiée au mode *bist* qui force les *crossbars* de données et de contrôle afin que chaque paquet passe, effectivement, par tous les chemins possibles dans le *crossbar*. Les blocs *icb* et *ocb*, bien que court-circuités, réagissent normalement à chaque paquet qui se présente. C'est-à-dire que la logique de contrôle d'entrée (*icb*) envoie des requêtes, alors que la logique de contrôle de sortie (*ocb*) envoie des commandes d'allocation vers les *crossbars* de données et de contrôle. Ces commandes d'allocation ne sont pas prises en compte par les *crossbars* de données et de contrôle mais elles sont capturées dans un *MISR* de contrôle qui fournit également une signature à la fin de l'auto-test interne. Le *MISR* de contrôle est distribué sur le routeur *RSPIN*. La signature peut être récupérée à l'aide du chemin de tests et comparée à une signature de référence.

Un paquet passe par tous les chemins possibles dans le routeur *RSPIN* en 32 étapes dont l'ordre est représenté sur la Figure 33. Sur cette figure, chaque chemin a une entrée (verticale) et une sortie (horizontale). Ces 32 étapes sont répétées un grand nombre de fois pour améliorer la couverture de l'auto-test.

Les *FIFOs* ne sont que très partiellement testées par cet auto-test car il n'y a pas de contention et les *FIFOs* ne se remplissent donc pas. Elles sont testées plus complètement par l'auto-test externe.

Les largeurs des *MISR* de données et de contrôles et du *LFSR* sont respectivement de 38 bits, 36 bits et 33 bits. Il faut donc prendre 38 cycles pour les initialiser en parallèle, d'une façon déterministe, au début de l'auto-test interne. Les signatures récupérées après l'auto-test interne sont totalement déterminées et ne dépendent théoriquement que du nombre de cycles effectués par le micro-réseau *SPIN* en mode *bist*.
L'auto-test interne coûte 6,83 % de la surface totale du routeur *RSPIN*.

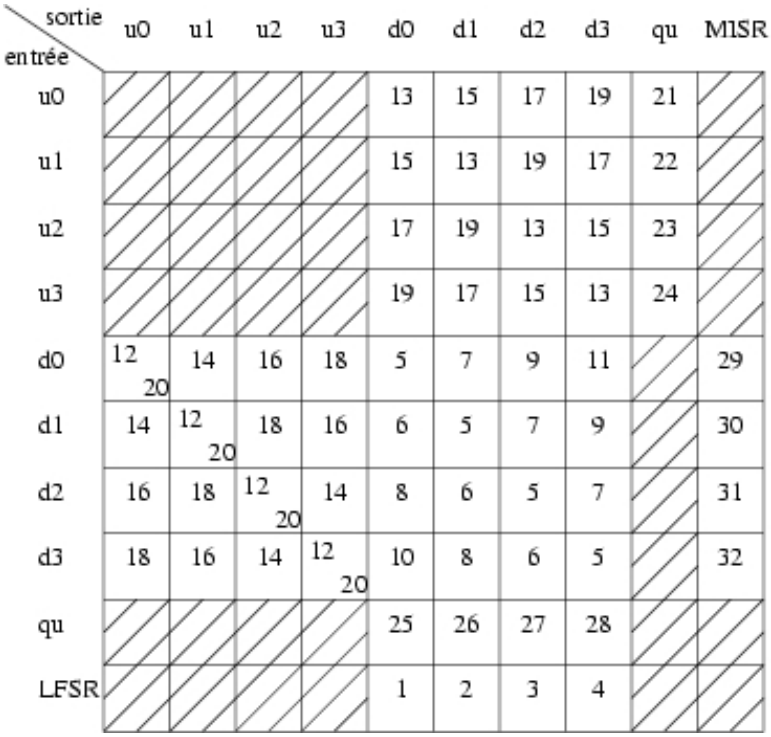

Figure 33 : Ordre des connexions établies dans le routeur RSPIN en mode bist

4 L'auto-test externe

L'auto-test interne, ou mode *bist*, permet de tester les composants internes du routeur *RSPIN*. L'auto-test externe, ou mode *eulérien*, a été conçu pour vérifier les fils d'interconnexion entre les routeurs *RSPIN* du point de vue des délais de propagation, ainsi que les *FIFOs* contenues dans les routeurs *RSPIN*. Dans ce mode, un certain nombre de paquets sont envoyés dans le micro-réseau *SPIN*. Ce dernier est configuré de telle sorte que chaque paquet passe par tous ses liens avant d'être capturé dans un *MISR* de signature. La validation des fils d'interconnexion entre les routeurs *RSPIN* se fait en comparant la signature obtenue avec la signature de référence obtenue par simulation avant fabrication.
Le passage de chaque paquet par tous les liens est assuré car les paquets suivent un circuit eulérien à travers le micro-réseau *SPIN*. Le micro-réseau *SPIN* peut être représenté par un graphe orienté dont les liens représentent les arêtes et les routeurs *RSPIN* représentent les sommets. Les routeurs *RSPIN* permettent d'établir des connexions entre les liens suivant la valeur de l'adresse destination contenue dans l'en-tête du paquet. Les arêtes correspondantes forment un chemin dans le graphe. Si le *wrapper* de l'abonné destinataire d'un paquet modifie cette adresse et ré-injecte le paquet dans le micro-réseau *SPIN*, il est possible de réaliser, en mettant bout à bout ces chemins, un circuit eulérien constitué de chemins couvrant toutes les arêtes du graphe. La topologie du micro-réseau *SPIN* garantit qu'un tel chemin existe mais il dépend évidemment du nombre de ports du micro-réseau *SPIN* [1]. Un exemple de circuit eulérien est donné en annexe (page 108) pour un micro-réseau *SPIN* à 32 ports. L'auto-test externe ne coûte rien en surface dans les routeurs *RSPIN* puisque ceux-ci sont en mode de fonctionnement normal lors de cet auto-test. La logique de test est située dans les *wrappers*.
En utilisant le mécanisme de rétro-propagation de la contention, on peut obliger les *FIFOs* à se remplir de façon à améliorer leurs taux de couverture.

5 Conclusion

Le routeur *RSPIN* se prête mal aux méthodes de test habituellement utilisées dans le domaine de la microélectronique. C'est pour cette raison que nous avons intégré dans le routeur *RSPIN* un mécanisme d'*auto-test* interne, qui utilise un *scan-path*, ainsi que deux *MISRs* et un *LFSR*. Ce matériel représente 8.23 % de la surface totale du routeur *RSPIN*, mais nous verrons au Chapitre 7, qu'il ne coûte rien car la surface du routeur *RSPIN* est définie par l'encombrement des nappes de fils d'entrée/sortie.

Chapitre 7 *Réalisation physique du routeur RSPIN*

Dans ce chapitre, nous décrivons précisément la méthode suivie pour obtenir une macro-cellule synchrone optimisée pour le routeur *RSPIN*.

1 Modèle VHDL

Le modèle *VHDL* du routeur *RSPIN* a été conçu, simulé, synthétisé et validé à l'aide de la chaîne de conception *SYNOPSYS*, selon le flot de conception de la Figure 34.

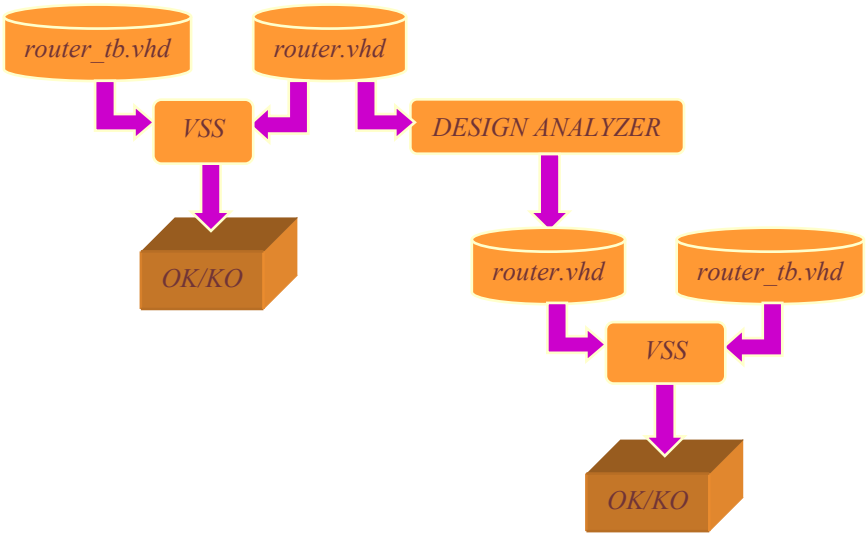

Figure 34 : Le flot de conception du routeur RSPIN

Le modèle *VHDL* du routeur *RSPIN* est une hiérarchie à 3 niveaux qui contient les sous-blocs suivants (Figure 35) :

- les blocs du haut (*u0, u1, u2* et *u3*) contiennent chacun : une *FIFO* d'entrée du haut, un lien descendant (*link*) contenant des multiplexeurs et des *tristates*, un bloc de contrôle d'entrée (*icb*), un bloc de contrôle de sortie (*ocb*) et enfin un bloc divers (*x*) (Figure 36) ;
- les blocs du bas (*d0, d1, d2* et *d3*) contiennent chacun : une *FIFO* d'entrée du bas, un lien descendant (*link_down*) contenant des multiplexeurs et des *tristates*, un lien montant (*link_up*) contenant des multiplexeurs et des *buffers*, un bloc de contrôle d'entrée (*icb*), un bloc de contrôle de sortie (*ocb*) et enfin un bloc divers (*x*) (Figure 37) ;

- les blocs queues centrales (*qu* et *qd*) contiennent chacun : une *FIFO* tenant lieu de queue centrale, un lien vers la queue centrale (*link*) contenant des multiplexeurs, un bloc de contrôle d'entrée (*icb*), un bloc de contrôle de sortie (*ocb*) et enfin un bloc divers (*x*) (Figure 36).

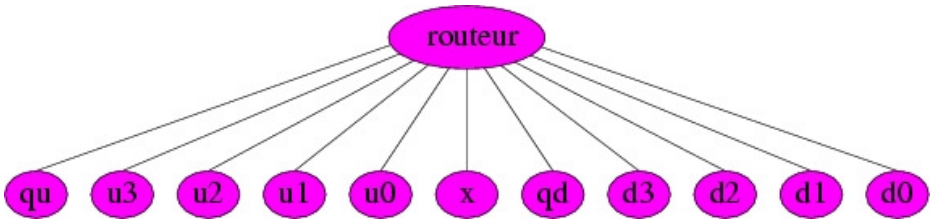

Figure 35 : Niveau le plus haut dans la hiérarchie

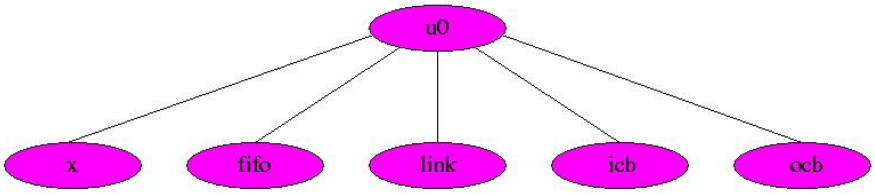

Figure 36 : Composition des blocs du haut et des queues centrales

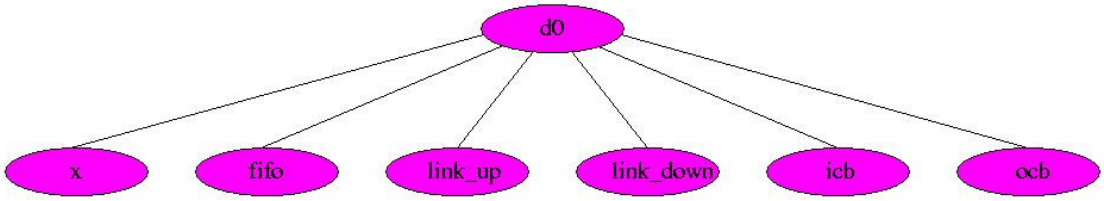

Figure 37 : Composition des blocs du bas

2 Le placement/routage

2.1 Principe de la méthode

Nous voulons contrôler le placement/routage pour conserver la régularité des chemins de données et éviter le foisonnement des fils de routage. Pour cela, nous avons mélanger le placement/routage manuel et le placement/routage automatique et avons effectuer le

placement/routage du routeur *RSPIN* en deux étapes. Le placement/routage du chemin de données a été optimisé et décrit de façon explicite au moyen du langage procédural de l'outil *GENLIB*. Ensuite, le placement/routage de la partie contrôle a été effectué de façon automatique avec l'outil *SILICON ENSEMBLE*.

2.2 Application sur le routeur RSPIN

Le routeur *RSPIN* a été conçu en utilisant le *layout* symbolique de la chaîne *ALLIANCE*. Cela nous permet d'être entièrement indépendant du processus de fabrication utilisé pour le mettre sur silicium.
Nous nous sommes appuyés sur les bibliothèques de cellules *SXLIB* et *DP_SXLIB* :

- *SXLIB* est une bibliothèque de cellules précaractérisées élémentaires (cellules un bit) comportant une soixantaine de cellules et destinées à la synthèse des parties contrôles,
- *DP_SXLIB* est une bibliothèque de macro-cellules précaractérisées (colonne de multiplexeurs, de *tristates*, ….) et destinées à la conception des chemins de données.

2.2.1 Le chemin de données

Le chemin de données ayant une topologie très régulière a été préplacé. En plus du préplacement de toutes les cellules de la partie régulière, tous les fils réalisant le *crossbar* et les nappes de fils de données ont été préroutés par le langage procédural de l'outil *GENLIB*.
La Figure 38 montre qu'à partir d'une description en langage *C*, l'outil *GENLIB* génère un fichier de préplacement et de préroutage au format *.ap* qui va être utilisé par la suite pour router le routeur *RSPIN* en entier. L'outil *GENLIB* fait partie de la chaîne *ALLIANCE*.

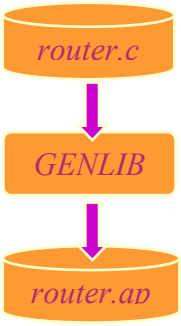

Figure 38 : Génération du fichier de préplacement et de préroutage

Nous avons choisi d'effectuer un placement explicite des opérateurs du chemin de données pour préserver la régularité topologique :

- les 4 *FIFO*s servant de tampons d'entrées pour les paquets qui viennent du haut (Figure 39) ;
- les 4 *FIFO*s servant de tampons d'entrées pour les paquets qui viennent du bas (Figure 39) ;
- les 2 *FIFO*s servant de queues centrales pour les paquets qui viennent du haut et du bas (Figure 39) ;
- toutes les cellules composant le *crossbar* de données (multiplexeurs, *tristates*, *buffers* et *NAND*) sur 36 bits (Figure 40) ;

- toutes les cellules (*tristates* et inverseurs) composant le *crossbar* de contrôle (Figure 41) ;
- toutes les cellules nécessaires au préroutage et les multiplexeurs nécessaires pour le test interne du routeur *RSPIN* (Figure 42). Les cellules nécessaires au préroutage sont les cellules qui ne font pas partie du chemin de données mais qui sont connectées à un fil du chemin de données. Ces cellules doivent être préplacées pour pouvoir entièrement prérouter le fil du chemin de données, car il est impossible de faire du préroutage partiel (le fil qui a commencé à être prérouté doit être entièrement prérouté).

Les signaux préroutés sont les nappes de fils régulières qui connectent les cellules du *crossbar* de données entre elles ou les nappes de fils constituant les connecteurs d'entrée ou de sortie du routeur *RSPIN*.

La stratégie de préroutage est donnée ci-dessous :

- les 16 nappes de fils verticales de 38 bits qui composent les 8 liens bidirectionnels sont préroutées au-dessus des cellules en métal 5. Comme les 8 nappes de fils de 38 bits d'entrées qui arrivent sur les *FIFO*s ne traversent pas le routeur *RSPIN* de haut en bas mais seulement la moitié du routeur *RSPIN*, elles sont mises l'une au-dessus de l'autre, et deux par deux pour gagner de la place (Figure 43) ;
- des segments horizontaux en métal 4 et des *VIA*s sont utilisés pour ramener les nappes de fils d'entrées et de sorties vers les cellules correspondantes (Figure 44) ;
- des segments horizontaux en métal 4 sont utilisés pour relier les sorties de *FIFO* et les entrées des multiplexeurs entre elles ;
- les cellules multiplexeurs et *tristates* ont la particularité de pouvoir se placer les unes sur les autres et de créer des colonnes de multiplexeurs ou de *tristates* avec un seul fil de commande vertical. Ces fils de commande sont préroutés en métal 3 ;
- les distributions d'horloge et *reset* sont préroutés en métal 4 au milieu du routeur *RSPIN* (Figure 49) ;
- des segments en métal 4 sont utilisés pour les rappels d'alimentation horizontaux et des segments en métal 3 pour les rappels d'alimentation verticaux (Figure 47).

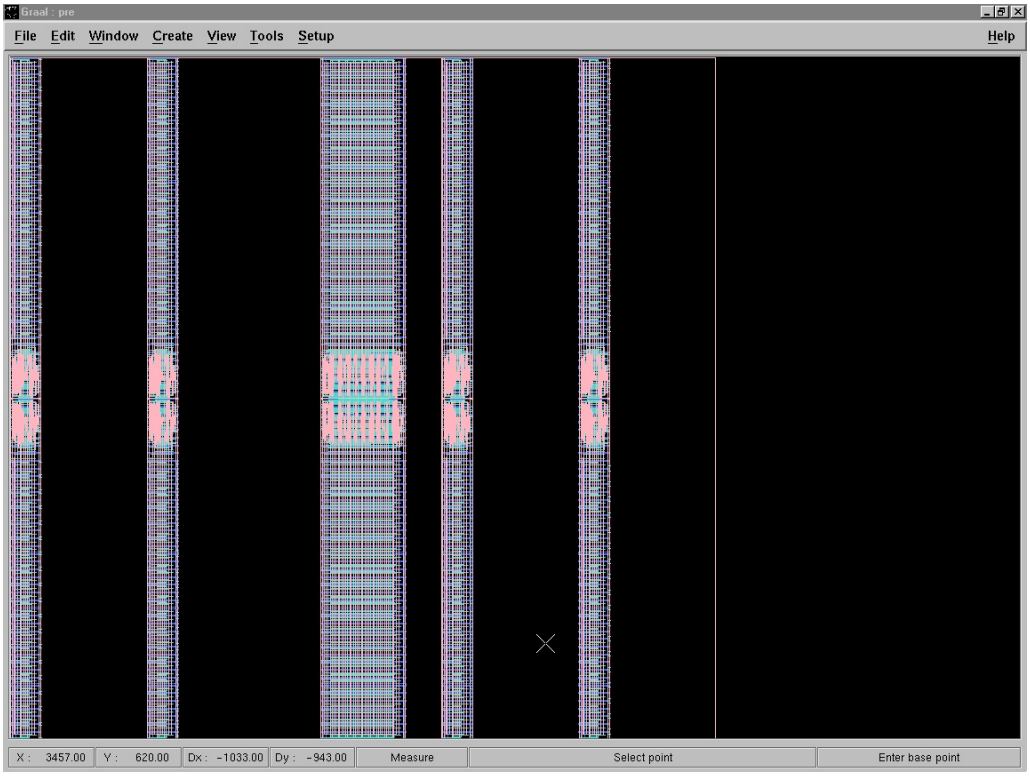

Figure 39 : Placement des queues centrales et des FIFOs

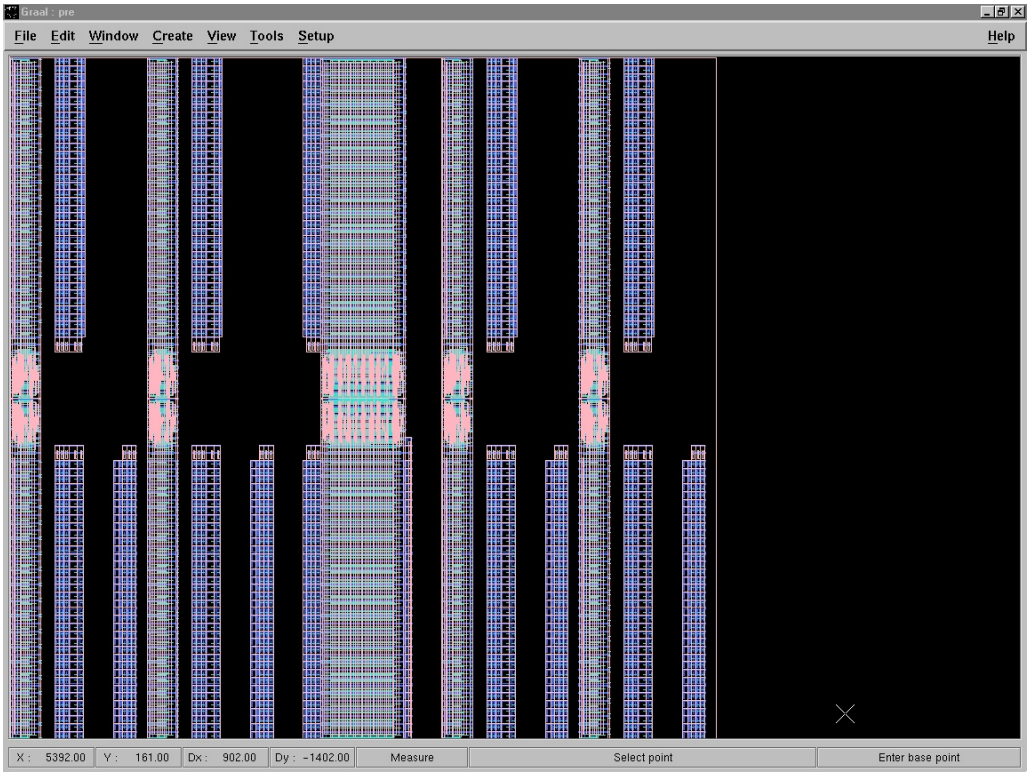

Figure 40 : Placement des cellules du crossbar de données

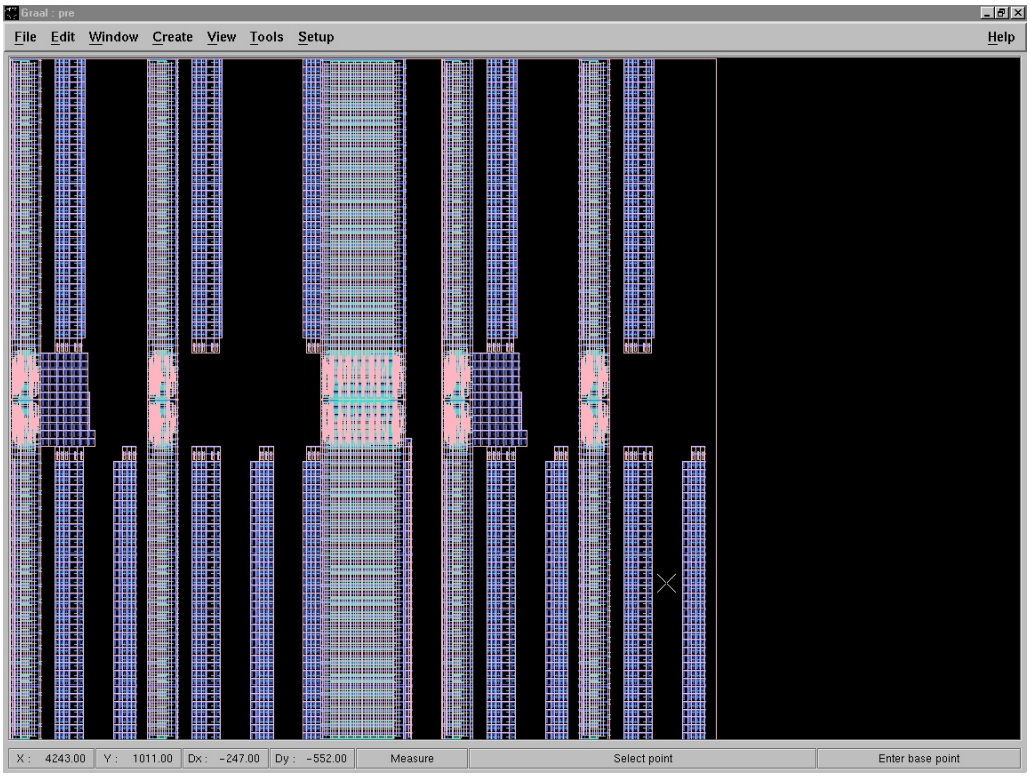

Figure 41 : Placement du crossbar de contrôle

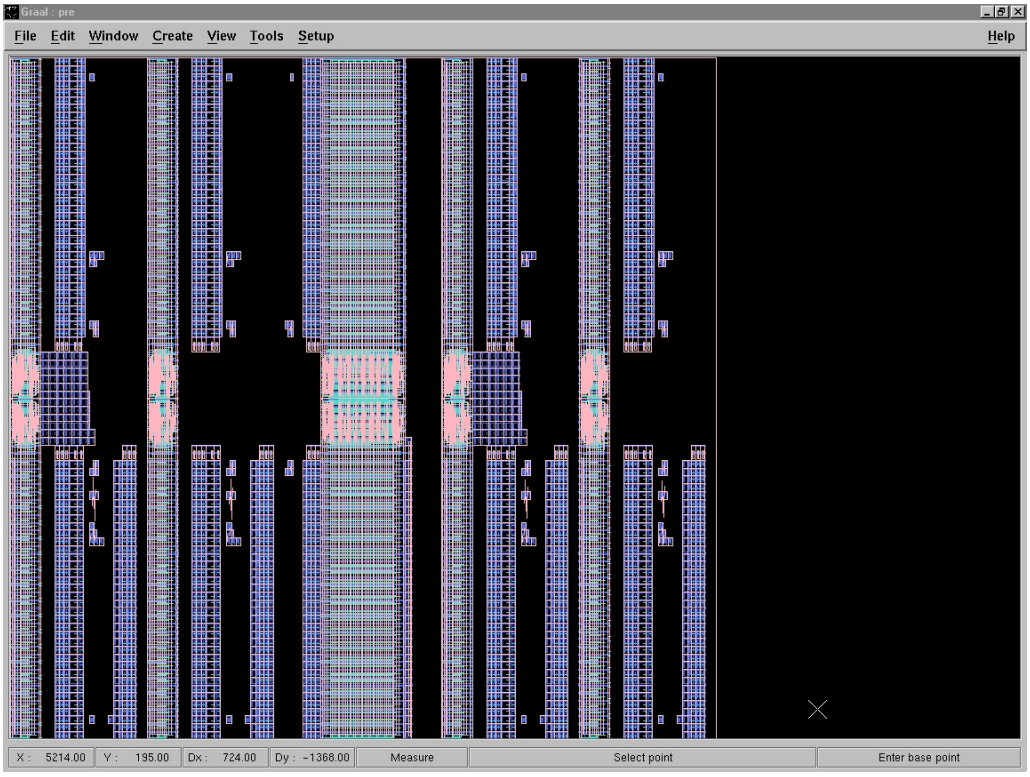

Figure 42 : Placement des diverses cellules nécessaires au préroutage

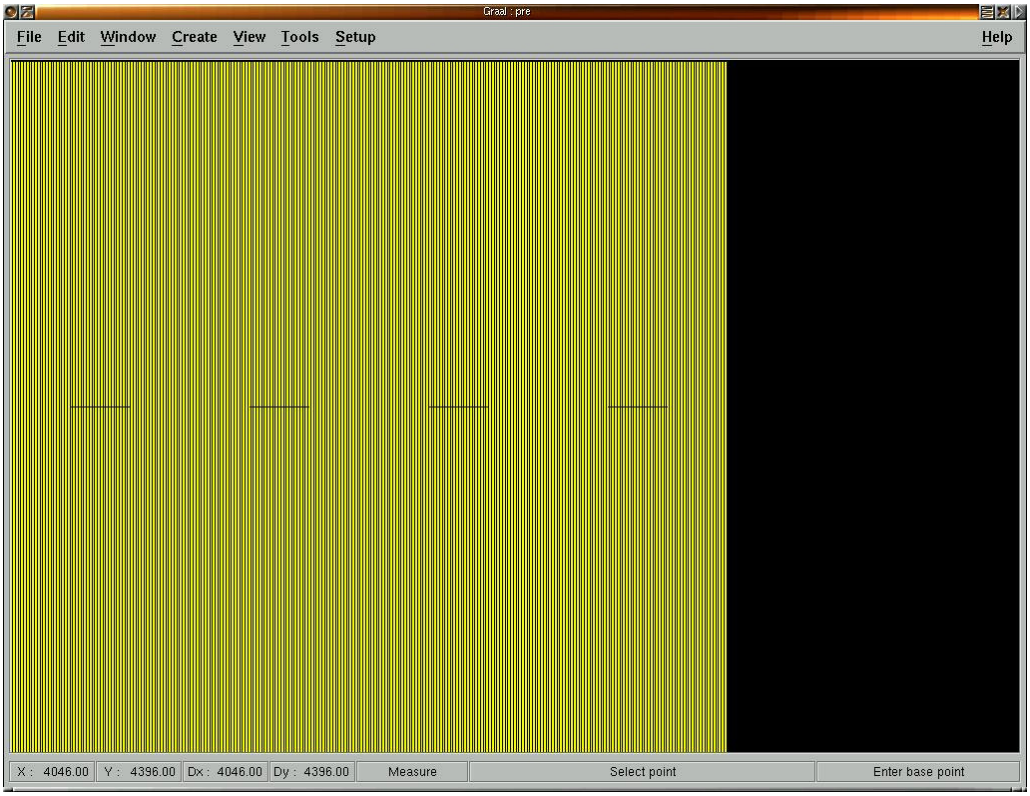

Figure 43 : Routage des nappes de fils d'entrées/sorties

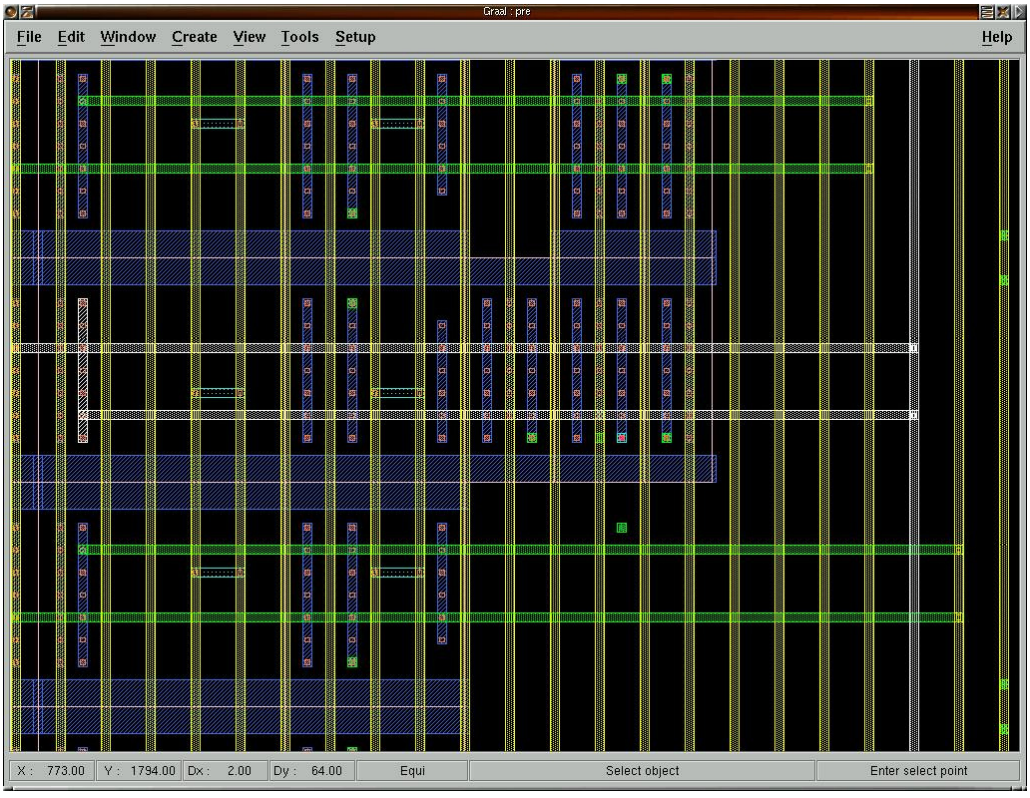

Figure 44 : Routage vers les cellules

2.2.2 Le placement automatique de la partie contrôle

Le fichier de préplacement et préroutage (*router.ap*) est associé à la *netlist* du routeur *RSPIN* par l'outil *A2DEF*. Cette association génère un seul fichier contenant toutes les informations utiles pour le placement/routage du routeur *RSPIN* dans un format *.def* acceptable par l'outil de placement/routage *SILICON ENSEMBLE* (Figure 45).

Le placement/routage des cellules de la partie contrôle se fait à l'aide de l'outil *SILICON ENSEMBLE*. Certaines directives de localité ont été fournies afin de regrouper ensemble toutes les cellules de chaque bloc (*qu*, *qd*, *u0*, *u1*, *u2*, *u3*, *d0*, *d1*, *d2*, *d3*) (Figure 45). Ces directives de localité, associées avec le fichier de préplacement et préroutage généré par l'outil *GENLIB*, permettent à l'outil *SILICON ENSEMBLE* de placer les cellules de la partie irrégulière dans les places laissées libres par les cellules de la partie régulière, sous les nappes de fils de 38 bits (Figure 46). Il utilise pour cela uniquement le métal 2 et le métal 3.

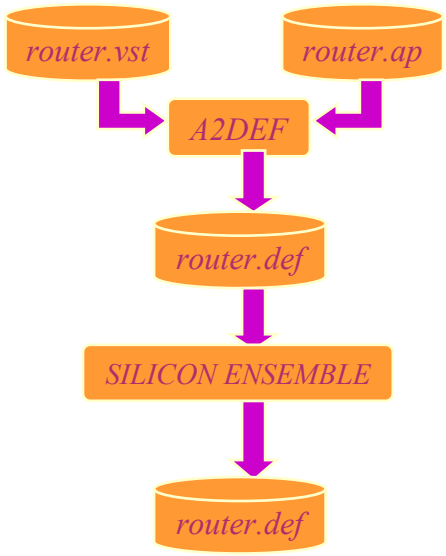

Figure 45 : Flot de conception pour le placement/routage

2.2.3 La distribution des alimentations

Le placement/routage explicite permet également de contrôler précisément le maillage des alimentations. Notre objectif est de garantir que les deux niveaux d'alimentation *vdd* et *vss* soient accessibles et soient les plus constants possibles sur toute la surface du routeur *RSPIN*. Le réseau d'alimentation doit théoriquement être une équipotentielle sur toute la surface du routeur *RSPIN*. En réalité, ce sont les résistances des fils qui sont la cause principale de la dégradation du niveau de l'alimentation. Le degré de dégradation va dépendre de la région où l'on se trouve dans le circuit. Nous cherchons donc à diminuer les chutes de tension sur les fils d'alimentation *vdd* et *vss*. Pour cela, nous avons conçu un réseau d'alimentation maillé (Figure 47) dont les rappels d'alimentation horizontaux sont routés en métal 1 et en métal 4 et les rappels d'alimentation verticaux sont routés en métal 3. Cette structure doit être un compromis. D'un coté, si la maille d'alimentation occupe trop de place, il n'y en aura plus assez pour le routage. De l'autre coté, s'il n'occupe pas assez de place, sa résistance sera trop élevée et la dégradation du niveau de l'alimentation sera trop grande.

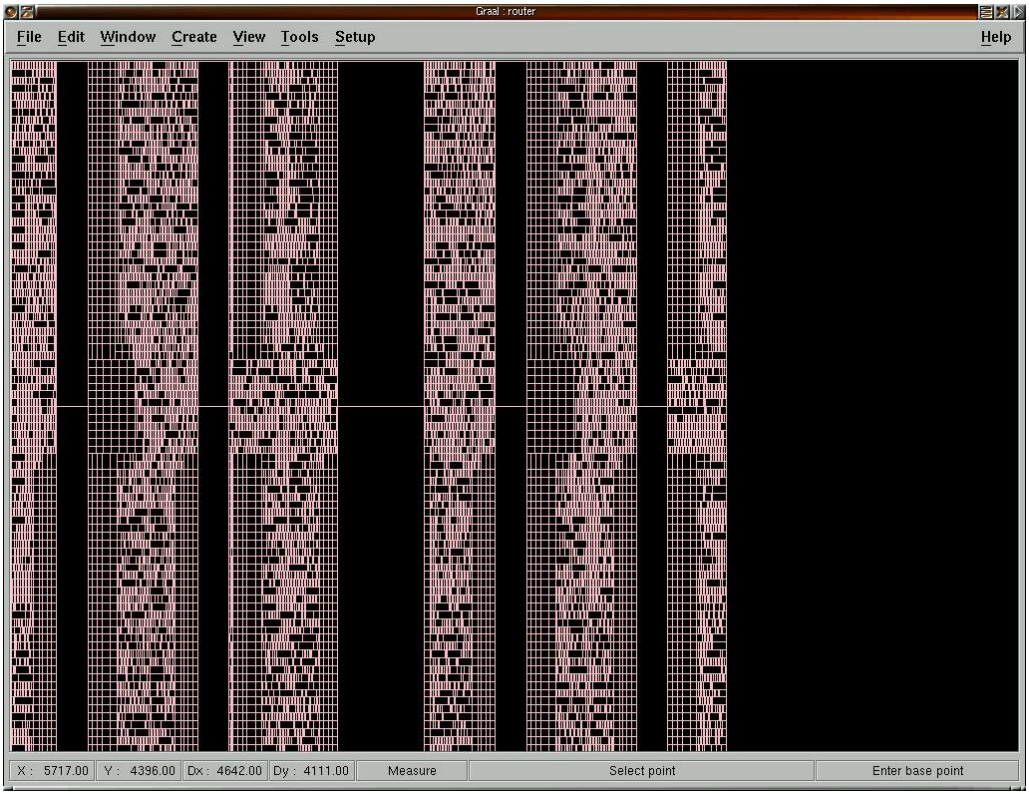

Figure 46 : Placement final du routeur RSPIN

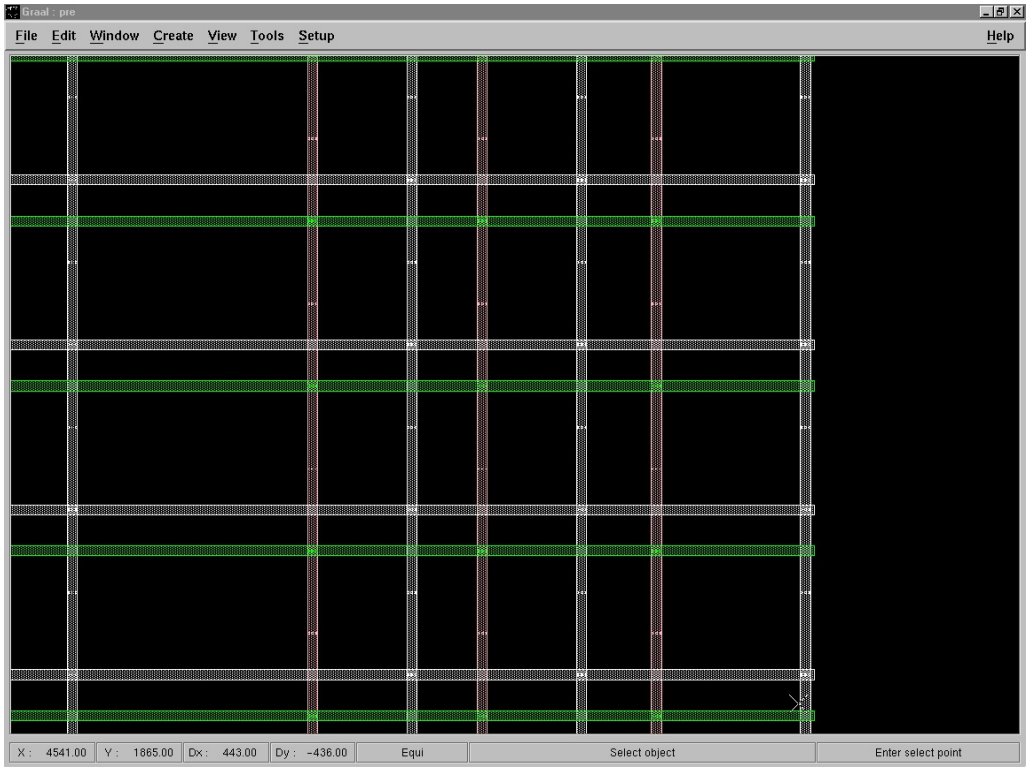

Figure 47 : Maille d'alimentation dans le routeur RSPIN

2.3 Principe de la méthode

Le routeur *RSPIN* se comporte comme un générateur de courant qui doit charger une certaine capacité. Au moment du chargement, nous allons admettre une chute de tension sur les alimentations d'une valeur maximale ΔV de ±10 %, c'est-à-dire *vdd* – 10% et *vss* + 10 % (Figure 48). La chute de tension sur les alimentations est déterminée par l'Équation 1, où *R* représente la résistance du réseau d'alimentation maillé et *I* représente le courant qui passe par le réseau d'alimentation maillé.

$$\Delta V = R.I$$

Équation 1 : Chute de tension sur les alimentations

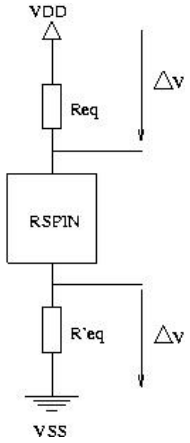

Figure 48 : Chute de l'alimentation

D'abord, nous avons estimé le courant maximum qui peut passer par les fils d'alimentation. Ensuite, nous en avons déduit la résistance maximale pour que la chute de tension sur les alimentations soit inférieure à 10%. Finalement, nous avons vérifié que la résistance du réseau d'alimentation maillé est inférieure à la résistance maximale autorisée.

Nous nous intéressons au courant instantané maximum qui peut être approximé comme le double du courant moyen (Équation 2). Le courant moyen peut être calculé à l'aide de l'Équation 3 où *P* représente la puissance dissipée et *U* la tension d'alimentation.

$$I \approx 2 * I_{moyen}$$

Équation 2 : Courant maximum instantané

$$I_{moyen} = \frac{P}{U}$$

Équation 3 : Courant moyen

Si nous négligeons la puissance dissipée par la logique interne du routeur *RSPIN*, la puissance réelle maximale dissipée par le routeur *RSPIN* est approximée comme la moitié de la puissance P_a susceptible de passer à travers les deux alimentations *vdd* et *vss* (Équation 4 où *C* représente la capacité de charge, *U* la tension d'alimentation et *T* le cycle d'horloge). En

effet, pendant un cycle, soit le routeur *RSPIN* charge ses sorties, soit il les décharge. Les deux rappels d'alimentation *vdd* et *vss* sont dimensionnés pour le maximum de puissance.

$$P_{\max} = \frac{1}{2} * P_a = \frac{1}{2}\left(\frac{1}{2} CU * 2\right)\frac{1}{T}$$

Équation 4 : Puissance maximale dissipée

Pour estimer le courant instantané maximum, nous avons donc commencé par estimer la capacité *C* que le réseau d'alimentation maillé doit charger.

2.4 Application sur le routeur RSPIN

Nous estimons que le routeur *RSPIN* exige le maximum de puissance des alimentations lorsqu'il doit faire basculer de la valeur logique '0' à la valeur logique '1' toutes ses sorties, c'est-à-dire les 8 * 38 = 304 fils en métal 5. C'est la capacité de ces nappes de fils qu'il faut donc calculer pour avoir une estimation de la puissance dissipée.
Pour la technologie 0.13 µm de *STMicroelectronics*, la capacité *C* des 304 fils en métal 5 est de l'ordre de 70 pF.
Avec une alimentation *U* de 1.2 V et une fréquence de fonctionnement visée de 200 MHz (donc un temps de cycle *T* de 5 ns), la puissance maximale est :

$P_{max}^{réelle} = ½ * (½ * 70.10^{-12} * 1{,}2 * 2) / 5.10^{-9} \approx 8.4$ mW

En prenant un peu de marge, nous utilisons une puissance maximale de 10 mW pour le reste du calcul. Ce qui nous donne un courant instantané maximum *I* estimé à :

$I \approx 2 * 10$ mW / 1.2 V = 16.66 mA

La chute de tension doit être inférieure à 10 % de la tension d'alimentation :

$\Delta V = RI < 10\% * V_{dd} = 120$ mV

Ce qui nous donne, en arrondissant le courant maximum à 20 mA, une résistance maximale admise de : $R = \Delta V / I = 120$ mV / 20 mA = 6 Ω.

Les rappels d'alimentation horizontaux en métal 1, intégrés dans les cellules de base développées par l'équipe *ASIM* du *LIP6*, ont été doublés par des rappels d'alimentation horizontaux en métal 4 dans le routeur *RSPIN*. Le métal 4, moins résistif, permet de diminuer les résistances équivalentes des rappels d'alimentation horizontaux. Elles sont estimées à $R_h \approx$ 305 mΩ pour *vss* et *vdd*.
Nous avons 8 rappels d'alimentation verticaux pour *vdd* et 10 rappels d'alimentation verticaux pour *vss*. Routés en métal 3, ils ont une résistance totale d'environ $R_v \approx 2{,}75$ Ω pour *vdd* et $R_v' \approx 2{,}2$ Ω pour *vss*.
Les résistances équivalentes des rappels d'alimentation (R_{eq} pour *vdd* et R'_{eq} *vss*) sont estimées par la mise en série des résistances des rappels d'alimentation horizontaux (R_h) et de celles des rappels d'alimentation verticaux (R_v pour *vdd* et R_v' pour *vss*). Elles ne dépassent pas 4 Ω, ce qui est largement en dessous de la résistance maximale autorisée.

3 La distribution de l'horloge et du reset

Le principal objectif est de minimiser le retard (*skew*) des signaux d'horloge et de *reset*. Ici encore, le placement/routage explicite permet de contrôler le *skew* d'horloge. Nous avons utilisé la méthode classique qui consiste à réamplifier ces signaux par l'intermédiaire de *buffers* avant qu'ils arrivent sur les éléments mémorisants.

Il y a deux types d'éléments mémorisants dans le routeur *RSPIN* :

- Les registres contenus dans les *FIFO*s du chemin de données,
- Les registres définissant l'état des automates de contrôle des blocs *icb* et *ocb*.

Pour ce qui concerne les *FIFO*s, les signaux d'horloge et de *reset* sont réamplifiés par des *buffers* internes aux *FIFO*s. Pour les registres des blocs *icb* et *ocb* obtenus par synthèse, nous avons introduit des *buffers* locaux pour amplifier le signal d'horloge et le signal *reset* dans chaque bloc. Le retard d'horloge dans les *FIFO*s a été mesuré en simulation *spice*. Il doit être égal au retard sur les registres de la partie contrôle.

Tous les *buffers* sont placés dans la zone centrale de la macro-cellule *RSPIN*, de telle sorte que la distribution du signal d'horloge et du signal *reset* puisse se résumer à un fil horizontal et à quelques fils verticaux pour descendre ou pour monter vers les *buffers*. Les segments horizontaux sont préroutés en métal 4, tandis que les segments verticaux sont préroutés en métal 3 (Figure 49).

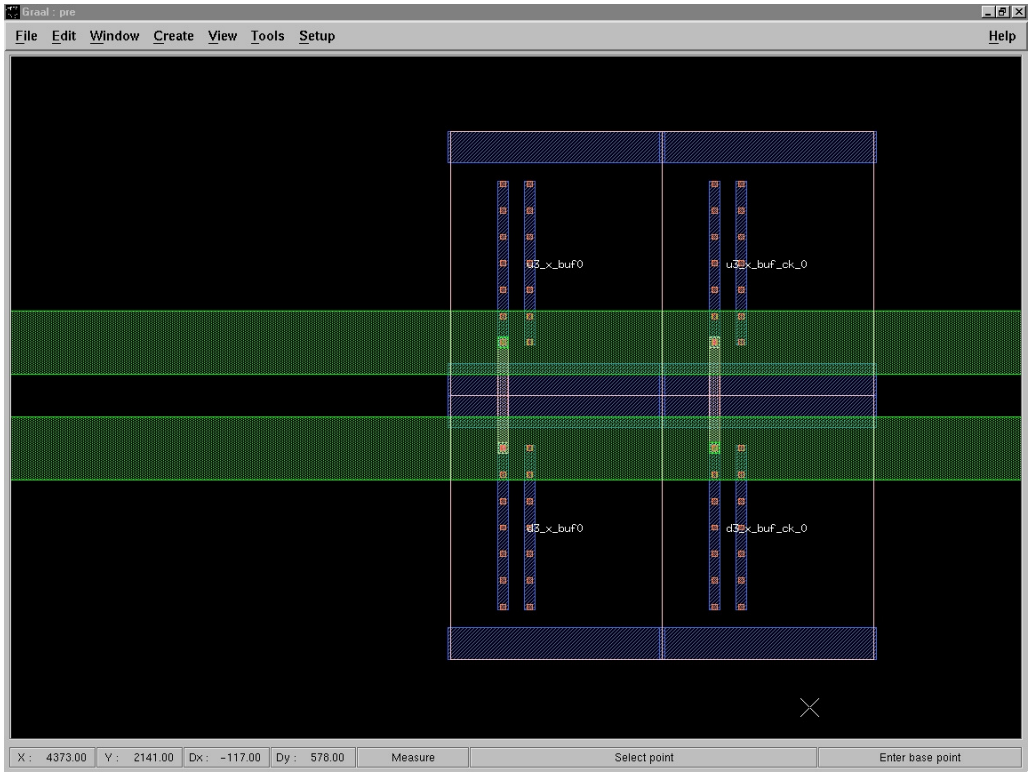

Figure 49 : Distribution d'horloge et du signal reset

La largeur des segments pour le signal d'horloge est de 22λ en métal 4. Celle pour le signal *reset* est également de 22λ. Leur longueur est de 4560λ et leur résistance est d'environ 16 Ω. La capacité mesurée dans un cas typique (1.2 V et 25 °C) est d'environ 288.42 fF pour le signal d'horloge et d'environ 332.33 fF pour le signal de *reset*.

Le signal d'horloge attaque 11 *buffers* avec une capacité d'entrée de 6.12 fF chacun, ainsi que 10 *FIFO*s avec une capacité d'entrée de 9.02 fF chacun. Nous obtenons donc une capacité totale de 157.52 fF et un *skew* total de 7.14 ps sur le signal d'horloge. Cette valeur est largement inférieure à 1 % de la période d'horloge. Pour éviter le bruit sur l'horloge, nous avons blindé le fil d'horloge en le mettant entre deux fils d'alimentation. Ce qui explique la valeur inférieure de la capacité mesurée sur le signal d'horloge.
Le signal *reset* attaque un *buffer* avec une capacité d'entrée de 6.12 fF, 10 portes *OR* avec une capacité d'entrée de 4.46 fF chacun, ainsi que 10 *FIFO* avec une capacité d'entrée de 5.95 fF chacun. Ce qui représente une capacité totale de 110.22 fF. Le retard sur le signal de *reset* est d'environ 7.08 ps, également largement inférieure à 1 % de la période d'horloge.

4 Surface occupée par le routeur RSPIN

Les dimensions du routeur *RSPIN* sont déterminées en hauteur par la hauteur des *FIFO* (soit 88 tranches de un bit, de hauteur 50 λ chacune), et en largeur par les nappes de fils des ports d'entrée/sortie (soit 12 nappes de 38 bits, en métal 5, au pas de 10 λ). La hauteur du routeur *RSPIN* est de 4400 λ, sa largeur est de 4560 λ. Il est composé de 9144 portes logiques et environ 29,6 % de sa surface est occupée par les *FIFOs*. Il reste 17,2 % d'espace inutilisé sous les fils de routage. La Figure 50 montre le *layout* symbolique du routeur *RSPIN*. Sur cette figure, les *FIFOs* sont représentées en rouge, jaune et noir. Le bloc de contrôle et le chemin de données sont représentés en rose sur la Figure 50. L'espace libre, rempli par des cellules de bourrage, est représenté en gris sur la Figure 50.

Nous voyons bien que la largeur du routeur *RSPIN* n'est pas déterminée par la logique interne mais bien par la largeur des nappes de fils d'entrée/sortie, comme le montre les 17,2 % d'espace libre laissés dans le *layout*.

L'intérêt du *layout* symbolique de la chaîne de conception *ALLIANCE* est de pouvoir produire un *layout* réel, au format *GDS II*, pour n'importe quel fondeur. Comme le routeur *RSPIN* utilise 5 niveaux de métal, la seule contrainte est que le fondeur puisse fournir un processus de fabrication possédant au moins 5 niveaux de métal. Le Tableau 2 résume les dimensions du routeur *RSPIN* pour les processus de fabrication récents.

Technologie	λ	Largeur	Hauteur	Surface
0.25 µm	0,20 µm	912 µm	880 µm	0,803 mm^2
0,18 µm	0,16 µm	729,6 µm	704 µm	0,514 mm^2
0,13 µm	0,12 µm	592.8 µm	0.572 µm	0,289 mm^2

Tableau 2 : Dimensions du routeur RSPIN

5 La validation du layout

La validation du *layout* du routeur *RSPIN* se fait avec les outils de la chaîne *ALLIANCE* (Figure 51). Après avoir converti le *layout* du routeur *RSPIN* du format *.def* de l'outil *SILICON ENSEMBLE* au format *.ap* de la chaîne *ALLIANCE* avec l'outil *DEF2A*, on fait d'abord une vérification des règles de dessins à l'aide de l'outil *DRUC*. Puis on fait une extraction du *layout* au niveau porte avec l'outil *COUGAR* pour obtenir une *netlist*. Ensuite, la *netlist* d'origine est mise à plat avec l'outil *FLATLO*. Enfin, on fait une comparaison de la *netlist* produite par l'outil *FLATLO* avec la *netlist* produite par l'outil *COUGAR* grâce à l'outil *LVX*. Cette comparaison valide le *layout* du routeur *RSPIN*. Les outils *DEF2A*, *DRUC*, *COUGAR*, *FLATLO*, *LVX* font partie de la chaîne de conception *ALLIANCE*.

6 Conclusion

Nous avons proposé une méthode de réalisation d'une macro-cellule optimisée comportant des chemins de données très réguliers et de blocs de contrôle obtenus par synthèse. Cette méthode permet de concevoir des composants virtuels réutilisables décrits jusqu'au niveau physique (*hard IP core*) grâce à l'utilisation du ''*layout* symbolique''. Nous avons également proposé une méthode très pertinente de dimensionnement des réseaux de distribution d'alimentation, d'horloge et de *reset* pour ce type de composant. Cette méthode, nécessitant un nombre très réduit de paramètres, a été mise en œuvre de bout en bout pour le routeur *RSPIN*.

Nous avons obtenu une macro-cellule *VLSI* de 0,29 mm^2 pour un procédé de fabrication 0.13 µm et avons démontré que la surface du routeur *RSPIN* n'est pas déterminée par la logique et les *FIFOs*, mais par les nappes de fils réalisant le *crossbar* et permettant la communication entre les routeurs *RSPIN*.

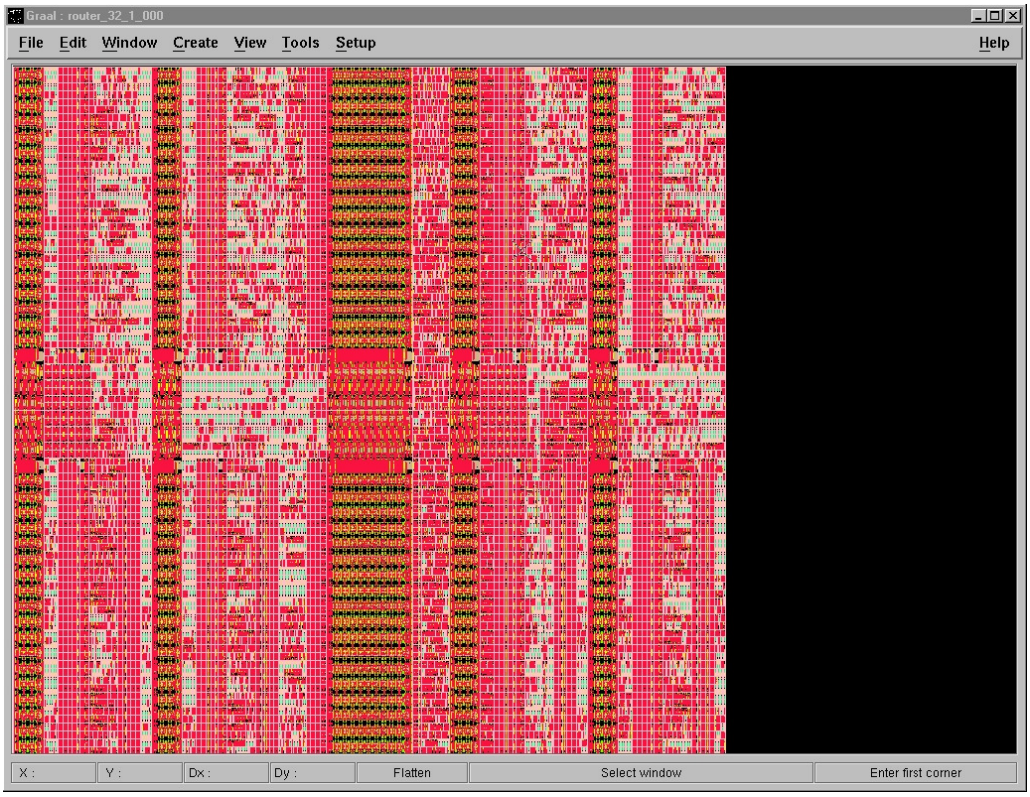

Figure 50 : Layout symbolique du routeur RSPIN

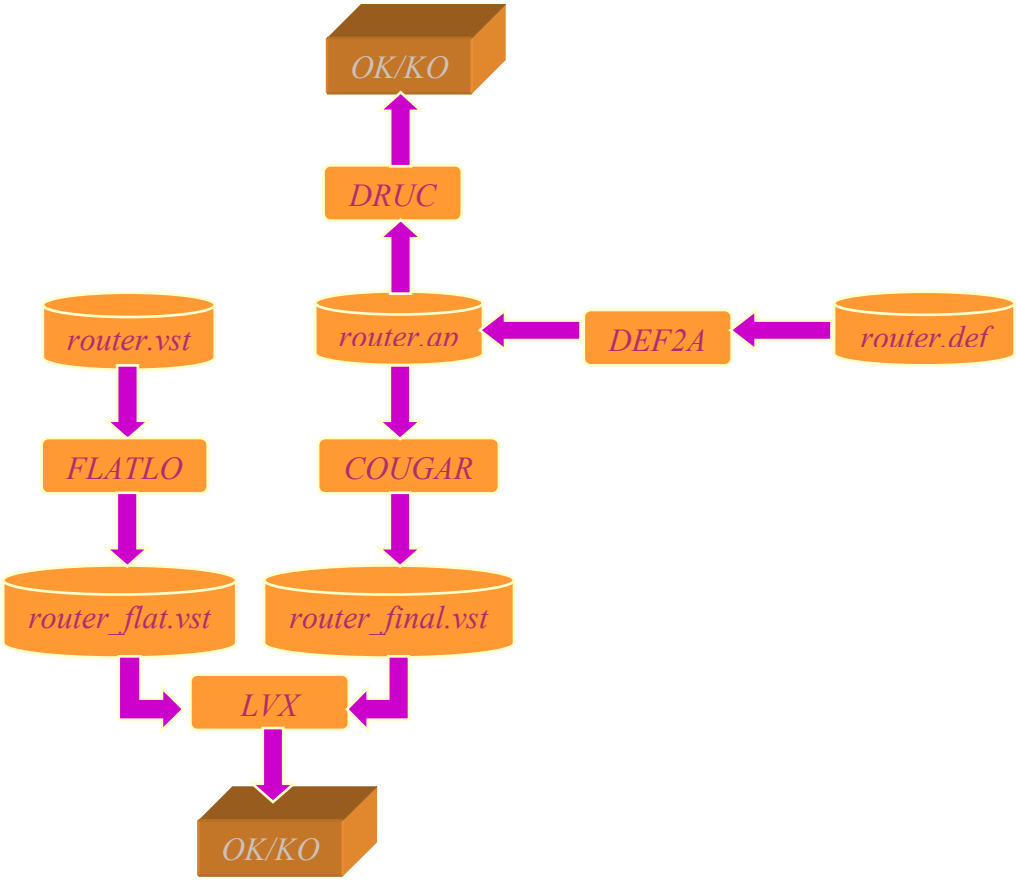

Figure 51 : Flot de validation du layout du routeur RSPIN

Chapitre 8 *Puce d'évaluation d'un micro-réseau 32 ports*

Un micro-réseau *SPIN* à 32 ports a été conçu dans le but de permettre une caractérisation physique de la technologie *SPIN,* et d'analyser les problèmes posés par l'industrialisation de cette technologie. Nous avons donc conçu d'une part une macro-cellule *SPIN32,* totalement synchrone, et d'autre part la logique d'instrumentation destinée à être intégrée dans une puce d'évaluation afin de valider, au niveau physique, les différents résultats obtenus par simulation ou analyse théorique : latence du micro-réseau *SPIN* en fonction de la charge, temps de cycle, consommation électrique, taux d'erreurs.

1 *La macro-cellule SPIN32*

Le graphe du micro-réseau *SPIN* à 32 ports est composé de deux micro-réseaux *SPIN* à 16 ports mis tête-bêche. Il contient 16 routeurs *RSPIN* (Figure 52) : le premier indice correspond au niveau du routeur *RSPIN* dans *l'arbre quaternaire élargi*, le second correspond au numéro de *cluster* ou grappe.

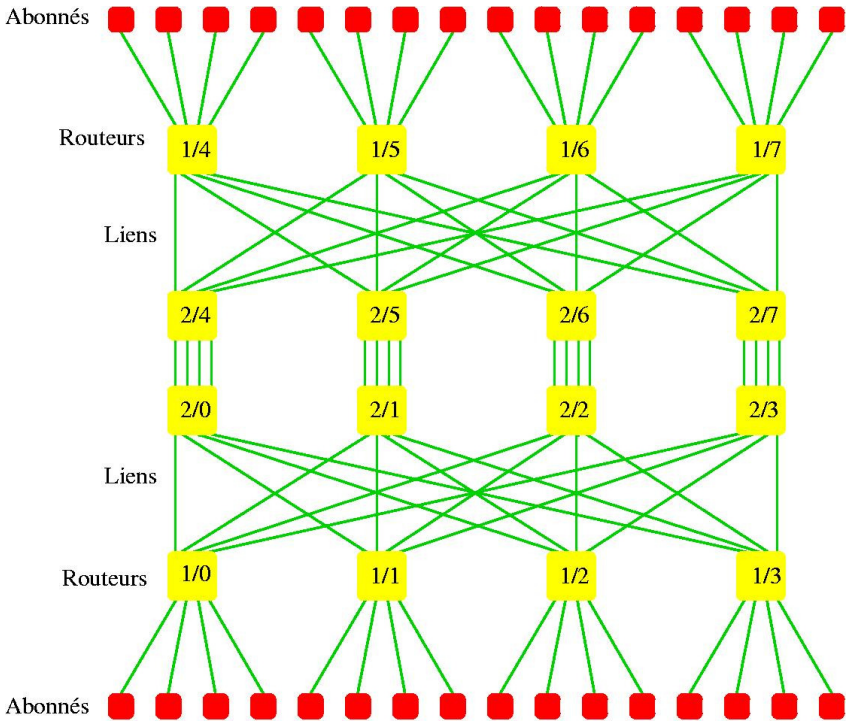

Figure 52 : Micro-réseau SPIN à 32 ports

1.1 Topologie de la macro-cellule SPIN32

Le dessin des masques du routeur *RSPIN* est conçu pour être aboutable horizontalement et verticalement, ce qui permet d'abouter les 16 routeurs *RSPIN* de façon à avoir le micro-réseau *SPIN* à 32 ports le plus compact possible.
La Figure 53 montre comment les nappes de fils d'interconnexion entre les routeurs *RSPIN* peuvent être routées ''*au dessus*'' des routeurs *RSPIN*. Chaque segment bipoint horizontal ou vertical représente deux nappes de 38 bits (76 fils routés en métal 4 et 6).

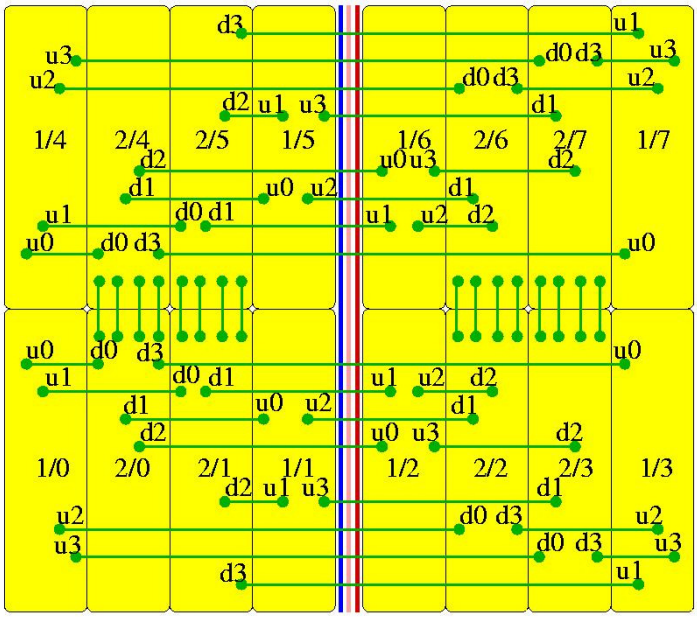

Figure 53 : Plan de masse du micro-réseau SPIN à 32 ports

Le micro-réseau *SPIN* à 32 ports est composé de 8 routeurs *RSPIN* en largeur et de 2 routeurs *RSPIN* en hauteur. Le maillage d'alimentation du micro-réseau *SPIN* est construit en interconnectant les maillages d'alimentation de tous les routeurs *RSPIN* entre eux. Le routeur *RSPIN* a été conçu de façon à ce que les rappels d'alimentation coïncident si deux routeurs *RSPIN* sont aboutés horizontalement ou verticalement. Les résistances équivalentes totales de la grille d'alimentation maillée pour chaque alimentation *vdd* et *vss* ne dépassent pas 1.3 Ω.
Les fils d'interconnexion entre les 16 routeurs *RSPIN* sont routés au-dessus des routeurs *RSPIN* en métal 4 et 6.

1.2 La distribution du signal d'horloge

Le pin d'horloge de la macro-cellule *SPIN32* se trouve au milieu de la macro-cellule sur les faces Nord et Sud. Ce signal doit attaquer les fils d'horloge des 16 routeurs *RSPIN* (4 en haut à gauche, 4 en bas à gauche, 4 en haut à droite et 4 en bas à droite) d'une capacité totale 1.23 pF (dans un cas typique) et un *skew* inacceptable de 1.77 ns. Nous avons rajouté une colonne de 172 *buffers* au milieu de la macro-cellule *SPIN32* afin d'amplifier le signal d'horloge et diminuer le *skew* d'horloge (Tableau 3). Le nombre 172 a été obtenu par simulation *spice*. Nous avons également inséré 172 *buffers* pour amplifier les signaux de *reset* et de *mode*.

	Meilleur cas	Cas typique	Pire cas
Sans éléments parasites	237 ps	344 ps	528 ps
Avec résistances et capacités parasites	363 ps	505 ps	765 ps
Avec capacités de couplages	313 ps	467 ps	725 ps

Tableau 3 : Skew d'horloge dans différents cas

1.3 Surface occupée par la macro-cellule SPIN32

Puisque les nappes de fils d'interconnexion entre les routeurs *RSPIN* sont routées en métal 6 et métal 4 au-dessus des routeurs *RSPIN*, les dimensions de la macro-cellule *SPIN32* sont déterminées en hauteur et en largeur par le nombre de routeurs *RSPIN* nécessaires pour construire la macro-cellule *SPIN32*. En largeur, il faut ajouter la place prise par la distribution de l'horloge avec son blindage, soit 160 λ. Cela fait une surface de (2 * 4400 λ) * (8 * 4560 λ + 160 λ) = 322432000 λ^2 donc 4,64 mm^2 en technologie 0.13 µm. Il est composé de 146304 portes logiques et de 1411136 transistors (Figure 54).

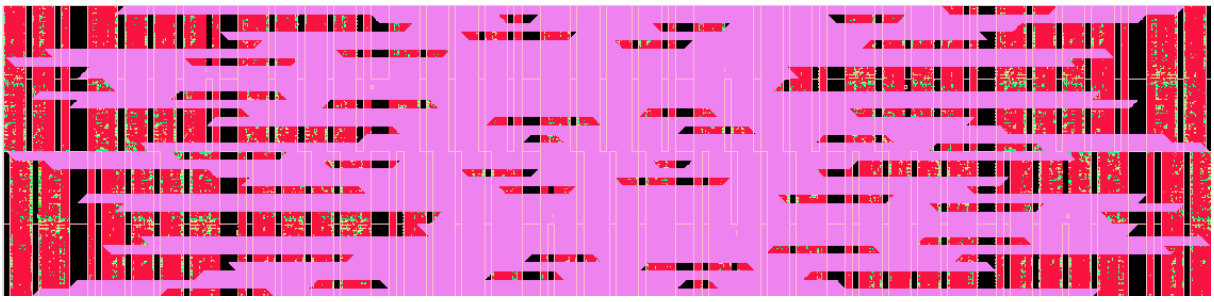

Figure 54 : Layout symbolique de la macro-cellule SPIN32

1.4 Génération du fichier GDSII

La macro-cellule *SPIN32* a été conçue en utilisant le *layout* symbolique de la chaîne de conception *ALLIANCE*, de façon à ce qu'elle soit indépendante du processus de fabrication utilisé. Nous pouvons ainsi livrer la macro-cellule *SPIN32* à n'importe quel fondeur utilisant un processus de fabrication pouvant utiliser au moins 6 niveaux de métal.
Lorsque nous avons commencé le projet *SPIN*, la technologie cible était le processus de fabrication 0.25 µm. Au fil du temps, nous sommes passés au processus de fabrication 0.18 µm pour finir aujourd'hui avec le processus de fabrication 0.13 µm de *STMicroelectronics*. Pour livrer la macro-cellule *SPIN32* à *STMicroelectronics*, il faut donc passer du *layout* symbolique au *layout* réel. Pour cela, nous suivons le flot de la Figure 55. Nous utilisons l'outil *S2R* de la chaîne *ALLIANCE*, qui prend en entrée un fichier de *layout* symbolique au format *.ap* et fourni un fichier de *layout* réel en technologie 0.13 µm au format *GDS II*.

Pour vérifier le dessin des masques, nous faisons une extraction au format *Spice,* avec l'outil *COUGAR* de la chaîne *ALLIANCE*, permettant d'obtenir une *netlist* au niveau transistors qui peut être simulée avec l'outil *Spice.*
D'autre part, la validation de la macro-cellule *SPIN32* se fait, également, par la comparaison de la *netlist* au format *Spice* que nous avons fournie et de la *netlist* extraite à partir du format *GDS II*. Cette extraction/comparaison se fait avec l'outil *CALIBRE*.

1.5 Caractérisation de la macro-cellule SPIN32

Une fois la macro-cellule *SPIN32* validée du point de vue du schéma de principe, il faut la caractériser du point de vue du *timing*. Pour cela, elle a été extraite en tenant compte, cette fois-ci, des capacités parasites et des résistances des fils d'interconnexions. L'extraction a été faite avec l'outil *ARCADIA* associé avec l'interface *PLS*. Le résultat est une *netlist* à plat, au format *Spice*, contenant les résistances et les capacités parasites. L'extraction a été faite de 3 manières différentes. Dans le premier cas, nous avons fait une extraction sans éléments parasites. Dans le deuxième cas, l'extraction a été faite avec les capacités et les résistances parasites. Enfin dans le troisième cas, en plus des capacités et des résistances parasites, nous avons également extrait les capacités de couplages.
Cette *netlist* à plat a été, en partie, simulée avec l'outil *HSIM* pour valider son comportement. Elle a été, ensuite, caractérisée à l'aide de l'outil d'analyse temporelle *HiTas*. La caractérisation fournit un fichier au format *.lib* qui contient les informations sur les temps de *setup* et de *hold* ainsi que les capacités sur les entrées. Le fichier contient également les temps d'accès et les capacités sur les sorties de la macro-cellule *SPIN32*. Ce fichier de caractérisation au format *.lib* sera utilisé par l'outil *DESIGN COMPILER* lors de la synthèse de la puce d'évaluation.

L'ensemble du flot de génération, de vérification et de caractérisation est présenté sur la Figure 55.

	Meilleur cas	Cas typique	Pire cas
Sans éléments parasites	998 ps	1528 ps	2492 ps
Avec résistances et capacités parasites	2195 ps	3255 ps	5183 ps
Avec capacités de couplages	2109 ps	3175 ps	5107 ps

Tableau 4 : Chaîne longue

	Meilleur cas	Cas typique	Pire cas
Sans éléments parasites	1 GHz	654 MHz	401 MHz
Avec résistances et capacités parasites	455 MHz	307 MHz	192 MHz
Avec capacités de couplages	474 MHz	314 MHz	195 MHz

Tableau 5 : Fréquence de fonctionnement

Le Tableau 4 représente le temps de propagation de la chaîne longue pour différentes caractéristiques du procédé de fabrication. Le Tableau 5 fournit les fréquences de fonctionnement correspondantes. On constate qu'il est possible de garantir dans le pire cas une fréquence de fonctionnement de 192 MHz, ce qui nous donne une bande passante cumulée maximale d'environ 209 Gbit/s. Nous verrons que les simulations vont estimer le

seuil de saturation à 52 % de la charge maximale, soit une bande passante cumulée pratique d'environ 109 Gbit/s.

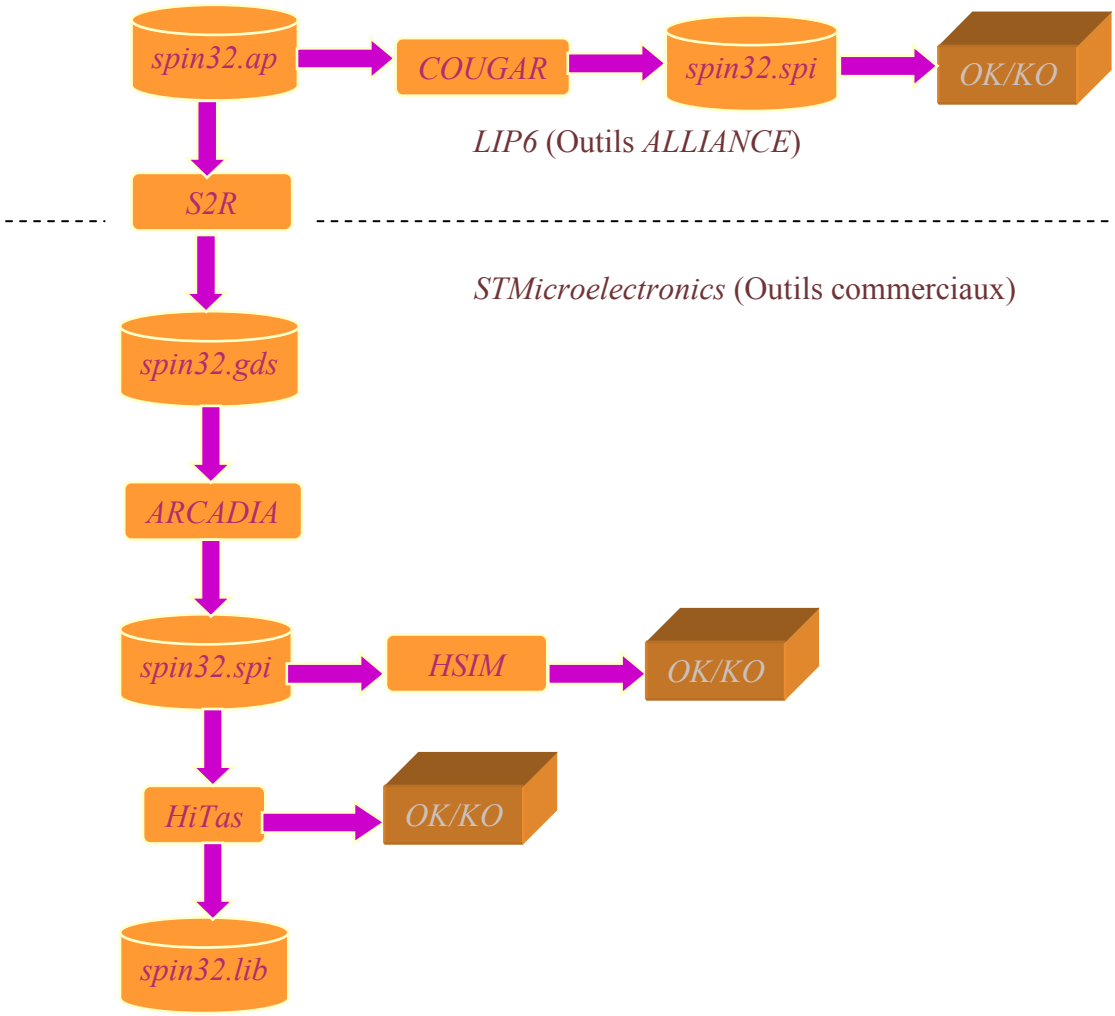

Figure 55 : Flot de transfert technologique

2 Instrumentation

Pour caractériser la macro-cellule *SPIN32* sur silicium, nous avons conçu une *puce d'évaluation* composée de la macro-cellule *SPIN32*, habillée de toute la logique nécessaire à son instrumentation. Nous avons ajouté, entre autres, un analyseur et un générateur de trafic aléatoire *TGA* (*Traffic Generator and Analyzer*) sur chacun de ses ports. Nous avons également conçu un contrôleur central *CC* (*Central Controller*) qui va gérer toute l'instrumentation. L'ensemble des 32 *TGA*, du *CC* et de la macro-cellule *SPIN32* constitue la *puce d'évaluation* (Figure 56). Son interface est une interface mémoire de type *RAM*, permettant d'adresser, en lecture ou en écriture, les différentes ressources internes depuis l'extérieur de la puce.

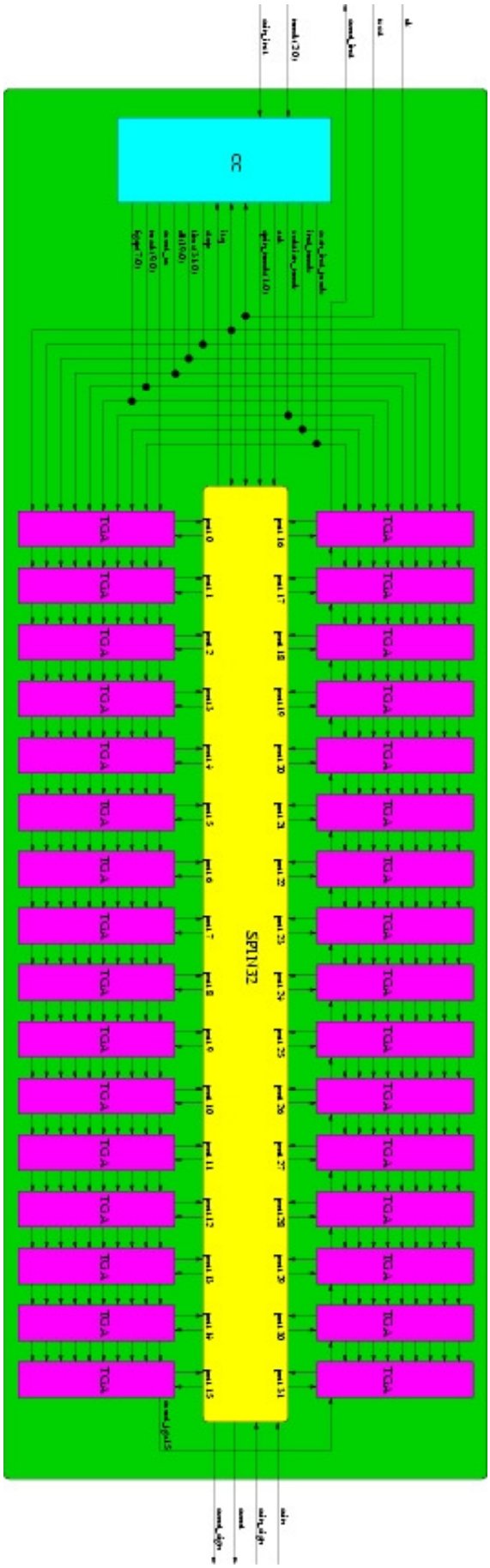

Figure 56 : Environnement d'instrumentation

2.1 Objectif de l'instrumentation

L'objectif est de caractériser la macro-cellule physique *SPIN32* selon 4 critères : la fiabilité, la latence, la consommation et la bande passante.

2.1.1 La fiabilité

La fiabilité est une caractéristique intrinsèque à tout micro-réseau d'interconnexion. C'est l'assurance que chaque paquet injecté dans la macro-cellule *SPIN32* ressort à la destination prévue, sans aucune détérioration de l'information contenue. La fiabilité consiste donc à mesurer le taux d'erreurs de transmission dans la macro-cellule *SPIN32* en faisant varier des paramètres physiques tels que la fréquence de fonctionnement, la température ou la tension d'alimentation.

Pour savoir si un paquet a été corrompu durant la transmission, un *CRC* (*Cyclic Redundancy Code*) [42] est attaché à la fin du paquet. Le *CRC* est calculé avec tous les mots du paquet, sauf évidemment le dernier mot, qui lui contient le *CRC*.
Lors de la réception, un nouveau calcul de *CRC* est fait sur tous les mots du paquet, sauf le dernier. Le *CRC* calculé à la réception est comparé avec le *CRC* calculé à la transmission. L'égalité de ces deux *CRC* assure que le paquet a été transmis sans être corrompu.
Le calcul du *CRC* à la réception se fait avec l'adresse locale et non avec l'adresse de destination transmise. Autrement dit, nous remplaçons le champ *dest* de l'en-tête par l'adresse locale du port de sortie. De cette façon, si les deux *CRC* sont égaux, nous sommes assurés que le paquet qui a été routé par la macro-cellule *SPIN32* est parvenu au bon destinataire.

2.1.2 La latence

L'architecture *SPIN*, comme tous micro-réseaux à paquets commutés, a la particularité de ne pas offrir une garantie de latence bornée ni de latence fixe, mais une garantie statistique de la latence. Puisque la latence de la macro-cellule *SPIN32* n'est pas fixe, nous cherchons à mesurer la latence moyenne, en faisant varier plusieurs paramètres tels que : la charge, l'utilisation des queues centrales, la longueur des paquets, l'acheminement des paquets dans l'ordre chronologique, l'absence d'interblocage et la localité du trafic. Nous cherchons également à évaluer le seuil de saturation (le pourcentage de la charge à partir de laquelle la latence augmente de façon significative) de la macro-cellule *SPIN32* selon les mêmes paramètres. Enfin, nous voulons mesurer la distribution de la latence pour deux valeurs de la charge : une charge dans la zone de saturation et une autre hors de cette zone.

Pour mesurer la latence, la date au moment de la transmission est insérée dans chaque paquet. Lors de la réception, la date de transmission est extraite du paquet. Elle est soustraite de la date actuelle pour donner la latence du paquet.

2.1.3 La consommation

Nous cherchons à évaluer la puissance consommée par la macro-cellule *SPIN32*, en fonction de paramètres tels que la charge et la fréquence de fonctionnement.

Pour mesurer la puissance consommée par la macro-cellule *SPIN32*, nous avons séparé les réseaux d'alimentation *Vdd/Vss* pour la macro-cellule *SPIN32*, et pour la logique d'instrumentation. Nous utiliserons un ampèremètre sur l'alimentation *vdd* pour mesurer le courant entrant dans la macro-cellule.

2.1.4 La fréquence de fonctionnement

Nous cherchons à évaluer la fréquence maximale de fonctionnement de la macro-cellule *SPIN32*. L'analyse statique de performances réalisée avec *HiTas*, confirmée par simulation *spice*, fournit une fréquence maximale de fonctionnement de 192 MHz pour la macro-cellule *SPIN32*.

2.2 Caractéristique du trafic

Pour effectuer les mesures, les blocs *TGA* génèrent, vers la macro-cellule *SPIN32*, un trafic synthétique. Ce dernier est tel que :

- au cours d'une session de mesures, tous les paquets émis par les blocs *TGA* ont la même longueur *pl* ;
- pour chaque bloc *TGA*, le choix de la destination *dest* est une variable aléatoire à distribution uniforme parmi les 32 destinations possibles. Les destinations possibles sont définies en prenant en compte la localité du trafic voulu ;
- le *gap* (nombre de cycles entre deux paquets consécutifs) est également une variable aléatoire à distribution uniforme dont on peut contrôler la valeur moyenne *gapm.*

Vu les caractéristiques du trafic synthétique décrites ci-dessus, la charge fournie à la macro-cellule *SPIN32* est donnée ci-dessous. Elle peut être changée en variant la valeur moyenne *gapm* et la longueur du paquet *pl*.

$$load = \frac{pl}{pl + gapm}$$

Il y a 4 niveaux de localité du trafic (Figure 57):

- le *trafic très local* est composé de paquets dont les destinations se trouvent réparties uniformément dans le même *cluster* que son port d'entrée (en bleu sur la Figure 57). Pour générer ce type de trafic, il faut mettre la *puce d'évaluation* dans le mode *full* à l'aide du signal *mode*. Dans ce mode, seuls les 2 bits de poids faibles de la destination *dest* sont aléatoires. Les 3 bits de poids forts sont fixes et dépendent du *cluster* où se trouve le bloc *TGA* ;
- le *trafic moyennement local* est composé de paquets dont les destinations se trouvent réparties uniformément dans le même *cluster* que son port d'entrée ou dans le *cluster* voisin (en rose sur la Figure 57). Pour générer ce type de trafic, il faut mettre la *puce d'évaluation* dans le mode *high* à l'aide du signal *mode.* Dans ce mode, seuls les 3 bits de poids faibles de la destination *dest* sont aléatoires ; les 2 bits de poids forts sont fixes et dépendent du cluster où se trouve le bloc *TGA* ;
- le *trafic faiblement local* est composé de paquets dont les destinations se trouvent réparties uniformément dans le même demi-réseau que son port d'entrée (en violet sur la Figure 57). Pour générer ce type de trafic, il faut mettre la *puce d'évaluation* dans le

mode *medium* à l'aide du signal *mode*. Dans ce mode, seuls les 4 bits de poids faibles de la destination *dest* sont aléatoires. Le bit de poids fort est fixe et dépend du demi-réseau où se trouve le bloc *TGA* ;

- le *trafic aléatoire*, est composé de paquets dont les destinations se trouvent réparties uniformément sur tous les ports de la macro-cellule *SPIN32*. Pour générer ce type de trafic, il faut mettre la *puce d'évaluation* dans le mode *low* à l'aide du signal *mode*. Dans ce mode, les 5 bits de la destination *dest* sont tous aléatoires.

2.3 Le bloc TGA

Le bloc *TGA* a la double tâche de générer et d'analyser le trafic qui va charger la macro-cellule *SPIN32*. Le bloc *TGA* est répliqué 32 fois, à quelques détails près : la partie s'occupant du test eulérien est différente et dépend du port sur lequel le bloc *TGA* est connecté. Un bloc *TGA* est connecté à chaque port de la macro-cellule *SPIN32* (Figure 56).

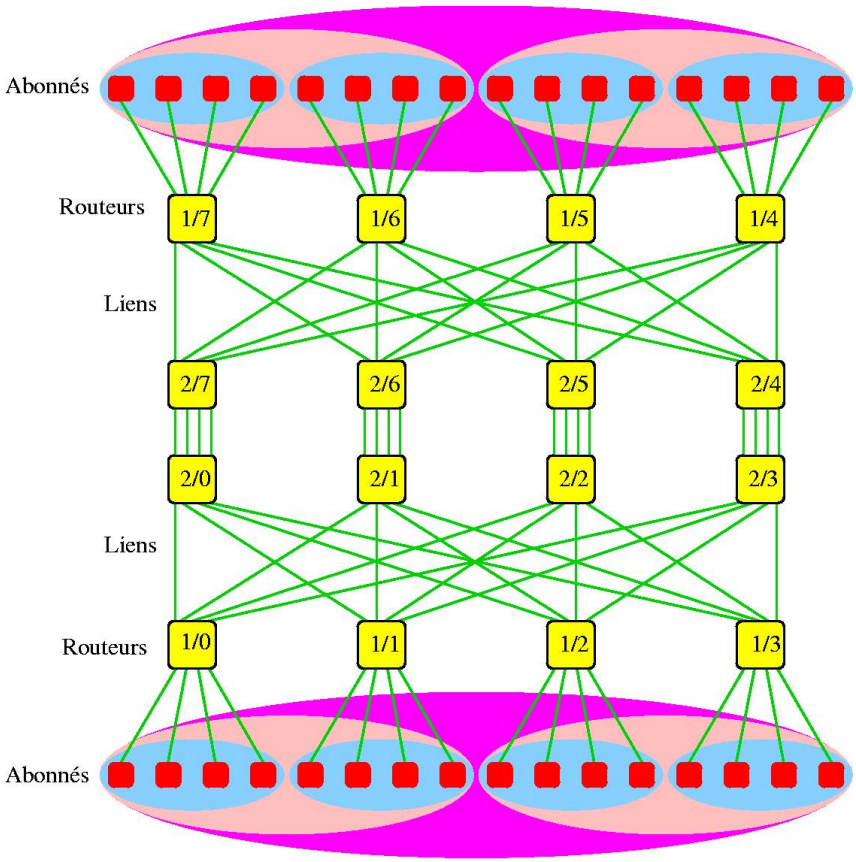

Figure 57 : Localité du trafic synthétique

2.3.1 La partie génération du bloc TGA

En génération, le rôle du bloc *TGA* est de construire les paquets et de les injecter dans la macro-cellule *SPIN32*. La longueur *pl* d'un paquet est égale à *dl* + 3, où *dl* est un paramètre codé sur 20 bits qui permet de définir la longueur du paquet. Le paramètre *dl* est envoyé par le contrôleur central (*CC*) aux 32 blocs *TGA*. Une valeur nulle du paramètre *dl* définit la longueur minimale d'un paquet, qui est alors de 3 mots. Le premier contient la destination, le deuxième contient la date d'injection du paquet dans la macro-cellule *SPIN32* et le dernier mot contient le *CRC*.

Pour pouvoir construire le paquet, le bloc *TGA* doit calculer les informations qu'il doit y inclure. Ces informations sont la destination *dest* du paquet et le *CRC* pour la vérification de l'intégrité des données. Le bloc *TGA* contient un registre *crc_tg* et un registre *crc_ta*. Le registre *crc_tg* sert pour le calcul du *CRC* au moment de la génération ; tandis que le registre *crc_ta* sert au calcul du *CRC* au moment de l'analyse. L'égalité des deux *CRC* valide le paquet reçu.

Le bloc *TGA* contient également un compteur *nbpkt*, qui contient le nombre total de paquets envoyés. Il est incrémenté d'une unité à chaque fois qu'un paquet est envoyé par le générateur. Le compteur *nbpkt* peut être lu grâce à l'interface *RAM* de la *puce d'évaluation*. Pour le bloc *TGA* connecté sur le port 0, le compteur *nbpkt* se trouve à l'adresse 0x018.

2.3.2 L'automate de génération de paquets

Cet automate est responsable de la génération des paquets, en respectant les paramètres *dl*, *fgap* et *mask* qui lui sont transmis par le bloc *CC*. Il met également à jour le compteur *nbpkt*. Il contient 4 états qui sont : *idle*, *body_state*, *ep_state* et *wait* (Figure 58). Les tâches exécutées dans chaque état sont décrites ci-dessous.

Dans l'état *idle* :

- mise à jour du registre *dl_tg* si *stop* = '1'. Le signal *stop* est à la valeur logique '0' si le signal *reset* vaut '0' et si la macro-cellule *SPIN32* peut consommer des mots. Sinon, il est à la valeur logique '1'. Le registre *dl_tg* contient le décompte du nombre de mots du paquet en cours d'envoi ;
- envoi de l'en-tête du paquet si *stop* = '0' ;
- envoi d'un signal indiquant que la donnée envoyée est valide : le signal *dv* est mis à la valeur logique '1' si *stop* = '0'. Le signal *dv* valide le mot envoyé ;
- initialisation du *CRC* si *stop* = '0' ;
- incrémentation du *LFSR* qui détermine la destination *dest* du paquet et la valeur *vgap* si *stop* = '0'. La valeur *vgap* est la valeur moyenne de la bulle. La bulle représente le nombre de cycles insérés entre l'envoi deux paquets consécutifs ;
- mise à jour du compteur *nbpkt* si la macro-cellule *SPIN32* peut consommer des mots.

Dans l'état *body_state* :

- envoi d'un mot constituant le corps du paquet,
- envoi d'un signal indiquant que la donnée envoyée est valide,
- incrémentation du *CRC* si la macro-cellule *SPIN32* a consommé le mot,

- décrémentation et mise à jour du registre *dl_tg* si la macro-cellule *SPIN32* a consommé le mot,
- calcul de *gap_tg = fgap + vgap*. Le registre *gap_tg* contient un décompte de la bulle insérée entre l'envoi de deux paquets consécutifs,
- comparaison de la valeur du registre *dl_tg* à zéro.

Dans l'état *ep_state* :
- mise à jour du registre *gap_tg*,
- envoi de la fin du paquet,
- envoi d'un signal indiquant que la donnée envoyée est valide.

Dans l'état *wait* :
- décrémentation et mise à jour du registre *gap_tg* si *stop* = '0',
- comparaison de la valeur du registre *gap_tg* à zéro.

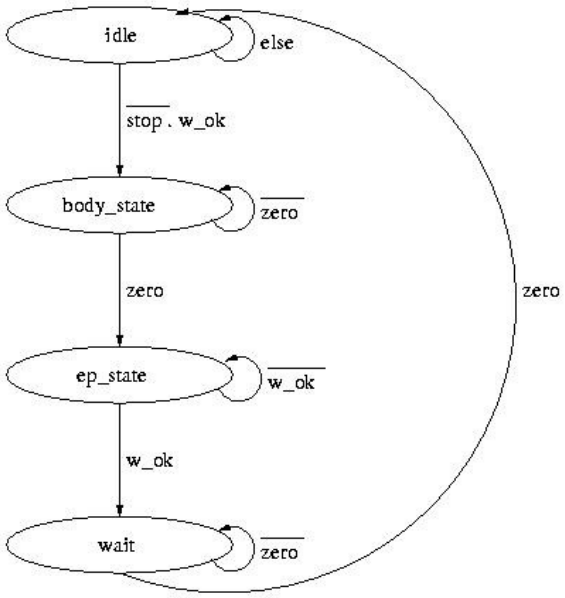

Figure 58 : Automate de génération de paquets

2.3.3 La partie analyse du bloc TGA

En réception, le rôle du bloc *TGA* est d'analyser les paquets sortant de la macro-cellule *SPIN32*. D'une part, il mesure la latence de chaque paquet et d'autre part, il vérifie l'intégrité des données contenues dans le paquet. Il vérifie également que le paquet arrive bien à sa destination après son passage dans la macro-cellule *SPIN32*.

Afin de vérifier l'intégrité des données, le bloc *TGA* doit calculer un *CRC* portant sur tous les mots du paquet qu'il reçoit, sauf le dernier. Il doit alors extraire le *CRC* transmis se trouvant dans le dernier mot du paquet et le comparer au *CRC* qui vient d'être calculé. L'égalité des deux *CRC* implique, d'une part, l'intégrité des données et, d'autre part, que le paquet arrive bien à sa destination. Le bloc *TGA* contient un registre *crc_ta* permettant le calcul du *CRC* lors de l'analyse du paquet.

Pour calculer la latence, le bloc *TGA* doit extraire la date d'injection du paquet dans la macro-cellule *SPIN32*, qui se trouve dans le deuxième mot du paquet.

Pour pouvoir analyser les paquets sortant de la macro-cellule *SPIN32*, le bloc *TGA* contient 12 registres de mesure, qui peuvent être relus grâce à l'interface *RAM* de la *puce d'évaluation* :

- le compteur *lat_sum* contient la somme des latences de tous les paquets reçus. Il est incrémenté de la valeur de la latence de chaque paquet non corrompu reçu avant que le compteur de cycle n'atteigne la valeur zéro. Pour le bloc *TGA* qui est connecté sur le port 0, les 32 bits de poids fort se trouvent à l'adresse 0x00B, tandis que les 32 bits de poids faible se trouvent à l'adresse 0x00C (Tableau 6) ;
- le compteur *npktno* contient le nombre total de paquets non corrompus reçus après que le compteur de cycle ait atteint la valeur zéro. Il est incrémenté d'une unité à chaque fois qu'un paquet non corrompu est reçu. Pour le bloc *TGA* qui est connecté sur le port 0, le compteur *npktno* se trouve à l'adresse 0x00D (Tableau 6) ;
- le compteur *npktok* contient le nombre total de paquets non corrompus reçus avant que le compteur de cycle n'atteigne la valeur zéro. Il est incrémenté d'une unité à chaque fois qu'un paquet non corrompu est reçu. Pour le bloc *TGA* qui est connecté sur le port 0, le compteur *npktok* se trouve à l'adresse 0x00E (Tableau 6) ;
- le compteur *npktko* contient le nombre total de paquets corrompus reçus. Il est incrémenté d'une unité à chaque fois qu'un paquet corrompu est reçu. La vérification du *CRC* permet de détecter les paquets corrompus. Il faut évidemment attendre le dernier mot du paquet pour vérifier le *CRC*. Pour le bloc *TGA* qui est connecté sur le port 0, compteur *npktko* se trouve à l'adresse 0x00F (Tableau 6) ;
- le registre *lat_max* contient la latence maximale observée. Pour chaque paquet reçu, la valeur de la latence est comparée à la valeur de la latence maximale observée. Si elle est supérieure, le registre *lat_max* est mise à jour. Pour le bloc *TGA* qui est connecté sur le port 0, registre *lat_max* se trouve à l'adresse 0x010 (Tableau 6) ;
- les registres *nb_ref0*, *nb_ref1*, *nb_ref2*, *nb_ref3*, *nb_ref4*, *nb_ref5* et *nb_ref6* contiennent le nombre de paquets dont la latence est comprise entre les différents intervalles de latence. Dans l'ordre : inférieure à *lat_ref0*, comprise entre *lat_ref0* et *lat_ref1*, comprise entre *lat_ref1* et *lat_ref2*, comprise entre *lat_ref2* et *lat_ref3*, comprise entre *lat_ref3* et *lat_ref4*, comprise entre *lat_ref4* et *lat_ref5*, et enfin supérieure à *lat_ref5*. Pour le bloc *TGA* qui est connecté sur le port 0, les registres *nb_ref0*, *nb_ref1*, *nb_ref2*, *nb_ref3*, *nb_ref4*, *nb_ref5* et *nb_ref6* se trouvent en ordre croissant entre les adresses 0x011 et 0x017 (Tableau 6).

Le bloc *TGA* connecté sur le port 0 contient également le *MISR* (*Multiple-Instruction Shift Register*) de signature de 36 bits utilisé pour le test *eulérien*. Les 4 bits de poids fort se trouvent à l'adresse 0x1CB et les 32 bits de poids faible à l'adresse 0x1CC.

Pour les blocs *TGA* connectés aux autres ports de la macro-cellule *SPIN32*, les adresses des registres internes sont dans le même ordre avec en plus, le compteur *nbpkt*. Les blocs *TGA* sont mis dans l'ordre croissant du numéro de port auquel ils sont connectés (Tableau 6).

	lat_sum	npktno	npktok	npktko	lat_max	nb_ref0	nb_ref1	nb_ref2	nb_ref3	nb_ref4	nb_ref5	nb_ref6	nbpkt
Port 0	0x00C	0x00D	0x00E	0x00F	0x010	0x011	0x012	0x013	0x014	0x015	0x016	0x017	0x018
Port 1	0x01A	0x01B	0x01C	0x01D	0x01E	0x01F	0x020	0x021	0x022	0x023	0x024	0x025	0x026
Port 2	0x028	0x029	0x02A	0x02B	0x02C	0x02D	0x02E	0x02F	0x030	0x031	0x032	0x033	0x034
Port 3	0x036	0x037	0x038	0x039	0x03A	0x03B	0x03C	0x03D	0x03E	0x03F	0x040	0x041	0x042
Port 4	0x044	0x045	0x046	0x047	0x048	0x049	0x04A	0x04B	0x04C	0x04D	0x04E	0x04F	0x050
Port 5	0x052	0x053	0x054	0x055	0x056	0x057	0x058	0x059	0x05A	0x05B	0x05C	0x05D	0x05E
Port 6	0x060	0x061	0x062	0x063	0x064	0x065	0x066	0x067	0x068	0x069	0x06A	0x06B	0x06C

Port 7	0x06E	0x06F	0x070	0x071	0x072	0x073	0x074	0x075	0x076	0x077	0x078	0x079	0x07A
Port 8	0x07C	0x07D	0x07E	0x07F	0x080	0x081	0x082	0x083	0x084	0x085	0x086	0x087	0x088
Port 9	0x08A	0x08B	0x08C	0x08D	0x08E	0x08F	0x090	0x091	0x092	0x093	0x094	0x095	0x096
Port 10	0x098	0x099	0x09A	0x09B	0x09C	0x09D	0x09E	0x09F	0x0A0	0x0A1	0x0A2	0x0A3	0x0A4
Port 11	0x0A6	0x0A7	0x0A8	0x0A9	0x0AA	0x0AB	0x0AC	0x0AD	0x0AE	0x0AF	0x0B0	0x0B1	0x0B2
Port 12	0x0B4	0x0B5	0x0B6	0x0B7	0x0B8	0x0B9	0x0BA	0x0BB	0x0BC	0x0BD	0x0BE	0x0BF	0x0C0
Port 13	0x0C2	0x0C3	0x0C4	0x0C5	0x0C6	0x0C7	0x0C8	0x0C9	0x0CA	0x0CB	0x0CC	0x0CD	0x0CE
Port 14	0x0D0	0x0D1	0x0D2	0x0D3	0x0D4	0x0D5	0x0D6	0x0D7	0x0D8	0x0D9	0x0DA	0x0DB	0x0DC
Port 15	0x0DE	0x0DF	0x0E0	0x0E1	0x0E2	0x0E3	0x0E4	0x0E5	0x0E6	0x0E7	0x0E8	0x0E9	0x0EA
Port 16	0x0EC	0x0ED	0x0EE	0x0EF	0x0F0	0x0F1	0x0F2	0x0F3	0x0F4	0x0F5	0x0F6	0x0F7	0x0F8
Port 17	0x0FA	0x0FB	0x0FC	0x0FD	0x0FE	0x0FF	0x100	0x101	0x102	0x103	0x104	0x105	0x106
Port 18	0x108	0x109	0x10A	0x10B	0x10C	0x10D	0x10E	0x10F	0x110	0x111	0x112	0x113	0x114
Port 19	0x116	0x117	0x118	0x119	0x11A	0x11B	0x11C	0x11D	0x11E	0x11F	0x120	0x121	0x122
Port 20	0x124	0x125	0x126	0x127	0x128	0x129	0x12A	0x12B	0x12C	0x12D	0x12E	0x12F	0x130
Port 21	0x132	0x133	0x134	0x135	0x136	0x137	0x138	0x139	0x13A	0x13B	0x13C	0x13D	0x13E
Port 22	0x140	0x141	0x142	0x143	0x144	0x145	0x146	0x147	0x148	0x149	0x14A	0x14B	0x14C
Port 23	0x14E	0x14F	0x150	0x151	0x152	0x153	0x154	0x155	0x156	0x157	0x158	0x159	0x15A
Port 24	0x15C	0x15D	0x15E	0x15F	0x160	0x161	0x162	0x163	0x164	0x165	0x166	0x167	0x168
Port 25	0x16A	0x16B	0x16C	0x16D	0x16E	0x16F	0x170	0x171	0x172	0x173	0x174	0x175	0x176
Port 26	0x178	0x179	0x17A	0x17B	0x17C	0x17D	0x17E	0x17F	0x180	0x181	0x182	0x183	0x184
Port 27	0x186	0x187	0x188	0x189	0x18A	0x18B	0x18C	0x18D	0x18E	0x18F	0x190	0x191	0x192
Port 28	0x194	0x195	0x196	0x197	0x198	0x199	0x19A	0x19B	0x19C	0x19D	0x19E	0x19F	0x1A0
Port 29	0x1A2	0x1A3	0x1A4	0x1A5	0x1A6	0x1A7	0x1A8	0x1A9	0x1AA	0x1AB	0x1AC	0x1AD	0x1AE
Port 30	0x1B0	0x1B1	0x1B2	0x1B3	0x1B4	0x1B5	0x1B6	0x1B7	0x1B8	0x1B9	0x1BA	0x1BB	0x1BC
Port 31	0x1BE	0x1BF	0x1C0	0x1C1	0x1C2	0x1C3	0x1C4	0x1C5	0x1C6	0x1C7	0x1C8	0x1C9	0x1CA

Tableau 6 : Adresses des registres de mesures

2.3.4 L'automate d'analyse de paquets

Cet automate est responsable de la réception des paquets. Il doit calculer la latence et le *CRC* de chaque paquet entrant, mettre à jour les registres *lat_sum*, *npktno*, *npktok*, *npktko*, *lat_max*, *nb_ref0*, *nb_ref1*, *nb_ref2*, *nb_ref3*, *nb_ref4*, *nb_ref5* et *nb_ref6* tant que le compteur de cycles ne contient pas la valeur zéro. Afin de ne pas fausser les mesures, cet automate doit pouvoir analyser chaque paquet sans aucune attente et ne pas ralentir la macro-cellule *SPIN32* en créant de la contention à ses ports de sortie.
Puisque la longueur minimale d'un paquet est de 3 mots, cet automate doit contenir un maximum de 3 états pour analyser les paquets sans provoquer de contentions. Les états sont : *idle*, *body_state* et *ep_state* (Figure 59). Leurs équivalents pour les paquets contenant des erreurs de transmission sont : *idle_error*, *body_error* et *ep_error*. Les tâches qui s'exécutent dans chaque état sont décrites ci-dessous.

Dans l'état *idle* :

- détection de la première source d'erreur. Une erreur est détectée si le premier mot reçu n'est pas un début de paquet,
- initialisation du *CRC* s'il y a un paquet qui sort de la macro-cellule *SPIN32*,
- consommation d'un mot s'il y a un paquet qui sort de la macro-cellule *SPIN32*,
- mise à jour des registres *lat_sum*, *npktok*, *bp_1*, *lat_max* et *dl_ta* si un paquet sort de la macro-cellule *SPIN32*. Le registre *bp_1*, sur 1 bit, prend la valeur logique '1' si le mot est le deuxième mot du paquet. Sinon, il prend la valeur logique '0'. Le registre *dl_ta* contient le décompte du nombre de mots déjà lus du paquet en cours de lecture.

Dans l'état *idle_error*, l'automate exécute les mêmes tâches que dans l'état *idle* sauf que le seul registre mise à jour est le registre *npktko*.

Dans l'état *body_state* :

- détection de la deuxième source d'erreur. Une erreur est détectée si le mot reçu est un début ou une fin de paquet,
- mise à jour du registre *bp_1*. Ce registre, sur 1 bit, prend la valeur logique '1' si le mot est le deuxième mot du paquet. Sinon, il prend la valeur logique '0',
- calcul de la latence,
- mise à jour du registre *lat* si *bp_1* = 1. Le registre *lat* contient la latence du paquet en cours de lecture,
- consommation d'un mot s'il y a un paquet qui sort de la macro-cellule *SPIN32*,
- incrémentation du *CRC*,
- décrémentation et mise à jour du registre *dl_ta*,
- comparaison de la valeur du registre *dl_ta* avec zéro si *bp_1* = '0'.

Dans l'état *body_error*, l'automate exécute les mêmes tâches que dans l'état *body_state*.

Dans l'état *ep_state* :

- consommation d'un mot si un paquet sort de la macro-cellule *SPIN32*,
- détection de la troisième source d'erreur. Une erreur est détectée si le mot reçu n'est pas une fin de paquet,
- détection de la quatrième source d'erreur. Une erreur est détectée si le *CRC* reçu n'est pas égal au *CRC* calculé localement,
- calcul de la somme des latences,
- comparaison de la valeur du registre *lat_max* avec la valeur du registre *lat*,
- mise à jour des registres *lat_sum*, *bp_1* et *lat_max*,
- incrémentation et mise à jour des registres *npktno*, *npktok*, *nb_ref0*, *nb_ref1*, *nb_ref2*, *nb_ref3*, *nb_ref4*, *nb_ref5* et *nb_ref6* selon les cas.

Dans l'état *ep_error*, l'automate exécute les mêmes tâches que dans l'état *ep_state* mais seul le registre *npktko* est incrémenté et mise à jour.

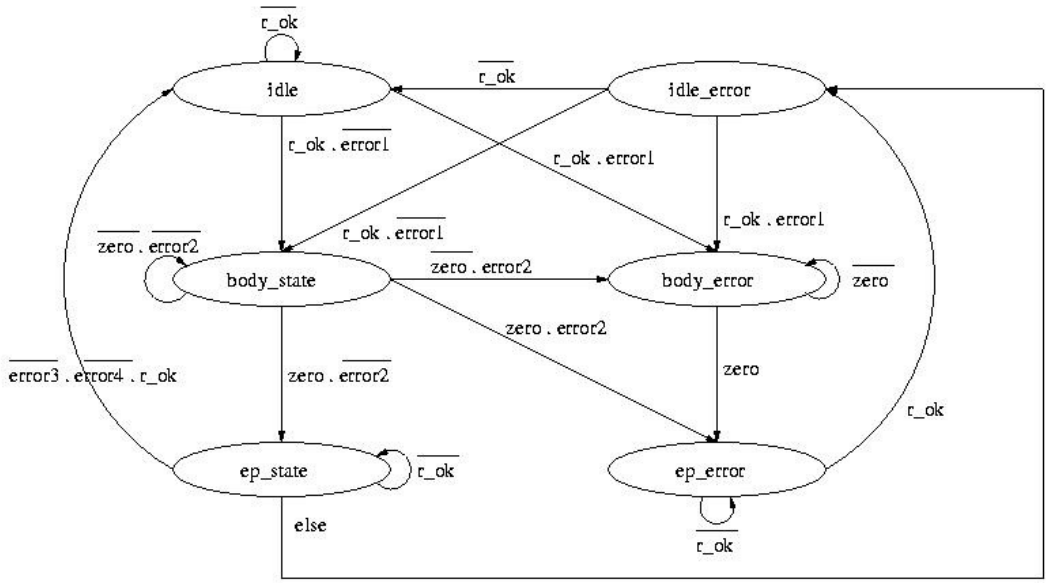

Figure 59 : Automate d'analyse de paquets

2.3.5 Variables aléatoires

Les valeurs *dest* et *vgap* sont des variables aléatoires. Elles sont réalisées à l'aide d'un *LSFR* (*Linear Feedback Shift Register*). Pour cela, chaque bloc *TGA* contient un *LSFR* sur 32 bits. Les 5 bits de poids forts sont utilisés pour désigner une destination *dest* au hasard, tandis que les *m* bits de poids faible sont utilisés pour désigner la valeur de la variable aléatoire *vgap*.

Le *LFSR* est initialisé par le bloc *CC*. L'initialisation doit évidemment être différente pour chaque bloc *TGA*, de façon à éviter les corrélations entre les différents blocs *TGA*. Concrètement, les 5 bits de poids fort, qui représentent la destination, doivent être différents pour chaque bloc *TGA* au début de la mesure.

2.4 Le bloc CC

Le bloc *CC* (*Central Controller*) permet de définir les valeurs des paramètres durant une session de mesures en initialisant les registres correspondants. L'initialisation se fait en entrant les valeurs des paramètres grâce à l'interface *RAM*. Pour cela, nous plaçons la *puce d'évaluation* en mode d'écriture (mode *write*) à l'aide du signal mode. Une fois les registres initialisés avec les bonnes valeurs, le rôle du bloc *CC* est d'envoyer les valeurs de ces paramètres vers tous les blocs *TGA*.

Le bloc *CC* contient les registres correspondants aux paramètres suivants :

- le temps *time*, codé sur 32 bits. Le temps *time* est le compteur de cycles qui représente la date pendant les mesures. Il se trouve à l'adresse 0x001 (Tableau 7) ;
- la longueur *dl*, codée sur 20 bits. Elle est initialisée à une valeur quelconque, comprise entre 0 et 2^{20-1}. Sa valeur reste constante pendant toute la mesure. La longueur du paquet s'obtient en rajoutant 3 à cette valeur (*dl* + 3). Le registre *dl* se trouve à l'adresse 0x002 (Tableau 7) ;
- le masque *mask*, codé sur 10 bits. Les *m* bits de poids faible doivent être initialisés à la valeur logique '1', et les 10-*m* bits de poids fort à la valeur logique '0'. Cette valeur reste constante pendant la mesure. Elle permet la définition de la charge sur la macro-cellule *SPIN32*. Le masque se trouve à l'adresse 0x003 (Tableau 7) ;
- la valeur *fgap*, codée sur 20 bits. Elle est initialisée à une valeur quelconque, comprise entre 0 et 2^{20-1}. Cette valeur permet l'ajustement de la valeur de la charge sur la macro-cellule *SPIN32*. Elle reste constante pendant toute la mesure et se trouve à l'adresse 0x004 (Tableau 7) ;
- les références de latence qui déterminent les différents intervalles de latences. Le nombre de paquets dont la latence est comprise dans ces intervalles sera déterminé. Les 6 références de latence se trouvent en ordre croissant entre les adresses 0x005 et 0x00A (Tableau 7).

Le compteur de cycles *time* est initialisé à la valeur *t_max*, par l'intermédiaire de l'interface *RAM* de la *puce d'évaluation*. Le compteur de cycles *time* est décrémenté à chaque cycle dès que le signal *reset* prend la valeur logique '0'. La valeur *t_max* représente la durée de la mesure. La durée maximale d'une mesure est de 4 milliards de cycles, soit un temps de mesure de 20 secondes pour une fréquence de fonctionnement de 200 MHz.

adresse	registre
0x000	*reset*
0x001	*time*
0x002	*dl*
0x003	*mask*
0x004	*fgap*
0x005	*lat_ref0*
0x006	*lat_ref1*
0x007	*lat_ref2*
0x008	*lat_ref3*
0x009	*lat_ref4*
0x00A	*lat_ref5*

Tableau 7 : Adresse des registres de paramètres à travers l'interface RAM

Le *gap* représente le nombre de cycles que le bloc *TGA* insère entre deux paquets consécutifs. Dans chaque bloc *TGA*, la valeur du *gap* est la somme d'une constante et d'une variable aléatoire donnée par l'Equation 5.

$$gap = fgap + vgap$$

Equation 5 : Nombre de cycles entre l'injection de deux paquets

La valeur *fgap* est une constante identique pour tous les blocs *TGA*. Codée sur 8 bits, elle est stockée dans un registre du bloc *CC*. Le registre *fgap* est accessible par l'intermédiaire de l'interface *RAM* de la *puce d'évaluation.* La constante *fgap* est envoyée à tous les blocs *TGA* par le bloc *CC*. Elle permet d'ajuster finement la valeur moyenne *gapm* et donc la valeur de la charge *load* injectée dans la macro-cellule *SPIN32*.

La valeur *vgap* est une variable aléatoire, spécifique à chaque bloc *TGA*. Codée sur *m* bits, sa valeur est uniformément distribuée entre 0 et 2^{m-1}. La valeur de *m* est définie par un mot de 10 bits qui sert de masque, et dont les *m* bits de poids faible sont à la valeur logique '1'. Elle est stockée dans le registre *mask* du bloc *CC* et elle est envoyée aux 32 blocs *TGA*. Le registre *mask* est accessible par l'intermédiaire de l'interface *RAM* de la *puce d'évaluation.*

2.5 Floorplan de la puce d'évaluation

La Figure 60 représente le *floorplan* de la puce d'évaluation. Cette figure a été fournit par *STMicroelectronics*. La macro-cellule *SPIN32* est représentée en vert au milieu et les *TGA* (*Traffic Generator and Analyzer*) sont représentés en rouge. La puce d'évaluation occupe une surface de 22.64 mm² (6240 µm * 3628 µm).

2.6 Résultats des mesures

La *puce d'évaluation* (Figure 61) a été effectivement fabriquée chez *STMicroelectronics*. On voit distinctement la macro-cellule *SPIN32* au milieu de la *puce d'évaluation*.

Malheureusement, nous n'avons pas pu effectuer les mesures que nous avions prévues avant la soutenance de cette thèse. Elles feront probablement l'objet d'une future publication.

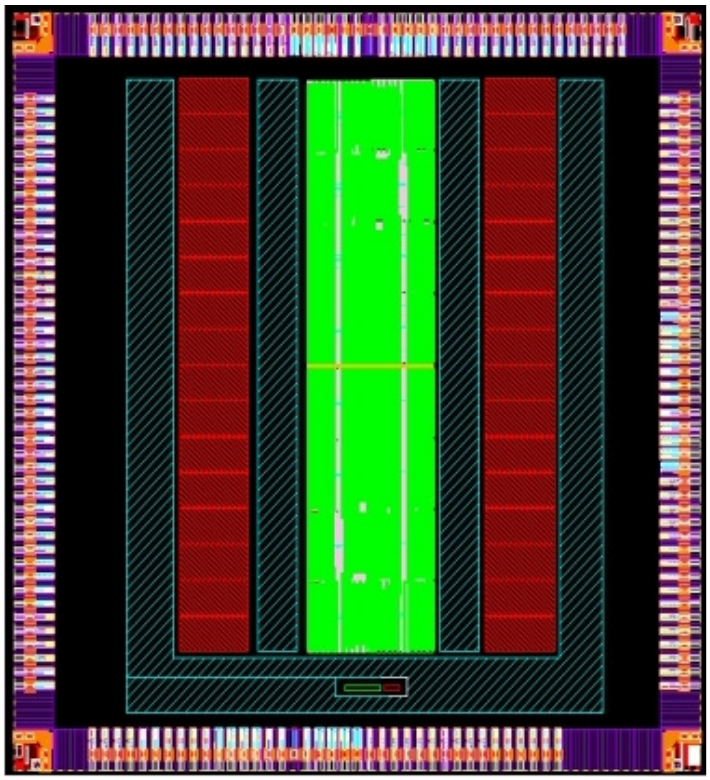

Figure 60 : Floorplan de la puce d'évaluation

3 Conclusion

Ce chapitre détaille la conception d'un micro-réseau *SPIN* à 32 ports comportant 16 routeurs *RSPIN*. Ce micro-réseau se présente comme une macro-cellule ''dure'', placée, routée et caractérisée occupant une surface de 4.6 mm² pour un procédé de fabrication CMOS 0.13 µm. Nous avons prouvé la faisabilité matérielle d'un micro-réseau *SPIN*. Nous avons, entre autres, prouvé que les nombreuses nappes de fils ne sont pas un facteur limitant en ce qui concerne la surface de la macro-cellule *SPIN32*. Enfin, la caractérisation de la macro-cellule *SPIN32* permet de garantir une fréquence de fonctionnement de 192 MHz dans le pire cas de dispersion des caractéristiques technologiques du procédé de fabrication.
Ce chapitre détaille également la conception d'une *puce d'évaluation*, contenant une macro-cellule *SPIN32*, entourée de toute la logique nécessaire à son instrumentation. L'instrumentation consiste d'abord à tester la macro-cellule *SPIN32* puis, à caractériser sa fiabilité, sa latence, sa consommation et sa bande passante. Pour ce faire, la logique d'instrumentation permet d'injecter et d'analyser différents types de trafics dans la macro-cellule *SPIN32*. Nous avons conçu toute la logique d'instrumentation et nous avons fourni les modèles comportementaux, ainsi que la macro-cellule *SPIN32* au format *GDS II* à *STMicroelectronics*. Le *layout* de la *puce d'évaluation* occupe une surface de 22.64 mm².

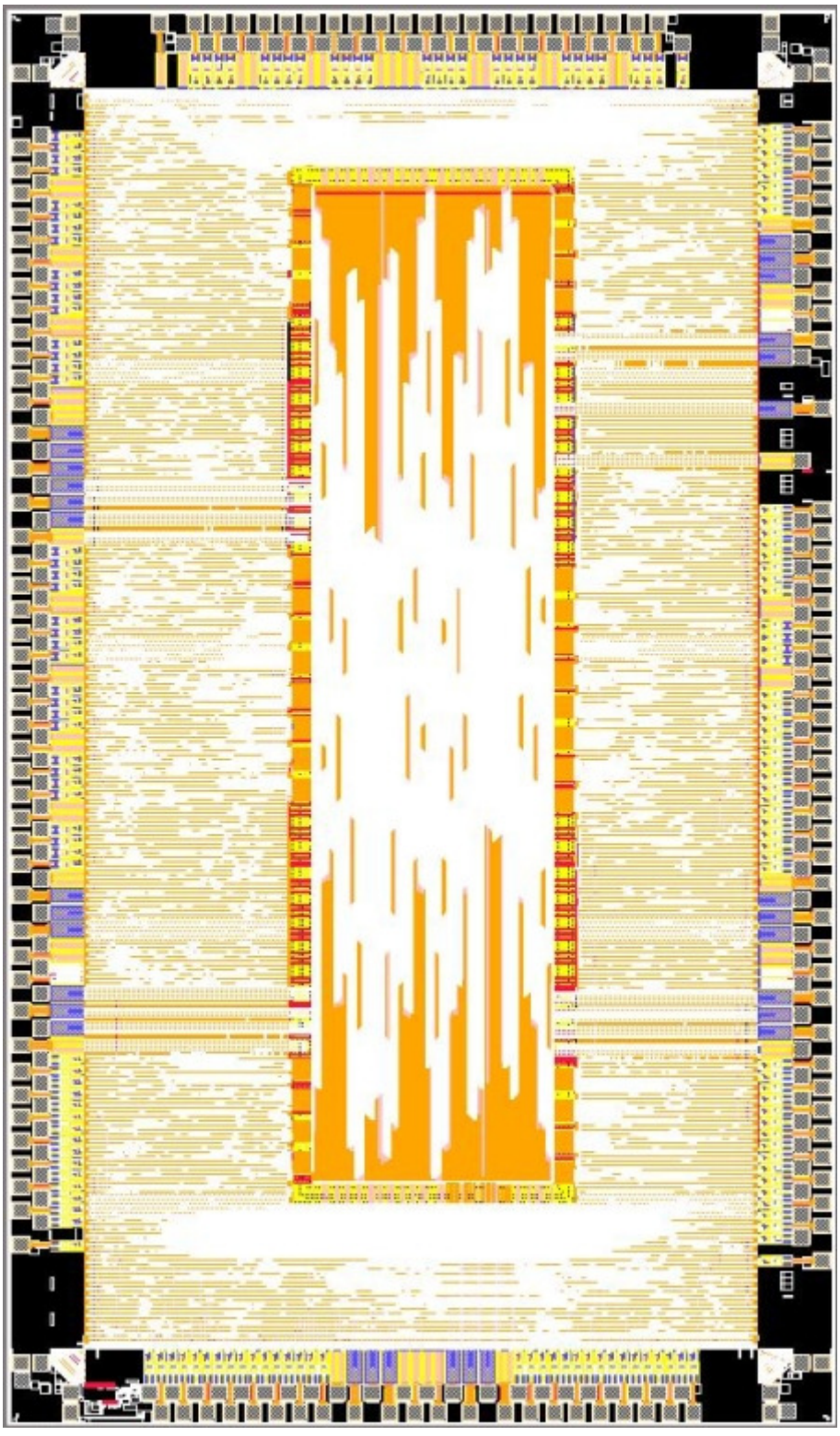

Figure 61 : Layout de la puce d'évaluation

Chapitre 9 *Modélisation SystemC*

1 Modèle systemC

En attendant l'implémentation matérielle de la *puce d'évaluation*, nous l'avons modélisée en *systemC* pour estimer les performances de la macro-cellule *SPIN32*. Le modèle *systemC* de la *puce d'évaluation* a été conçu de façon à être un modèle *CABA* (*Cycle Accurate Bit Accurate*), précis au cycle et au bit près.

1.1 Organisation du modèle

Les modèles *systemC* du générateur et analyseur de trafic *TGA*, du routeur *RSPIN* et du contrôleur central *CC* sont les modèles *systemC* de base. La Figure 62 représente la hiérarchie à 3 niveaux du modèle *systemC* de la *puce d'évaluation* :

- la *puce d'évaluation* est au plus haut niveau de la hiérarchie. Elle est composée de 3 modèles (*CC*, *TGA* et *SPIN32*) ;
- le contrôleur central *CC* est instancié une seule fois ;
- le générateur et analyseur de trafic *TGA* est instancié 32 fois, à raison d'un *TGA* sur chaque port ;
- la macro-cellule *SPIN32* est instanciée une fois. Elle est composée de 16 instances de routeurs *RSPIN* ;

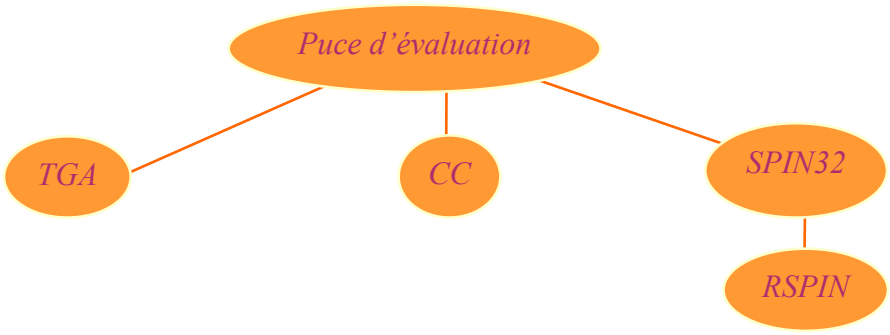

Figure 62 : Organisation de modèle systemC

1.2 Définition de la latence

La simulation du modèle *systemC* ne permet pas d'évaluer la fréquence de fonctionnement. Elle ne permet pas, non plus, de mesurer la consommation ni la fiabilité de la macro-cellule

SPIN32 sur silicium. En revanche, la simulation du modèle *systemC* permet de faire des mesures de latence sous différentes conditions de charge.
Le modèle *systemC* a été conçu pour permettre la mesure de deux types de latence : la latence du micro-réseau *SPIN*, appelée *temps de traversée*, et la latence vue par le composant, appelée *latence effective*. Le *temps de traversée* définit le nombre de cycles effectivement passés par le paquet à l'intérieur du micro-réseau *SPIN*. Il correspond au temps mis par le paquet depuis son entrée dans le micro-réseau *SPIN* jusqu'à sa sortie. La *latence effective* définit le nombre de cycles entre l'envoi du paquet par le composant émetteur jusqu'à sa réception par le composant destinataire. La *latence effective* comprend 3 temps :

- le *temps d'attente à l'envoi*, représentant le nombre de cycles entre le moment où le composant émetteur veut envoyer un paquet et le moment où le micro-réseau *SPIN* lui permet effectivement de l'envoyer ;
- le *temps de traversée*, représentant le nombre de cycles effectivement passés par le paquet à l'intérieur du micro-réseau *SPIN* ;
- le *temps d'attente à la réception*, représentant le nombre de cycles entre le moment où un paquet sort du micro-réseau *SPIN* et le moment où il commence effectivement à être analyser par le composant destinataire.

Le *temps d'attente à l'envoi* est nul quand la macro-cellule *SPIN32* accepte les paquets qui arrivent sur ses entrées, c'est-à-dire tant qu'il n'y a pas de contention dans la macro-cellule *SPIN32*. Le *temps d'attente à l'envoi* reste faible tant qu'il n'y a pas de saturation dans la macro-cellule *SPIN32*. Par contre, en cas de saturation du micro-réseau, le *temps d'attente à l'envoi* augmente brusquement puisque la contention se rétro-propage vers le *TGA* émetteur.

Le *temps d'attente à la réception* est toujours nul car la partie analyse des *TGA* a été conçue pour pouvoir consommer chaque mot qui sort de la macro-cellule *SPIN32*, même dans le cas d'un débit maximal de un mot par cycle. De cette manière, l'analyse des paquets ne constitue pas un goulot d'étranglement. Hors de la zone de saturation, la *latence effective* et le *temps de traversée* restent donc très voisins.

1.3 Objectifs de la simulation

L'objectif de cette simulation est de caractériser la valeur moyenne de la *latence effective* en fonction de la charge. Nous voulons mettre en évidence le seuil de saturation de la macro-cellule *SPIN32* ; qui est la valeur de la charge pour laquelle la valeur moyenne de la *latence effective* augmente d'une manière soudaine.
Le second objectif est de déterminer l'effet de plusieurs paramètres sur le seuil de saturation, ainsi que sur la valeur moyenne du *temps de traversée*. Ces paramètres sont :

- la localité du trafic,
- la longueur des paquets,
- l'utilisation ou non de la queue centrale,
- l'activation du mécanisme de prévention de l'interblocage (*deadlock*),
- le routage déterministe.

Le modèle *systemC* du *TGA* peut générer 4 types de trafic selon sa localité : *trafic très local*, *trafic moyennement local*, *trafic faiblement local* et *trafic aléatoire*. Il peut également générer 2 types de paquets : les paquets marqués comme pouvant utiliser les queues centrales et ceux

qui ne les utilisent pas. Enfin, le modèle *systemC* du *TGA* peut générer des paquets de différentes tailles. Le modèle *systemC* de la macro-cellule *SPIN32* possède plusieurs paramètres de configuration (absence de *deadlock*, acheminement dans l'ordre chronologique, …)

1.4 Méthode utilisée

Pour tracer chaque courbe, nous avons effectué 32 simulations. Nous avons configuré les 32 *TGA* pour qu'ils injectent tous la même charge dans la macro-cellule *SPIN32*. Tous les *TGA* injectent des paquets de 16 mots. Entre deux simulations, nous avons fait varier la *charge fournie* par chaque *TGA*. Pour les 32 simulations, la *charge fournie* varie entre 5 % et 100 %. Chaque simulation se fait sur 55039 cycles d'horloge. Sur une machine possédant un processeur de 1 GHz et 1 Go de *RAM*, une simulation dure environ 45 minutes.

1.5 Résultats des simulations

1.5.1 Charge acceptée

Dans la zone de saturation, la macro-cellule *SPIN32* n'est plus en mesure d'accepter la *charge fournie* par les *TGA* à cause de la contention qui y règne. Cela nous amène à définir deux charges : la *charge fournie* et la *charge acceptée*.
La *charge fournie* est la charge que chaque *TGA* essaie d'injecter dans la macro-cellule *SPIN32*. Sa valeur est donnée ci-dessous, où *pl* est la longueur du paquet et *gapm* est la valeur moyenne du nombre de cycles insérés entre deux paquets consécutifs.

$$load_{provided} = \frac{pl}{pl + gapm}$$

La *charge acceptée* est la charge que la macro-cellule *SPIN32* accepte et qui est effectivement acheminée par celle-ci. Sa valeur est donnée ci-dessous, où *nbpkt* est le nombre de paquets et *time* est la durée de la simulation (nombre de cycles d'horloge).

$$load_{accepted} = \frac{nbpkt * pl}{time}$$

Les 32 simulations nous permettent de tracer la courbe de la Figure 63. Cette courbe représente en ordonnée la *charge acceptée* et en abscisse la *charge fournie*. Elle met en évidence qu'au moment de la saturation, la *charge acceptée* cesse effectivement de croître avec la *charge fournie*. Le seuil de saturation se situe autour d'une *charge fournie* d'environ 52 %.

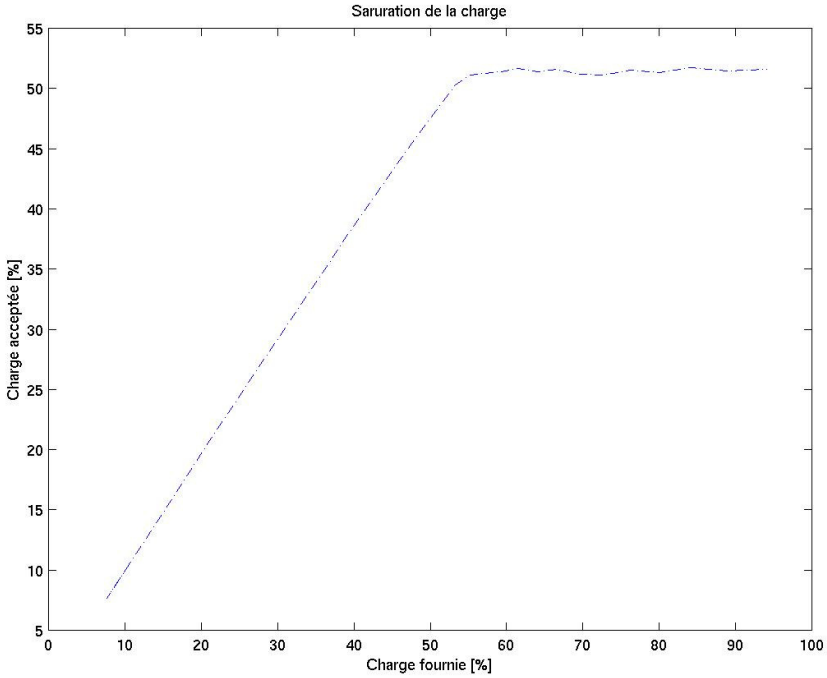

Figure 63 : Charge acceptée en fonction de la charge fournie

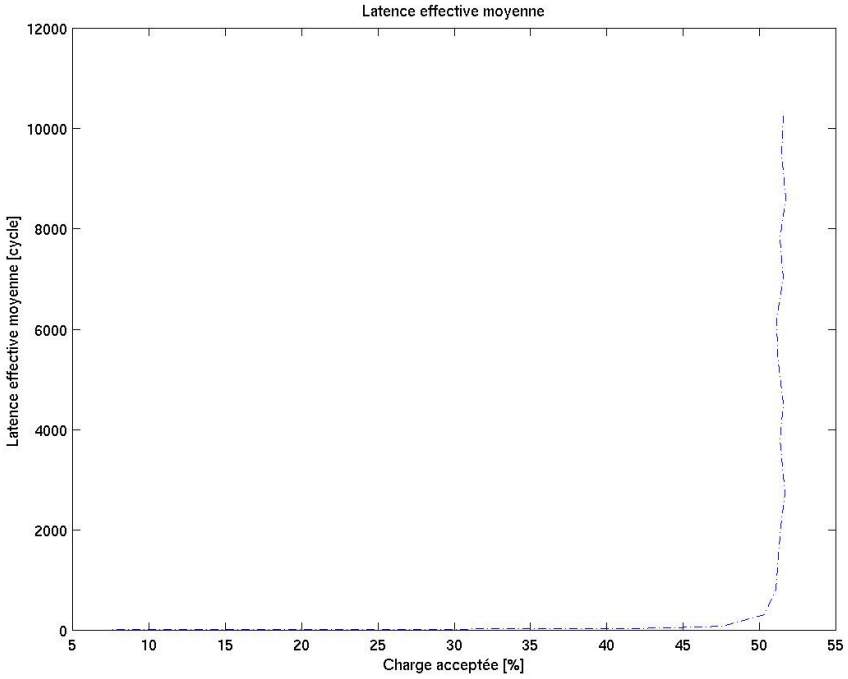

Figure 64 : Latence effective moyenne en fonction de la charge acceptée

1.5.2 Latence moyenne

Nous voulons caractériser la latence moyenne en fonction de la *charge acceptée*. La latence moyenne est calculée en divisant la somme des latences de tous les paquets (*lat_sum*) par le nombre de paquets (*nbpkt*) reçus par la partie analyseur de paquets de tous les *TGA*.

$$latence_{moyenne} = \frac{lat\_sum}{nbpkt}$$

Les simulations effectuées permettent de tracer la courbe de la Figure 64, dans laquelle l'ordonnée représente la valeur moyenne de la *latence effective* et l'abscisse représente la *charge acceptée*. La courbe de la Figure 64 confirme que le seuil de saturation se trouve bien autour d'une *charge acceptée* d'environ 52 %. En effet, au-delà de cette limite, la valeur moyenne de la *latence effective* augmente brusquement. La courbe de la Figure 65 est un agrandissement de celle de la Figure 64. Elle montre qu'en dehors de la zone de saturation, la valeur moyenne de la latence effective est d'environ 20 cycles.

La Figure 66 nous montre le *temps de traversée* de la macro-cellule *SPIN32* en fonction de la *charge acceptée*. Cette courbe nous confirme qu'effectivement la valeur moyenne du *temps de traversée* et celle de la *latence effective* restent très voisines (moins de 30 cycles de différence), tant que la *charge acceptée* reste éloignée de la zone de saturation (inférieure à 40 %). La courbe de la Figure 66 nous montre surtout que, contrairement à la valeur moyenne de la *latence effective*, la valeur moyenne du *temps de traversée* ne dépasse pas 42 cycles. Cette valeur est atteinte au moment où la *charge acceptée* franchit le seuil de saturation.

1.5.3 Coût de la garantie de l'absence d'interblocage

Nous avons établi que la seule façon de garantir l'absence d'interblocage (*deadlock*) dans le micro-réseau *SPIN*, lorsque le trafic est du type *requête/réponse* (*VCI*, *OCP*, …), était de séparer les chemins utilisés par les paquets *requêtes* de ceux utilisés par les paquets *réponses*. Lors de ces simulations, nous avons voulu quantifier le coût de cette garantie sur le seuil de saturation de la macro-cellule *SPIN32*. Les deux courbes de la Figure 67 ont en abscisse la *charge acceptée* et en ordonnée la valeur moyenne de la *latence effective*. La première courbe représente la valeur moyenne de la *latence effective* en fonction de la *charge acceptée*, lorsqu'on ne distingue pas les paquets *requêtes* des paquets *réponses*. La seconde représente exactement la même chose mais cette fois-ci avec la garantie de l'absence de l'interblocage. La Figure 67 montre que le seuil de saturation baisse d'environ 3 % et se retrouve vers 49 % de la charge acceptée. Le coût de cette garantie n'est donc pas exorbitant.

1.5.4 Coût du routage adaptatif

Dans son en-tête, un paquet *SPIN* possède un bit appelé *in_order*. Lorsque ce bit est activé sur plusieurs paquets venant du même composant, le micro-réseau *SPIN* les achemine dans l'ordre chronologique. L'acheminement des paquets dans l'ordre chronologique est une spécificité du micro-réseau *SPIN* dont nous avons voulu quantifier le coût sur le seuil de saturation. En abscisse de la Figure 68, nous représentons la *charge acceptée* et en ordonnée la valeur moyenne de la *latence effective*. La Figure 68 montre que l'acheminement des

paquets dans l'ordre chronologique ne dégrade presque pas le seuil de saturation, puisque la dégradation est inférieure à 0.5 %. Cela est un coût plus qu'acceptable par rapport à l'inconvénient de devoir remettre les paquets dans l'ordre au niveau du composant récepteur.

1.5.5 Gain fourni par l'utilisation des queues centrales

Les queues centrales ont été introduites dans le routeur *RSPIN* pour diminuer la contention dans le micro-réseau *SPIN*. En effet, un paquet dont le port de sortie est occupé est dirigé vers une queue centrale. Il libère ainsi le lien et la *FIFO* d'entrée en amont. Cela permet aux paquets suivants d'être acheminés vers leurs destinations sans attendre la libération du port voulu par le premier paquet. La Figure 69 représente en abscisse la *charge acceptée* et en ordonnée la valeur moyenne de la *latence effective*. Cette figure montre que le gain de l'utilisation des queues centrales est d'environ 5 %. En effet, le seuil de saturation passe de 47 % à environ 52 % lorsque nous utilisons les queues centrales. Ce gain de 5 % nous semble dérisoire par rapport au coût de ces queues centrales en terme de surface de silicium. En effet, rappelons que les queues centrales occupent 12 % de la surface totale de la macro-cellule *SPIN32*.

1.5.6 Localité du trafic

La *latence effective* et le *temps de traversée* dépendent évidemment de la localité du trafic. En effet, la *latence effective* et le *temps de traversée* pour un paquet faisant partie d'un trafic entre deux composants se trouvant dans le même *cluster* ne sont pas les mêmes que pour un paquet faisant partie d'un trafic entre deux composants très distants.
La Figure 70 nous permet de quantifier l'effet de la localité du trafic sur le seuil de saturation. Elle représente en abscisse la *charge acceptée* et en ordonnée la valeur moyenne de la *latence effective*. Nous distinguons 4 courbes sur la Figure 70. Elles représentent la valeur moyenne de la *latence effective* en fonction de la *charge acceptée* respectivement pour un *trafic aléatoire*, un *trafic faiblement local*, un *trafic moyennement local* et un *trafic très local*. La Figure 70 montre que le seuil de saturation est beaucoup plus élevé pour un *trafic moyennement local* (63 %) et pour un *trafic très local* (62 %). Cela nous conforte évidemment dans notre choix de placer les interlocuteurs fréquents dans un même *cluster*. La Figure 71 montre un agrandissement de la Figure 70. Elle nous confirme qu'en dehors de la zone de saturation, la valeur moyenne de la *latence effective* est de 5 cycles pour un *trafic très local*, 10 cycles pour un *trafic moyennement local*, 15 cycles pour un *trafic faiblement local* et 20 cycles pour un *trafic aléatoire*. La Figure 72 représente en abscisse la *charge acceptée* et en ordonnée la valeur moyenne du *temps de traversée*. Elle nous confirme que la *latence effective* et le *temps de traversée* restent dans le même ordre de grandeur lorsque la charge effective reste éloignée de la zone de saturation. Au-delà de la limite de saturation, la valeur moyenne du *temps de traversée* ne dépasse pas 18 cycles pour un *trafic très local*, 22 cycles pour un *trafic moyennement local*, 32 cycles pour un *trafic faiblement local* et 42 cycles pour un *trafic aléatoire*.

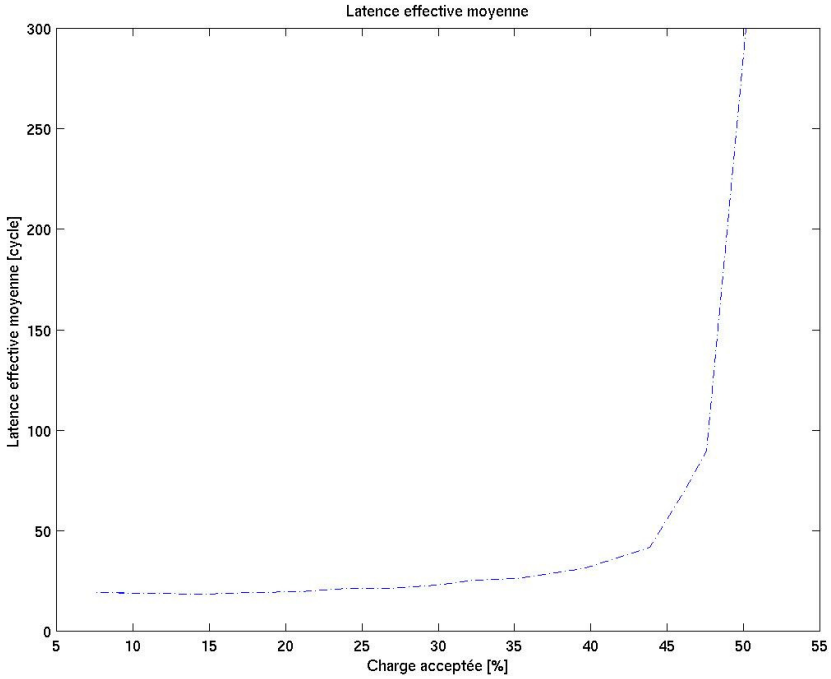

Figure 65 : Latence effective moyenne en fonction de la charge acceptée

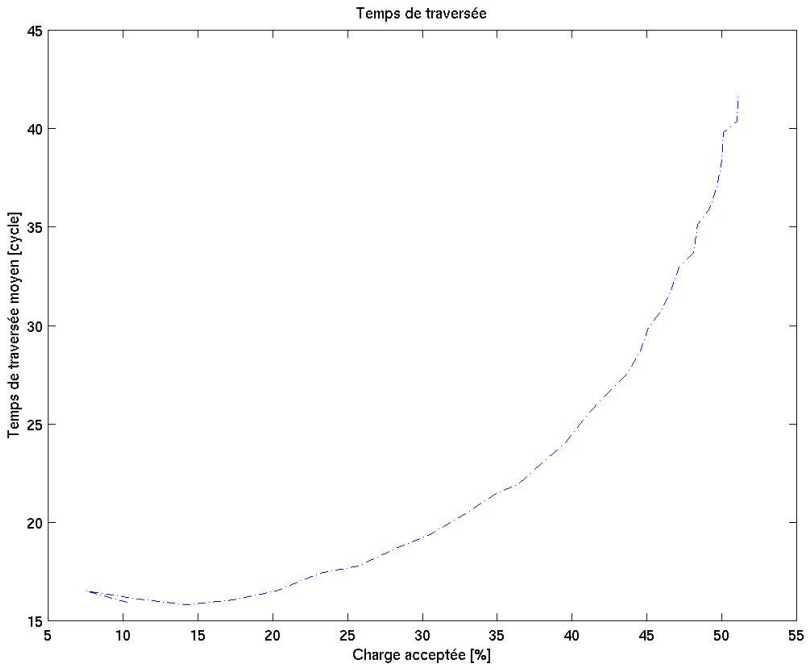

Figure 66 : Temps de traversée en fonction de la charge acceptée

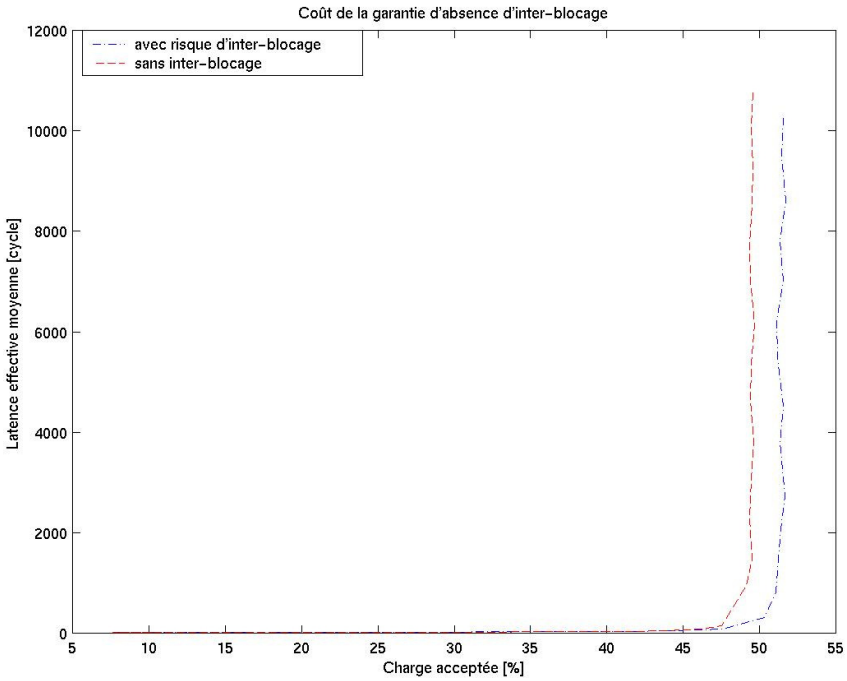

Figure 67 : Coût de l'absence d'interblocage sur le seuil de saturation

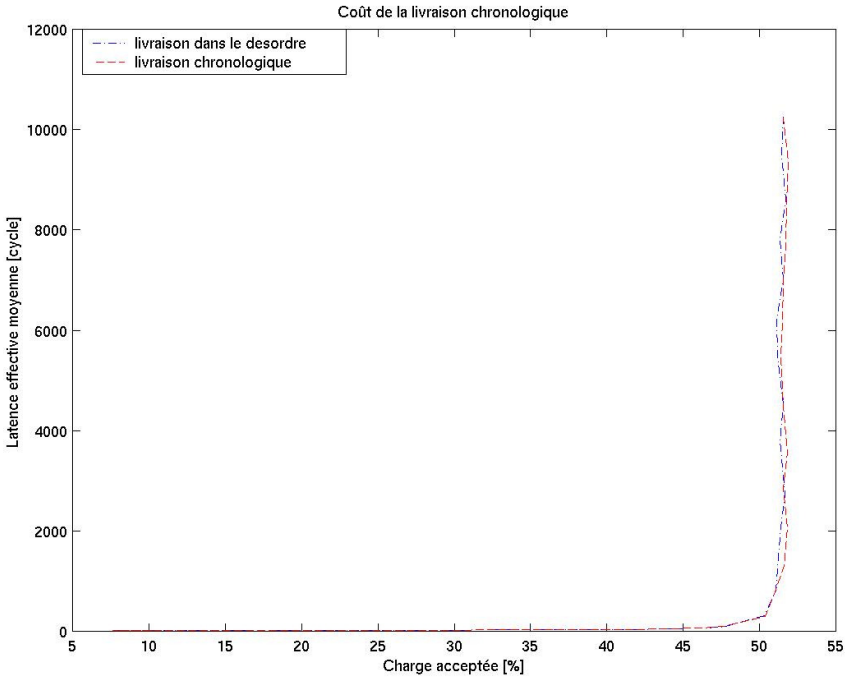

Figure 68 : Coût de l'acheminement des paquets dans l'ordre chronologique

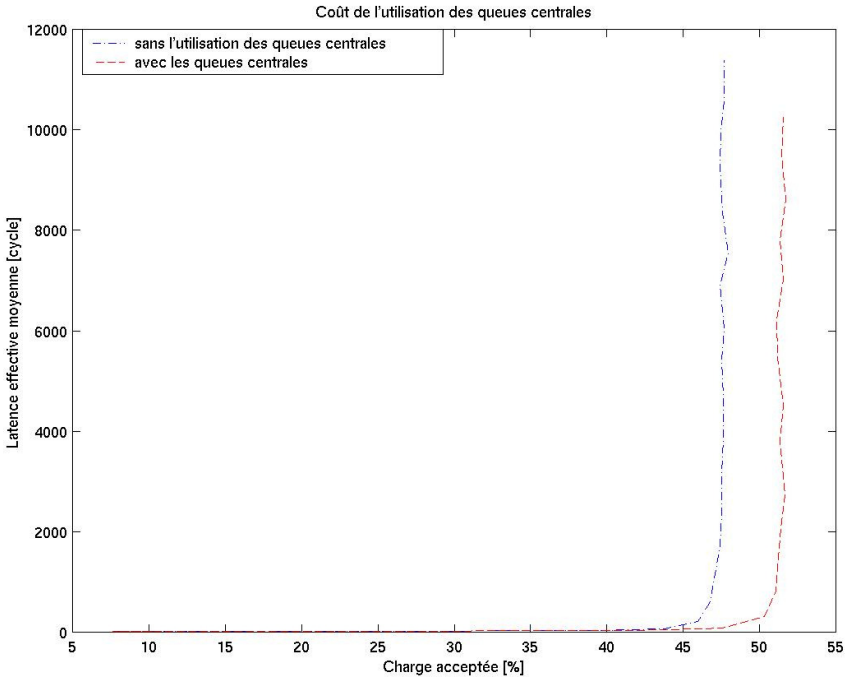

Figure 69 : Gain fourni par l'utilisation des queues centrales

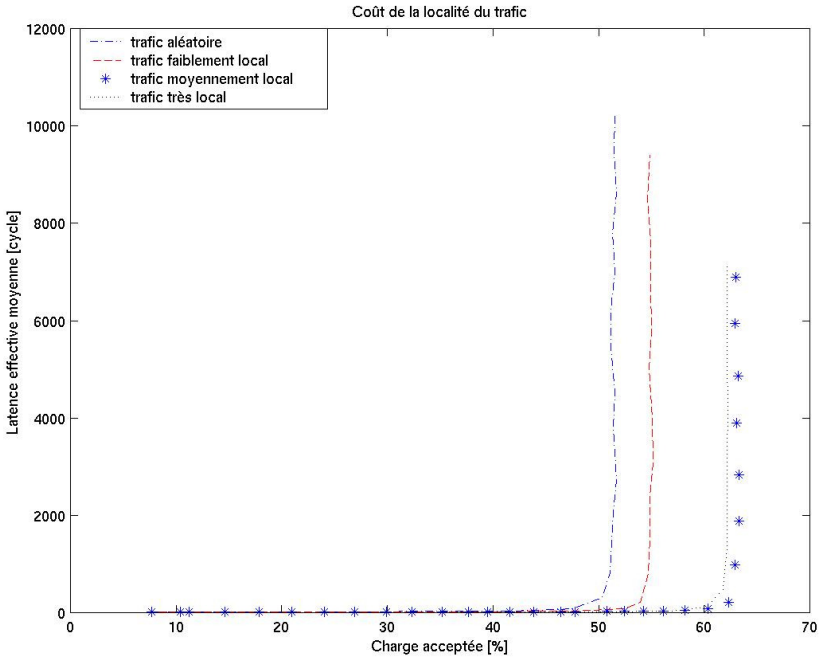

Figure 70 : Effet de la localité du trafic sur le seuil de saturation

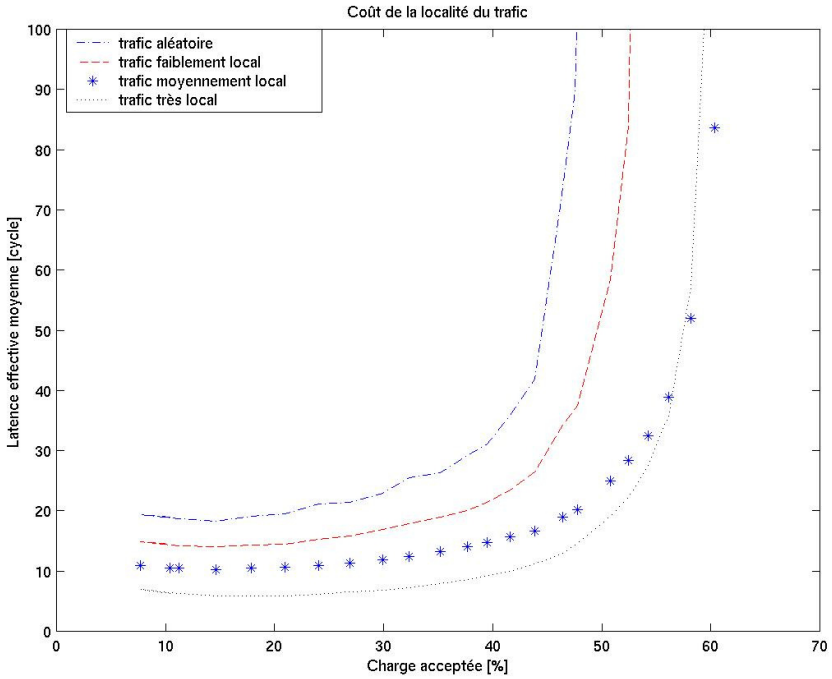

Figure 71 : Latence effective moyenne en fonction de la localité du trafic

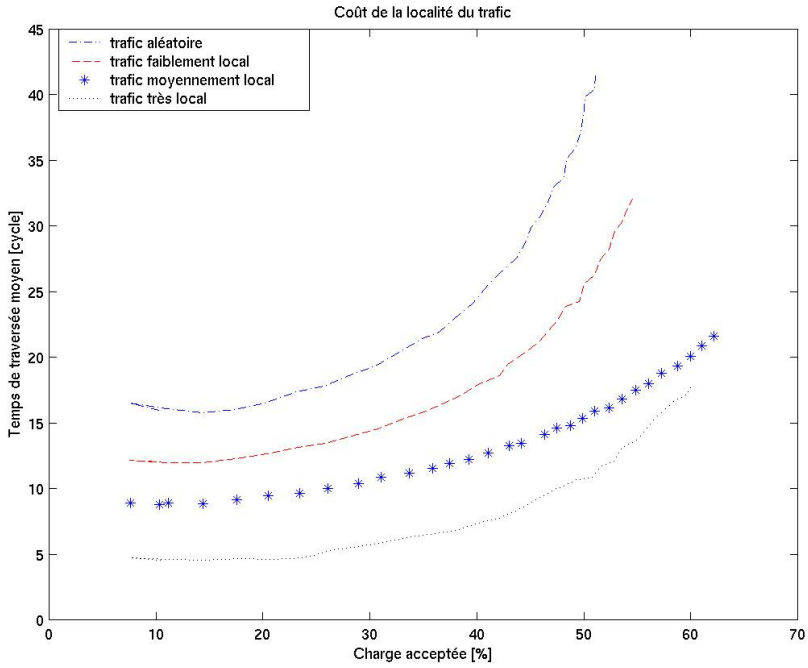

Figure 72 : Temps de traversée moyen en fonction de la localité du trafic

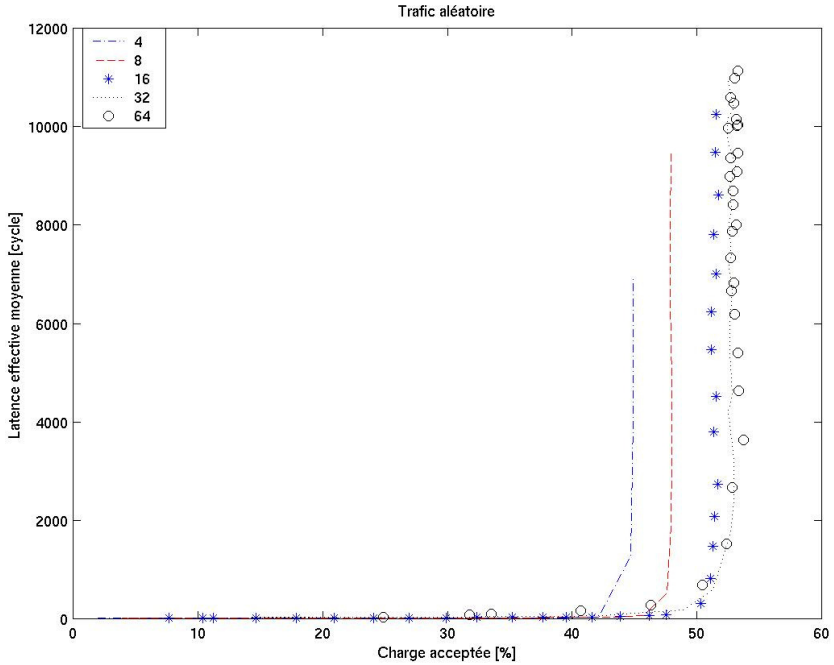

Figure 73 : Effet de la longueur des paquets sur le seuil de saturation

1.5.7 Longueur du paquet

Nous désirons maintenant quantifier l'effet de la *longueur des paquets* sur le seuil de saturation et sur la latence. Pour cela, nous avons fait varier la *taille des paquets* de 4 mots à 64 mots pour un *trafic aléatoire*. La Figure 73 représente en abscisse la *charge acceptée* et en ordonnée la valeur moyenne de la *latence effective*. Elle montre la valeur moyenne de la *latence effective* en fonction de la *charge acceptée* pour 5 longueurs de paquets différents (4, 8, 16, 32 et 64 mots). La Figure 73 nous indique, contrairement à une idée reçue, qu'augmenter la *longueur des paquets* permet de repousser le seuil de saturation. En effet, le seuil de saturation passe de 44 % pour des paquets de 4 mots à 54 % pour des paquets de 64 mots. Par contre, le fait d'augmenter la *longueur des paquets* entraîne une augmentation de la valeur moyenne de la *latence effective*, puisque le gain sur le seuil de saturation est minime au-delà de paquets de 16 mots. Nous réaffirmons donc que la macro-cellule *SPIN32* a été conçue, dimensionnée et optimisée pour router des paquets de 16 mots.

1.5.8 Distribution de la latence

Notre objectif est de caractériser la distribution de la latence de la macro-cellule *SPIN32* afin de fournir des garanties statistiques aux utilisateurs. Dans ce but, nous avons mesuré la distribution de la latence pour deux valeurs de la charge (Figure 74) : une charge de 21.33% qui ne sature pas la macro-cellule *SPIN32* et une charge de 72.73% qui se trouve dans la zone de saturation de la macro-cellule *SPIN32*. L'histogramme de la Figure 74 représente le nombre de paquets en ordonnée et les intervalles de latence en abscisse :

- *lat0* représente les paquets dont la latence est inférieure à 16 cycles,
- *lat1* représente les paquets dont la latence est comprise entre 16 et 32 cycles,
- *lat2* représente les paquets dont la latence est comprise entre 32 et 64 cycles,
- *lat3* représente les paquets dont la latence est comprise entre 64 et 128 cycles,
- *lat4* représente les paquets dont la latence est comprise entre 128 et 256 cycles,
- *lat5* représente les paquets dont la latence est comprise entre 256 et 512 cycles,
- *lat6* représente les paquets dont la latence est supérieure à 512 cycles.

L'histogramme de la Figure 74 nous montre que 94,30% des paquets ont une latence supérieure à 512 cycles dans la zone de saturation. Lorsque la macro-cellule *SPIN32* n'est pas saturée, cette tendance s'inverse : 92,73% des paquets ont une latence inférieure à 32 cycles et 71.83% des paquets ont une latence inférieure à 16 cycles.

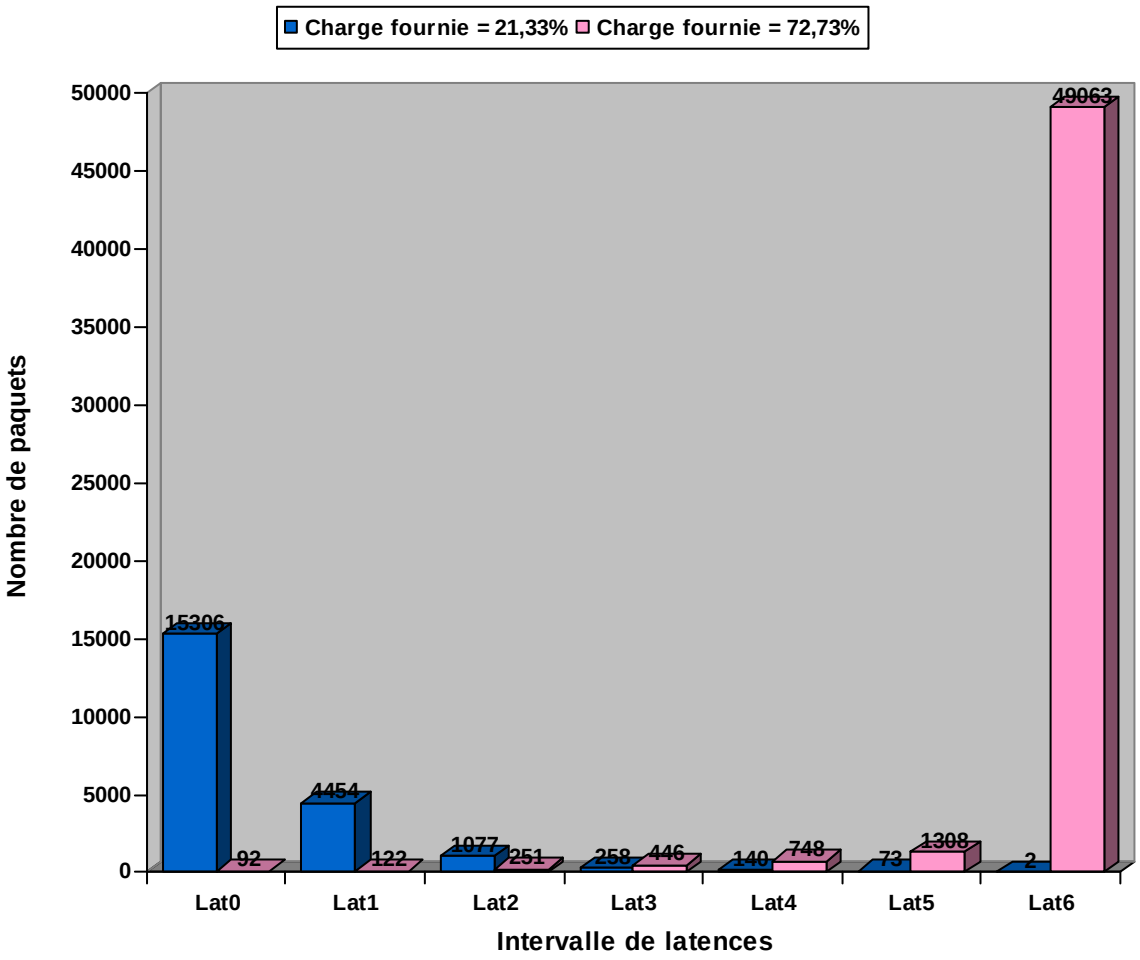

Figure 74 : Distribution de la latence

2 Conclusion

Dans ce chapitre nous avons présenté les résultats des simulations que nous avons effectuées pour caractériser le comportement de la macro-cellule *SPIN32* en termes de latence et de *charge acceptée* :

- la *charge acceptée* est identique à la *charge fournie* avant le seuil de saturation, qui apparaît à environ 52 % ;
- la valeur moyenne de la *latence effective* augmente brusquement lorsque le seuil de saturation est franchi. La valeur moyenne de la *latence effective* et la valeur moyenne du *temps de traversée* de la macro-cellule *SPIN32* restent dans le même ordre de grandeur (environ 20 cycles) en dehors de la zone de saturation ;
- la valeur moyenne du *temps de traversée* de la macro-cellule *SPIN32* ne dépasse pas 42 cycles. Cette valeur est atteinte au moment où la *charge acceptée* franchit le seuil de saturation.
- la garantie d'absence d'interblocage réduit d'environ 3 % le seuil de saturation, qui se retrouve vers 49 % de la *charge acceptée* ;
- l'utilisation des queues centrales repousse le seuil de saturation d'environ 5 %, puisque le seuil de saturation passe de 47 % à 52 % de la *charge acceptée* ;
- les interlocuteurs fréquents doivent être placés dans le même *cluster* car le *trafic très local* a une latence plus faible que le *trafic aléatoire* ;
- hors de la zone de saturation, plus de 90% des paquets ont une latence inférieure à 32 cycles.

Nous en concluons que le trafic injecté par chaque composant ne doit pas être supérieur à 50% de la charge maximale, et que les interlocuteurs fréquents doivent être placés dans le même *cluster*.

Une seconde conclusion est que l'amélioration de la résistance à la contention apportée par les queues centrale et l'adaptativité ne justifie probablement pas le coût en surface.

Chapitre 10 *Conclusion*

1 Contexte

Les recherches sur le micro-réseau *SPIN* ont commencé en 1997, et l'architecture *SPIN* a probablement été la première proposition d'utilisation d'un micro-réseau à commutation de paquets pour résoudre le problème de la bande passante dans les systèmes multi-processeurs intégrés sur puce. Aujourd'hui, les mots *NoC* (*Network On Chip*) et *SoC* (*System On Chip*) sont à la mode, et plusieurs dizaines d'équipes travaillent sur différents projets de micro-réseaux intégrés. Dans cette thèse nous nous sommes attaqués aux problèmes posés par l'industrialisation effective de cette technologie.

2 Analyses critiques

2.1 Flot de conception

La décision de *STMicroelectronics* de lancer la fabrication d'une *puce d'évaluation* nous a conduit à en définir l'architecture. Elle contient la macro-cellule *SPIN32*, entourée de toute la logique nécessaire à son instrumentation. Nous avons suivi un flot de conception qui utilise les bibliothèques de cellules portables (*layout* symbolique), ainsi que le langage de *layout* procédural *GENLIB*, disponibles dans la chaîne de *CAO ALLIANCE*. Nous avons également utilisé les outils commerciaux *VSS* et *Design Analyzer* de *SYNOPSYS* et enfin *Silicon Ensemble* de *CADENCE*. Ce flot de conception nous a permis de développer les modèles comportementaux en langage *VHDL* pour la macro-cellule *SPIN32*, ainsi que pour la logique d'instrumentation. Nous avons également réalisé la macro-cellule *SPIN32* au format *GDS II*, pour le procédé de fabrication cible.

L'envoi en fabrication ne se fait qu'après de nombreuses vérifications (analyse de *crosstalk*, analyse de *timing*, analyse de stabilité, …) sur le dessin des masques après *placement/routage*. Ces vérifications doivent être faites avec le flot de conception de *STMicroelectronics,* qui n'a pas été conçu pour traiter des macro-cellules « dures » (fichier *GDSII*) contenant plus d'un million de transistors. Il nous a donc fallu construire une passerelle entre le flot de conception interne au *LIP6* et le flot de conception *STMicroelectronics*. Cette passerelle a été beaucoup plus difficile à réaliser techniquement que prévu. Par exemple, il a fallu presque une année entière pour avoir une *netlist* extraite du *layout* contenant les capacités parasites et de couplages, ainsi que les résistances parasites. *STMicroelectronics* a dû recourir à des outils de vérification qui ne faisaient pas partie du flot standard (Analyseur de timing *HiTas*). Un des principaux enseignements a donc été que l'approche macro-cellule « dure » au format *GDSII* pour le micro-réseau *SPIN* ne permettait pas une industrialisation facile de cette technologie. Elle n'est pas synthétisable et ne peut utiliser la bibliothèque de cellules de *STMicroelectronics*. C'est incontestablement un désavantage qui a contribué aux difficultés

rencontrées lors de la phase de validation avant l'envoi en fabrication. C'est, entre autres, pour cette raison qu'il a fallu utiliser l'outil d'analyse temporelle *HiTas* de la société *Avertec*, puisque les outils dont disposait *STMicroelectronics* n'étaient pas adaptés.
Par ailleurs, la macro-cellule *SPIN32* en *layout* symbolique devait être portable vers n'importe quel processus de fabrication contenant au moins 6 niveaux de métallisation. Là encore, nous nous sommes heurtés à des difficultés de portabilité inattendues. En effet, les règles de dessins sont devenues tellement complexes que l'intégration de nouvelles règles de dessins dans notre outil de conversion de *layout* symbolique vers le *layout* réel ne s'est pas faite sans peine.

2.2 Caractéristiques physiques

Cette thèse apporte un début de réponse au défi de la bande passante dans les systèmes multi-processeurs intégrés sur puce. Nous avons prouvé la faisabilité matérielle d'un micro-réseau à commutation de paquet dont la bande passante cumulée croît linéairement avec le nombre d'abonnés connectés : environs 109 Gbit/s pour 32 abonnés sur une surface de 4.6 mm², pour un procédé de fabrication 0.13 µm. La fréquence de fonctionnement, obtenue par analyse temporelle de la *netlist* extraite du dessin des masques après *placement/routage* confirme les résultats des simulations *Spice*, et garantit un fonctionnement correct à 192 MHZ, dans le pire cas de dispersion des caractéristiques technologiques.
Grâce aux améliorations architecturales introduites au cours du projet, le micro-réseau *SPIN* peut supporter différents types de trafics (unidirectionnel ou de type *requête/réponse*) en garantissant l'absence d'interblocage et l'acheminement des paquets dans l'ordre chronologique lorsque cela s'avère nécessaire. Il intègre un mécanisme d'auto-test *at speed* qui lui confère un bon niveau de testabilité malgré sa spécificité, y compris vis-à-vis des fautes temporelles, et dont le coût en terme de surface est nul car la surface du routeur *RSPIN* est définie par l'encombrement des nappes de fils d'entrée/sortie. Les mesures qui vont être effectuées sur la *puce d'évaluation* permettront d'aller plus loin dans la caractérisation du micro-réseau *SPIN* (consommation, taux d'erreur sous différentes conditions, …). Mais l'analyse du dessin des masques nous a déjà fourni les réponses en ce qui concerne la fréquence de fonctionnement, le seuil de saturation, la latence, la bande passante et la surface.

2.3 Perspectives

Les enseignements tirés de cette première tentative d'implémentation matérielle de l'architecture *SPIN* ont fortement influencé les recherches dans le domaine des micro-réseaux sur puce au laboratoire *LIP6*. Le successeur du micro-réseau *SPIN*, l'architecture *DSPIN* (*Distributed SPIN*) s'appuie sur une technique de synthèse logique utilisant une bibliothèque de cellules précaractérisées, conforme au flot de conception standard de *STMicroelectronics*.
L'architecture *DSPIN* est adaptée au paradigme *GALS* (*Globally Asynchronous, Locally Synchronous*), ou différents sous-systèmes sont interconnectés par le micro-réseau *DSPIN*, et où chaque sous-système est cadencé par sa propre horloge. Le micro-réseau *DSPIN* est donc distribué et non plus concentré en une seule macro-cellule. Il y a un routeur par sous-système, et chaque routeur est cadencé par l'horloge du sous-système auquel il appartient. Enfin le micro-réseau *DSPIN* utilise une technique de routage déterministe, renonçant à toute forme d'adaptativité, qui possède plus d'inconvénients que d'avantages, et permet de minimiser la surface de silicium occupée.

Chapitre 11 *Annexe*

1 *Interface du routeur RSPIN*

1.1 *Interface logique*

L'interface logique du routeur *RSPIN* est composée des signaux (Figure 75) :

- *ck*, sur un bit, représente l'horloge ;
- *reset*, sur un bit, représente la remise à zéro ;
- *mode*, sur 2 bits, représente le mode de fonctionnement du routeur *RSPIN*. La valeur "00" indique que le routeur *RSPIN* est en mode *fonctionnel*, "01" en mode *scan*, "10" en mode *bist* et "11" n'est pas utilisé actuellement ;
- *ack*, sur un bit, représente l'acquittement d'une interruption ;
- *scin*, sur un bit, représente l'entrée du chemin de test ;
- *scin_sign*, sur un bit, représente l'entrée du chemin de test dédié au test natif de la macro-cellule ;
- *c_in*, sur 8 bits, représente les retours de crédits sur les ports d'entrée. Chaque bit correspond, dans l'ordre croissant, à *d0*, *d1*, *d2*, *d3*, *u0*, *u1*, *u2* et *u3* ;
- *dv_in*, sur 8 bits, indique que les données sont valides sur les ports d'entrée (*data_valid*). Chaque bit correspond, dans l'ordre croissant, à *d0*, *d1*, *d2*, *d3*, *u0*, *u1*, *u2* et *u3* ;
- *u3_in*, *u2_in*, *u1_in* et *u0_in*, sur 36 bits, représentent les données sur les ports d'entrée du haut ;
- *d3_in*, *d2_in*, *d1_in* et *d0_in*, sur 36 bits, représentent les données sur les ports d'entrée du bas ;
- *irq*, sur un bit, représente une interruption. Ce signal est activé lorsqu'un paquet emprunte une sortie inactive ;
- *scout*, sur un bit, représente la sortie du chemin de test ;
- *scout_sign*, sur un bit, représente la sortie du chemin de test dédié au test natif de la macro-cellule ;
- *c_out*, sur 8 bits, représente les retours de crédits sur les ports de sortie. Chaque bit correspond, dans l'ordre croissant, à *d0*, *d1*, *d2*, *d3*, *u0*, *u1*, *u2* et *u3* ;
- *dv_out*, sur 8 bits, indique que les données sont valides sur les ports d'entrée (*data_valid*). Chaque bit correspond, dans l'ordre croissant, à *d0*, *d1*, *d2*, *d3*, *u0*, *u1*, *u2* et *u3* ;
- *u3_out*, *u2_out*, *u1_out* et *u0_out*, sur 36 bits, représentent les données sur les ports de sortie du haut ;

- *d3_out*, *d2_out*, *d1_out* et *d0_out*, sur 36 bits, représentent les données sur les ports de sortie du bas.

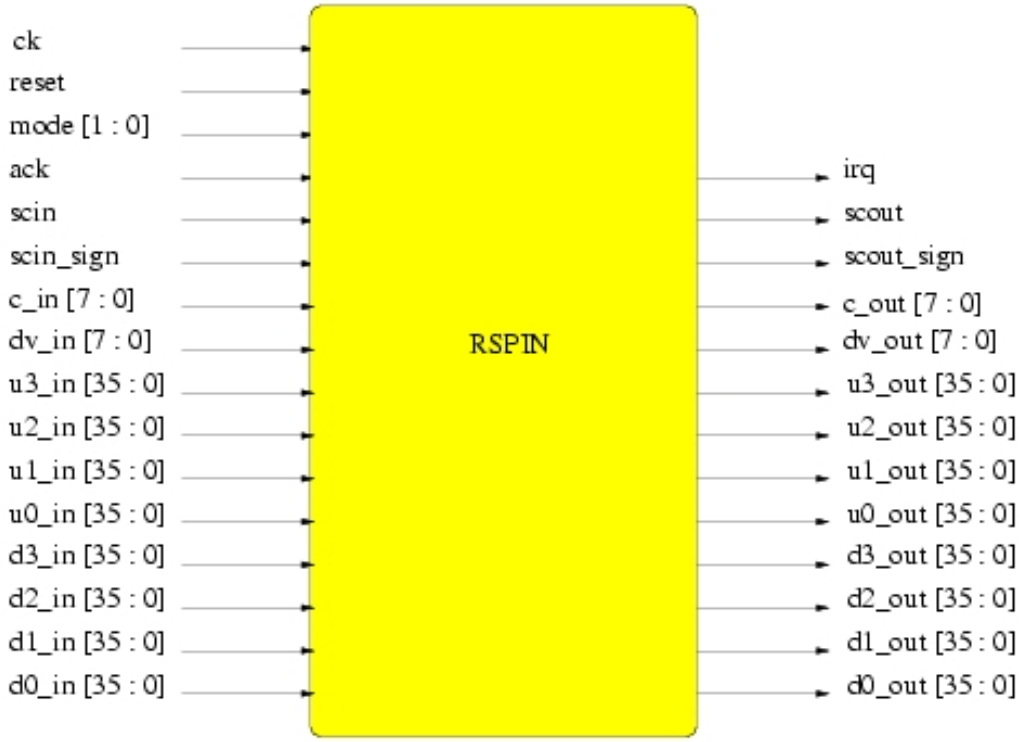

Figure 75 : Interface logique du routeur RSPIN

1.2 Interface physique

L'interface physique du routeur *RSPIN* s'étale sur 3 niveaux de métal (Figure 76) :

- les connecteurs pour les signaux *mode*, *ack*, *scin* et *scin_sign* sont situés sur la face Ouest en métal 2 (en bleu clair sur la Figure 76) ;
- les connecteurs pour les signaux *ck* et *reset* sont situés sur les faces Ouest et Est en métal 4 (en vert sur la Figure 76) ;
- les connecteurs pour les signaux *scout* et *scout_sign* sont situés sur la face Est en métal 2 (en bleu clair sur la Figure 76) ;
- le connecteur pour le signal *irq* est situé sur la face Sud en métal 3 (en rose sur la Figure 76) ;
- les connecteurs pour les signaux des liens sortants *c_in*, *u0_out*, *u1_out*, *u2_out*, *u3_out*, *d0_out*, *d1_out*, *d2_out*, *d3_out* et *dv_out* s'étendent sur toute la hauteur du routeur *RSPIN* en métal 5 (en bleu sur la Figure 76) ;
- les connecteurs pour les signaux des liens entrants *c_out*, *u0_in*, *u1_in*, *u2_in*, *u3_in*, *d0_in*, *d1_in*, *d2_in*, *d3_in* et *dv_in* s'étendent sur la moitié de la hauteur du routeur *RSPIN* en métal 5 (en bleu sur la Figure 76).

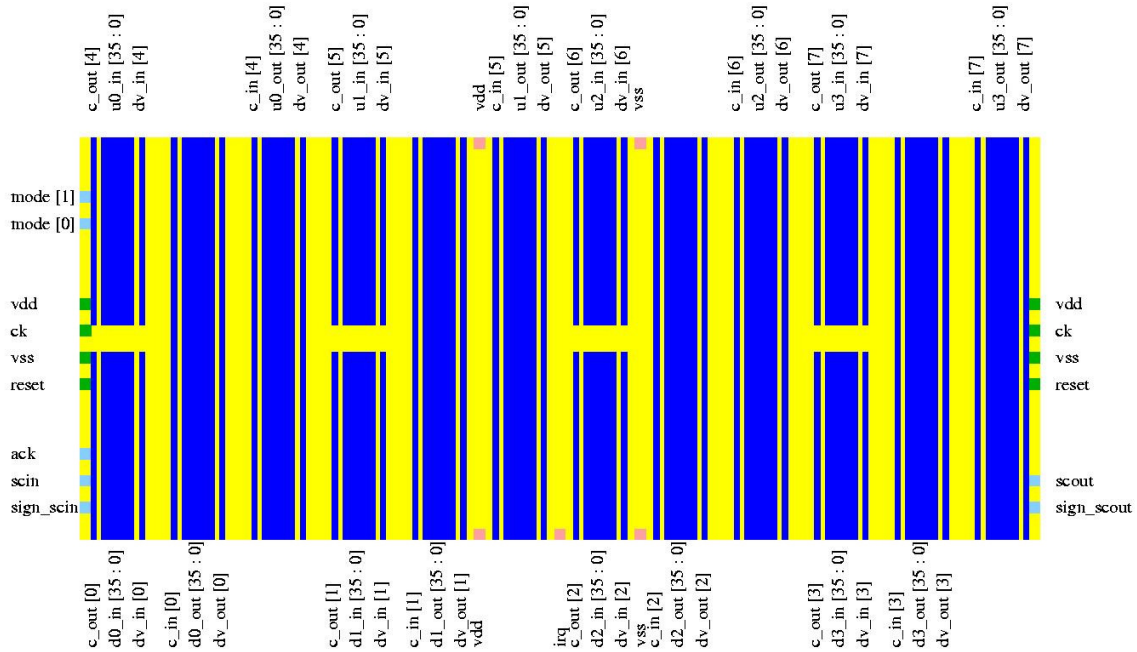

Figure 76 : Interface physique du routeur RSPIN

2 Interface de la macro-cellule SPIN32

2.1 Interface logique

L'interface logique de la macro-cellule *SPIN32* est composée des signaux suivants (Figure 77):

- *ck*, sur un bit, représente l'horloge ;
- *reset*, sur un bit, représente la remise à zéro ;
- *mode*, sur 2 bits, représente le mode de fonctionnement. La valeur "00" indique que le routeur *RSPIN* est en mode *fonctionnel*, "01" en mode *scan* et "10" en mode *bist*. La valeur "11" est utilisée à un plus haut niveau pour le test *eulérien* ;
- *ack*, sur un bit, représente l'acquittement des interruptions ;
- *scin*, sur un bit, représente l'entrée du chemin de tests ;
- *scin_sign*, sur un bit, représente l'entrée du chemin de tests pour les registres de la logique de tests ;
- *c_in*, sur 32 bits, représente les retours de crédits sur les ports de sortie.
- *dv_in*, sur 32 bits, indique que les données sont valides (*data_valid*) sur les ports d'entrée ;
- *router_1_n_d3_in*, *router_1_n_d2_in*, *router_1_n_d1_in* et *router_1_n_d0_in*, sur 36 bits, représentent les données sur les ports d'entrée pour le routeur *RSPIN* numéro n du niveau 1 ;
- *irq,* sur 16 bits, représente les interruptions lorsqu'un paquet emprunte une sortie invalidée. Les bits [7:0] correspondent aux routeurs *RSPIN* du niveau 1. Les bits [15:8] correspondent aux routeurs *RSPIN* du niveau 2 ;
- *scout*, sur un bit, représente la sortie du chemin de tests ;
- *scout_sign*, sur un bit, représente la sortie du chemin de tests pour les registres de la logique de tests ;
- *c_out,* sur 32 bits, représente les retours de crédits sur les ports ;
- *dv_out*, sur 32 bits, donne l'information que les données sont valides (*data_valid*) sur les ports de sortie ;
- *router_1_n_d3_out*, *router_1_n_d2_out*, *router_1_n_d1_out* et *router_1_n_d0_out*, sur 36 bits, représentent les données sur les ports de sortie pour le routeur *RSPIN* numéro *n* du niveau 1.

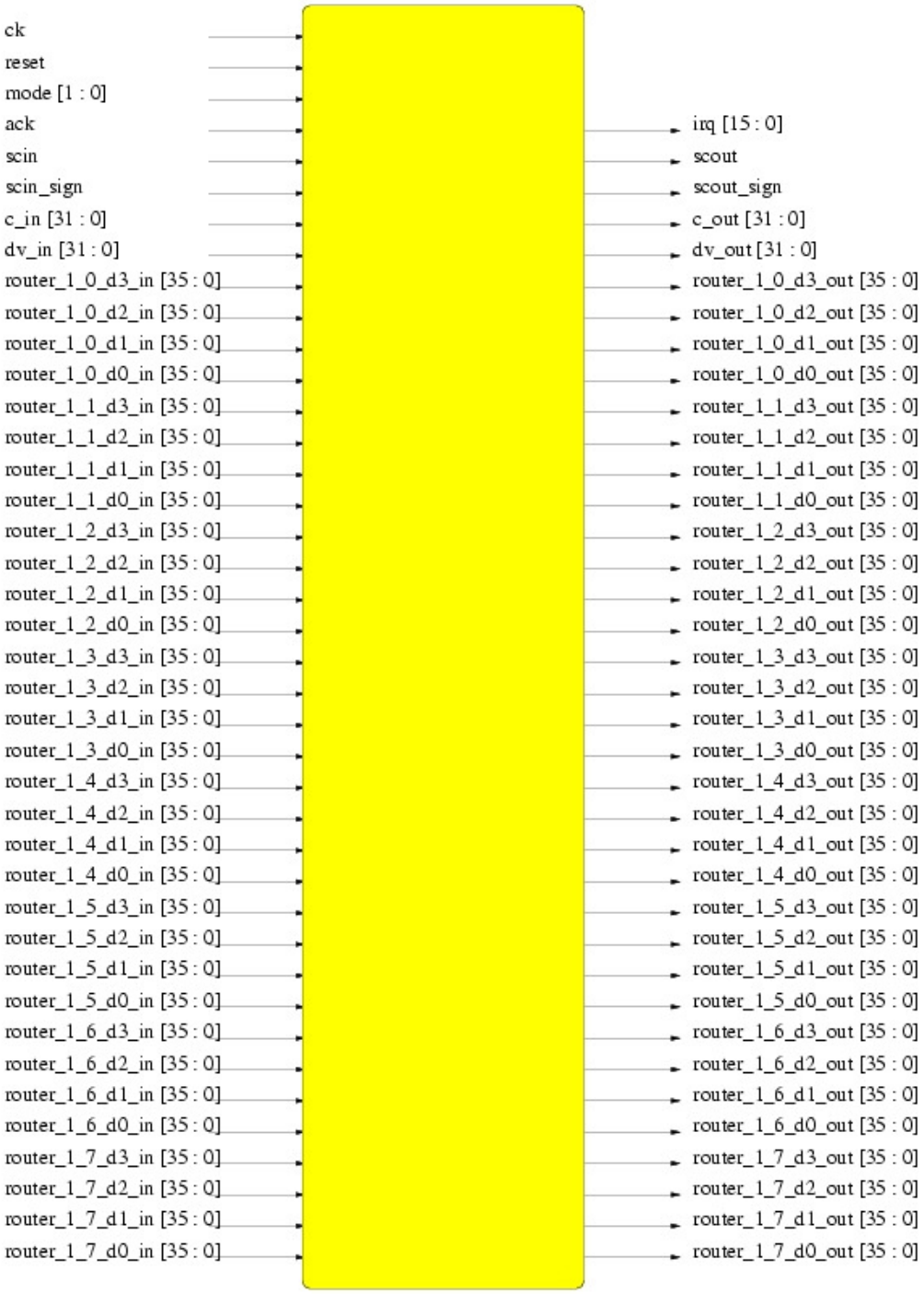

Figure 77 : Interface de la macro-cellule SPIN32

2.2 Interface physique

L'interface physique de la macro-cellule *SPIN32* est sur 3 niveaux de métal comme le montre la Figure 78. Pour plus de clarté, la Figure 78 a été décomposée en deux parties, dénommées partie A et partie B. La Figure 79 représente la partie A, tandis que la partie B est représentée par la Figure 80. La répartition des connecteurs est décrite ci-dessous :

- les connecteurs pour les signaux *ck*, *reset*, *mode* et *ack* sont situés sur la face Nord en métal 3 ;
- les connecteurs pour les signaux *scin*, *scin_sign*, *scout* et *scout_sign* sont situés sur la face Ouest en métal 2 ;
- les connecteurs pour les signaux *irq [3:0]*et *irq [11:8]* sont situés sur la face Sud en métal 3. Ceux pour les signaux *irq [7:4]*et *irq [15:12]* sont situés sur la face Nord en métal 3 ;
- les connecteurs pour les signaux d'entrées/sorties *dv_in[15:0]*, *c_in[15:0]*, *routeur_1_n_d0_out*, *routeur_1_n_d1_out*, *routeur_1_n_d2_out*, *routeur_1_n_d3_out*, *dv_out[15:0]*, *c_out[15:0]*, *routeur_1_n_d0_in*, *routeur_1_n_d1_in*, *routeur_1_n_d2_in* et *routeur_1_n_d3_in* (où $n \in [0, 1, 2, 3]$ représente le numéro du routeur de niveau 1) sont situés sur la face Sud en métal 5 ;
- les connecteurs pour les signaux d'entrées/sorties *dv_in[31:16]*, *c_in[31:16]*, *routeur_1_n_d0_out*, *routeur_1_n_d1_out*, *routeur_1_n_d2_out*, *routeur_1_n_d3_out*, *dv_out[31:16]*, *c_out[31:16]*, *routeur_1_n_d0_in*, *routeur_1_n_d1_in*, *routeur_1_n_d2_in* et *routeur_1_n_d3_in* (où $n \in [4, 5, 6, 7]$ représente le numéro du routeur de niveau 1) sont situés sur la face Nord en métal 5.

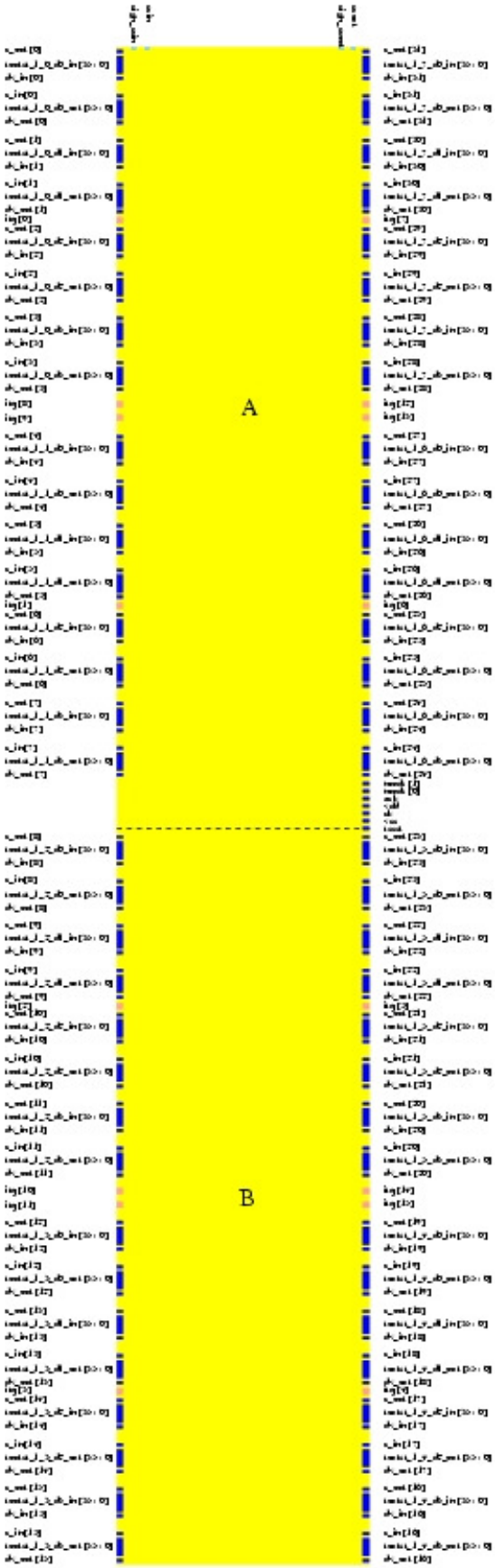

Figure 78 : Interface physique du micro-réseau SPIN à 32 ports

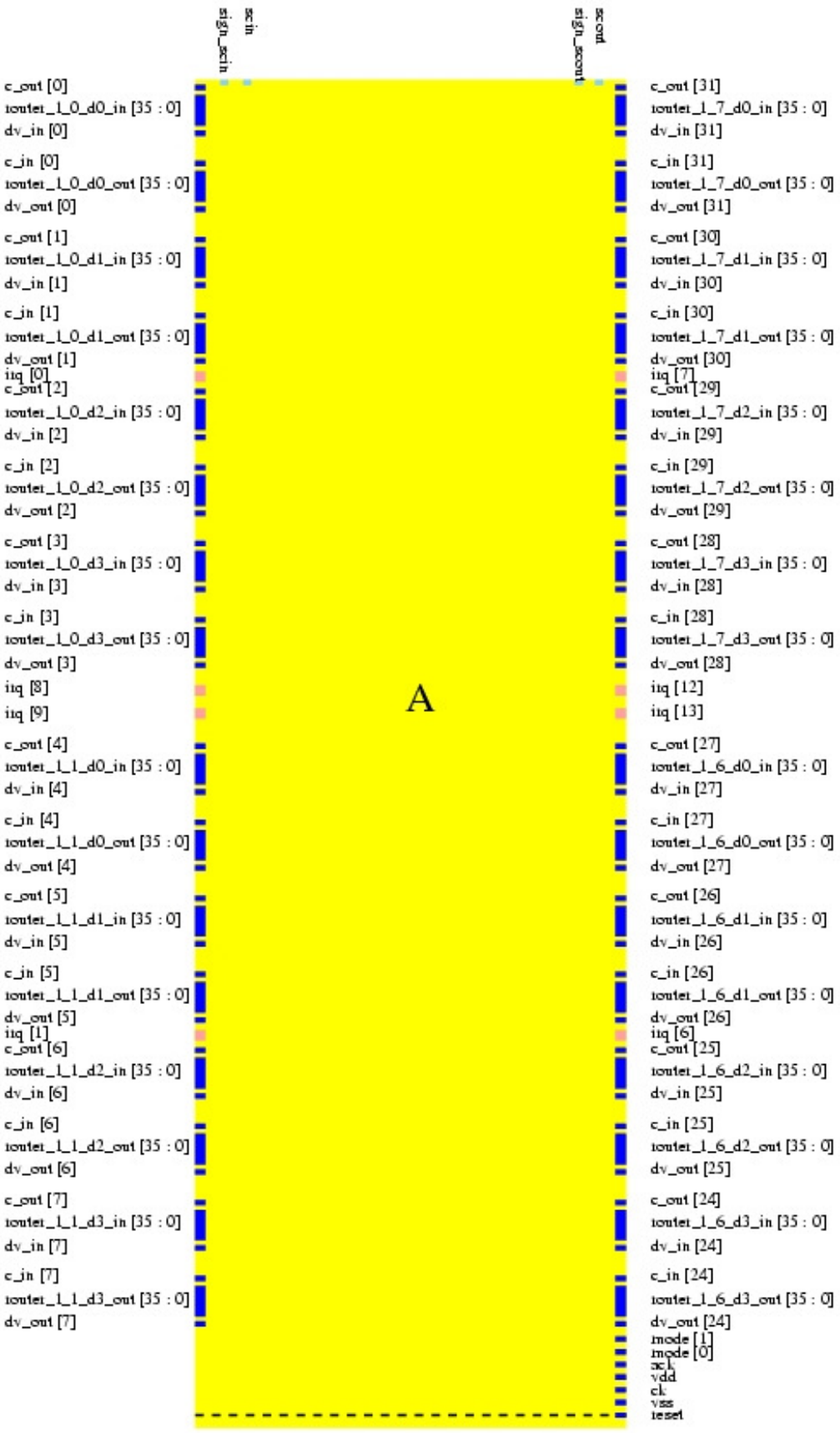

Figure 79 : Partie A de l'interface physique du micro-réseau SPIN à 32 ports

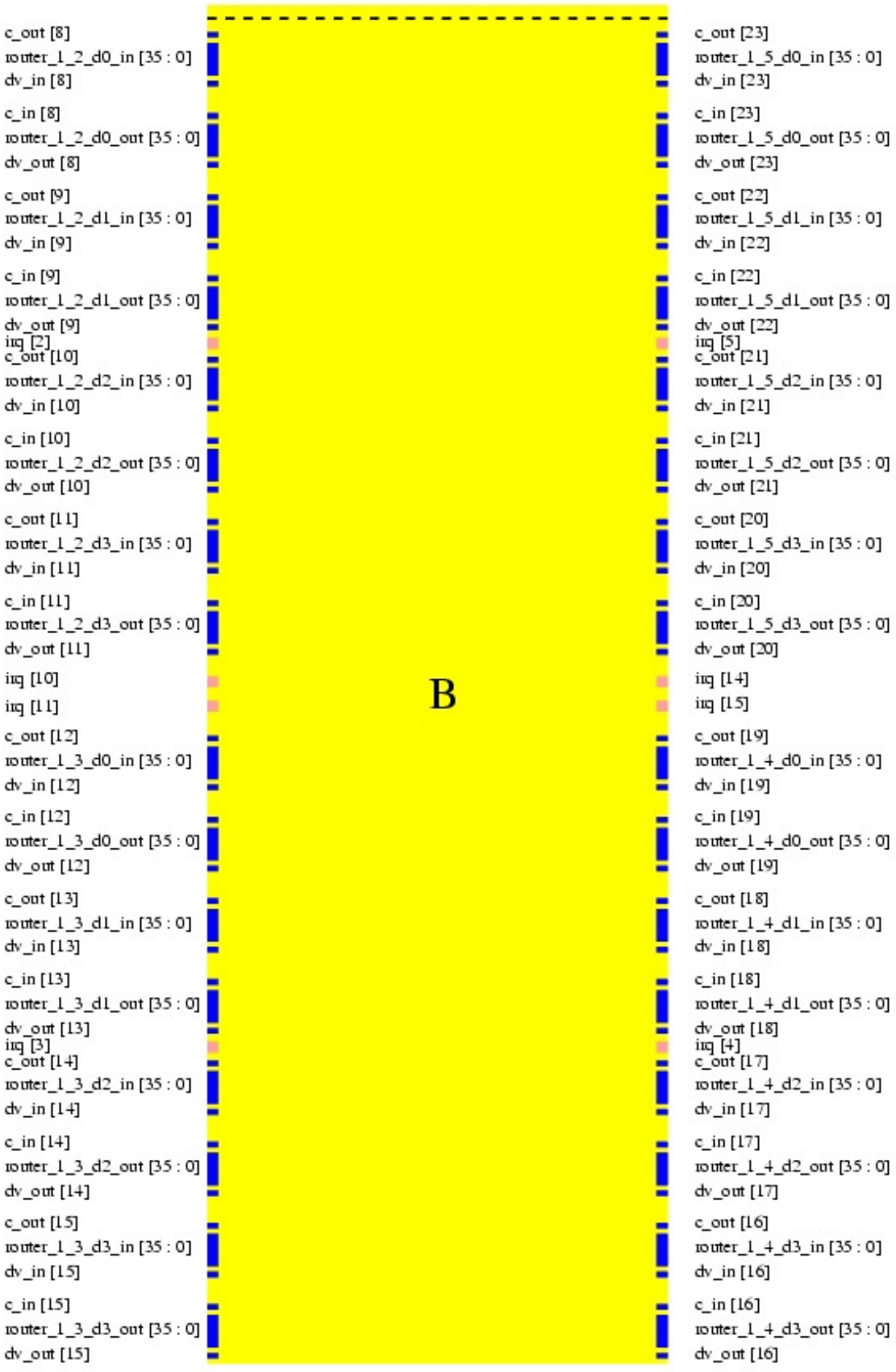

Figure 80 : Partie B de l'interface physique du micro-réseau SPIN à 32 ports

3 La puce d'évaluation

3.1 Circuit eulérien dans un micro-réseau SPIN à 32 ports

Dans un micro-réseau *SPIN* à 32 ports, le circuit eulérien choisi est une succession de chemins qui passe, une fois et une seule, par tous les ports du micro-réseau *SPIN* à 32 ports dans l'ordre décrit sur le Tableau 8.
Ce dernier indique également le port d'entrée et le port de sortie pour chaque chemin, ainsi que l'action exercée par le *wrapper* accroché à ce port de sortie.

On distingue 4 types d'actions :

- génération et injection des paquets dans le micro-réseau *SPIN* à 32 ports,
- capture des paquets sortant sur le port dans un *MISR* de signature,
- incrémentation de l'adresse de destination et réinjection dans le micro-réseau *SPIN* à 32 ports,
- réinjection dans le micro-réseau *SPIN* à 32 ports.

Les *wrappers* connectés sur les ports 1 à 15 se contentent de réinjecter les paquets qu'ils reçoivent sur leurs ports de sortie dans leurs ports d'entrée (Figure 81).

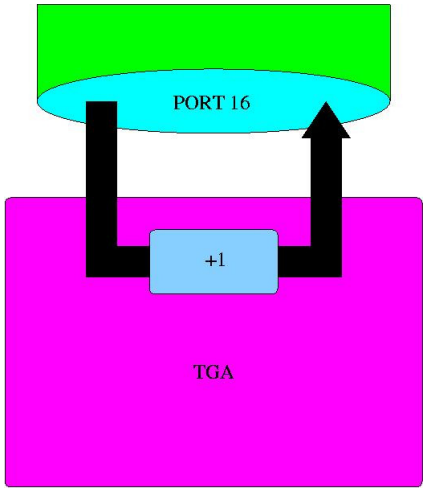

Figure 81 : Ports 1à 15 en mode eulérien

Les *wrappers* connectés sur les ports 16 à 31, par contre, incrémentent l'adresse de destination avant de réinjecter les paquets qu'ils reçoivent (Figure 82).

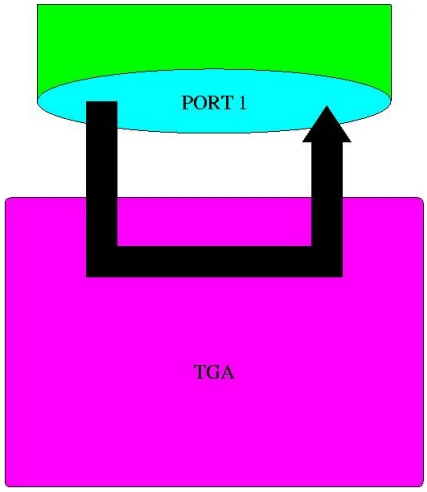

Figure 82 : Ports 16 à 31 en mode eulérien

Enfin, sur le *wrapper* connecté sur le port 0, on trouve d'une part, le *MISR* de signature afin de capturer les paquets arrivant sur son port de sortie. D'autre part on trouve, sur son port d'entrée, le générateur de paquets qui doit envoyer les paquets dans le circuit eulérien du micro-réseau *SPIN* à 32 ports (Figure 83).

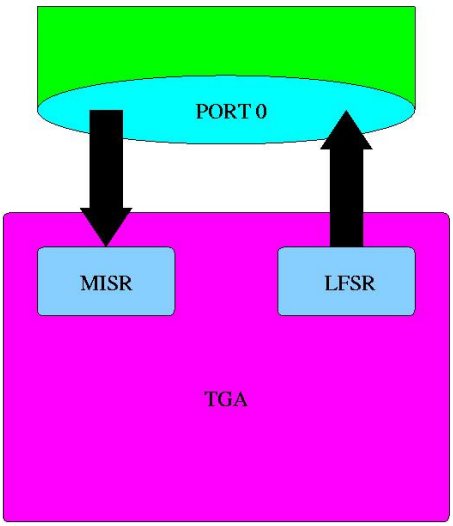

Figure 83 : Port 0 en mode eulérien

Numéro du chemin	Port d'entrée	Port de sortie	Action sur la destination
0	Port 0	Port 16	Incrémentation et réinjection
1	Port 16	Port 1	réinjection
2	Port 1	Port 17	Incrémentation et réinjection
3	Port 17	Port 2	réinjection
4	Port 2	Port 18	Incrémentation et réinjection
5	Port 18	Port 3	réinjection
6	Port 3	Port19	Incrémentation et réinjection
7	Port 19	Port 4	réinjection
8	Port 4	Port 20	Incrémentation et réinjection
9	Port 20	Port 5	réinjection
10	Port 5	Port 21	Incrémentation et réinjection
11	Port 21	Port 6	réinjection
12	Port 6	Port 22	Incrémentation et réinjection
13	Port 22	Port 7	réinjection
14	Port 7	Port 23	Incrémentation et réinjection
15	Port 23	Port 8	réinjection
16	Port 8	Port 24	Incrémentation et réinjection
17	Port 24	Port 9	réinjection
18	Port 9	Port 25	Incrémentation et réinjection
19	Port 25	Port 10	réinjection
20	Port 10	Port26	Incrémentation et réinjection
21	Port 26	Port 11	réinjection
22	Port 11	Port 27	Incrémentation et réinjection
23	Port 27	Port 12	réinjection
24	Port 12	Port 28	Incrémentation et réinjection
25	Port 28	Port 13	réinjection
26	Port 13	Port 29	Incrémentation et réinjection
27	Port 29	Port 14	réinjection
28	Port 14	Port 30	Incrémentation et réinjection
29	Port 30	Port 15	réinjection
30	Port 15	Port 31	Incrémentation et réinjection
31	Port 31	Port 0	réinjection

Tableau 8 : Description du circuit eulérien dans le micro-réseau SPIN à 32 ports

3.2 Chemin de tests dédié à l'instrumentation

Afin de pouvoir faire, le cas échéant, une analyse approfondie de la logique d'instrumentation de la *puce d'évaluation*, un chemin de tests (*scan-path*) dédié à l'instrumentation y est introduit. Ce chemin de tests est totalement distinct des chemins de test de la macro-cellule *SPIN32*. Il est composé de tous les registres du bloc *CC* suivis de tous les registres de tous les blocs *TGA*.

3.3 Mode de fonctionnement

La *puce d'évaluation* possède 16 modes de fonctionnement définis par le signal *mode* qui comporte 4 bits. Le codage des bits du signal *mode* est :

- "0000" correspond au mode *fonctionnel*,
- "0001" correspond au mode *scan*, où tous les registres de la macro-cellule *SPIN32* sont accessibles à travers les chemins de test correspondants,
- "0010" correspond au mode *bist*, où chaque routeur *RSPIN* se met dans un mode qui teste toute sa logique interne,
- "0011" correspond au mode *eulerian*, où l'on effectue un test eulérien sur la macro-cellule *SPIN32*,
- "0100" correspond au mode *inst_scan*, où tous les registres de la logique d'instrumentation sont accessibles à travers le chemin de test correspondant,
- "0101" correspond au mode *read*, où l'on peut lire dans les registres à travers l'interface *RAM*,
- "0110" correspond au mode *write*, où l'on peut écrire dans les registres à travers l'interface *RAM*,
- "0111" n'est pas utilisé à ce jour,
- "1000" correspond au mode *random*, où la logique d'instrumentation injecte un *trafic aléatoire*. Les paquets ne peuvent pas utiliser les queues centrales,
- "1001" correspond au mode *low*, où la logique d'instrumentation injecte un *trafic faiblement local*. Les paquets ne peuvent pas utiliser les queues centrales,
- "1010" correspond au mode *medium*, où la logique d'instrumentation injecte un *trafic moyennement local*. Les paquets ne peuvent pas utiliser les queues centrales,
- "1011" correspond au mode *high*, où la logique d'instrumentation injecte un *trafic très local*. Les paquets ne peuvent pas utiliser les queues centrales,
- "1100" correspond au mode *random_q*, où la logique d'instrumentation injecte un *trafic aléatoire*. Les paquets peuvent utiliser les queues centrales,
- "1101" correspond au mode *low_q*, où la logique d'instrumentation injecte un *trafic faiblement local*. Les paquets peuvent utiliser les queues centrales,
- "1110" correspond au mode *medium_q*, où la logique d'instrumentation injecte un *trafic moyennement local*. Les paquets peuvent utiliser les queues centrales,
- "1111" correspond au mode *high_q*, où la logique d'instrumentation injecte un *trafic très local*. Les paquets peuvent utiliser les queues centrales.

3.4 Interface logique de la puce d'évaluation

L'interface logique de la *puce d'évaluation* est composée des 84 ports externes suivants (Figure 84):

- le signal *ck* correspond à l'horloge. L'ensemble de la *puce d'évaluation* est synchrone sur le front montant,
- le signal *reset* permet l'initialisation. La réinitialisation de la *puce d'évaluation* est asynchrone,
- le signal *scin* correspond à l'entrée du chemin de tests de la macro-cellule *SPIN32*,
- le signal *scout* correspond à la sortie du chemin de tests de la macro-cellule *SPIN32*,
- le signal *scin_sign* correspond à l'entrée du chemin de tests dédié à la signature de la macro-cellule *SPIN32*,

- le signal *scout_sign* correspond à la sortie du chemin de tests dédié à la signature de la macro-cellule *SPIN32*,
- le signal *scin_inst* correspond à l'entrée du chemin de tests dédié à l'instrumentation,
- le signal *scout_inst* correspond à la sortie du chemin de tests dédié à l'instrumentation,
- le signal *mode*, sur 4 bits, gère le mode de fonctionnement de la *puce d'évaluation*. Les valeurs "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110" et "1111" correspondent respectivement aux modes : *fonctionnel*, *scan*, *bist*, *eulerian*, *scan_inst*, *read*, *write*, non utilisé, *low*, *medium*, *high*, *full*, *low_q*, *medium_q*, *high_q* et *full_q*,
- le signal *address*, sur 9 bits, correspond à l'adresse sur l'interface *RAM* de la *puce d'évaluation*. Les adresses sont décrites sur le Tableau 7 et le Tableau 6,
- le signal *data_in*, sur 32 bits, correspond à la donnée entrante sur l'interface *RAM* de la *puce d'évaluation*,
- le signal *data_out*, sur 32 bits, correspond à la donnée sortante sur l'interface *RAM* de la *puce d'évaluation.*

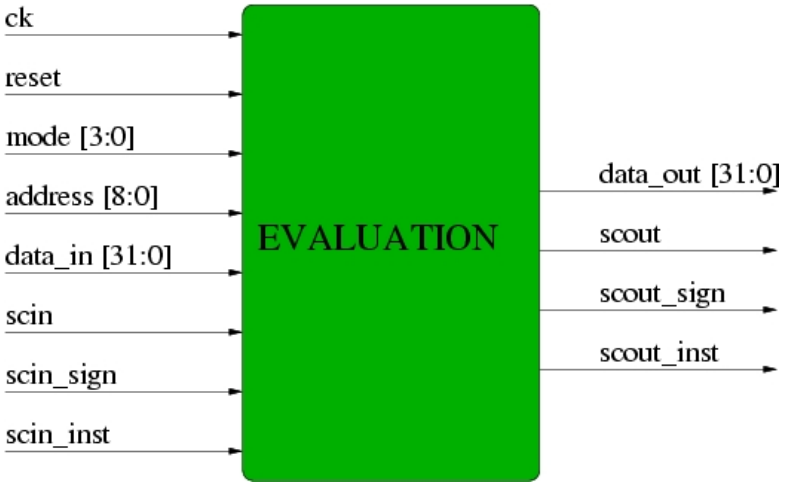

Figure 84 : Interface logique du chip de test

3.5 Flot de caractérisation

La caractérisation de la macro-cellule *SPIN32* se fera selon le chronogramme de la Figure 85. La séquence "*zone 6 – zone 7 – zone 8 – zone 9*" est répétée autant de fois que nécessaire pour effectuer le nombre de mesures voulues.

La description de chaque zone est la suivante :

- dans la *zone 0*, la *puce d'évaluation* est en mode *write*. Pendant ce temps, un signal d'initialisation est envoyé. Ce mode constitue à écrire, par l'intermédiaire de l'interface *RAM*, à l'adresse 0x0, afin que chaque routeur *RSPIN* qui compose la macro-cellule *SPIN32* soit dans un état stable et connu. Ici, la valeur écrite n'est pas importante ; c'est la commande d'écriture à l'adresse 0x0 qui déclanche l'initialisation. Le signal d'initialisation peut également être envoyé en mettant le signal *reset* à la valeur logique '1' ;
- dans la *zone 1*, la *puce d'évaluation* est dans le mode *bist* où chaque routeur *RSPIN* de la macro-cellule *SPIN32* exécute un test interne afin de valider la logique et les interconnexions internes ;

- dans la *zone 2*, la *puce d'évaluation* est dans le mode *scan* où la signature de chaque routeur *RSPIN* de la macro-cellule *SPIN32* est récupérée par l'intermédiaire du chemin de tests dédiée à la signature. Les signatures issues de chaque routeur *RSPIN* sont chaînées pour être mises en série. Ensuite, elles sont comparées aux signatures de référence obtenues par simulation ;
- dans la *zone 3*, la *puce d'évaluation* est en mode *write* où l'interface *RAM* est utilisée pour envoyer un signal d'initialisation. L'initialisation consiste à écrire une valeur quelconque à l'adresse 0x0, afin de remettre chaque routeur *RSPIN* de la macro-cellule *SPIN32* dans un état stable et connu. Ce signal initialise également le *LFSR* et le *MISR* eulérien, ainsi que tous les registres de la *puce d'évaluation.* Le signal d'initialisation peut également être envoyé en mettant le signal *reset* à la valeur logique '1' ;
- dans la *zone 4*, la *puce d'évaluation* est dans le mode *eulerian* où les interconnexions inter-routeurs sont vérifiées dans la macro-cellule *SPIN32* ;
- dans la *zone 5*, la *puce d'évaluation* est dans le mode *read* où l'interface *RAM* est utilisée pour récupérer la signature dans le *MISR* eulérien ;
- dans la *zone 6*, la *puce d'évaluation* est en mode *write* où l'interface *RAM* est utilisée pour envoyer un signal d'initialisation. L'initialisation consiste à écrire une valeur quelconque à l'adresse 0x0, afin de remettre chaque routeur *RSPIN* de la macro-cellule *SPIN32* dans un état stable et connu. Le signal d'initialisation peut également être envoyé en mettant le signal *reset* à la valeur logique '1' ;
- dans la *zone 7*, la *puce d'évaluation* est en mode *write* où l'interface *RAM* est utilisée pour initialiser les paramètres de mesure (*time*, *dl*, *mask*, *fgap*, *lat_ref0*, *lat_ref1*, *lat_ref2*, *lat_ref3*, *lat_ref4* et *lat_ref5*), en écrivant aux adresses correspondantes ;
- dans la *zone 8*, la *puce d'évaluation* peut être dans les modes *full*, *high*, *medium*, *low*, *full_q*, *high_q*, *medium_q* ou *low_q* selon la localité du trafic souhaitée et l'utilisation ou non des queues centrales. Dans ces modes, les mesures sont faites. Les modes *full_q*, *high_q*, *medium_q* et *low_q* permettent aux paquets d'utiliser les queues centrales ;
- dans la *zone 9*, la *puce d'évaluation* est dans le mode *read* où l'interface *RAM* est utilisée pour récupérer le résultat des mesures.

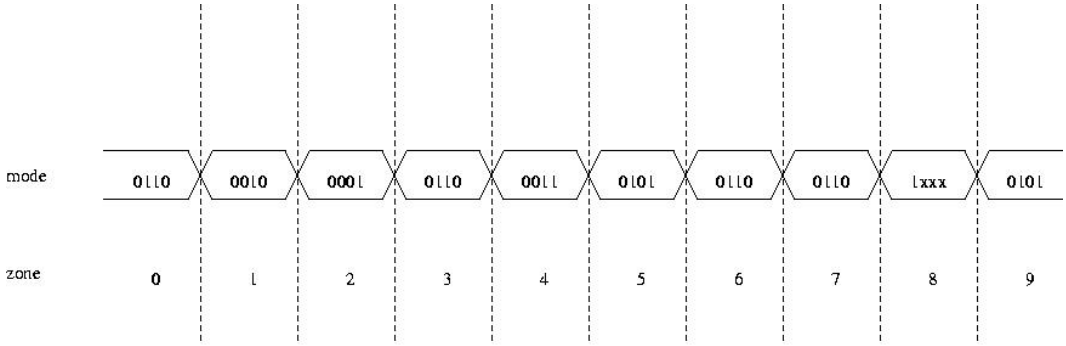

Figure 85 : Chronogramme pour la caractérisation de la macro-cellule SPIN32

Bibliographie

[1] Guerrier Pierre "*Un Réseau d'interconnexion pour systèmes intégrés*", Université Paris VI – Pierre et Marie CURIE, U.F.R. d'informatique, mai 2000
[2] L. Benini and G. De Micheli, "*Networks on Chips: A New SoC Paradigm*" IEEE Computer, vol. 35, no. 1, Jan. 2002, pp. 70-80
[3] A. Jantsch and H. Tenhunen, editors. "*Networks on Chip*". Kluwer, 2003
[4] http://www-asim.lip6.fr
[5] Abdelhafid Bouhraoua, "*Design, Performance Evaluation and Validity Verification of Topologies and Routing Schemes for Interconnection Networks*", Université Paris VI – Pierre et Marie CURIE, U.F.R. d'informatique, mai 1998
[6] John L. Hennessy, David A. Patterson , David Goldberg , Krste Asanovic, in "*Computer architecture : a quantitative approach*" 3 éd., (M. Kaufmann , 2003)
[7] C. Clos, "*A Study of Nonblocking Switching Networks*", Bell System Technical Journal, vol. 32, no. 2, 1953, pp. 5-8
[8] William J. Dally, Charles L. Seitz, "*Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*", *IEEE Trans. Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553
[9] M. J. Karol, M. G. Hluchyj, S. P. Morgan, "*Input Versus Output Queueing on a Space-Division Packet Switch*", *IEEE Transactions on Communications*, december 1987, pp. 1347-1356
[10] Alain Goeury, "*Conception d'un Routeur Intégré possédant un Tampon Central et un Mécanisme d'Auto-configuration*", Université Paris VI – Pierre et Marie CURIE, U.F.R. d'informatique, novembre 1998
[11] William Dally, "*Virtual-Channel Flow Control*", *IEEE Trasactions on Parallel and Distributed Systems*, vol. 3, no. 2, mars 1992, pp. 194-205
[12] C. Stunkel, D. Shea, B. Abali, M. Atkins, C. Bender, D. Grive, P. Hockschild, D. Joseph, B. Nathanson, R. Swetz, R. Stucke, M. Tsao, P. Valker, "*The SP/2 High Performance Switch*", IBM Journal, vol. 34, no. 2, 1995, pp. 185-204
[13] IEEE Standards Department, "*IEEE 1355 High Speed Link Standard*", 1355 Association , http://www.1355-association.org , 1995
[14] Belkacem Zerrouk, Vincent Reibaldi, Frédéric Potter, Alain Greiner, Anne Derrieux, "*Rcube : A Gigabit Serial Link Low Latency Adaptive Router*", Records of the IEEE Interconnects IVth Symposium, Palo Alto, USA, august 1996
[15] http://www.vsi.org
[16] http://www.sonicsinc.com/sonics/index_html
[17] http://www.sonicsinc.com/sonics/products/index_html
[18] http://www.ocpip.org
[19] http://www.sonicsinc.com/sonics/products/siliconbackplaneIII
[20] http://www-3.ibm.com/chips/products/coreconnect
[21] http://www.ibm.com
[22] http://www-306.ibm.com/chips/products/asics
[23] http://www.arm.com/products/solutions/AMBAHomePage.html
[24] ARM. AMBA AXI Protocol Specification, June 2003
[25] http://www.arm.com
[26] http://www.renpar.org/Actes/sympaaa/S4-1.ps
[27] http://www.homepages.inf.ed.ac.uk/kgoossen/2002-date.ps

[28] Kees Goossens, John Dielissen, and Andrei Radulescu, "*The Aethereal network on chip: Concepts, architectures, and implementations*", IEEE Design and Test of Computers, Vol 22(5):21--31, 7-Oct 2005
[29] K. Goossens et al., "*A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification*" Proc. Design, Automation and Test in Europe(DATE 05), IEEE CS Press, 2005, pp. 1182-1187
[30] http://www.phillips.com
[31] E. Rijpkema et al., "*Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip*" Proc. IEEE Computers and Digital Techniques, IEEE Press, 2003, vol. 150, no. 5, pp. 294-302
[32] Andrei Radulescu, John Dielissen, Santiago Gonzalez Pestana, Om Gangwal, Edwin Rijpkema, Paul Wielage, and Kees Goossens, "*An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming*", IEEE Transactions on CAD of Integrated Circuits and Systems, 24(1), January 2005
[33] H. Zhang. "*Service disciplines for guaranteed performance service in packet-switching networks*". Proceedings of the IEEE, 83(10):1374-96, Oct. 1995
[34] "*Les wrappers VCI/SPIN*", H. Charlery, A. Greiner, Rapport technique, Université Pierre et Marie Curie, 2001.
[35] William Dally, Charles Seitz, "*Deadlock-free Message Routing in Multiprocessor Interconnection Network*", IEEE Transactions on Computers, vol. C-36, no. 5, mai 1987, pp. 547-553
[36] IEEE Standard Board. IEEE std 1149.1-1990, "*standard test access port and boundary scan architecture*". 345 East 47th Street, New York, NY 10017-2394, 1990
[37] "*Self-Test Services*", The STS ASIC Testability Seminar, Self-Test Services, Ambler, PA, 1989
[38] W.W. Peterson and E.J. Weldon, Jr. "*Error Correcting Codes*", MIT press, Cambridge, MA 1972
[39] Texas Instruments, "*SCOPE Cell Design Manual*", Texas Instruments Incorporated, Dallas, TX, 1990
[40] Andrzej Hlawiczka, Michal Kopec. "*Dependable Testing of Compactor MISR:A Imperceptible Problem?* " *etw*, p. 31, Seventh IEEE European Test Workshop (ETW'02), 2002
[41] P. H. Bardell, W. H. McAnney, and J. Savir, "*Built-In Test for VLSI: Pseudorandom Techniques*", Wiley Interscience, New York, 1987
[42] http://whatis.techtarget.com/definition/0,,sid9_gci213868,00.html