Thèse de doctorat de l'université Paris VI

Spécialité Informatique

Présentée par

Hervé CHARLERY

Pour obtenir le titre de

Docteur de l'université Pierre et Marie Curie

Intégration d'un micro-réseau à commutation de paquets dans un système multiprocesseur à mémoire partagée intégré sur puce

Soutenue le 9 décembre 2005

Devant le jury composé de

M. Alain Greiner	Directeur de thèse
M. Daniel Litaize	Rapporteur
M. Frédéric Pétrot	Rapporteur
M. François Charot	Examinateur
M. Habib Mehrez	Examinateur
M. Olivier Temam	Examinateur

Remerciements

Je remercie tout d'abord le professeur Alain Greiner pour m'avoir pris en thèse et encadré pendant un peu plus de quatre ans. Ses conseils m'ont guidés dans la voie du réalisme. Et grâce à son pragmatisme j'ai pu garder les pieds sur terre.

Je remercie Frédéric Pétrot pour m'avoir donné le goût de la micro-électronique dès la Licence, et m'avoir ensuite accompagné patiemment à chaque instant.

Je remercie les professeurs Daniel Litaize, Olivier Temam et François Charot d'avoir bien voulu être rapporteur et examinateurs de ma thèse.

Je remercie Emmanuelle Encrenaz pour ses lumières en vérification formelle et pour sa précieuse aide dans la deuxième partie de mon travail.

Je remercie également Habib Mehrez, Franck Wajsburt, Anne Dérieux, Marie Martine Paget, François Dromard et Pirouz Bazargan, pour leur aide, leur présence et l'oreille attentive qu'ils ont toujours su me prêter. Et un grand merci à Danièle Dromard pour tout ce qui précède et aussi pour son important travail de relecture.

Je remercie les administrateurs systèmes Manuel Bouyer et Jean-Paul Chaput qui nous réconcilient tous les jours avec nos machines.

Je remercie Nicole Fouque, Elisabeth Sergent, Annick Lablanche, Bouchra Guesmi, Chantale Darin et Shahin Mahmoodian sans lesquelles la thèse serait administrativement parlant un vrai parcours du combattant.

Je remercie Pierre Nguyen Tuong et Cedric Roux pour leur soutien et les discussions que nous avons eus, ainsi que tous les thésards pour leur accompagnement et la bonne ambiance du labo à laquelle ils ont contribués.

Je remercie tous les stagiaires qui ont contribués à l'avancement de cette thèse : César Zéférino (stage de thèse), Philippe Aldana Flores, Thomas Morgand, Evgueni Talal, François Zanati, Jean-Yves Ingea, Ilhem Benachour, Xavier Libault, Hamid Ait Ibrahim, Stéphane Dubuisson, Daniel Zaoui, Damien Dupuis et Sophie Belloeil.

Enfin, et surtout, je remercie ma famille pour son soutien, sa compréhension et ses encouragements.

A ma famille, pour son soutien sa patience, sa compréhension et ses encouragements.

Hâtez-vous lentement, et sans perdre courage.
Vingt fois sur le métier remettez votre ouvrage.
Polissez-le sans cesse et le repolissez.
(Nicolas Boileau - L'Art poétique)

Per ardua ad astra.
- A travers les épreuves vers les astres -
(Devise de la Royal Air Force)

Résumé

Ce travail de recherche dans le domaine des "Network On-Chip", porte sur les conséquences directes de l'insertion de communications parallèles au sein d'un système sur puce. L'un des objectifs d'une telle action est d'optimiser le traitement des informations : en traiter plus en moins de temps et, par la même occasion, réaliser ainsi une économie d'énergie. L'impact d'une telle approche parallèle a été étudié à travers des évaluations de performance entre un système ne permettant pas de communication parallèle (organisé autour d'un PI-Bus) et un système massivement parallèle, organisé autour du micro-réseau SPIN (Scalable Programmable Integrated Network). Le micro-réseau est un réseau d'interconnexion à commutation de paquets pour systèmes intégrés sur puce développé au Laboratoire d'Informatique de Paris 6 (LIP6). SPIN fournit un mécanisme de communication très général entre les différents composants virtuels connectés dans le système. La bande passante disponible du micro-réseau croit linéairement en fonction du nombre de processeurs intégrés.
Des modèles de simulation de haut niveau en SystemC ont été développés afin de valider le concept et le comportement du réseau SPIN. Nous avons ensuite évalué ses performances face au PI-Bus. La transcription de certains composants en VHDL a également permis d'avoir une estimation de leur surface et de leur fréquence de fonctionnement après routage.
Nous avons également étudié les complications liées à l'insertion de communications parallèles au sein du réseau d'interconnexion. Nous avons constaté que des paquets ne pouvaient plus progresser dans le réseau parce qu'ils étaient bloqués par d'autres, eux même déjà bloqués pour des raisons similaires. Cette situation d'inter-blocage a été analysée en utilisant des outils de preuve formelle, puis une solution à ce problème a ensuite été validée.
Enfin, des complications architecturales sont mises en relief dans l'étude topologique des sept réseaux SPIN possibles, allant de 4 à 256 ports. Cette étude a servi de base à la modélisation d'un générateur de réseaux SPIN/VCI, composants SystemC permettant de générer et de configurer un réseau SPIN/VCI avec le nombre de ports souhaité.

Mots clés

SPIN, Interface VCI, PI-Bus, SystemC, CASS, SoCLib, Wrapper, Micro-réseau, Système sur puce, Réseau d'Interconnexion, Vérification Formelle, Preuve, ProMeLa, Disydent, MUTEK, Synthèse, Placement, Routage, Analyse Temporelle.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Le contexte de l'étude

Un système intégré est un ensemble de composants communiquant les uns avec les autres au sein d'une même puce. Avec les progrès de l'intégration, c'est-à-dire la possibilité de mettre de plus en plus de transistors sur une même surface (Cf Figure 1.1 : Loi de Moore), les fabricants de puces ont voulu intégrer un maximum de fonctionnalités. Une telle politique répondait au triple objectif de diminuer la taille des circuits, réduire leur consommation, et augmenter leur vitesse.
Lorsqu'un système contient peu de composants, ces derniers peuvent directement communiquer par le biais de connexions propriétaires. C'est le cas des systèmes *mono-processeur*. Cependant cette façon de procéder n'est pas adaptée pour la communication au sein de systèmes intégrant de nombreux processeurs, certains devant communiquer avec plusieurs autres. La section suivante montre les avancées faites dans ce sens.

1.2 Propriétés des bus et des réseaux

Le bus fournit un mécanisme général d'interconnexion permettant à plusieurs composants de communiquer entre eux. L'utilisation du bus est une façon de standardiser les interfaces des composants et leur manière de communiquer, car les intégrateurs utilisent de plus en plus des composants provenant de divers fournisseurs.
Cependant les limites des architectures à base de bus commencent à être atteintes. Les systèmes actuels sont bien souvent multi-processeurs (ou multi-maîtres) et nécessitent une bande passante élevée avec un fort degré de parallélisme au niveau des communications. Pour de tels systèmes le bus a l'inconvénient d'être un goulot d'étranglement, du fait de sa bande passante non extensible, de l'absence de communications simultanées et de sa fréquence de fonctionnement limitée.

1.3 Systèmes multiprocesseurs à mémoire partagée

Pour permettre une migration aisée d'une architecture à base de bus vers une architecture utilisant un réseau commuté tel que SPIN [Gue00][Pie00], il faut que cette dernière solution fournisse aux concepteurs de systèmes la même interface et le même type de service que le bus. Ces deux conditions sont remplies par l'utilisation du standard VCI. Le standard VCI [Vir97] (`Virtual Component Interface`), normalisé par le consortium VSIA, définit à la fois une interface et un protocole de communication de type "espace mémoire adressable partagé", qui peut être implanté aussi bien avec un bus système traditionnel de type AMBA [ARM99] ou PI-Bus [Ope94], qu'avec un réseau d'interconnexion à

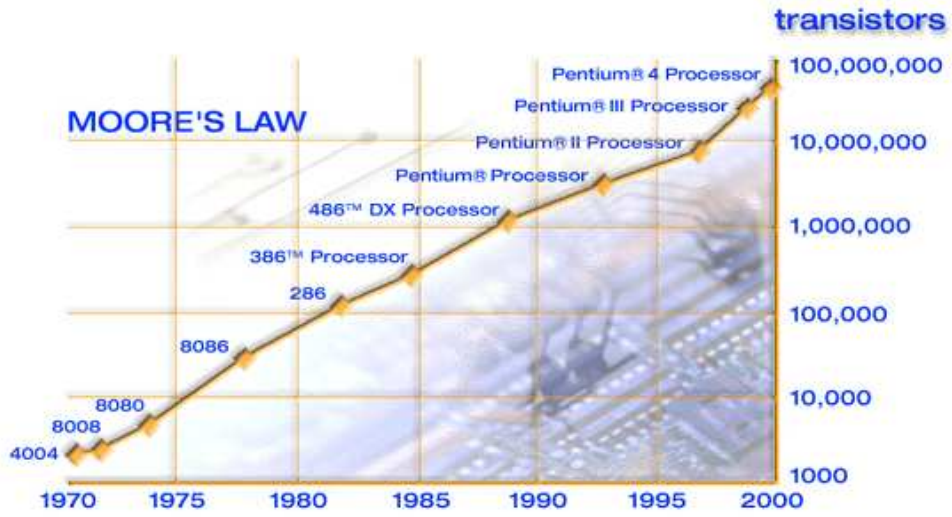

FIG. 1.1 – Loi de Moore (source : http ://www.intel.com/research/silicon/mooreslaw.htm)

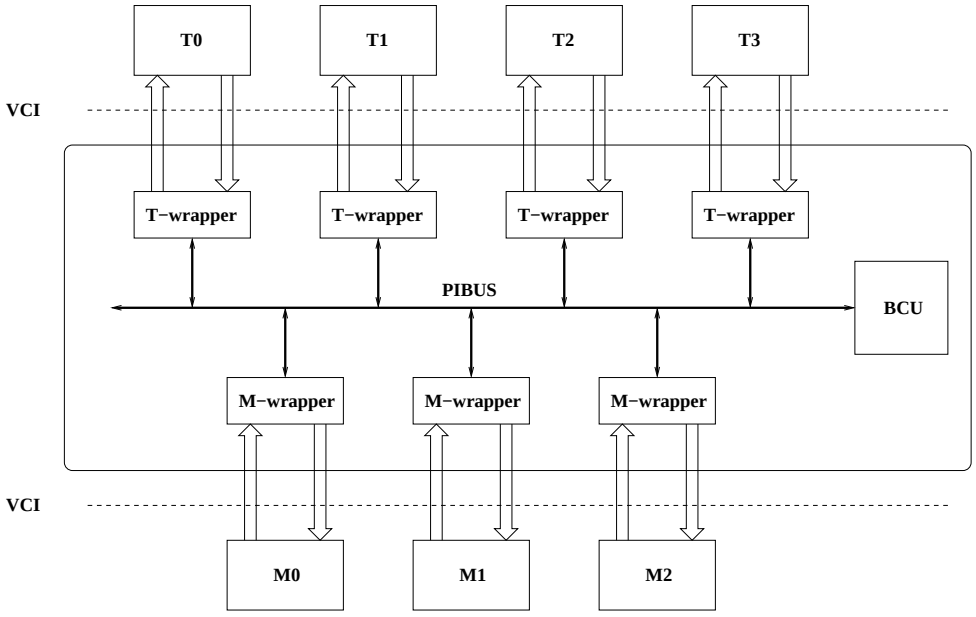

FIG. 1.2 – Exemple d'architecture VCI avec un PI-Bus

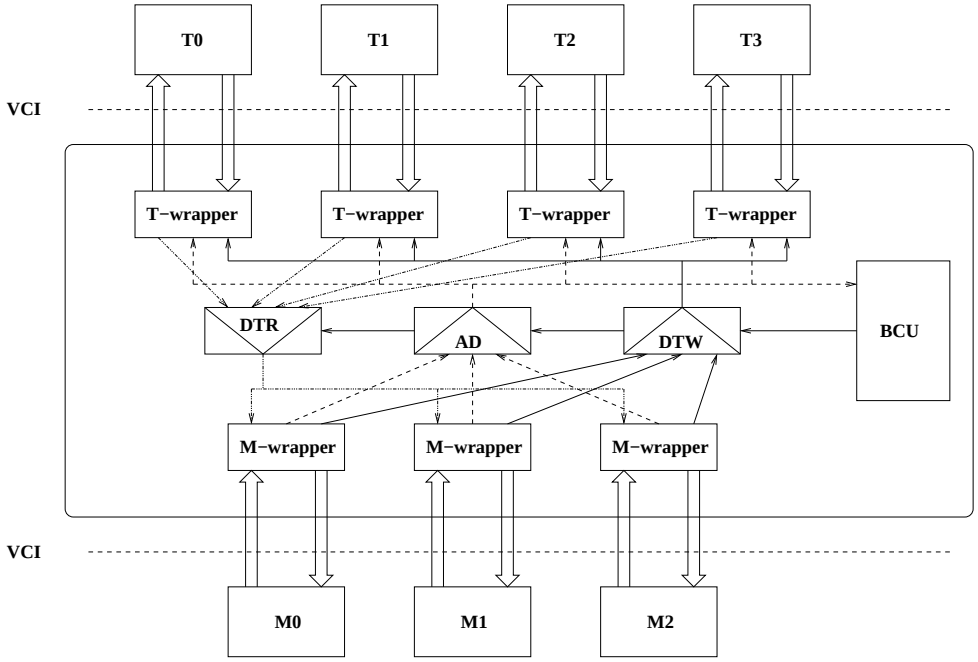

FIG. 1.3 – Exemple d'architecture VCI avec un bus AMBA

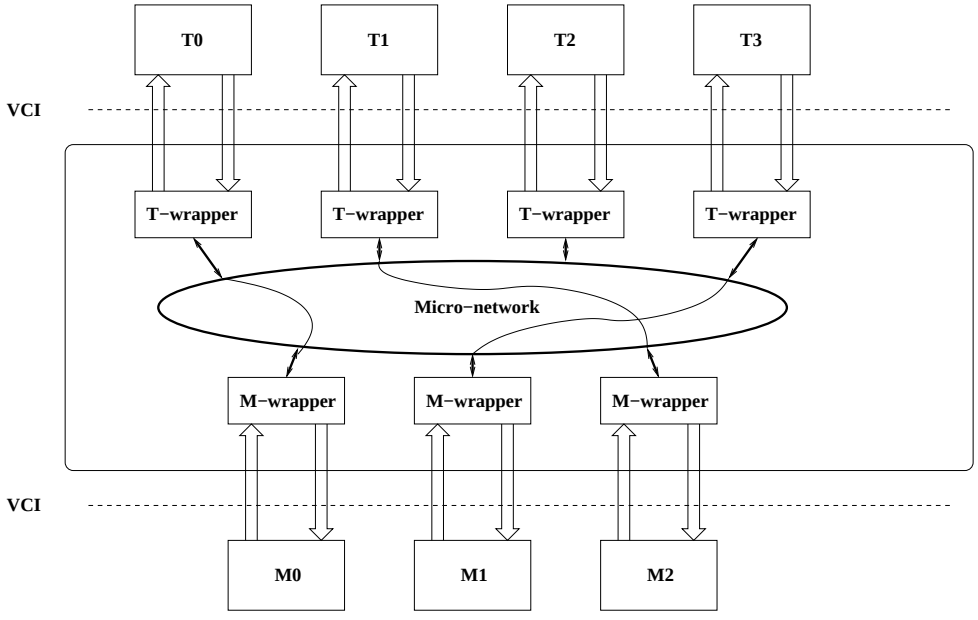

FIG. 1.4 – Exemple d'architecture VCI avec un réseau SPIN

commutation de paquets. Les Figures 1.2, 1.3 et 1.4 montrent des exemples d'architecture VCI avec respectivement un PI-Bus, un Bus AMBA et un réseau SPIN. Les initiateurs sont représentés par les rectangles $M0$, $M1$ et $M2$. Les cibles sont $T0$, $T1$, $T2$ et $T3$. Ces initiateurs et ces cibles ont tous des interfaces VCI et, pour les connecter aux différentes structures d'interconnexion, nous utilisons des wrappers. Les wrappers sont des convertisseurs de protocole propres à chaque type de structure d'interconnexion. On a donc respectivement des wrappers VCI/PI-Bus, des wrappers VCI/AMBA et des wrappers VCI/SPIN. De plus, pour chacune des ces catégories, nous avons ceux rattachés aux initiateurs, les $M - wrapper$ et ceux rattachés aux cibles, les $T - wrapper$.
Grâce à VCI nous pouvons donc réutiliser les mêmes composants (IP cores : Intellectual Property cores) d'une architecture à l'autre en ne faisant varier que la structure d'interconnexion. Cependant, l'apparition de nouveaux concepts nécessite de nouveaux outils pour les exploiter et, dans bien des cas, une nouvelle façon de voir les choses.[Hug97].

1.4 Les problèmes posés par l'utilisation d'un micro-réseau

On utilise de manière croissante des structures d'interconnexion de type cross-bar ou réseau commuté. Elles ont le double avantage de permettre la montée en fréquence ainsi que la réalisation de plusieurs communications simultanées.
Le concept `SPIN` (*Scalable Programmable Integrated Network*) d'architecture de communication à haut débit pour systèmes intégrés sur puce, mis en oeuvre au laboratoire LIP6, dérive de l'expérience acquise dans le domaine des calculateurs parallèles [M. 93][C. 91][Jac93]. Ces machines ont de gros besoins en bande passante et utilisent souvent comme substitut au traditionnel bus système, des réseaux d'interconnexion multi-étages mettant en oeuvre des liaisons point à point et des routeurs.

Le remplacement du bus par un micro-réseau pose de nombreux problèmes :

- Celui de la latence : l'utilisation d'un micro-réseau en remplacement d'un bus permet d'augmenter massivement la bande passante au prix d'une augmentation de la latence. Une première question est de comparer les performances de ces deux types de communications.
- Celui des inter-blocages : le partage des mêmes liens du réseau par des paquets de nature divers (requêtes et réponses) entraîne non seulement des problèmes de contention, mais aussi d'inter-blocage.
- Enfin celui de la génération : un réseau SPIN peut avoir différentes tailles (en nombre de ports). Lorsque l'on a plusieurs dizaines de composants, il devient difficile de

configurer le réseau et de connecter les abonnés avec les bons paramètres.
Beaucoup de questions sont posées. Nous y apporterons les réponses au fil de la thèse suivant le plan développé dans la section ci-dessous.

1.5 Organisation du manuscrit

Dans ce manuscrit, nous partons des concepts généraux qui nous ont guidés dans les solutions concrètes que nous proposons. Chaque fois que c'est possible, nous indiquons les cheminements et les raisonnements qui nous ont amenés à la solution présentée.

Dans le chapitre 2, `Problématique`, nous présentons en détail les problèmes auxquels nous nous sommes attaqués.

Dans le chapitre 3, `Norme VCI et structures d'interconnexion PI-Bus et SPIN`, nous traitons de la norme VCI du bus standard PI-Bus, du réseau SPIN et des mécanismes de contrôle de flux de type FIFO et CREDITS.

Dans le chapitre 4, `Règles de transformation VCI/SPIN`, nous précisons la terminologie utilisée pour la description des entités circulant sur le réseau, ainsi que la structure des mots et des paquets. Nous décrivons également comment on passe d'un *paquet VCI* à un *paquet SPIN* et inversement.

Dans le chapitre 5, `Architecture des wrappers VCI/SPIN`, nous présentons les choix architecturaux, ainsi que la méthodologie de conception bas niveau des wrappers. C'est-à-dire que nous présentons les phases de description en VHDL, de synthèse, de placement, de routage et d'analyse temporelle.

Dans le chapitre 6, `Comparaison des performances SPIN / PI-Bus`, nous évaluons les architectures à base de PI-Bus face à celles à base de réseau SPIN. La comparaison se fait d'abord sur une charge synthétique (plus facile à mettre en oeuvre et à caractériser) puis sur une application réelle.

Dans le chapitre 7, `Vérification formelle de l'inter-blocage`, nous analysons le problème des inter-blocages et proposons une solution, puis nous validons cette dernière à travers la modélisation d'un petit système SPIN en ProMeLa.

Dans le chapitre 8, `Génération et configuration automatique d'un réseau SPIN`,

nous proposons un algorithme d'interconnexion suffisamment souple pour permettre de générer automatiquement des réseaux SPIN, tout en respectant une topologie sans interblocages.

Dans le chapitre 9, `Conclusion`, nous résumons le travail effectué au cours de cette thèse et nous mettons en avant les difficultés rencontrées.

Les annexes suivantes servent à éclairer notre propos :

Dans l'annexe A, `Différents environnements de simulation`, nous présentons les différents environnements élaborés pour les tests et mises au point des wrappers ainsi que des autres composants VCI.

Dans l'annexe B, `Architecture détaillée des wrappers`, nous complétons le chapitre 5 en détaillant les architectures internes des wrappers VCI/SPIN.

Chapitre 2

Problématique

Sommaire

Ce chapitre a pour objectif de soulever différents problèmes liés à l'insertion du parallélisme au niveau des communications dans un système multiprocesseurs à espace d'adressage partagé. Ces problèmes se traduiront par des questions qui concluront le chapitre.

2.1 Un micro-réseau pour remplacer le bus

La loi de Moore met clairement en évidence le problème de la complexité croissante des systèmes intégrés, qui contiennent des dizaines de composants et qui nécessitent une bande passante de plus en plus importante. La structure d'interconnexion devient alors le facteur limitant les performances du système.

Les structures d'interconnexion de type bus ont l'avantage de reproduire sur puce des mécanismes d'interconnexion modulaires, bien connus des architectes systèmes. Leur coût matériel est limité puisque les fils peuvent assez facilement être routés autour et par dessus les composants. Ce facteur a cependant de plus en plus tendance à s'effacer devant les problèmes posés par une forte intégration sur une même puce :

- L'absence de parallélisme au niveau des communications : avec l'augmentation du nombre de composants, le bus devient alors le goulot d'étranglement du système puisqu'une seule communication est possible à un instant donné.
- La lenteur : on utilise des technologies de plus en plus fines permettant d'aller de plus en plus vite, mais cet élan se heurte à la technologie intrinsèque des bus. En effet, le bus est conçu avec des techniques de circuiterie de type multi-émetteurs, ce qui l'empêche de monter en fréquence et le rend d'autant plus lent que le nombre d'abonnés est grand : plus il y a de composants connectés au bus et plus la charge capacitive est importante.

L'utilisation d'un micro-réseau supprime ces inconvénients :

- Les communications simultanées sont accessibles par construction,
- La montée en fréquence est autorisée grâce à l'utilisation de liaisons point-à-point entre deux routeurs du réseau et entre un routeur et un abonné. Le micro-réseau offre un ensemble de liaisons point à point reliant deux composants qui échangent des informations.

Le prix à payer est évidemment une augmentation de la latence.

Le schéma de la figure 2.1 montre dans sa partie gauche une architecture de bus et dans sa partie droite une architecture utilisant un réseau multi-étages (telle que SPIN). Nous voyons clairement dans cette dernière structure la possibilité de communications simultanées.

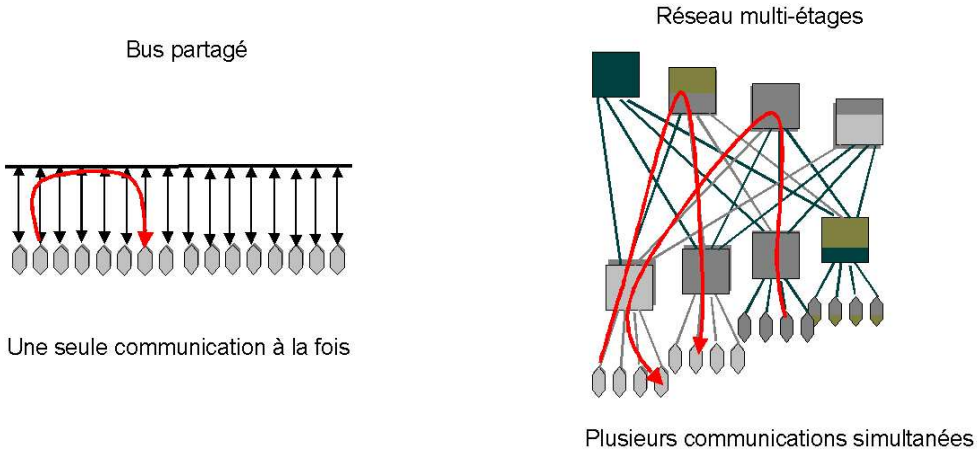

FIG. 2.1 – Schémas des communications selon le type d'interconnexion

2.2 Système multiprocesseur à espace d'adressage partagé

Les bus sont généralement utilisés dans des systèmes à espace d'adressage partagé [Joh96] avec des primitives de communication de type `read` ou `write`. Le défi à relever est de pouvoir fournir les mêmes services autour d'une structure d'interconnexion de type micro-réseau.

Nous définissons un système à espace d'adressage partagé comme étant un système où tous les composants ont la même vision de l'espace d'adressage. Ces composants sont soit des initiateurs, soit des cibles.
Nous appelons `initiateur` un composant pouvant initier une transaction vers une cible, en émettant une requête. Quant à la `cible`, elle se contente de répondre à l'initiateur à travers une réponse. Une `transaction` est l'ensemble des deux transferts constitués par une requête et une réponse. A chaque cible est attribué un segment de l'espace d'adressage. L'initiateur fournit l'adresse de la case mémoire qui fera l'objet de la transaction. Les bits de poids forts de l'adresse sont utilisés par le système de routage pour la sélection de la cible et l'acheminement des données.

Dans un bus, la sélection de la cible se fait dans le contrôleur de bus ou BCU (*Bus Controller Unit*) et l'acheminement est réalisé sans ambiguïté car une seule cible est sélectionnée par le BCU. Dans le cas d'un micro-réseau, la sélection est faite à l'entrée du

réseau où une table de transcodage donne, pour chaque adresse, le numéro de la cible correspondante au sein du réseau. Ce numéro est ensuite réutilisé par le système de routage pour acheminer correctement le paquet, c'est-à-dire que l'aiguillage s'effectue au niveau de chaque noeud du réseau. L'algorithme de routage est donc un algorithme distribué et installé sur tous les noeuds du réseau.

Les requêtes sont des lectures et des écritures. Plus précisément, nous pouvons lire des mots et écrire des mots, des demi-mots ou des octets. Lors d'une lecture, l'initiateur demande à la cible de lui fournir une donnée ; pour une écriture, il envoie à la cible une adresse suivie de la donnée à stocker. La réponse de la cible constitue alors un accusé de réception.

2.3 Utilisation de la norme VCI

Le problème de l'hétérogénéité des composants se pose dès que l'on parle d'assemblage ou d'intégration : les composants sont souvent conçus par des équipes différentes dans des contextes d'utilisation particuliers. Le travail des intégrateurs est encore compliqué par le fait que chaque composant dispose d'une interface propriétaire ainsi que d'un protocole de communication plus ou moins complexe. De ce constat découle l'idée de standardiser les interfaces des composants, et donc d'uniformiser leur manière d'accéder au réseau. Nous avons opté pour le standard `VCI` (`Virtual Component Interface`), qui présente deux grands avantages :

- La simplicité du protocole d'accès au réseau, reposant sur un mécanisme de contrôle de flux de type FIFO,
- La possibilité d'effectuer des transactions éclatées, c'est-à-dire qu'un même initiateur peut envoyer plusieurs requêtes avant que les réponses aux précédentes requêtes ne soient revenues. Ce mécanisme permet de masquer une forte latence de la structure d'interconnexion ou des cibles.

Les détails de cette norme seront décrits au chapitre suivant.

Les communications point à point entre les routeurs du réseau SPIN utilisent un protocole à CREDITS qui définit l'interface SPIN. Les interfaces VCI et SPIN diffèrent tant par le nombre de signaux que par leur protocole de communication. Pour connecter un composant doté d'une interface VCI à un port SPIN du réseau, il faut un adaptateur doté de ces deux interfaces. Cet adaptateur, encore appelé `wrapper` est un convertisseur de protocole qui, dans le cas présent, permet le passage du protocole VCI au protocole

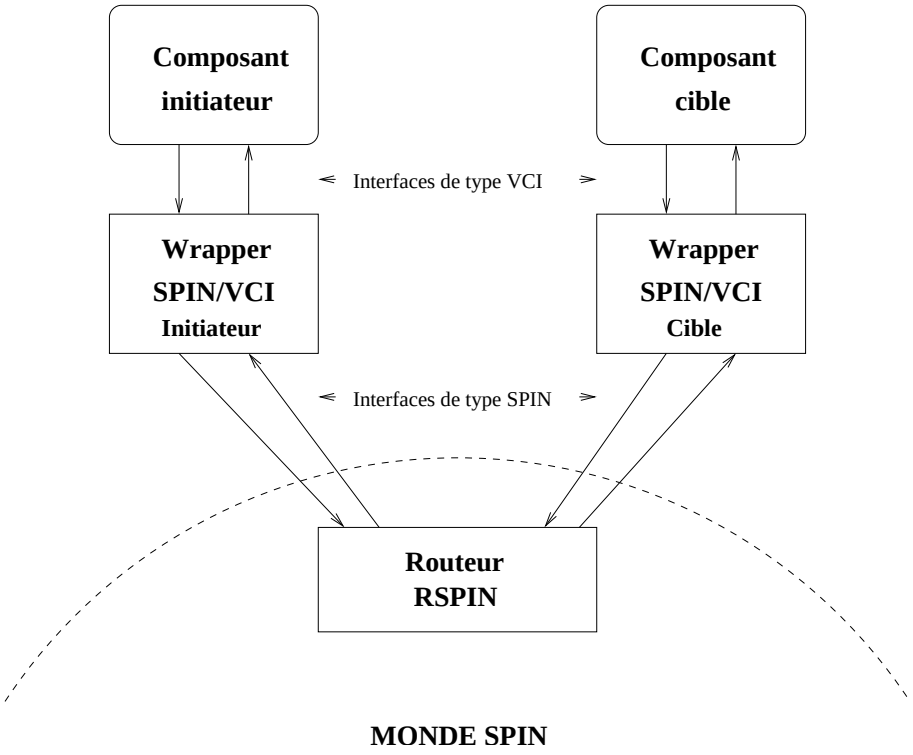

FIG. 2.2 – Représentation simplifiée d'une liaison initiateur-cible

SPIN et inversement. La distinction entre abonnés de type initiateur et abonnés de type cible impose de définir deux types de wrappers :

- les wrappers initiateurs rattachés aux initiateurs ;
- et les wrappers cibles rattachés aux cibles.

Les deux types de wrappers n'ont pas les mêmes fonctions. Le wrapper initiateur devra notamment, à partir de l'adresse VCI du paquet, générer un numéro de port SPIN correspondant à l'adresse (ou numéro) de la cible sur le réseau. Ce transcodage est réalisé en utilisant une petite table de routage, qui fait correspondre les bits de poids forts de l'adresse au numéro du port SPIN utilisé. La figure 2.2 présente un composant initiateur et un composant cible, munis de leur wrappers, qui communiquent par le biais d'un routeur.

Enfin, on parlera indifféremment de wrapper VCI/SPIN ou de wrapper SPIN/VCI ; seul importe le fait qu'il soit de type initiateur ou cible.

2.4 Evaluation des performances

Nous voulons comparer une architecture à base de PI-Bus à une architecture utilisant un réseau SPIN. Pour être pertinente, une telle comparaison doit se faire sur des archi-

tectures identiques en termes d'abonnés (initiateurs et cibles) et pour des applications identiques générant des charges identiques. La seule différence entre ces architectures doit être la structure d'interconnexion.

Il est donc nécessaire d'avoir des composants initiateurs ou cibles dont le comportement ne fait aucune hypothèse sur la nature de la structure d'interconnexion. La norme VCI ne fait pas l'hypothèse d'un acheminement dans l'ordre. Le réseau SPIN par exemple ne garantit pas l'ordre d'acheminement car il emploie un routage adaptatif. Pour aller d'un point à un autre, plusieurs chemins sont possibles. Par conséquent, il appartiendra aux initiateurs qui souhaitent émettre plusieurs requêtes sans attendre les réponses aux requêtes précédentes de réaliser l'appariement entre requêtes et réponses. Il devront pour cela utiliser les identificateurs de transaction définis par la norme VCI.

L'évaluation des performances est réalisée par deux types de tests :

- Le premier test permet d'obtenir un trafic synthétique dans le but de surcharger la structure d'interconnexion et de vérifier la robustesse architecturale des composants et des protocoles de communication. Des générateurs analyseurs de paquets permettent d'injecter dans le réseau un trafic que l'on peut ajuster de façon précise et contrôlée.
- Le second test met en oeuvre une application réelle, pour avoir un trafic ayant des caractéristiques différentes de celles obtenues avec un trafic synthétique. Pour ce faire, nous utilisons une architecture comportant plusieurs processeurs dotés de caches, des bancs mémoires, un contrôleur de sémaphores et un écran d'affichage. Donc des composants un peu plus variés et élaborés, pour un travail moins élémentaire. Aux simples requêtes de lecture du test de surcharge viennent s'ajouter des requêtes d'écriture de mots, de demi-mots et d'octets. Les tailles des requêtes ne sont alors plus fixes et il est désormais possible de tester des comportements un peu plus "intelligents", par exemple la synchronisation des processeurs, la gestion des sections critiques à l'aide de sémaphores, ou encore des changements de contexte aussi bien logiciels que matériels.

2.5 Analyse des inter-blocages

Initialement, le réseau SPIN a été conçu pour un trafic unidirectionnel de type `flux continu`. Pierre Guerrier a montré dans sa thèse [Gue00] que, pour un tel trafic, la topologie en arbre quaternaire élargi était exempt d'inter-blocage (ou `deadlock`).
Cependant, en utilisant le réseau SPIN pour des systèmes à mémoire partagée, on change la nature du trafic. En effet, toute transaction VCI entre un initiateur i et une cible j est

constituée de deux transferts de données : une requête de i vers j et une réponse de j vers i. Cela change alors les dépendances entre les paquets circulant dans le réseau : en mettant au dessus du réseau un service VCI, les informations qui circulent sont désormais des requêtes et des réponses avec, comme contrainte forte, que l'avancement du traitement des requêtes dépend de l'avancement du traitement des réponses. Autrement dit, le blocage d'une réponse entraîne alors le blocage de la requête correspondante.
Dans un réseau à commutation de paquets véhiculant simultanément des requêtes et des réponses sur le même support, les paquets peuvent donc se trouver dans une configuration telle qu'aucun ne puisse arriver à destination, car ils se bloquent les uns les autres. On se retrouve alors en situation d'inter-blocage. Il est donc nécessaire de définir un mécanisme général garantissant l'absence d'inter-blocage, même dans le cas d'un trafic de type requête/réponse.

2.6 Le problème de la génération et de la configuration du micro-réseau SPIN

Le réseau SPIN est une structure d'interconnexion dont la configuration est complexe. Il se compose de plusieurs étages de routeurs, connectés entre eux selon une topologie déterminée, avec un paramétrage particulier. En effet, chaque routeur a un comportement qui dépend de sa position dans le réseau. Tous les routeurs ont donc des paramétrages différents. Les wrappers VCI/SPIN ont également des comportements qui dépendent de leur position dans le réseau. Là encore les valeurs des paramètres ne sont pas triviales.

Afin de permettre une utilisation ou une réutilisation aisée du réseau, il est nécessaire d'automatiser sa génération. En effet, cela permet non seulement de gagner du temps dans sa mise en place, mais aussi de réduire les risques d'erreurs de codage. Enfin, sous cette forme, le réseau est aussi plus facilement exportable.
L'utilisation du réseau peut-être réalisée simplement à travers un composant générique qui sera instancié dans le système avec les bonnes valeurs des paramètres. Certains paramètres sont donnés par l'utilisateur, mais d'autres doivent être calculés automatiquement. En effet, on peut paramétrer de l'extérieur le nombre d'abonnés au réseau et l'emplacement des wrappers initiateur et cible. En revanche, il est difficile d'attribuer à chaque routeur un identifiant (une donnée utile pour le routage). C'est donc le générateur qui fournira ce paramètre à chaque routeur.
Le défi de cette étude est d'élaborer et de coder un algorithme permettant la création de réseaux SPIN de n'importe quelle taille, mais aussi trouver une manière simple de paramétrer l'ensemble.

2.7 Conclusion

Les trois questions qui suivent constituent l'objet de cette thèse :

1. L'intérêt d'utiliser un réseau sur puce est directement lié au phénomène de saturation des bus, lorsqu'il s'agit d'interconnecter des dizaines de composants et de faire circuler de gros volumes de données. A partir de quelle complexité, en nombre de composants, le réseau SPIN est-il meilleur qu'un bus partagé tel que le PI-Bus ?

2. Alors que sur un bus, une seule communication est possible à la fois, dans un réseau SPIN on peut avoir plusieurs communications simultanées. Le traitement d'un trafic de type "requête/réponse", caractéristique des systèmes à espace d'adressage partagé, introduit un risque d'inter-blocage dans le réseau. Quelles solutions architecturales peut-on mettre en oeuvre pour éviter les inter-blocages ? Peut-on prouver que cette solution supprime les inter-blocages ?

3. Enfin, lorsque l'on parle de systèmes contenant plusieurs dizaines de composants, on pense aussi aux milliers de lignes de code pour les mettre en place et les configurer. L'élaboration du réseau d'interconnexion étant fort complexe, nous avons besoin d'un générateur de réseau SPIN/VCI. Comment automatiser la génération et la configuration du réseau SPIN/VCI ?

Chapitre 3

Norme VCI et structures d'interconnexion PI-Bus et SPIN

Sommaire

Dans ce chapitre nous présentons en détail la norme VCI ainsi que les interconnect PI-Bus et SPIN. La norme VCI déterminera la façon dont les paquets seront construits et routés.

3.1 La norme VCI

3.1.1 Présentation générale

La norme `VCI` ou `Virtual Component Interface` [Vir97] a été définie par le consortium VSIA (*Virtual Socket Interface Alliance*) dans le but de standardiser l'interface entre les composants (initiateurs ou cibles) et la structure d'interconnexion. En effet, chaque constructeur dispose de ses propres `IP` (ou `Intellectual Property`) avec leurs interfaces propriétaires. L'utilisation de VCI vise donc à faciliter leur réutilisation par n'importe quel assembleur de systèmes, en standardisant leur interface et leur protocole de communication. VCI permet donc un découplage complet entre l'élaboration des composants, la mise au point de la structure d'interconnexion et leur assemblage.
Du coté des composants (initiateurs ou cibles), il existe deux manières de procéder :

- concevoir dès le départ un composant ayant une interface VCI ;
- englober le composant préexistant dans un wrapper lui donnant ainsi une interface VCI. La Figure 3.1 illustre ce dernier cas.

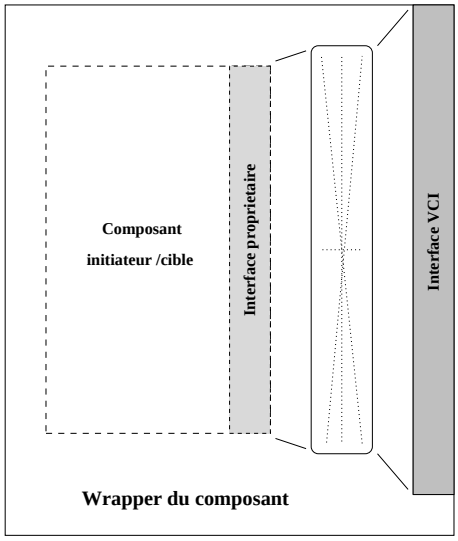

FIG. 3.1 – Wrapper d'un composant(initiateur ou cible)

Notons qu'une confusion peut être faite au niveau des termes employés. Le wrapper englobant l'initiateur pour lui donner une interface VCI est le `wrapper de l'initiateur`,

et le wrapper permettant l'accès au réseau SPIN est le `wrapper du réseau`. Par la suite on ne parlera que du second type de wrapper car tous les composants utilisés auront une interface VCI.

Il existe trois familles d'interfaces dans VCI : `Peripheral`, `Basic` et `Advanced`. Ces familles se distinguent par une complexification croissante, la première étant contenue dans la deuxième, et la deuxième dans la troisième. Dans le cadre de notre étude, nous avons exploité la version complète de la norme VCI (i.e VCI Advanced).

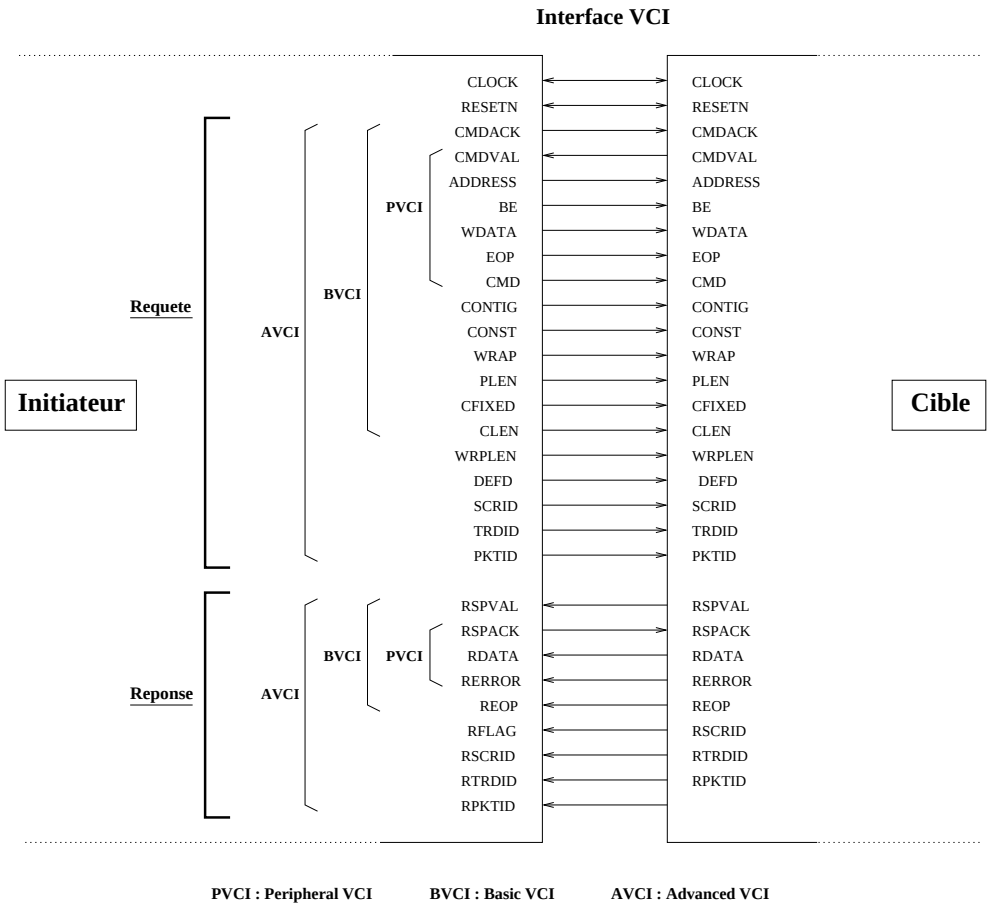

FIG. 3.2 – Schéma d'une liaison initiateur/cible à l'aide de VCI

La figure 3.2 présente une liaison VCI entre un initiateur et une cible. Les signaux d'interface se répartissent en deux groupes : ceux gérant et caractérisant les requêtes et ceux qui gèrent et caractérisent les réponses. Dans chacun des groupes nous avons fait ressortir les différentes familles VCI : PVCI pour Peripheral VCI, BVCI pour Basic VCI et AVCI pour Advanced VCI. Au sein de chaque famille VCI, nous trouvons *l'interface*

initiateur qui est celle de l'initiateur VCI et *l'interface cible* qui est celle de la cible VCI. Au niveau de ces interfaces, les signaux de sortie de l'une sont les signaux d'entrée de l'autre.
L'interface initiateur se reconnaît notamment au sens du signal `CMDVAL` qui indique l'émission d'une information de l'initiateur vers la cible. L'initiateur va activer ce signal tandis que la cible va le recevoir. Un initiateur VCI possède une interface VCI initiateur. En revanche, le wrapper initiateur (qui est une cible VCI) a une interface VCI cible. Par symétrie, le wrapper cible (qui est un initiateur VCI) a une interface VCI initiateur, et la cible VCI possède une interface VCI cible.

3.1.2 Signification des signaux et comportement

Nous appelons *mot* un ensemble de bits transmis en un cycle. L'ensemble des bits regroupés en signaux sur l'interface VCI constitue un *mot VCI*. Ce mot VCI contient 126 bits pour une requête et 64 bits pour une réponse. Ce qui suit décrit la signification des signaux constituant les mot-requêtes et mot-réponses de l'interface VCI.

Les significations des signaux du mot-requête sont les suivantes :

- **CMDVAL** : `Command Valid` indique que l'initiateur veut transmettre un mot d'un paquet requête à la cible.
- **CMDACK** : `Command Acknowledge` est utilisé par la cible pour indiquer à l'initiateur qu'elle est prête à recevoir un mot du paquet requête. Un mot est transféré lorsque les signaux `CMDVAL` et `CMDACK` sont simultanément à "1".
- **ADDRESS [31 :0]** : `address` est l'adresse de la cible de la transaction. La taille maximale d'une adresse est de 32 bits.
- **BE [3 :0]** : `Byte Enable` est un masque indiquant les octets valides du mot transféré.
- **CFIXED** : `Chain Fixed` indique que les champs `CMD`, `CONTIG`, `WRAP`, `CONST` et `PLEN` sont constants durant tout l'état haut du signal, et que la signification du champ `ADDRESS` est la même au sein des paquets constituant la chaîne.
- **CLEN [7 :0]** : `Chain Length` indique le nombre de paquets de la chaîne n'ayant pas encore été transférés.
- **CMD [1 :0]** : `Command` indique le type de l'opération effectuée par l'initiateur. Il est encodé sur deux bits de la façon suivante :
  - 00 : NOP, aucune donnée n'est transférée (optionnel)
  - 01 : Lecture, une donnée est demandée par l'initiateur à la cible.
  - 10 : Ecriture, une donnée est transférée de l'initiateur vers la cible.

. 11 : Lecture verrouillée, une donnée est demandée par l'initiateur à la cible, en utilisant en plus un mécanisme de verrouillage de la cible (optionnel).

- **CONTIG** : `Contiguous` indique que les adresses au sein d'un même paquet sont contiguës : elles sont deux à deux distantes de 4 (nombre d'octets constituant une adresse).
- **WDATA [31 :0]** : `Data` est la donnée transférée de l'initiateur vers la cible durant une opération d'écriture. Les octets valides sont définis par le signal `byte enable`.
- **EOP** : `End Of Packet` est positionné à "1" lors du transfert du dernier mot du paquet, de façon à indiquer que tous les mots du paquet on été transférés.
- **CONST** : `Constant` indique que l'adresse restera constante au sein du paquet. Lorsque le signal `CONST` est à "1", les signaux `CONTIG` et `WRAP` sont ignorés.
- **PLEN [7 :0]** : `Packet Length` indique la taille d'un paquet en octets.
- **WRAP** : `Wrap` est utilisé avec le signal `CONTIG` pour indiquer que les adresses seront wrappées avec une amplitude de `PLEN`.
- **WRPLEN [7 :0]** : `Wrap Length` indique l'amplitude du wrapping (2^{wrplen}).
- **DEFD** : `Defined` indique que l'initiateur et la cible se sont mis d'accord pour interpréter le `Byte Enable` d'une certaine façon.

Signaux optionnels constitutifs de la norme VCI avancée

- **SCRID [7 :0]** : `Source Identifier` est l'identifiant de la source, unique pour chaque initiateur du système.
- **TRDID [7 :0]** : `Thread Identifier` est l'identifiant local de la tâche. Il caractérise une tâche sur un initiateur. Des tâches envoyées par des initiateurs différents peuvent avoir le même numéro.
- **PKTID [7 :0]** : `Packet Identifier` est l'identifiant local du paquet. Il caractérise un paquet envoyé par un initiateur. Des paquets envoyés par des initiateurs différents peuvent avoir le même numéro.

Les significations des signaux du mot-réponse sont les suivantes :

- **RSPVAL** : `Response Valid` indique que la cible veut transmettre un mot d'un paquet réponse à l'initiateur.
- **RSPACK** : `Response Acknowledge` est utilisé par l'initiateur pour indiquer à la cible qu'il est prêt à recevoir un mot du paquet réponse. Un mot est transféré lorsque les signaux `RSPVAL` et `RSPACK` sont simultanément à "1".
- **RDATA [31 :0]** : `Response Data` est la donnée renvoyée par la cible à l'initiateur après une opération de lecture.

- **REOP** : `Response End Of Packet` est positionné à "1" lors du transfert du dernier mot du paquet de façon à indiquer que tous les mots du paquet on été transférés.
- **RERROR [2 :0]** : `Response Error` est positionné à "1" par la cible durant le transfert d'un paquet réponse pour indiquer qu'une erreur est survenue durant le traitement de la requête.
- **RFLAG [1 :0]** : `Response Flag` permet de préciser, de manière optionnelle, le type de réponse obtenue.

Signaux optionnels constitutifs de la norme VCI avancée

- **RSCRID [7 :0]** : `Response Source Identifier` est l'identifiant de la source. C'est la copie du champ SCRID retournée dans le paquet réponse.
- **RTRDID [7 :0]** : `Response Thread Identifier` est l'identifiant de la tâche. C'est la copie du champ TRDID retournée dans le paquet réponse.
- **RPKTID [7 :0]** : `Response Packet Identifier` est l'identifiant du paquet. C'est la copie du champ PKTID retournée dans le paquet réponse.

3.2 Le PI-Bus

3.2.1 Les concepts généraux

Le bus système est une structure d'interconnexion permettant à plusieurs processeurs ou coprocesseurs de partager de la mémoire. Le bus système utilisé pour cette étude est un bus standard pour les systèmes intégrés : le PI-Bus (Peripheral Interconnect Bus). Nous avons choisi ce bus comme référence en raison de sa présence dans un grand nombre d'environnement de développement aussi bien industriels qu'éducatifs.
Ce bus est le résultat de la collaboration de cinq grands acteurs européens de la micro-électronique : ARM (Advanced Risk Machine), Philips Semiconductors, SGS-Thompson Microelectronics, Siemens et Temic/Matra MHS. A l'époque l'objectif de cette collaboration était, vu le développement des capacités d'intégrations, de fournir une structure d'interconnexion rapide et suffisamment souple pour être utilisée avec toutes sortes de composants. Philips a notamment bâtis autour d'une hiérarchie de PI-Bus sa plate-forme Nexperia de traitement de flux mpeg-2.

Nous disposons au laboratoire de modèles de simulation précis au cycle et au bit près pour le contrôleur de bus (BCU) et pour les wrappers VCI/PI-Bus initiateur et cible.
Autour du bus coexistent trois entités : les maîtres, les esclaves et le contrôleur de bus (BCU).

Un maître est un abonné qui demande l'allocation du bus, qui émet des adresses ou des données à destination des esclaves. Il contrôle la transaction depuis son début.
Un esclave est un abonné qui répond à une transaction, en fournissant ou en acceptant les données. C'est lui qui contrôle la fin de la transaction.
Le contrôleur de bus à trois fonctions. La première est d'arbitrer entre les maîtres candidats pour l'allocation du bus. La deuxième est le décodage de l'adresse pour sélectionner l'esclave cible de la transaction. Enfin, le BCU contrôle la temporisation si l'esclave ne répond pas.

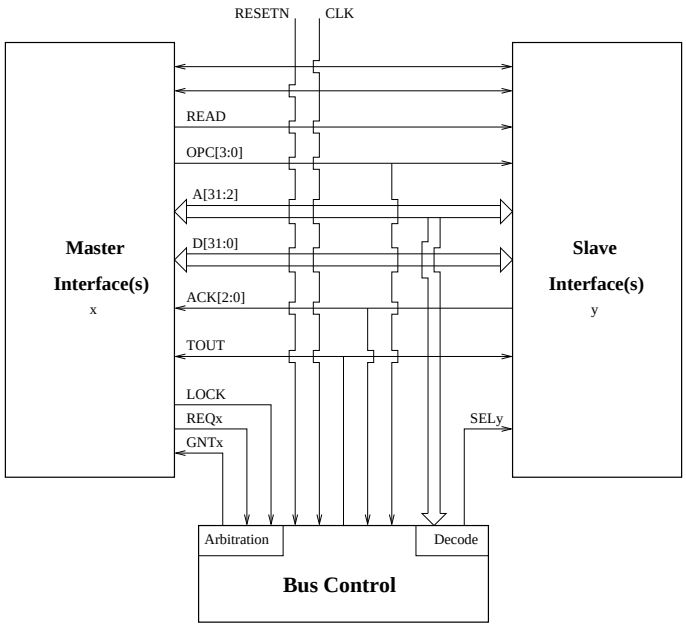

FIG. 3.3 – Architecture générale des connexion PI-Bus

3.2.2 Signification des signaux et comportement

Les signaux suivants constituent l'ensemble des fils reliant les initiateurs, les cibles et le BCU (Bus Controller Unit).

- **A [31 :2]** : `Address` est l'adresse à laquelle doit se faire la transaction. Elle est sur 30 bits car les 2 premiers bits de poids faibles sont toujours "00", les adresses étant alignées.
- **D [31 :0]** : `Data` est la donnée envoyée ou reçue. Du point de vue de l'initiateur c'est la donnée envoyée s'il s'agit d'une écriture ou la donnée reçue s'il s'agit d'une lecture.
- **READ** : `Read` indique si l'opération est une lecture ('1') ou une écriture ('0').

- **OPC [3 :0]** : `Op-code` indique la nature du transfert sur le bus.
- **ACK [2 :0]** : `Acknowledge` permet à l'esclave d'indiquer si l'opération s'est bien déroulée.
- **TOUT** : `Time out` est utilisé par le BCU pour mettre fin à une transaction après un silence prolongé de la cible.
- **LOCK** : `Lock` est utilisé par un maître pour verrouiller le bus.
- **REQx** : `Request x` indique au BCU que le maître x veut accéder au bus.
- **GNTx** : `Grant x` indique au maître x qu'il a remporté l'arbitrage.
- **SELy** : `Select y` sélectionne l'esclave y.

3.3 Le réseau SPIN

3.3.1 Généralités

Le concept du réseau SPIN a été développé par Pierre Guerrier durant sa thèse au Laboratoire d'Informatique de Paris 6 (LIP6) [Gue00]. Rappelons que les éléments fondamentaux nécessaires à la compréhension des chapitres suivants et de l'implantation matérielle du coeur du réseau, c'est-à-dire de la macro-cellule du routeur (`RSPIN`) qui est le sujet de la thèse d'Adrijean Andriahantenaina.

Le réseau SPIN est basé sur une topologie en arbre quaternaire élargi ("fat-tree") : chaque routeur RSPIN, noeud de l'arbre, possède quatre fils et quatre pères. Les chemins montants sont équivalents, mais les chemins descendants sont déterminés en fonction du numéro de destination du paquet à router. Les liens du réseau SPIN sont des liens bidirectionnels point-à-point. Nous donnons simplement à titre indicatif, à la figure 3.4, un exemple d'architecture de réseau SPIN à 16 ports.

Le mécanisme de contrôle de flux utilisé est de type `CREDITS` (Cf Figure 3.9). Le producteur utilise une estimation du taux de remplissage du consommateur afin de réguler l'envoi des données. Un tel mécanisme permet d'insérer des barrières de registres sans perte d'information, ce que ne permet pas un contrôle de flux de type `FIFO`. La section 3.3.7 présente une comparaison des mécanismes de contrôle de flux de type FIFO et CREDITS.

La communication au sein du réseau s'effectue selon une logique dite *postale* [Puj99] [Mél97] [J. 97]. Les hypothèses sont les suivantes :

- Aucun message n'est créé par le système de routage ;
- Le délai d'acheminement des messages est `fini` (bien que non déterminé) et il n'y a aucune perte de message ;

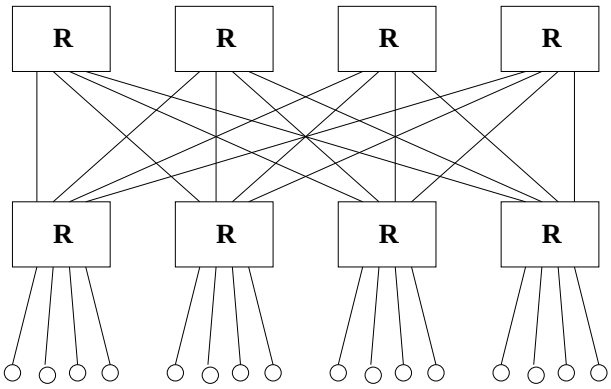

FIG. 3.4 – Schéma simplifié du réseau `SPIN` à 16 ports

- Les messages émis d'une entité vers une autre ne conservent pas forcément, lors de la réception, l'ordre qu'ils avaient lors de l'émission ; il peut y avoir déséquencement ;
- La transmission des messages se fait sans altération ni modification de ceux-ci.

3.3.2 L'arbitrage

On peut distinguer deux sortes d'arbitrage : les arbitrages entre abonnés et les arbitrages au sein d'un routeur.

. Au niveau des abonnés, il n'y a aucun mécanisme d'arbitrage entre les initiateurs comme dans le cas d'un bus. Donc lorsque plusieurs initiateurs veulent accéder à la même cible, ils envoient simultanément leurs requêtes. Ces requêtes arrivent à la cible dans un ordre arbitraire dépendant de la charge du réseau, et c'est selon cet ordre qu'elles sont traitées.

. Au sein d'un routeur, l'arbitrage est local au port de sortie. Comprenons par là que chaque port de sortie possède son propre mécanisme d'arbitrage. Si plusieurs ports d'entrée demandent le même port de sortie, un mécanisme de priorité tournante garantit l'absence de famine.

3.3.3 Structure d'un paquet SPIN

Les informations qui circulent sur le réseau SPIN sont des `paquets` (Cf Figure 3.5). Un paquet SPIN est une suite de mots de 36 bits dont le nombre n'est pas borné. Le premier mot d'un paquet, appelé `en-tête`, possède un marqueur de début de paquet (`BP : Begin of Packet`) et le dernier mot, un marqueur de fin de paquet (`EP : End of Packet`).
L'ensemble des mots qui suit le mot d'en-tête forme la `charge utile` du paquet. Chacun

de ces mots peut être marqué par le marqueur `ERR`, dans le cas où l'information qu'il transporte est erronée. En effet, la signalisation d'erreur dans VCI ne se fait pas par paquet, mais par mot.
Un mot de 36 bits est constitué de 32 bits de données, de 3 bits d'étiquette qui permettent de typer les mots, et d'un bit de parité. Ces quatre derniers bits constituent le *fanion* (Cf Figure 3.6). Les marqueurs EP, BP et ERR sont transportés dans le champ étiquette.

Remarque :

Un mot est transmis en un cycle. Donc si le réseau n'est pas congestionné, n mots seront transmis en n cycles. Pour la détection d'erreurs de transmission, on ajoute un bit de parité.

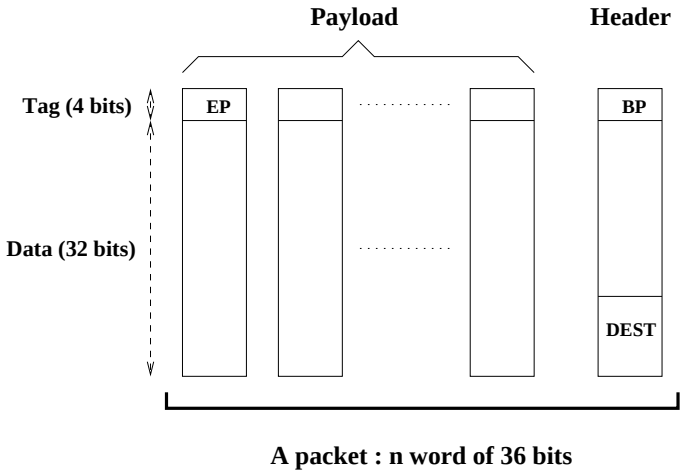

FIG. 3.5 – Structure d'un paquet SPIN

3.3.4 Routage d'un paquet SPIN

SPIN met en oeuvre un routage distribué, dynamique, adaptatif, de type `wormhole`. Il est distribué parce que le choix de l'aiguillage d'un paquet se fait dans chaque routeur et non pas dans un organe central comme le fait le BCU du PI-Bus. Ce routage est dynamique parce que les chemins sont refermés après la transmission du dernier mot de chaque paquet. Les chemins montants étant équivalents, plusieurs chemins sont possibles pour aller d'un abonné à un autre. L'adaptativité se traduit par le choix fait en fonction de la congestion du réseau.
Le routage de type wormhole a pour but de limiter la latence. En effet, un paquet est perçu par le réseau comme étant une entité atomique. L'ensemble des mots constituant

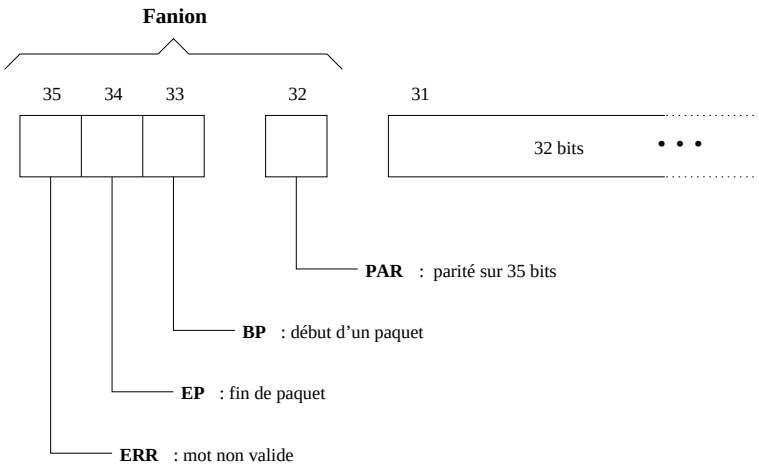

FIG. 3.6 – Structure du fanion

le paquet suit le chemin tracé par le premier mot. Ainsi un paquet pourra se trouver à cheval, sans risque de déséquencement, sur plusieurs routeurs simultanément. L'intérêt de ce choix est d'éviter d'avoir à mémoriser l'ensemble du paquet dans chaque routeur avant de le ré-émettre.
Le premier mot d'un paquet SPIN contient nécessairement le numéro du port destinataire. Ce numéro, codé sur 8 bits, est utilisé par le réseau pour acheminer le paquet vers le port destinataire. Chaque port est donc identifié par un numéro unique imposé par la topologie du réseau ; il ne peut donc être modifié.

3.3.5 Réalisation pratique du routage

Nous allons maintenant détailler plus concrètement la gestion des 8 bits d'adresse par les routeurs. Chaque routeur est caractérisé par un identifiant id et l'appartenance à un niveau l. L'adresse SPIN sur 8 bits (qui définit le numéro du port destination) se décompose en 4 groupes de 2 bits, chacun étant numéroté de 0 à 3. Lorsque le paquet arrive dans un routeur de niveau l, on compare les $[7 : 2 * l]$ bits à l'identifiant. S'ils sont différents, le paquet est routé vers l'un des ports supérieurs. S'ils sont identiques, on regarde les bits du groupe $l-1$. Ces deux bits codent le numéro de sortie du port inférieur vers lequel sera routé le paquet.

3.3.6 Architecture générale d'un réseau SPIN

L'élément de base du réseau SPIN est le routeur RSPIN, qui est un noeud du réseau d'interconnexion. La figure 3.7 représente l'architecture externe du routeur RSPIN. La

macro-cellule RSPIN possède 8 ports bidirectionnels pilotés par un mécanisme de contrôle de flux de type `CREDITS`. Ce mécanisme est décrit à la section suivante et comparé au mécanisme de type `FIFO`.
La structure d'un port SPIN est décrite à la figure 3.8. Elle se compose de deux nappes de fils de 36 bits pour la circulation des données (une nappe pour l'entrée et une pour la sortie) et de 4 bits de contrôle de flux (2 pour les données entrées et 2 pour les données sorties).

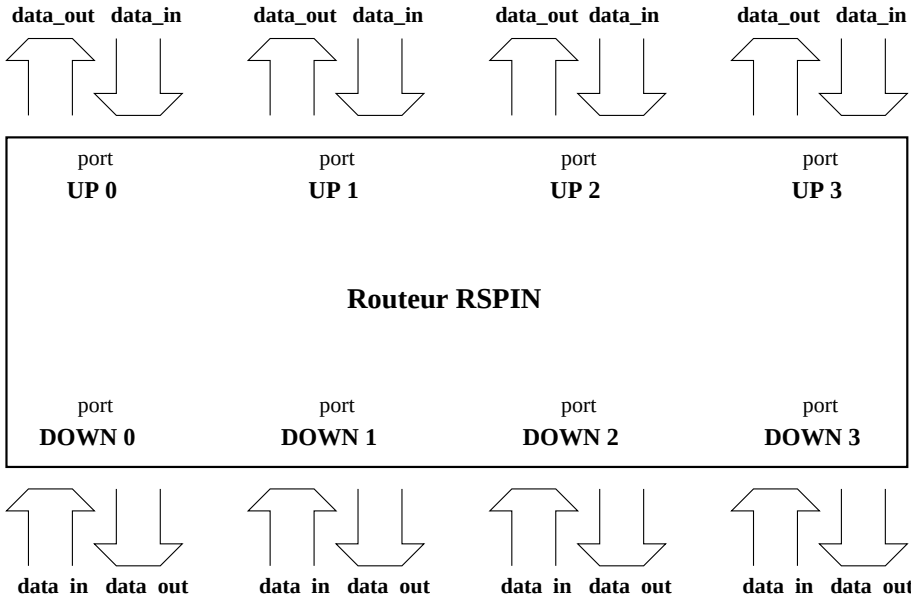

FIG. 3.7 – Architecture externe du routeur RSPIN

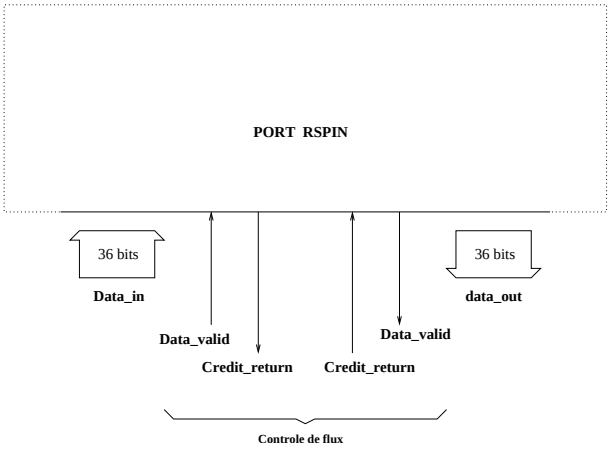

FIG. 3.8 – Description d'un port SPIN

Nous ne nous attardons pas sur la construction des réseaux SPIN, c'est-à-dire sur la façon d'interconnecter les routeurs, car nous développerons ce point au chapitre 8, en tenant compte des contraintes liées à l'utilisation de la norme VCI.

Le tableau 3.1 donne des informations quantitatives sur des réseaux SPIN de différentes tailles. Ces informations concernent le nombre de routeurs, de wrappers, et le nombre de liens bidirectionnels. Un lien se trouve soit entre deux routeurs, soit entre un routeur et un wrapper.

	Nb routeurs	Nb wrappers	Nb total de composants	Nb total de liens
SPIN 4 ports	1	4	5	4
SPIN 8 ports	2	8	10	12
SPIN 16 ports	8	16	24	32
SPIN 32 ports	16	32	48	80
SPIN 64 ports	48	64	112	192
SPIN 128 ports	96	128	224	448
SPIN 256 ports	256	256	512	1024

TAB. 3.1 – Nombre de composants dans un réseau SPIN

3.3.7 Les contrôles de flux de types `FIFO` et `CREDITS`

Nous allons dans cette section comparer les contrôles de flux FIFO et CREDITS afin de bien fixer les idées concernant les choix faits au niveau du réseau SPIN :

. Le contrôle de flux de type FIFO est assez simple. On considère qu'un transfert entre un producteur et un consommateur s'est effectué lorsque, dans le même cycle, le producteur indique qu'il veut écrire et que le consommateur indique que sa fifo n'est pas pleine. Le consommateur informe donc le producteur de l'état de sa fifo : `PLEINE / NON PLEINE`. C'est un contrôle de flux synchrone. Il est utilisé entre un wrapper et un abonné car on s'arrangera toujours pour que ces composants soient proches.

. Dans un contrôle de flux de type CREDITS, le producteur possède une estimation de la capacité de la fifo du consommateur. Il prendra donc la décision d'envoyer ou non une donnée. C'est un contrôle de flux asynchrone, car le producteur et le consommateur peuvent positionner leurs signaux respectifs à des instants différents.

L'utilisation du contrôle de flux de type CREDITS se justifie par la nécessité de pouvoir insérer des barrières de registres car les composants (wrappers et routeurs) ne sont pas toujours très proches les uns des autres. Certains fils, plus longs que les autres, font partie

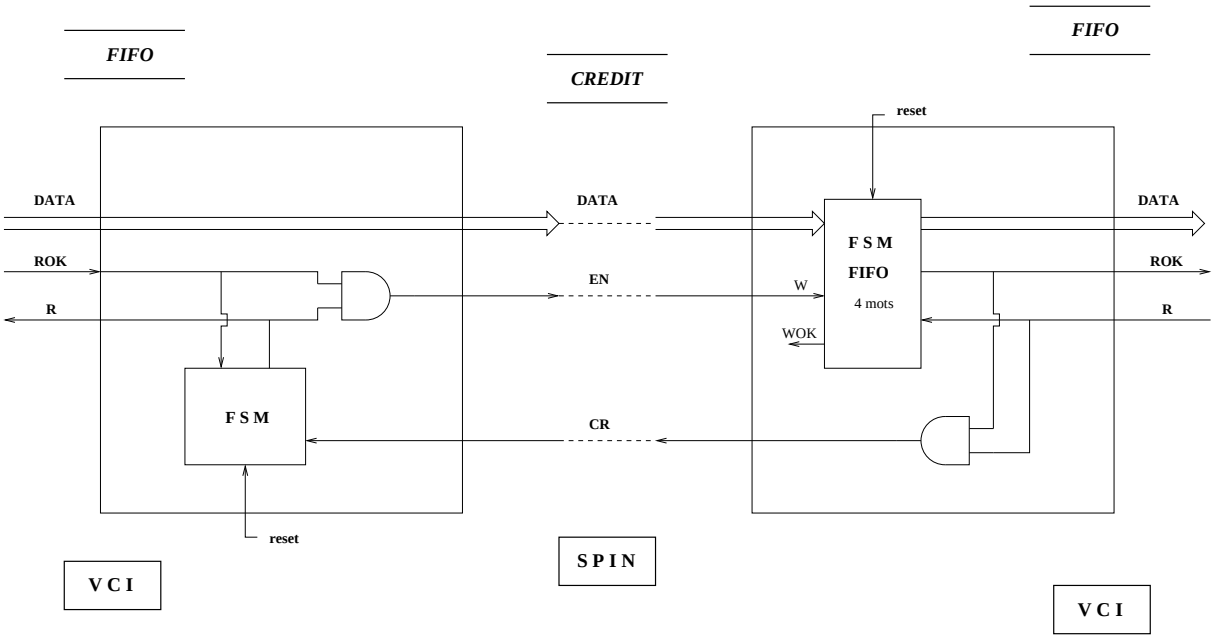

FIG. 3.9 – Le protocole CREDITS

de la chaîne longue. Pour la diminuer, il faut donc mettre des registres sur le chemin critique. Cette insertion de registres est impossible avec un protocole FIFO. En effet, le signal sortant du consommateur et devant indiquer au producteur que la fifo du consommateur est pleine devra franchir la barrière de registres avant de bloquer chez le producteur l'envoi des requêtes. Toutes les informations envoyées durant ce laps de temps seront perdues. Dans le cas du protocole CREDITS, il n'y aura pas de perte d'informations, car c'est le producteur lui-même qui bloquera l'envoi. On aura seulement un délai dans la reprise de l'envoi, égal en nombre de cycles, au nombre de registres.

3.4 La politique de routage dans le cas du trafic VCI

3.4.1 Le routage

Sur le réseau SPIN, les composants initiateurs et les composants cibles sont repérés par un numéro sur 8 bits. Ce numéro leur est attribué lors de la phase d'assemblage des composants du système. Il ne peut donc être modifié par la suite selon les besoins des utilisateurs. Il correspond plus précisément à l'emplacement physique de l'abonné sur le réseau, c'est-à-dire à l'adresse de l'abonné sur le réseau SPIN. C'est ce numéro/adresse qui

est utilisé par les routeurs pour pouvoir acheminer correctement les paquets SPIN/VCI d'un abonné à un autre.

Nous avons vu qu'à chaque abonné est associée une adresse mémoire sur 32 bits. Il faut dont un mécanisme réalisant la correspondance entre une adresse mémoire et une adresse SPIN (8 bits). Avec une structure d'interconnexion de type PI-Bus, ce service est assuré par le BCU. Dans le cas d'un réseau SPIN, ce service est dupliqué dans chaque wrapper initiateur. Donc, chaque wrapper initiateur contiendra le même module faisant la correspondance entre une adresse mémoire et une adresse SPIN. Ce module sera par la suite appelé `table de routage`.

3.4.2 Quelques principes du routage

Puisque toutes les tables de routage des différents wrappers initiateurs sont `identiques`, tout le monde voit le même espace d'adressage. Les tables sont `statiques`, c'est-à-dire qu'elles sont câblées en dur lors de la synthèse des wrappers. Le concepteur système définit une fois pour toutes le nombre de bits de poids forts de l'adresse qui doivent être décodés pour sélectionner une cible.
On découpe l'espace d'adressage en 2^n `zones` de taille fixe (2^{32-n} octets) identifiés par les n bits de poids forts de l'adresse. Un segment est un morceau contigu de l'espace adressable qui n'est pas forcément un nombre entier de zones.
Une cible SPIN peut contenir plusieurs segments (cas des RAM multiples). Chaque cible est identifiée par un numéro codé sur m bits, avec $m <= 8$. Le nombre total d'abonnés dans un système `VCI/SPIN` est donc inférieur ou égal à 2^m. On alloue à chaque cible i `une ou plusieurs` zones, de telle façon que les segments correspondant à la cible i soient complètement contenus dans les zones allouées à i. Une zone ne peut être allouée qu'à une seule cible.
La table de routage est donc fonctionnellement une ROM contenant 2^n mots de m bits. Elle peut être réalisée en logique câblée car m est petit. En effet, les valeurs typiques de n et m sont respectivement 8 et 6.
Le module de routage ne fait aucune vérification concernant les limites des segments. Pour déterminer le numéro de cible on n'aura pas besoin de vérifier que l'adresse est dans le bon intervalle, ce qui impliquerait l'utilisation de comparateurs de supériorité et d'infériorité ; il suffira juste de regarder les bits de poids forts. Puisqu'un segment peut recouvrir plusieurs zones, il faut que le réseau SPIN transmette à la cible les bits de poids forts de l'adresse et pas seulement les $(32 - n)$ bits de poids faibles, soit la totalité des bits d'adresse.

3.4.3 Le module de routage

Lorsqu'un abonné de type initiateur envoie une requête sur le réseau, il faut déterminer l'abonné SPIN, cible de cette transaction. Toutes les adresses se trouvant à l'intérieur d'un même paquet sont à destination de la même cible. Donc, pour localiser la cible de la transaction, il suffit d'analyser la première adresse du paquet et, plus précisément, les n bits de poids forts de cette adresse. C'est ce qui est fait lors de la génération du mot d'en-tête, à travers le `module de routage`[1]. Ce module convertit les n bits de poids forts de l'adresse en numéro de cible SPIN sur m bits. Ce numéro est ensuite inséré dans les bits de poids faibles de l'en-tête, dans le champ `destination`.
Le module de routage est répliqué dans tous les wrappers initiateurs, car le wrapper cible ne renvoie l'information qu'à l'auteur de la requête.

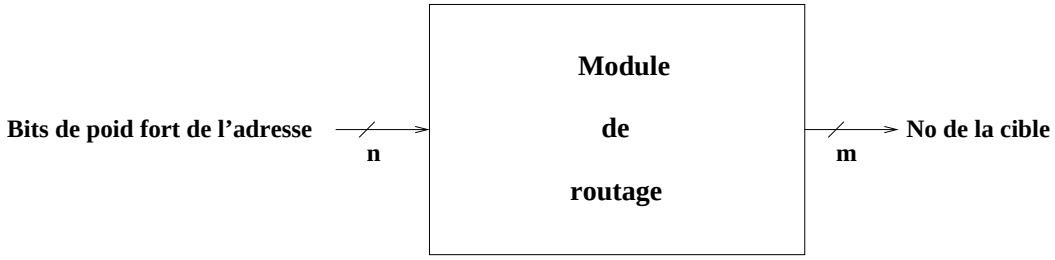

FIG. 3.10 – Interface du module de routage

3.4.4 Le comportement en cas d'erreur de routage

Lorsque le module de routage n'arrive pas à trouver un destinataire valide, c'est-à-dire que les bits de poids forts de l'adresse ne permettent pas désigner un abonné, il se produit une `erreur de routage`.
On peut imaginer différentes attitudes face à cette erreur :

- Il est possible de la signaler directement à l'initiateur en activant un signal d'erreur, car ce signal ne fait pas partie de la norme VCI.
- Il est en principe possible pour le wrapper initiateur de renvoyer lui-même un paquet réponse constitué d'une série d'acquittements négatifs. De cette façon, le réseau n'est pas encombré par un paquet erroné. En revanche, la complexité du wrapper initiateur augmente.
- enfin on a la possibilité de rejeter le problème à la périphérie, en chargeant la cible par défaut de générer l'erreur à travers le champ erreur de la nappe réponse VCI. En procédant de cette façon, on a l'avantage de ne pas augmenter la complexité

[1] CF : Figure 3.10 : Interface du module de routage

du wrapper initiateur et d'être indépendant de la structure d'interconnexion. Cette dernière solution sera explicitée dans le paragraphe suivant.

Nous avons voulu concevoir un mécanisme de gestion des erreurs de routage qui soit le plus général possible et donc indépendant de la structure d'interconnexion utilisée. La détection de l'erreur de routage et sa signalisation ont donc été rejetées à la périphérie du réseau, sur une cible dite `cible par défaut`. Cela signifie que la table de routage est "complète" : toutes les adresses correspondant à une zone "non définie" de l'espace adressable sont routées vers la "cible par défaut". Cette cible reçoit les paquets valides qui lui sont destinés, mais aussi les paquets issus d'erreurs de routage (destination non valide).

Afin de pouvoir faire la différence entre les paquets valides qui lui sont destinés et ceux qui ne le sont pas, cette cible devra analyser les bits de poids forts des adresses contenues dans la requête. D'une manière générale, une cible n'a pas vraiment besoin de regarder les bits de poids forts. On s'arrangera toutefois pour que dans tout réseau il y en ait au moins une qui le fasse. C'est une contrainte acceptable, dans la mesure où elle simplifie la structure des wrappers et fournit un mécanisme général de gestion des erreurs de routage. L'erreur ainsi signalée remonte jusqu'à l'initiateur à qui il incombe désormais de la traiter.

3.5 Conclusion

Nous avons présenté les concepts qui serviront de briques de base pour l'élaboration de notre étude. Les descriptions faites des architectures du réseau SPIN et du PI-Bus mettent en relief les avantages et les inconvénients de l'un et de l'autre. Le tableau 3.1 indiquant le nombre de composants d'un réseau SPIN en fonction du nombre de ports nous fait pressentir la principale difficulté liée à la mise au point des réseaux SPIN : leur complexité architecturale et leur taille.

Nous entrerons dans le détail des différentes réalisations dans les chapitres suivants.

Chapitre 4

Règles de transformation VCI/SPIN

Sommaire

Ce chapitre est destiné à préciser la nature des entités et des informations circulant sur le réseau SPIN. On appellera paquet SPIN/VCI un paquet VCI encapsulé dans un paquet SPIN. L'obtention des paquets VCI à partir des paquets SPIN/VCI et l'opération inverse seront explicitées, tout comme la structure des paquets SPIN/VCI requêtes et réponses.

4.1 L'en-tête du paquet SPIN/VCI

L'en-tête, également appelé `header`, est un mot de 36 bits se situant au début d'un paquet requête ou réponse. Il contient d'une part des informations exploitées par les routeurs du réseau SPIN, d'autre part des informations destinées aux wrappers VCI/SPIN. Le header est identifié dans un flot SPIN par la valeur à '1' du bit BP du fanion. Ce dernier a été décrit à la section 3.3 à propos du réseau SPIN.

Les informations à destination des routeurs sont utiles au bon acheminement du paquet à travers le réseau. Elles sont codées dans les 11 bits de poids faibles et les 3 bits de poids forts. Les figures 4.1 et 4.2 représentent la composition des headers pour des paquets SPIN/VCI requête et réponse. Il y a notamment dans le header le numéro du port destination du paquet. Ce numéro, codé sur les 8 bits de poids faibles, est utilisé pour l'aiguillage du paquet (cf section 3.3.5).

Les informations à destination des wrappers sont utiles au décodage du paquet. Elles sont codées entre les bits numéro 11 et 35 (cf figures 4.1 et 4.2). Contrairement aux informations à destination des routeurs, elles diffèrent selon le type de header (requête ou réponse). Vu que les wrappers initiateur et cible ont des comportements différents, les informations dont ils ont besoin sont également différentes. Les informations que nous retrouvons à la fois dans le header-requête et dans le header-réponse sont les bits codant les identifiants de tâche et de paquet. Ils sont utilisés à la fois par le wrapper initiateur et par l'initiateur lui-même. Le wrapper en a besoin afin d'éviter l'émission sur le réseau de deux requêtes ayant les mêmes identifiants. Quant à l'initiateur, il les utilise pour identifier la requête à laquelle la réponse est destinée.

Notons que routeurs et wrappers utilisent tous, les 3 bits de poids forts constituant, avec le bit de parité, le fanion. Les informations de début et de fin de paquet servent aux routeurs à ouvrir ou libérer des chemins pour le routage. Aux wrappers, ils permettent entre autre d'identifier les headers pour positionner les champs VCI.

4.1.1 Contenu de l'en-tête du paquet SPIN/VCI requête

Le header-requête est construit dans le wrapper initiateur. Il encapsule une grande partie des informations présentes sur l'interface VCI requête de l'initiateur. Nous présentons ici les différents champs le constituant. Cette description est à mettre en parallèle avec le schéma de la figure 4.1.

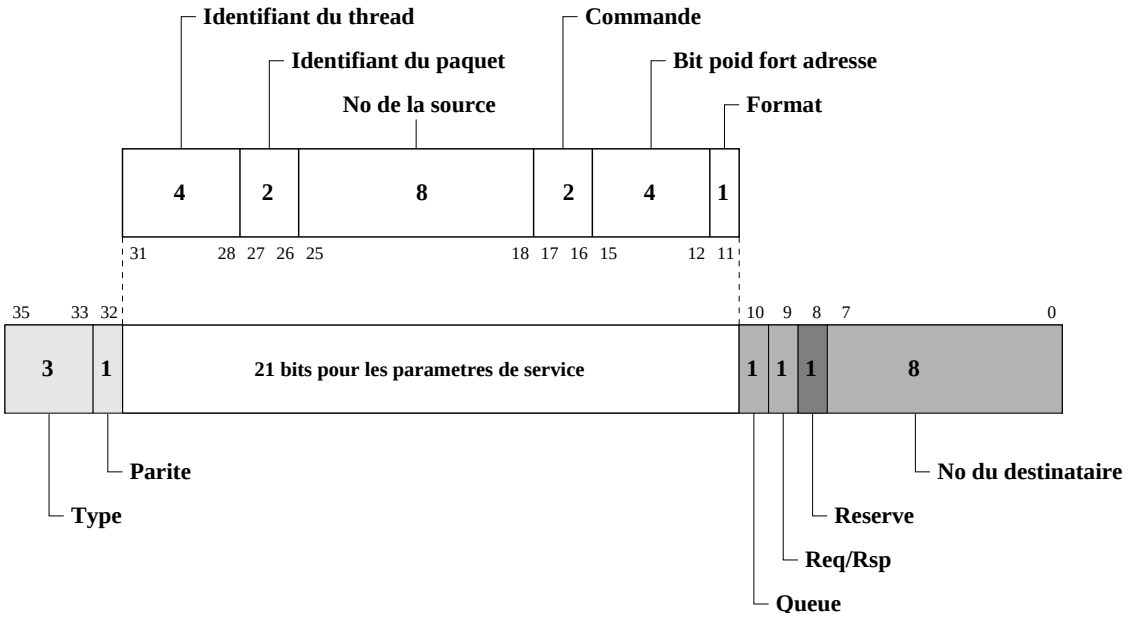

FIG. 4.1 – Composition de l'en-tête d'un paquet SPIN/VCI requête

Le header-requête se compose de :

- 11 bits à destination des routeurs :
  . 8 bits pour l'adresse de la destination ;
  . 1 bit réservé ;
  . 1 bit "reqrsp" indiquant s'il s'agit d'un paquet-requête ('0') ou d'un paquet-réponse ('1') ;
  . 1 bit "queue" indiquant si le paquet peut être rangé ou non dans les queues centrales (FIFO internes d'environ 18 mots), de façon à libérer plus rapidement le port d'entrée et ainsi optimiser le routage ;
- 21 bits à destination des wrappers, constituant les paramètres de service :
  . 1 bit pour le format du header (simple/double). Actuellement nous n'utilisons que des headers simples. L'utilisation de headers doubles (constitués de 2 mots) sera peut-être nécessaire par la suite, afin de pouvoir transmettre des informations

supplémentaires ;
. 4 bits contenant les 4 bits de poids forts de l'adresse. Vu qu'un paquet a pour destination une unique cible, toutes les adresses concernent le même segment mémoire. Elles ont donc des bits de poids forts identiques. On peut par conséquent se permettre de ne transmettre qu'une seule fois les 4 bits de poids forts dans le header, de façon à n'avoir plus que 28 bits à ranger dans les bits de poids faibles du mot d'adresse. Dans les 4 bits de poids forts de ce mot, on mettra le "byte enable" codant en "one hot", pour chaque adresse, les octets cibles de la transaction ;
. 2 bits pour la commande (read ("01"), write ("10"), read lock ("11"), nop ("00")) ;
. 8 bits pour le numéro de la source. Ce numéro (bits n°18 à 25) sera extrait par le wrapper cible et inséré dans le champ destination du header-réponse. Les routeurs utiliseront ensuite cette information pour router la réponse vers l'initiateur source de la transaction ;
. 2 bits pour l'identifiant du paquet. Les identifiants de tâche et de paquet (bits n°26 à 31) sont vus globalement comme un identifiant unique sur 6 bits. Ils sont repositionnés à l'identique dans le header-réponse, afin que l'initiateur puisse différencier les réponses qui lui arrivent ;
. 4 bits pour l'identifiant de la tâche ;

– 4 bits constituant le `fanion` répartis de la façon suivante :
. 1 bit de parité,
. 3 bits indiquant le type du mot dans le paquet :
NOP : pas de type particulier,
BP : début du paquet ("Begin Packet"), type de l'en-tête,
EP : fin du paquet ("End of Packet"), type du dernier mot de la charge utile,
ERR : erreur sur le mot du paquet.

4.1.2 Contenu de l'en-tête du paquet SPIN/VCI réponse

Le header-réponse est construit dans le wrapper cible. Il encapsule une grande partie des informations présentes sur l'interface VCI réponse de la cible. Nous présentons ici les différents champs le constituant. Cette description est à mettre en parallèle avec le schéma de la figure 4.2.

Le header-réponse se compose de :
– 11 bits à destination des routeurs :
. 8 bits pour l'adresse de la destination ;
. 1 bit réservé ;
. 1 bit reqrsp indiquant s'il s'agit d'un paquet-requête ou d'un paquet-réponse ;

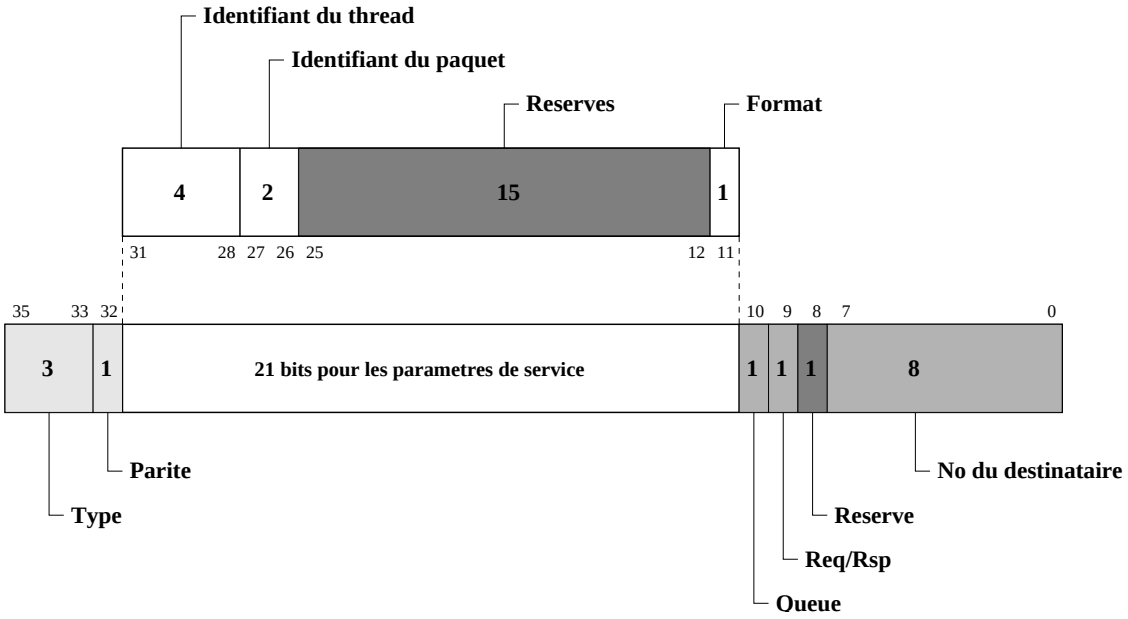

FIG. 4.2 – Composition de l'en-tête d'un paquet SPIN/VCI réponse

. 1 bit "queue" indiquant si le paquet peut être rangé ou non dans les queues centrales (FIFO internes d'environ 18 mots) de façon à libérer plus rapidement le port d'entrée et ainsi optimiser le routage ;

– 21 bits à destination des wrappers, constituant les paramètres de service :
  . 15 bits réservés ;
  . 2 bits pour l'identifiant du paquet ;
  . 4 bits pour l'identifiant de la tâche ;
– 4 bits constituant le `fanion`, répartis de la façon suivante :
  . 1 bit de parité ;
  . 3 bits indiquant le type du mot dans le paquet :
    NOP : pas de type particulier ;
    BP : début du paquet ("Begin Packet"), type de l'en-tête ;
    EP : fin du paquet ("End of Packet"), type du dernier mot de la charge utile ;
    ERR : erreur sur le mot du paquet.

4.2 La charge utile

La charge utile, également appelée `payload`, est la partie du paquet SPIN regroupant tous les mots se trouvant derrière le header. Ce sont également des mots de 36 bits, dont le nombre n'est pas limité. La taille de la charge utile peut aller d'un mot à plus d'une

centaine. Le dernier mot du paquet est identifié par la valeur à '1' du bit EP ("End of Packet") du fanion. Implicitement, le dernier mot de la charge utile est aussi le dernier mot du paquet.
La charge utile contient les informations à transmettre (données ou adresses), le header indiquant comment les utiliser. La nature de la charge utile varie selon le type de requête. Si c'est une requête de lecture, la charge utile ne contiendra que des adresses. Si c'est une requête d'écriture, elle contiendra, de manière alternée, adresses et données.
Le schéma de la figure 4.3 présente la structure d'un mot de la charge utile contenant une adresse, et celle d'un autre contenant une donnée.

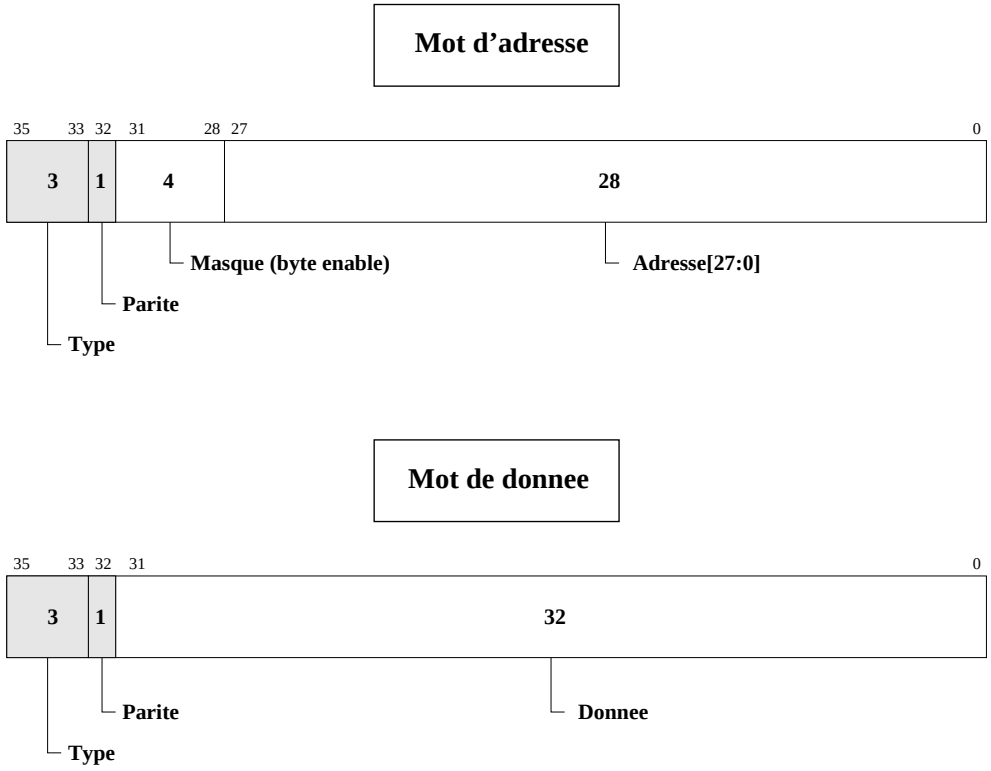

FIG. 4.3 – Composition des mots de la charge utile d'un paquet SPIN

Un *mot de donnée* est constitué des 4 bits de fanion et des 32 bits de la donnée à transmettre. Le *mot d'adresse* est un peu plus compliqué : aux 4 bits de fanions s'ajoutent 4 bits de masque et 28 bits d'adresse. En effet, à chaque adresse VCI est associé un masque ou "byte enable". Le byte enable désigne le ou les octet(s) cible(s) de la transaction. Il y a 1 bit par octet (codage "one hot"). Ses valeurs sont "1111" pour la lecture ou l'écriture d'un mot, "0011" ou "1100" pour l'écriture d'un demi-mot et "0001", "0010", "0100" ou

"1000" pour l'écriture d'un octet.
Pour la transmission du "byte enable" sur l'interface SPIN, nous avions le choix : soit l'envoyer en même temps que le mot d'adresse, soit immédiatement après l'adresse. L'insérer dans un mot supplémentaire à la suite du mot d'adresse reviendrait trop cher au niveau de la longueur des paquets. Nous avons donc fait le choix de le transmettre dans les bits de poids forts du mot d'adresse, les 4 bits de poids forts de l'adresse étant quant à eux transmis dans le mot d'en-tête du paquet.

4.3 La formation des paquets

Nous avons des abonnés VCI qui doivent communiquer par le biais du réseau SPIN. Cela nous donne quatre transformations générales :

- le passage du paquet-requête VCI au paquet-requête SPIN (dans le wrapper initiateur) ;
- le passage du paquet-requête SPIN au paquet-requête VCI (dans le wrapper cible) ;
- le passage du paquet-réponse VCI au paquet-réponse SPIN (dans le wrapper cible) ;
- le passage du paquet-réponse SPIN au paquet-réponse VCI (dans le wrapper initiateur).

Nous devons aussi prendre en compte le type du paquet : lecture ou écriture. Car dans le cas d'une écriture, en plus de l'adresse, il faudra fournir la donnée à écrire. Pour les 4 transformations ci-dessus, nous devons réaliser en tout huit transformations. Ces transformations ne sont pas toutes différentes. Par exemple, dans le format des paquets SPIN, les paquet-requêtes de lecture et les paquet-réponses de lecture et d'écriture sont semblables. Au final, seulement quatre transformations seront détaillées ci-après. Nous présenterons dans les sections qui suivent le schéma global de chaque transformation, sans entrer dans les détails de ce qui se passe au niveau bit.

4.3.1 Du paquet VCI au paquet SPIN/VCI

La transformation d'un paquet VCI en un paquet SPIN intervient dans le wrapper initiateur au niveau des requêtes et dans le wrapper cible au niveau des réponses.

Cas des paquets requêtes de lecture et des paquets réponses

Dans le cas d'une requête VCI de lecture, il faut un mot SPIN pour le header-requête et ensuite un mot par adresse. Donc un paquet de n mots VCI est transcrit en un paquet de $n + 1$ mots SPIN (Cf Figure 4.4).

Pour les réponses, il faut également un mot SPIN pour le header et un mot par donnée renvoyée (cas des lectures) ou par acquittement (cas des écritures). Donc, pour les réponses, un paquet de n mots VCI est également transcrit en un paquet de $n+1$ mots SPIN.

Cas des écritures

Pour écrire un mot en mémoire, il faut la donnée à écrire ainsi que l'adresse de l'emplacement en mémoire. Ces deux informations étant présentes dans le même mot VCI, il faut deux mots SPIN pour la traduire (Cf Figure 4.5). Par conséquent, n mots VCI seront transcrits, en $2n$ mots SPIN. Avec le header-requête on obtiendra donc un paquet de $2n+1$ mots SPIN.

4.3.2 Du paquet SPIN/VCI au paquet VCI

La transformation d'un paquet SPIN en un paquet VCI intervient dans le wrapper initiateur au niveau des réponses et dans le wrapper cible au niveau des requêtes.

Cas des lectures et des paquets réponses

Dans le cas des requêtes de lecture ou des paquets réponses, un paquet SPIN de $n+1$ mots est transcrit en paquet VCI de n mots (Cf Figure 4.6).

Cas des écritures

Dans le cas des écritures, un paquet SPIN de $2n+1$ mots génère un paquet VCI de n mots (Cf Figure 4.7).

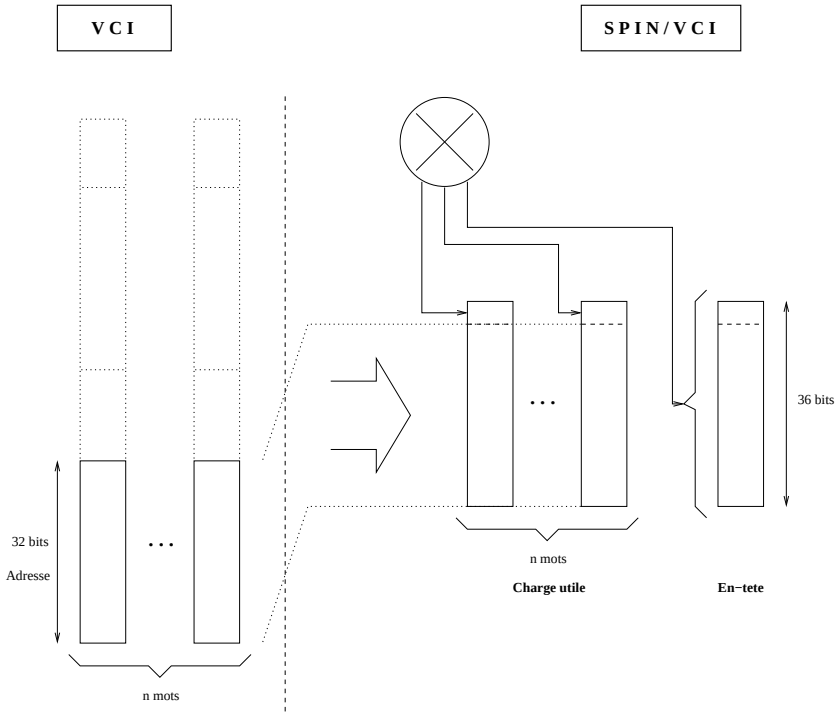

FIG. 4.4 – Du paquet VCI au paquet SPIN (cas des lectures et des paquets réponses)

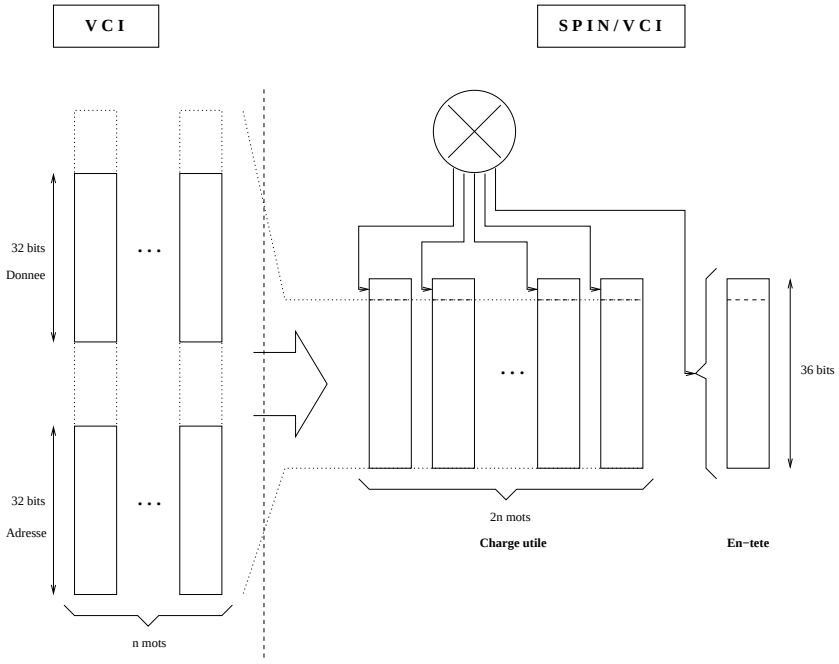

FIG. 4.5 – Du paquet VCI au paquet SPIN (cas des écritures)

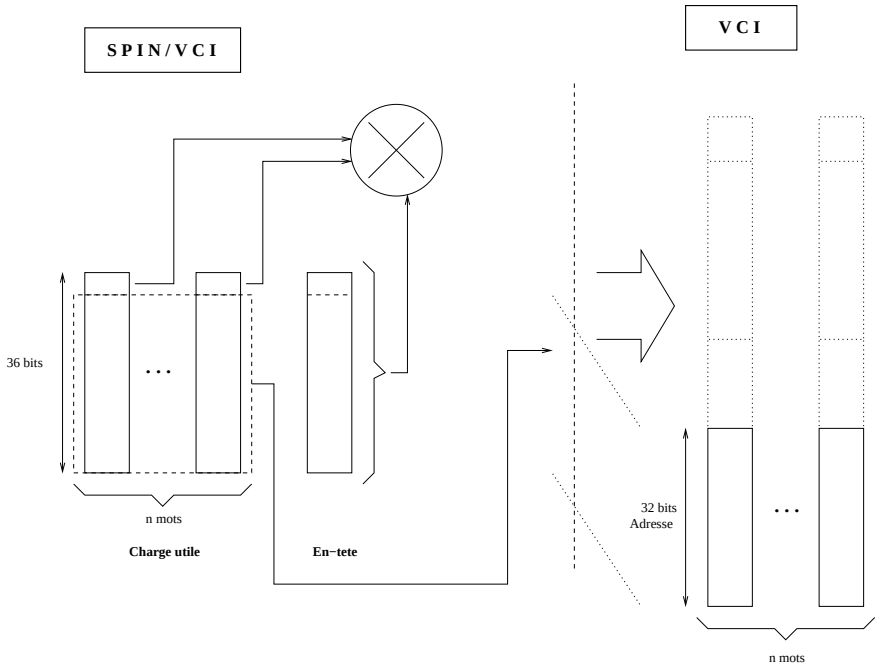

FIG. 4.6 – Du paquet SPIN au paquet VCI (cas des lectures et des paquets réponses)

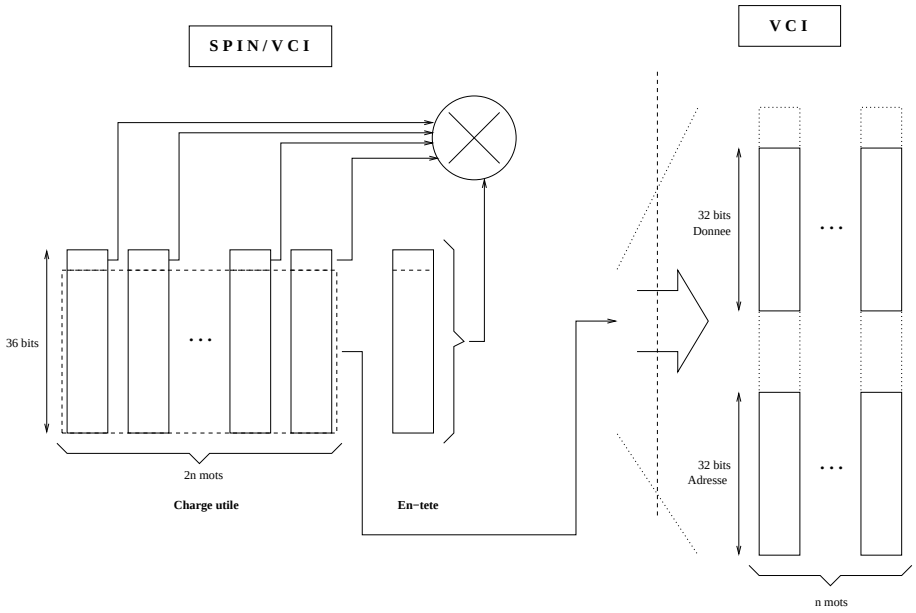

FIG. 4.7 – Du paquet SPIN au paquet VCI (cas des écritures)

4.4 Optimisations

Lorsque l'on passe d'un format de paquet à un autre nous gardons en tête deux grands objectifs : il faut que cela aille vite (un minimum de cycles) et que l'on perde le moins d'informations possible. Les wrappers transmettent les informations minimales à partir desquelles la cible pourra traiter la requête. Certains signaux VCI dits d'optimisation, ont été volontairement omis pour minimiser la longueur du paquet SPIN/VCI :

- `contig` : indique que les adresses du paquet sont contiguës ;
- `cons` : indique que les adresses du paquet sont constantes ;
- `wrap` : indique que les adresses sont wrappées ;
- `plen` : indique la taille en octet du paquet VCI au niveau données ;
- `cfixed` : indique que les champs d'OPCODE et PLEN seront constants durant tout l'état haut du signal ;
- `clen` : indique le nombre de paquets encore contenus dans la chaîne.

Pour une plus grande précision sur la signification des signaux, on peut se référer à la section 3.1 sur la norme VCI.
Pour l'instant, ces signaux, même s'ils sont présents sur l'interface, ne sont pas pris en compte par les wrappers. Etant facultatifs, ils n'influent en rien sur la nature du trafic VCI. En revanche, leur impact sur le trafic SPIN peut être intéressant.
Les wrappers VCI/SPIN initiateur et cible ne prennent pas en compte ces signaux. L'objectif de départ étant de mettre au point des composants simples permettant de tester rapidement les grands concepts généraux de VCI, à savoir sa capacité à encapsuler des informations de natures diverses ainsi qu'à gérer les transactions éclatées.
Prendre en compte ces signaux permettrait de répondre à une triple exigence :

- réduire le trafic sur le réseau SPIN/VCI et de ce fait les risques de congestion,
- optimiser la consommation,
- enfin ne pas pénaliser une cible qui prendrait en compte ces signaux, l'empêchant ainsi d'optimiser ses traitements.

Il est en effet possible de réduire le nombre de mots transmis sur le réseau. Par exemple, le cache d'instruction qui doit rapatrier une ligne de cache de 16 mots, n'est pas obligé d'envoyer un paquet-requête contenant les 16 adresses consécutives. Il lui suffit simplement d'envoyer un paquet-requête contenant la première adresse et d'indiquer, qu'à partir de celle-ci, quinze autres doivent être générées consécutivement (soit 2 mots plus un en-tête).

4.5 Conclusion

Nous avons donc présenté le contenu des paquets SPIN et expliqué comment passer d'un paquet VCI à un paquet SPIN et inversement, dans les cas des écritures et des lectures, pour les requêtes comme pour les réponses. Les changements de taille sont récapitulés ci-dessous :

Cas des écritures :

- Requêtes : n mots VCI $\longleftrightarrow$ 2n+1 mots SPIN
- Réponses : n mot VCI $\longleftrightarrow$ n + 1 mot SPIN

Cas des lectures :

- Requêtes : n mots VCI $\longleftrightarrow$ n+1 mots SPIN
- Réponses : n mots VCI $\longleftrightarrow$ n+1 mots SPIN

Chapitre 5

Architecture des wrappers VCI/SPIN

Sommaire

Nous allons dans ce chapitre spécifier les architectures externe et interne des wrappers initiateur et cible, puis nous traiterons de leur modélisation VHDL ainsi que des phases de synthèse, de placement et de routage.

5.1 Spécification des wrappers VCI/SPIN

5.1.1 Les fonctions du wrapper initiateur

Le wrapper initiateur permet de transformer un paquet requête VCI en un paquet requête SPIN/VCI et un paquet réponse SPIN/VCI en un paquet réponse VCI. Il assure le passage du contrôle de flux de type FIFO (sur l'interface VCI), au contrôle de flux de type CREDITS (sur l'interface SPIN) et inversement (Cf section 3.3.7 pour plus d'informations sur le contrôle de flux de type CREDITS). Il empêche aussi que deux requêtes ayant le même numéro d'identification ne soient présentes simultanément sur le réseau SPIN. Enfin, il analyse les bits de poids forts de l'adresse VCI et, à partir d'une table de routage (petite ROM de transcodage), il génère un numéro de port SPIN qu'il place dans les 8 bits de poids faibles du mot d'en-tête (header requête).

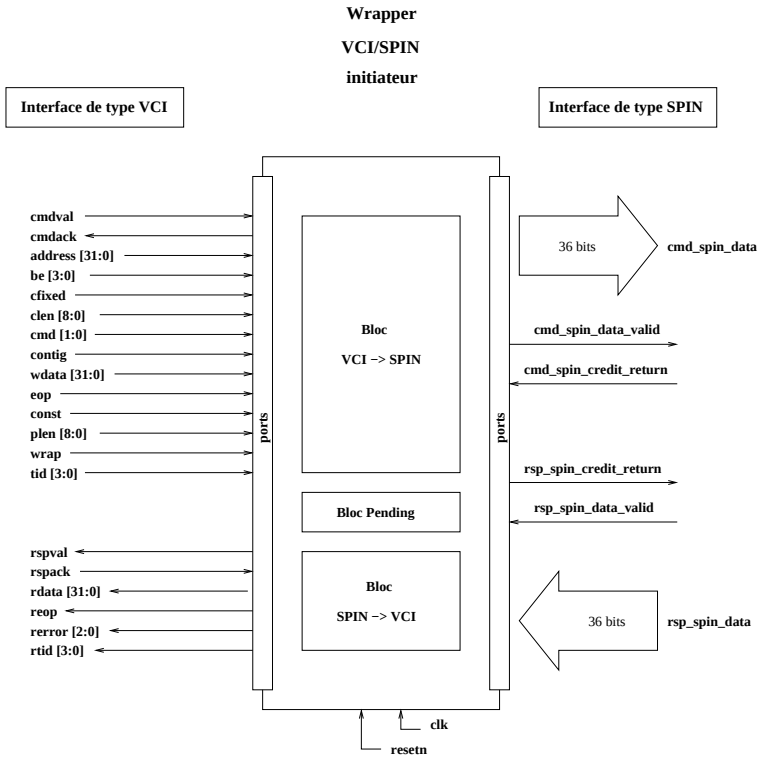

FIG. 5.1 – Interfaces du wrapper SPIN/VCI associé à l'initiateur

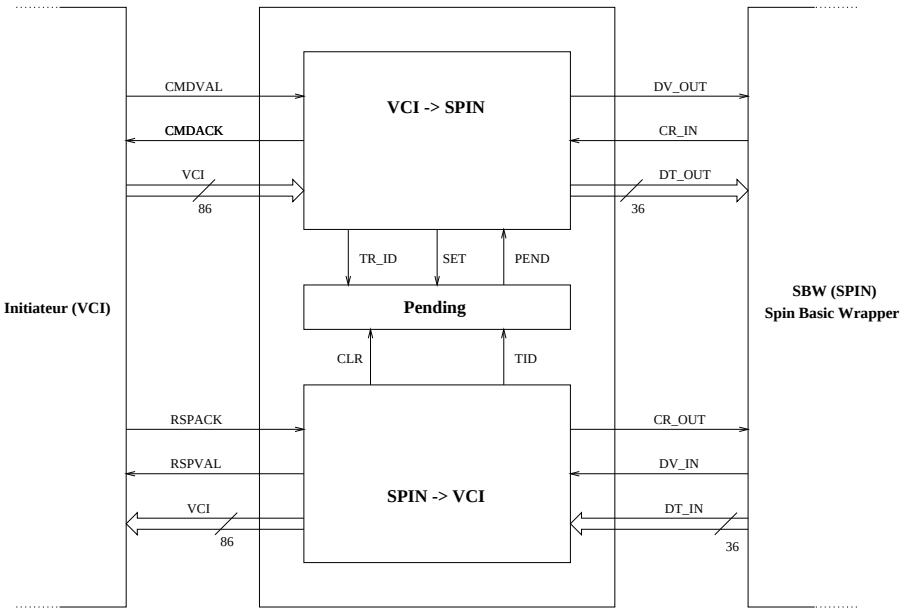

FIG. 5.2 – Architecture interne du wrapper VCI/SPIN initiateur

5.1.2 Architecture du wrapper initiateur

Le wrapper initiateur dispose d'une interface VCI "avancée" et d'une interface SPIN [1]. Le choix de l'interface VCI "avancée" a été fait pour permettre au wrapper de s'interfacer avec n'importe quel type de composant VCI. En effet, c'est la plus complète des trois interfaces VCI qui existe (cf section 3.1.1).
Nous avons vu à la section 4.4 que certains signaux dits d'optimisation ne sont pas encapsulés dans les paquet SPIN/VCI. Ces signaux sont tout de même présents sur l'interface VCI du wrapper. En effet des évolutions futures pourraient favoriser leur utilisation.

De plus, le wrapper initiateur se décompose en trois blocs : le bloc "VCI->SPIN" gère la transformation d'un paquet VCI requête en un paquet SPIN/VCI requête ; le bloc "SPIN->VCI" gère la transformation d'un paquet SPIN/VCI réponse en un paquet VCI réponse. Enfin, le bloc "Pending" gère les 16 bascules pending. Chaque bascule est associée à un numéro d'identification de paquet. Les valeurs '1' et '0' des bascules permettent respectivement, de savoir si un numéro d'identification est déjà utilisé ou pas.

Les figures 5.2 et 5.3 montrent le découpage en blocs du wrapper. Chaque bloc étant lui même divisé en sous-blocs, un sous-bloc pouvant être un automate ou de la logique

[1] Cf Figure 5.1 : Interfaces du wrapper SPIN/VCI associé à l'initiateur

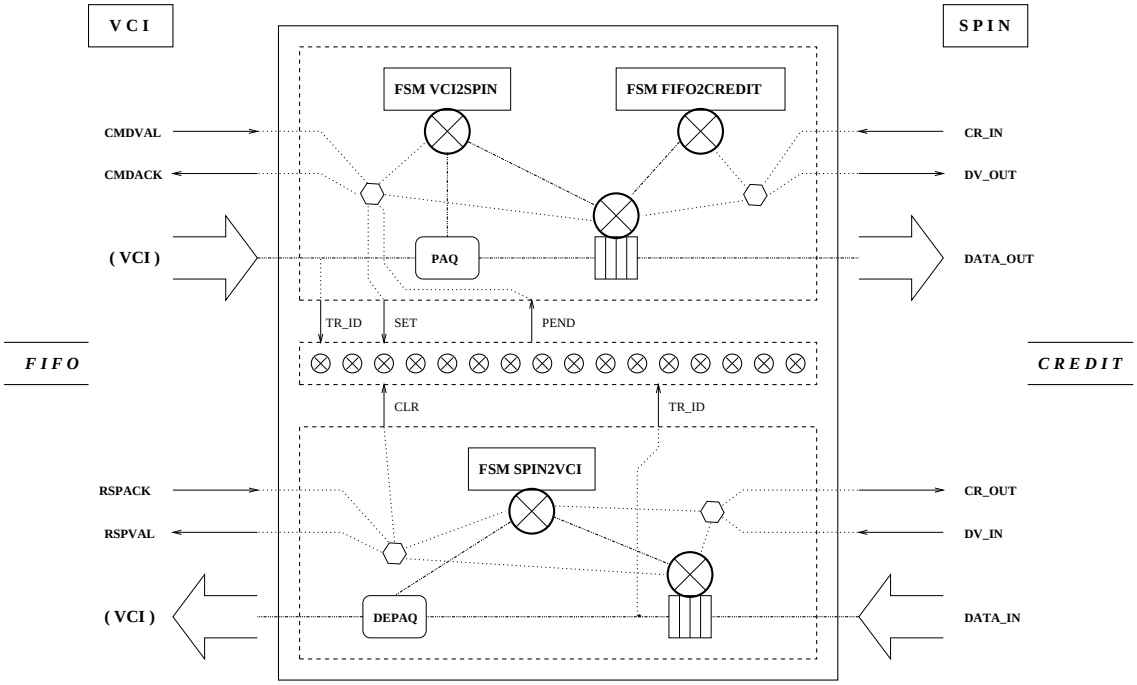

FIG. 5.3 – Architecture des automates du wrapper VCI/SPIN initiateur

combinatoire. Sur le schéma de l'architecture interne représentée à la Figure 5.3, les cercles en gras représentent des automates, les hexagones et les rectangles ("PAQ" et "DEPAQ") de la logique combinatoire. Le bloc de traitement des requêtes et celui de traitement des réponses sont synchrones au niveau temporel (ils utilisent la même horloge), mais asynchrones au niveau logique, puisque les traitements des requêtes et des réponses sont indépendants entre eux.

Remarquons bien que le fait d'utiliser des bascules pour gérer les requêtes pendantes est probablement une précaution non indispensable, car il est de la responsabilité de l'initiateur de ne pas envoyer simultanément deux requêtes ayant le même identifiant.
En effet, la norme VCI ne fait aucune hypothèse sur la nature de la structure d'interconnexion et la façon dont elle achemine les paquets. Certaines structures d'interconnexion garantissent l'ordre de transmission et de réception des paquets. C'est par exemple une garantie intrinsèque au fonctionnent du PI-Bus qui ne permet qu'une seule communication à la fois. Le réseau SPIN en revanche ne permet pas de garantir l'ordre de transmission des paquets, car il permet des communications simultanées sur plusieurs chemins pour aller d'un abonné à un autre.

5.1.3 Les fonctions du wrapper cible

Le wrapper cible permet de transformer un paquet VCI réponse en un paquet SPIN/VCI réponse et un paquet SPIN/VCI requête en un paquet VCI requête. Il assure aussi le passage du contrôle de flux de type FIFO au contrôle de flux de type CREDITS et inversement.
Le wrapper cible "séquentialise" les paquets VCI requête, en ne transmettant le paquet requête (n+1) qu'après avoir reçu complètement le paquet réponse (n). Le mécanisme mis en jeu à ce niveau est similaire à celui des bascules pending du wrapper initiateur. Dans le cas du wrapper cible, il n'y a qu'une seule bascule qui soit mise à '1' lorsque le premier mot du paquet VCI est transmis à la cible. La bascule est remise à '0' lorsque le wrapper reçoit le dernier mot du paquet réponse. Nous avons mis en place ce mécanisme car nous ne sommes pas sûrs que toutes le cibles puissent traiter plusieurs requêtes simultanément.

Dès la réception du paquet requête, le wrapper mémorise le header SPIN/VCI. De ce header, deux informations seront réutilisées : le numéro de la source et le numéro d'identification de la requête. Ces informations seront réinsérées dans le header du paquet réponse, la première dans le champ destination et la seconde dans le champ de l'identifiant. Ce mécanisme de précaution est utile dans le cas où la cible ne gère pas les identifiants. En effet, les identifiants ne sont présents que sur l'interface VCI "avancée", et beaucoup de cibles ont une interface VCI "basique". Il est ainsi possible d'accepter toutes sortes de cible.

5.1.4 Architecture du wrapper cible

Les interfaces du wrapper cible sont symétriques par rapport à celles du wrapper initiateur. Donc l'interface VCI de type "avancée" comporte également les signaux d'optimisation, même s'ils sont mis arbitrairement à '0'.

5.1.5 Remarque sur les wrappers

Un wrapper VCI/SPIN a une architecture modulaire, l'objectif étant de pouvoir réutiliser la même architecture avec des protocoles différents. L'entité essentielle dans les wrappers est la FIFO. Lorsqu'elle est asynchrone, elle autorise l'utilisation de plusieurs domaines d'horloges. Ainsi l'abonné peut avoir une fréquence de fonctionnement différente de celle du réseau.
Le changement de protocole de communication en entrée ou en sortie du wrapper n'entraîne la modification que d'une partie de celui-ci. En effet, prenons le cas du wrapper

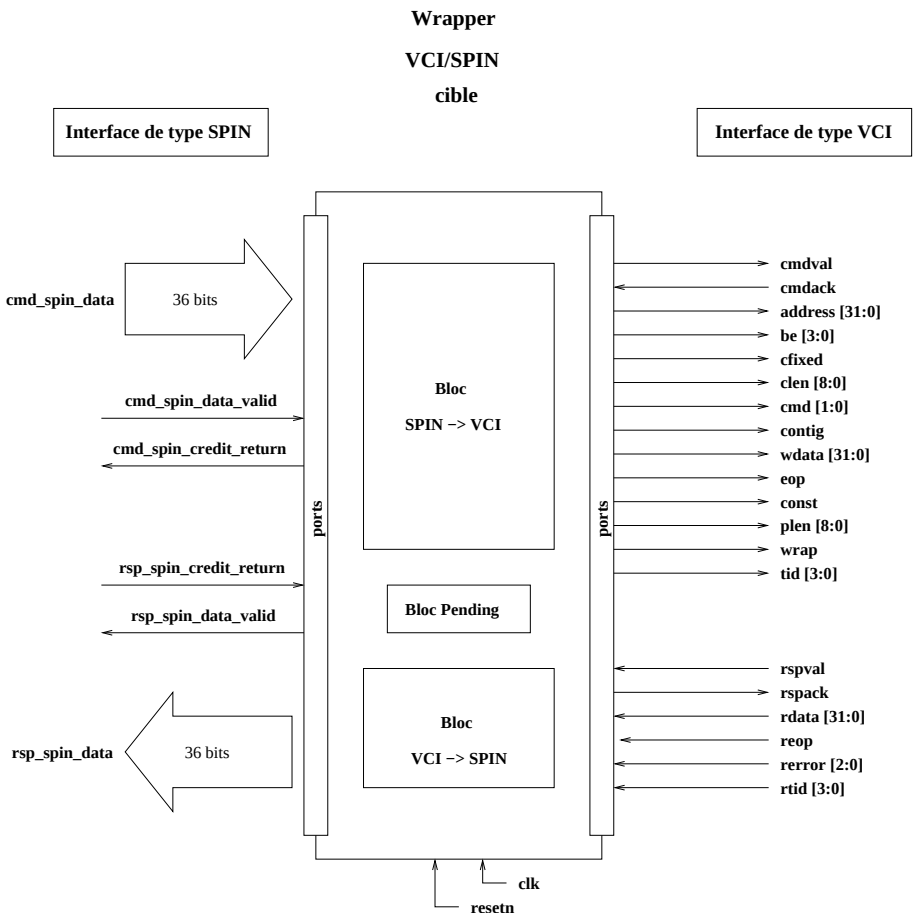

FIG. 5.4 – Interfaces du wrapper SPIN/VCI associé à la cible

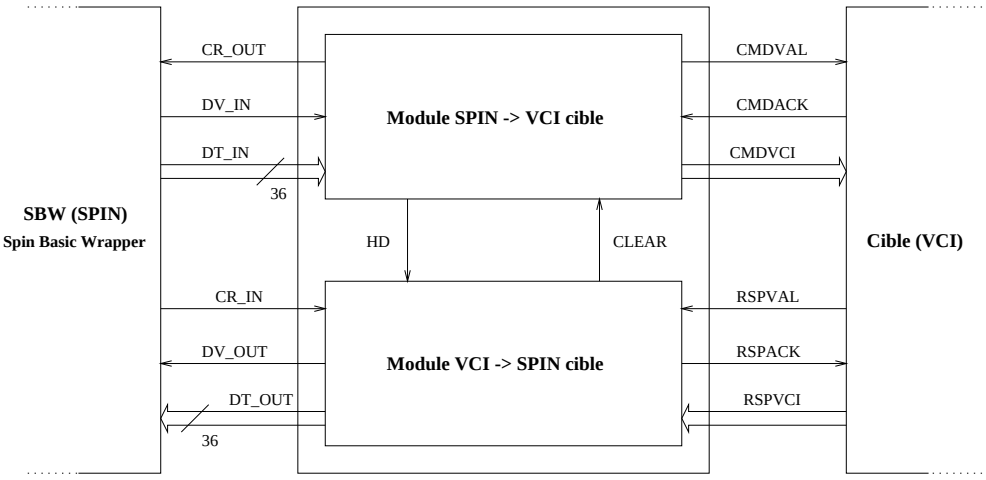

FIG. 5.5 – Architecture interne du wrapper VCI/SPIN cible

VCI/SPIN initiateur. Si on désire remplacer le protocole VCI par le protocole OCP, dans le cas des requêtes (resp. des réponses), il suffit de ne changer que ce qui est en amont (resp. en aval) de la FIFO.
De même si on veut utiliser VCI avec une autre structure d'interconnexion seule la partie du wrapper se trouvant entre la FIFO et l'interface SPIN est à changer.

5.1.6 Le test eulérien

Le test des liens du réseau SPIN, encore appelé test eulérien, est un test après fabrication dont la finalité est de valider les liens externes, c'est-à-dire les liens $routeur - routeur$ et $routeur - wrapper$. Les liens à tester sont représentés en gras sur la Figure 5.6 dans le cas d'un réseau à 16 ports.
L'idée est d'envoyer des paquets par le wrapper connecté au port 0 du réseau, de les faire circuler à travers tous les liens jusqu'au dernier wrapper (celui connecté au dernier port) qui les renverra au wrapper 0. La réception des paquets par le wrapper 0 est une condition nécessaire et suffisante pour la validation des liens. Les wrappers, selon leur emplacement et la nature du réseau, se contenteront de renvoyer le paquet soit en laissant l'adresse de la cible inchangée, soit en l'incrémentant d'une unité (Cf Figure 5.7).

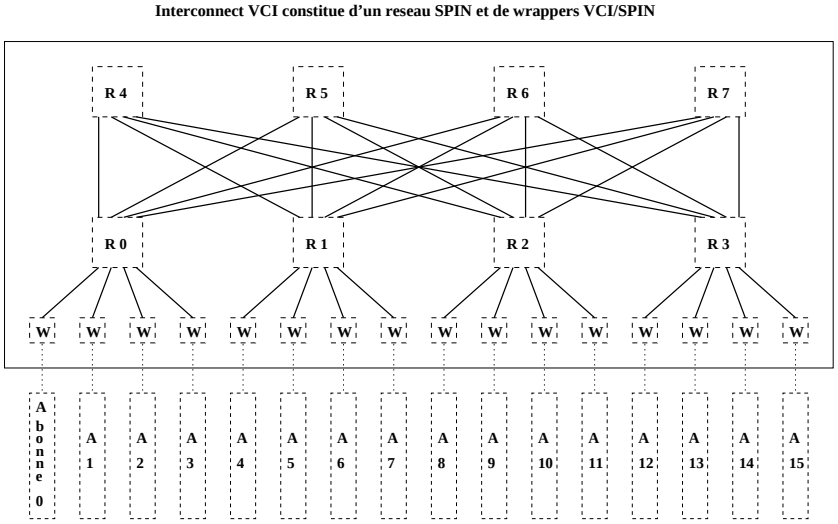

FIG. 5.6 – Représentation des interconnexions à tester pour un réseau à 16 ports

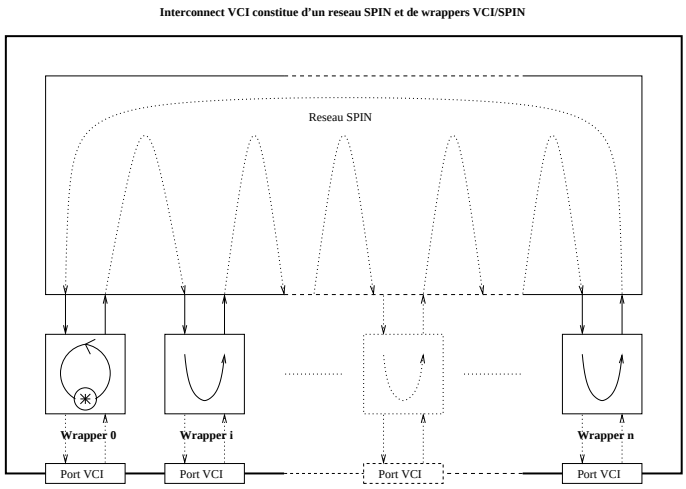

FIG. 5.7 – Parcours des paquets de test au sein du réseau et des wrappers

5.2 Modélisation VHDL des wrappers

5.2.1 Généralités

A travers la réalisation physique des wrappers, nous avons voulu vérifier trois choses. La première, que le protocole SPIN des wrappers était conforme à celui du routeur, donc que l'assemblage routeur/wrappers était cohérent. La deuxième, que les wrappers étaient suffisamment rapides pour ne pas limiter la fréquence de fonctionnement de la structure d'interconnexion. Enfin, que leurs surfaces étaient raisonnables par rapport à celle du routeur.

L'utilisation de la chaîne Alliance [All] a eu un impact direct sur la façon d'écrire les composants. En VHDL IEEE, il est possible d'avoir pêle-mêle dans un même fichier une vue comportementale, une vue en automate et une vue structurelle. En VHDL Alliance, ces trois vues doivent être dans des fichiers distincts. L'intérêt est une meilleure visibilité architecturale du composant. De plus, le VHDL Alliance n'implémente qu'un sous-ensemble du VHDL IEEE, ce qui permet d'avoir un code simplifié, plus lisible, et plus facilement débogable.
L'impact de l'utilisation de la chaîne Alliance se voit aussi sur les résultats obtenus. Bien que fournissant des résultats corrects, elle permet des optimisations moins poussées que les chaînes commerciales.

5.2.2 Méthodologie de conception

Comme méthodologie de conception, nous avons utilisé plusieurs outils de la chaîne de CAO Alliance (développée au LIP6) pour la synthèse, le placement et le routage. Pour l'analyse de timing, nous avons utilisé l'outil Tas d'AVERTEC. Le schéma de la Figure 5.6 résume l'enchaînement des différents outils.

Chaque étape a fait l'objet de vérifications et la totalité de l'exécution s'est faite sur la même machine. Il n'a donc pas été nécessaire de changer de système d'exploitation ou d'architecture matérielle. De plus, la chaîne de CAO Alliance a été conçue de telle sorte que chacun de ses outils puisse être appelé en ligne de commande, avec toutes les options nécessaires. Concrètement, la méthodologie de conception se traduit par un gigantesque Makefile générant, récupérant et analysant les résultats.

5.2.3 La synthèse

Afin de faciliter la phase de synthèse, les wrappers ont été découpés en plusieurs blocs, chaque bloc étant lui-même constitué de sous-blocs. Un bloc ou un sous-bloc est un fichier ou un ensemble de fichiers VHDL représentant une des trois vues suivantes : vue en automate, vue structurelle et vue comportementale.
Dans les descriptions hiérarchiques des wrappers représentées dans les figures 5.7 et 5.8, les feuilles sont des vues en automates ou des vues comportementales ; les noeuds sont représentés par des vues structurelles.

Le wrapper initiateur

Le découpage en blocs est un peu plus fin que celui présenté à la section 5.1.3. On retrouve les 3 blocs principaux : celui gérant la transformation d'une requête VCI en une requête SPIN/VCI, celui gérant le passage d'une réponse SPIN/VCI en une réponse VCI et, enfin, celui contenant les bascules pending.
Chacun de ces trois blocs est constitué de sous-blocs (Cf Figure 5.3). Contrairement à ce qui précède, l'insertion dans des blocs diffère de la description des automates et de la combinatoire.

Le wrapper cible

Comme pour le wrapper initiateur, la décomposition de l'architecture interne est plus poussée que celle présentée à la section 5.1.6. Le wrapper cible se décompose en 3 blocs principaux. Le premier gère la transformation d'une requête SPIN/VCI en une requête

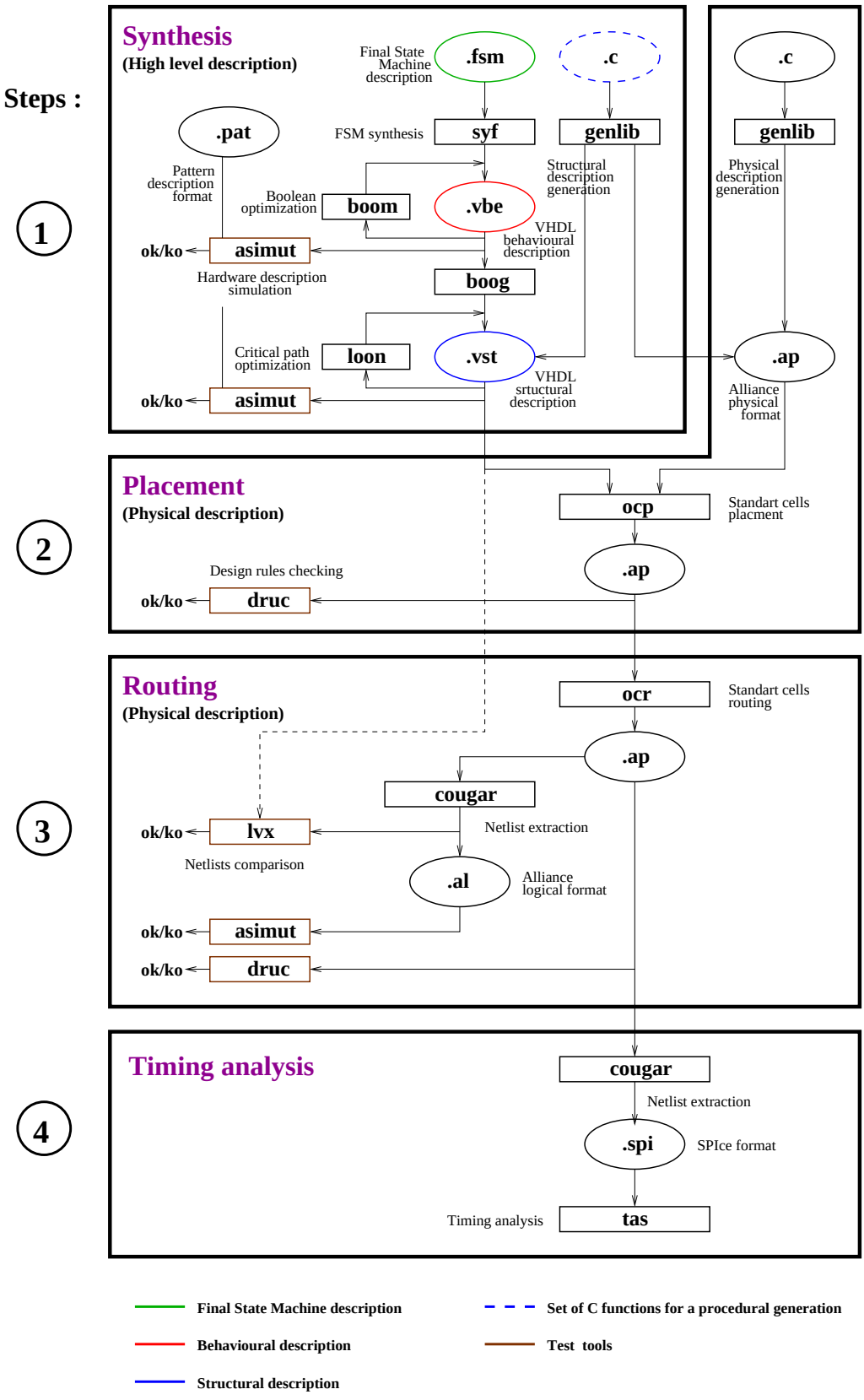

FIG. 5.8 – Méthodologie de conception

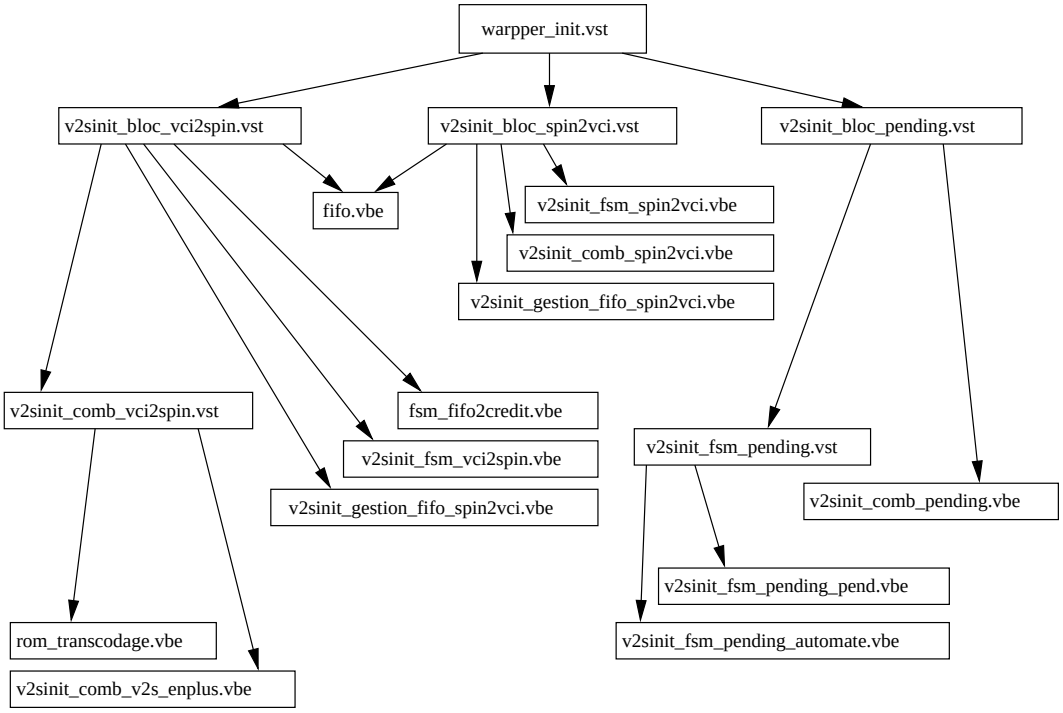

FIG. 5.9 – Hiérarchie architecturale du wrapper initiateur

VCI; le deuxième le passage d'une réponse VCI en une réponse SPIN/VCI; enfin, le troisième gère la bascule pending de sérialisation des requêtes. Chacun de ses trois blocs est constitué de sous-blocs [2].

5.2.4 Placement et routage

Les contraintes au niveau du placement et du routage sont extraites des différentes représentations de réseau à 4, 8, 16 et 32 ports. La figure 5.11 représente le placement théorique d'un réseau SPIN à 16 ports contenant 8 routeurs et 16 wrappers. Quelle que soit la taille du réseau, les schémas de placement montrent qu'une largeur de wrapper doit correspondre à la moitié de celle d'un routeur. Précisons aussi que les FIFO des routeurs et des wrappers sont toutes de profondeur 4. La largeur d'un routeur étant de $0,54\ mm$, celle des wrappers doit donc être inférieure à $0,27\ mm$.
Les dimensions du wrapper cible sont : $0,228\ X\ 0,276\ mm^2$ soit une surface de $0,062928\ mm^2$. Les dimensions du wrapper initiateur sont : $0,252\ X\ 0,288\ mm^2$ soit une surface de $0,072576\ mm^2$. Par rapport au routeur qui a une surface de $0,54\ X\ 0,54\ mm^2$ soit $0,2916\ mm^2$, la surface du wrapper cible représente $1/5$ de la surface du routeur, et celle

[2] Cf Figure 5.8 : Hiérarchie architecturale du wrapper cible

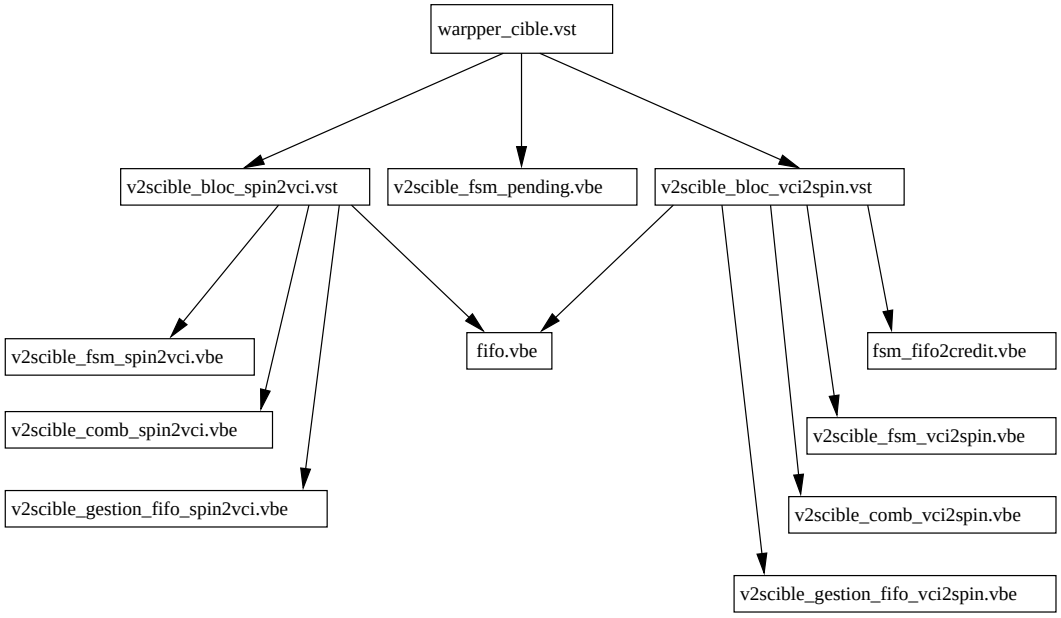

FIG. 5.10 – Hiérarchie architecturale du wrapper cible

du wrapper initiateur 1/4 de celle du routeur.

Pour le routage, la seule contrainte est que les quatre premiers niveaux de métal doivent être utilisés, les trois autres devant servir au routage de l'ensemble de la structure d'interconnexion (wrappers + routeurs) et du boîtier (structure d'interconnexion + abonnés). On se limite à 4 niveaux car c'est le nombre de niveaux utilisés pour le routage de la cellule du routeur RSPIN du réseau avec la chaîne Alliance.

5.2.5 Analyse temporelle

Une étude antérieure faite sur la fréquence maximale de fonctionnement du routeur a permis d'évaluer celle-ci à 690 MHz, à partir d'un fichier de transistors extrait sans les capacités et les résistances. Le routeur étant beaucoup plus complexe et imposant que les wrappers, c'est lui qui doit imposer sa fréquence de fonctionnement à la structure d'interconnexion. Il est donc nécessaire que le routeur et les wrappers aient des fréquences de fonctionnement peu différentes.
Sans réaliser d'optimisations poussées, les fréquences maximales trouvées pour les wrappers initiateur et cible sont respectivement de 600 MHz et de 700 MHz. Ces fréquences nous conviennent car elles sont du même ordre de grandeur que celle du routeur.

L'ensemble routeurs/wrappers fonctionne donc à une fréquence relativement élevée. Ceci illustre l'un des avantages de SPIN qui est de pouvoir monter en fréquence grâce à

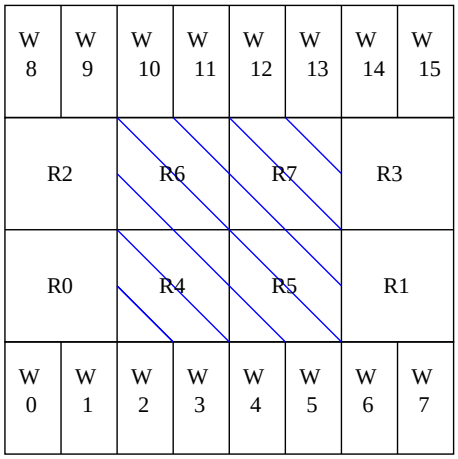

FIG. 5.11 – Placement des routeurs et des wrappers sur un réseau SPIN à 16 ports

ses liaisons point à point. Rappelons qu'en général la fréquence de fonctionnement des bus systèmes est de l'ordre de 150 MHz (par exemple c'est la fréquence d'un Pi-Bus en technologie $0.35\mu m$.
Enfin, contrairement à ce qui se passe sur un bus, le nombre de composants n'a pas d'influence sur la fréquence de fonctionnement d'un réseau SPIN.

5.3 Conclusion

Les résultats de ce chapitre, consacré à la réalisation physique des wrappers, permettent de valider les choix architecturaux et comportementaux faits lors de l'élaboration du cahier des charges.
Les trois questions du début du chapitre ont eu des réponses positives : le routeur et les wrappers communiquent correctement et les contraintes en surface, en routage et en fréquence sont bien respectées. Naturellement si l'on se place dans l'optique de réaliser physiquement un boîtier de test, il conviendra d'utiliser les outils des chaînes de CAO commerciales.

Chapitre 6

Comparaison des performances SPIN / PI-BUS

Sommaire

Cette étude de performances se découpe en deux parties. La première porte sur un test de résistance à la charge, la seconde sur l'exécution d'une application réelle. Nous présenterons donc brièvement, pour chacune des études, les différents abonnés et les environnements de test. Chaque étude se terminera par une analyse des résultats.

6.1 Etude du comportement du PI-Bus et du réseau SPIN sur un test de charge

6.1.1 Présentation générale

L'idée générale du test de charge est d'inonder le système de paquets, afin de tester la résistance à la charge de la structure d'interconnexion, la robustesse des architectures des composants et des protocoles de communication.
Des initiateurs appelés GAP (Générateur Analyseur de Paquets) envoient des rafales de requêtes de lecture à des cibles qui sont des RAM simplifiées. Une RAM simplifiée renvoie pour chaque paquet-requête, un paquet-réponse contenant les adresses de la transaction. La simulation se termine lorsque tous les GAP ont reçu toutes les réponses aux requêtes envoyées.
Le comportement des GAP est contrôlé par le biais de plusieurs paramètres. Il sera possible de spécifier entre autres, la taille des paquets requête, le nombre de paquets à envoyer et la charge offerte. Cette charge offerte (en pourcentage) est directement liée à l'intervalle moyen (en nombre de cycles) entre deux envois de paquets. Soit IM cet intervalle moyen et L la longueur d'un paquet. La valeur de la charge offerte est la suivante :

$$charge\ offerte = \frac{L}{L+IM}.$$

6.1.2 Les environnements de tests

Deux environnements de tests sont mis au point : le premier avec une structure d'interconnexion PI-Bus et le second avec un réseau SPIN. Ces deux environnements ne diffèrent que par le type de la structure d'interconnexion utilisée. Les abonnés et leur paramétrage restent rigoureusement identiques. Les Figures 6.1 et 6.2 représentent ces environnements. Il y a au total 16 abonnés : 8 initiateurs (GAP) et 8 cibles (RAM). Afin d'équilibrer au mieux la distribution des paquets, les initiateurs et les cibles sont entrelacés et en nombre égal, afin de pouvoir atteindre le nombre maximal de communications établies (qui est de $n/2$ communications pour un ensemble de n abonnés).

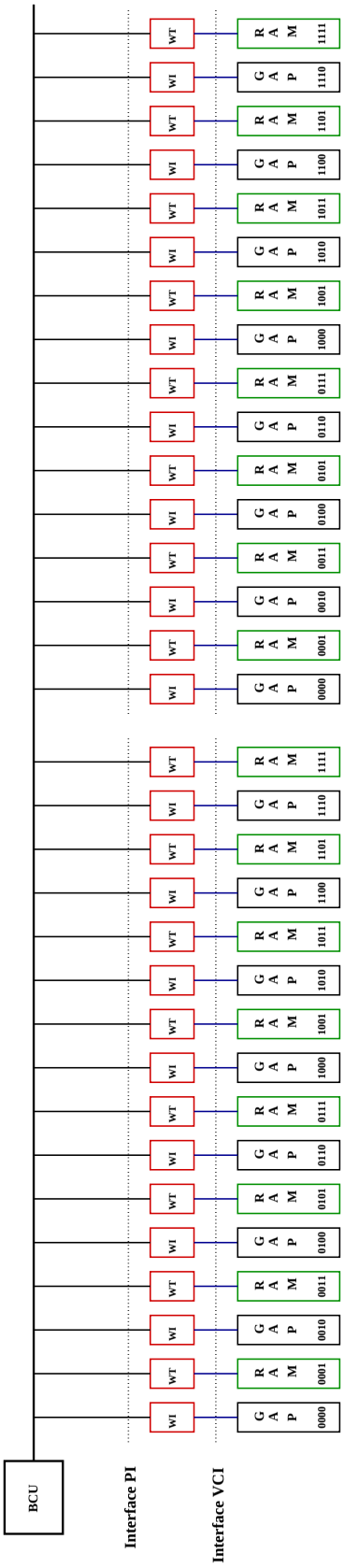

FIG. 6.1 – Environnement d'évaluation PI-Bus

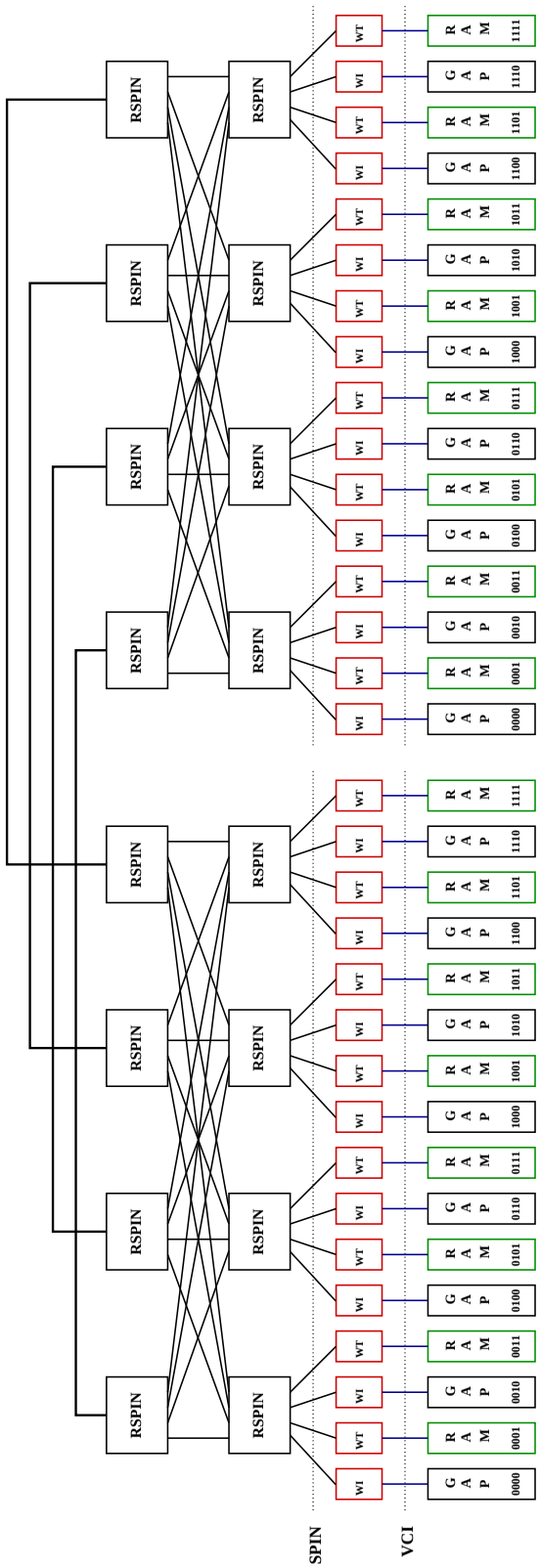

FIG. 6.2 – Environnement d'évaluation SPIN

Remarquons que les abonnés ont tous des interfaces VCI. Pour les connecter au réseau SPIN nous utilisons des wrappers VCI/SPIN initiateur et cible. Dans le cas d'une structure d'interconnexion PI-Bus, nous utilisons des wrappers VCI/PI-Bus maître et esclave.

Pour les simulations, nous avons le choix entre les simulateurs CASS [Den98][?][Hom01] (développé au LIP6) et SystemC [Syn00]. Les résultats obtenus avec l'un et l'autre sont similaires, les simulations étant dans les deux cas `cycle true / bit true`.
La grande différence entre ces deux outils est leur moteur de simulation. Celui de SystemC est événementiel, celui de CASS est à ordonnancement statique. CASS est donc en pratique près de 10 fois plus rapide que SystemC, cette différence s'accroissant avec la taille de l'environnement de tests.

6.1.3 Analyse des résultats

Les courbes de la figure 6.3, représentent la latence moyenne en fonction de la charge offerte pour les environnements précédemment cités. Les GAP envoient 100000 paquets de 8 mots.
Nous voyons que pour des systèmes faiblement chargés (en dessous de 5%), le PI-Bus fournit de meilleurs résultats. Dans le cas du réseau SPIN en revanche, la saturation se produit plus tardivement. Elle est obtenue à partir d'une charge de 28%. Cette différence s'explique par le haut niveau de parallélisme autorisé par SPIN. Evidemment, le gain en bande passante se paye par une augmentation de la latence (30 cycles au lieu de 10).

6.2 Etude sur une application réelle

6.2.1 Présentation générale

Nous avons tiré profit des possibilités offertes par le micro-noyau *MUTEK* [Gom02]. Ce système d'exploitation embarqué développé au LIP6 supporte les applications multi-tâches s'exécutant sur des architectures multi-processeurs et multi-bancs mémoire. Il permet au concepteur de contrôler précisément le placement (statique) des tâches sur les processeurs, et le placement (statique) des données sur les bancs mémoire. Cette fonctionnalité est d'une grande importance, parce qu'elle permet de mieux exploiter le parallélisme au sein des communications. Dans une application multi-tâches, les communications entre tâches peuvent être réalisées par le biais de canaux de communications réalisés comme des tampons mémoires partagés. Nous souhaitons pouvoir distribuer les canaux de communication sur plusieurs bancs mémoire, afin d'éviter que dans un système à mémoire distribuée, la mémoire ne devienne le goulot d'étranglement. Une autre fonctionnalité

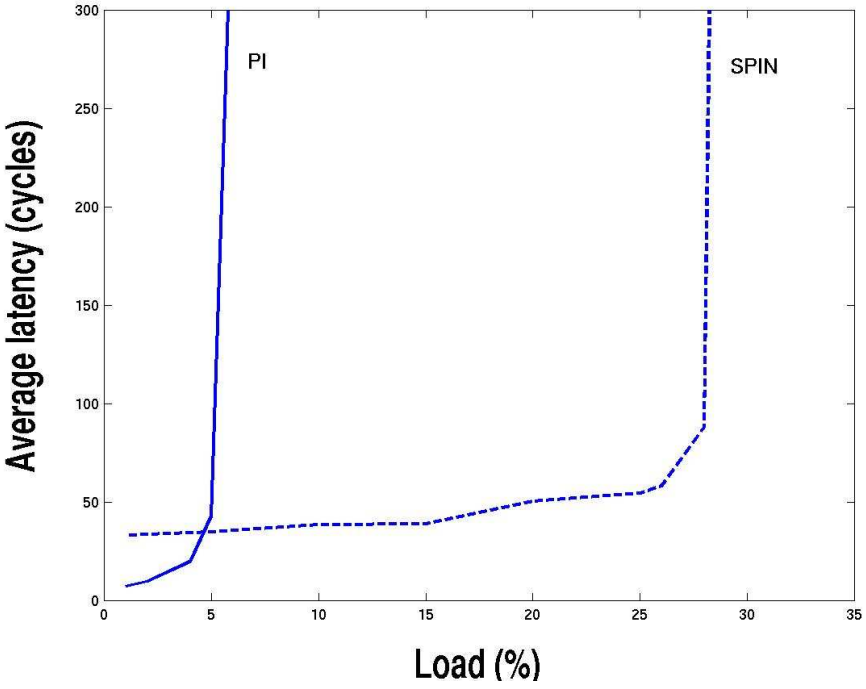

FIG. 6.3 – Evolution de la latence moyenne en fonction de la charge

non moins importante de MUTEK est de permettre la cohérence mémoire. Pour ce faire, nous déclarons non-cachables les tampons de communication.

L'application utilisée a pour objectif de charger la structure d'interconnexion en générant de gros transferts simultanés d'informations. Nous avons maintenant comme initiateurs des processeurs MIPS R3000 [G. 92] disposant de caches et générant un trafic un peu plus varié que de simples rafales de lecture. Enfin, dans le prolongement de ce qui précède, l'intérêt d'utiliser une application réelle est aussi de pouvoir tester la synchronisation des processeurs, la prise de sémaphores pour la gestion des sections critiques, ainsi que la capacité des wrappers à gérer un trafic varié.

6.2.2 Structure de l'application logicielle multi-tâches

Soit n le nombre de tâches qui participent aux échanges de données. A chacune de ces tâches est associé un processeur logique. Deux autres tâches sont destinées à l'initialisation de l'application et aux tests de fin de traitement. Ces deux tâches n'étant pas très gourmandes en ressource processeur, elles peuvent tourner toutes les deux sur le même processeur. Il y a donc $n + 2$ tâches s'exécutant sur $n + 1$ processeurs.

La figure 6.4 représente l'organisation de l'application. Les cercles représentent les tâches et les flèches les canaux de communication. La tâche d'initialisation (`INIT`) envoie un jeton à chaque tâche de traitement (`TH x`). Le traitement consiste à lire un tableaux d'entiers dans le canal de communication provenant de son voisin de gauche et à recopier ce tableau dans le canal de communication vers son voisin de droite. Dès qu'une tâche de traitement reçoit son jeton elle commence son traitement et envoie à la fin un jeton à la tâche de test de fin (`RECEP`). Une fois que cette dernière a reçu tous les jetons des tâches de traitement, elle met fin à la simulation.

Cette application est configurable de l'extérieur grâce à trois paramètres : le nombre de tâches de traitement (et donc le nombre de processeurs), le nombre de mots des tableaux à transférer et le nombre de transferts par processeur.

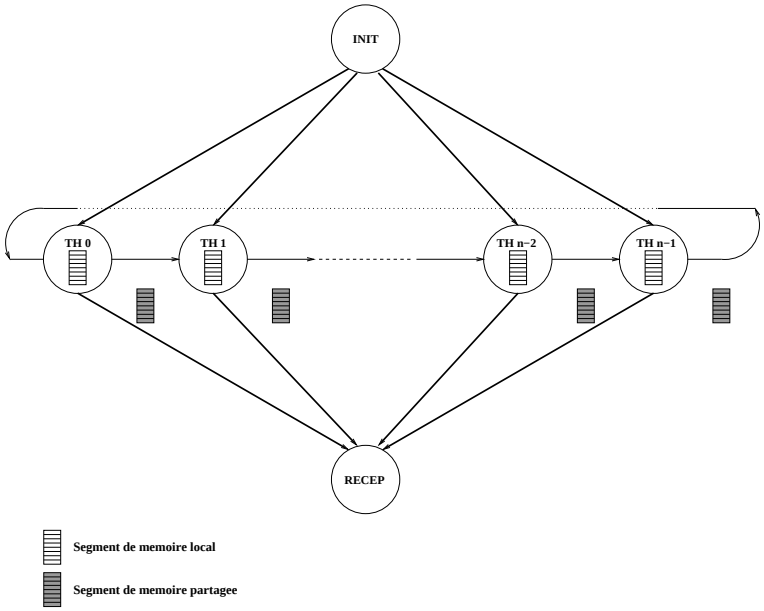

FIG. 6.4 – Organisation de l'application

Arbitrairement, nous fixons les actions des tâches de numéro pair et impair. Les pre-

miers commenceront par écrire dans le canal de "droite", les seconds par lire dans le canal de "gauche", puis les rôles s'inverseront. Cependant, un processeur ne peut lire dans un canal que s'il est non vide et un processeur ne peut écrire que si le canal est non plein.
Les canaux de communication sont mappés dans des segments mémoire partagés. Donc, pour un processeur, l'écriture consiste à transférer un tableau de son segment local vers un segment partagé ; la lecture consiste à transférer un tableau depuis un segment partagé vers son segment local.

6.2.3 Architecture matérielle

L'environnement de test, représenté à la figure 6.5, est construit autour d'un réseau à 32 ports. Le réseau en lui-même est constitué de 16 routeurs RSPIN et de 32 wrappers VCI/SPIN (16 de type initiateur et 16 de type cible). Chaque routeur de premier niveau est relié à deux initiateurs et deux cibles (on parlera de *cluster*). Le dernier cluster contient 1 initiateur et 3 cibles.
Les initiateurs VCI sont des caches instructions/données reliés à des processeurs MIPS R3000. Les caches et les processeurs sont tous multi-contextes. Il faut à ce stade préciser les notions de processeur logique et de processeur physique.
Un processeur physique est un composant matériel contenant plusieurs processeurs logiques. Un processeur logique est un contexte matériel d'exécution. Les processeurs physiques utilisés sont multi-contextes et fonctionnent sur un principe de multiplexage temporel : lorsque une tâche s'exécutant sur un processeur logique fait `MISS`, elle est évincée (en 1 cycle) et une autre tâche s'exécutant sur un autre processeur logique est sélectionnée. Les caches sont également multi-contextes. Un cache possède autant de contextes que le processeur physique qui lui est associé. Nous avons choisi d'utiliser des processeurs multi-contextes pour maximiser le trafic généré vers la mémoire.

Les cibles sont des RAM de données, une RAM d'instructions, une RAM des sémaphores et un écran d'affichage. Les RAM de données contiennent des segments mémoire qui sont partagés ou locaux. Au sein de chaque *cluster* on s'efforce de mettre un segment mémoire partagé et un segment mémoire local.

Les segments de *code*, d'*exception* et de *reset* sont mis dans la même RAM VCI. Leur répartition sur plusieurs cibles serait coûteuse en nombre de ports VCI et peu utile, vu la grande capacité des caches d'instructions (1024 lignes de 16 mots). Les segments associés aux afficheurs sont pour les mêmes raisons regroupés au sein de la même cible VCI au lieu d'être distribués sur plusieurs clusters. Ces deux choix précédemment cités font de la RAM d'instructions (*rst/excp/code*) et des afficheurs deux ressources partagées. Cependant, du fait de leur faible taux d'utilisation ce choix n'a pas d'impact significatif.

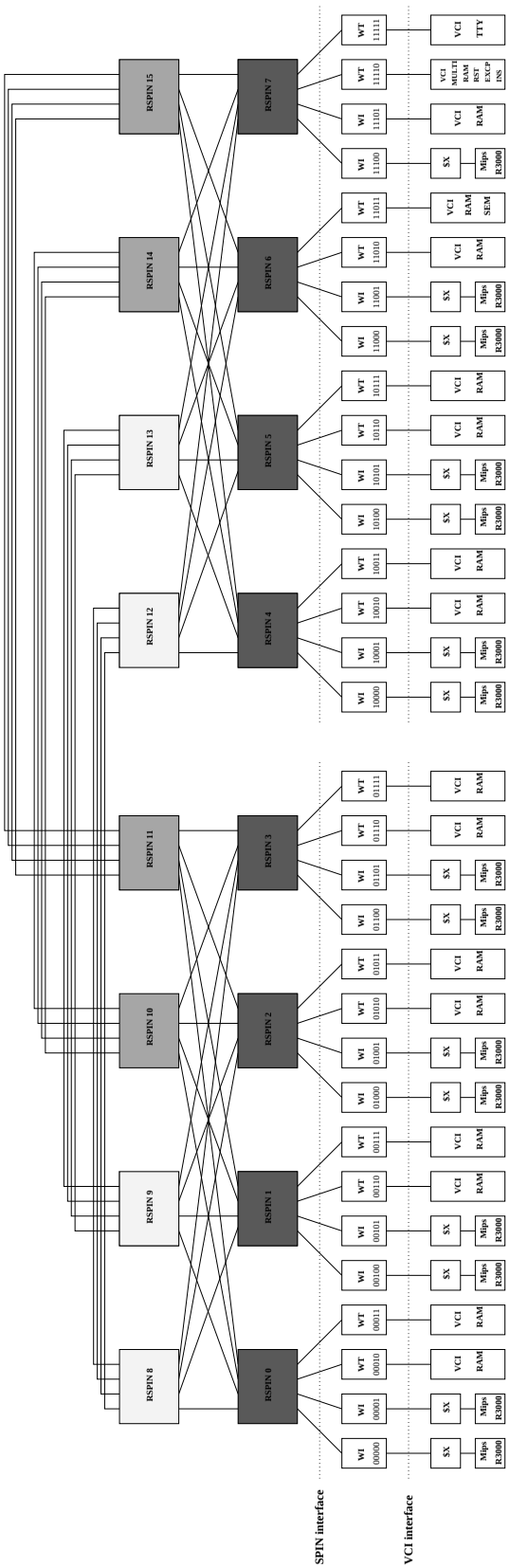

FIG. 6.5 – Réseau SPIN à 32 ports

6.2.4 La nature du trafic

Lorsque l'on exécute plusieurs tâches sur plusieurs processeurs, le micro-noyau a un important travail de synchronisation à effectuer. Cela se traduit par de nombreuses requêtes de lecture et d'écriture vers la RAM des sémaphores.
L'écriture et la lecture à travers des canaux de communication vont générer des requêtes vers le segment mémoire du noyau, afin de vérifier si les canaux sont "non plein" ou "non vide". Pour réduire au maximum ce trafic, nous donnons à ces canaux des profondeurs supérieures au nombre de mots à y écrire ou lire.
Les segments mémoires partagés sont déclarés "non cachable". Par conséquent les requêtes de lecture aux adresses qu'ils contiennent sont des requêtes VCI d'un seul mot. Donc, lire 1024 mots se traduira par 1024 paquets de lecture.
Ci-dessous nous donnons en exemple la nature du trafic sur l'interface VCI du cache instructions/données du processeur 0, au sein d'un système à 11 processeurs. Pour chaque cache nous avons la longueur en nombre de mots de la requête, puis le nombre de requêtes de ce type, et enfin, le nombre total de cycles associé à la requête.

Exemple de trafic sur un cache instructions/données :

```
CACHE D'INSTRUCTIONS - Stats des lectures :
16 mots  : 218    soit 3488   cycles

CACHE DE DONNEES - Stats des lectures :
1 mot    : 356204 soit 356204 cycles
16 mots  : 165    soit 2640   cycles

CACHE DE DONNEES - Stats des écritures :
1 mot    : 138673 soit 138673 cycles
2 mots   : 3375   soit 6750   cycles
3 mots   : 194    soit 582    cycles
4 mots   : 111    soit 444    cycles
5 mots   : 245    soit 1225   cycles
6 mots   : 201    soit 1206   cycles
7 mots   : 49     soit 343    cycles
8 mots   : 121    soit 968    cycles
9 mots   : 50     soit 450    cycles
11 mots  : 14     soit 154    cycles
12 mots  : 2      soit 24     cycles
13 mots  : 3      soit 39     cycles
```

```
14 mots  : 126    soit 1764   cycles
15 mots  : 16     soit 240    cycles
16 mots  : 7      soit 112    cycles
17 mots  : 1482   soit 25194  cycles
18 mots  : 4694   soit 84492  cycles
19 mots  : 5      soit 95     cycles
20 mots  : 57     soit 1140   cycles
21 mots  : 52     soit 1092   cycles
22 mots  : 203    soit 4466   cycles
23 mots  : 26     soit 598    cycles
24 mots  : 108    soit 2592   cycles
25 mots  : 2      soit 50     cycles
26 mots  : 2      soit 52     cycles
27 mots  : 124    soit 3348   cycles
28 mots  : 10     soit 280    cycles
29 mots  : 4      soit 116    cycles
30 mots  : 5      soit 150    cycles
31 mots  : 3      soit 93     cycles
32 mots  : 3      soit 96     cycles
34 mots  : 3      soit 102    cycles
36 mots  : 2      soit 72     cycles
38 mots  : 5      soit 190    cycles
40 mots  : 1      soit 40     cycles
44 mots  : 1      soit 44     cycles
48 mots  : 2      soit 96     cycles
53 mots  : 3      soit 159    cycles
54 mots  : 2      soit 108    cycles
56 mots  : 3      soit 168    cycles
57 mots  : 3      soit 171    cycles
58 mots  : 19     soit 1102   cycles
59 mots  : 1      soit 59     cycles
60 mots  : 3      soit 180    cycles
61 mots  : 2      soit 122    cycles
62 mots  : 4      soit 248    cycles
64 mots  : 2      soit 128    cycles
65 mots  : 3      soit 195    cycles
66 mots  : 6      soit 396    cycles
```

```
67 mots  : 2      soit 134    cycles
68 mots  : 50     soit 3400   cycles
104 mots : 30     soit 3120   cycles
```

D'une manière générale, nous remarquons qu'il y a beaucoup de paquets de lecture et d'écriture d'un mot. Au chapitre précédent, nous avons vu qu'à chaque envoi de paquet on perd un cycle pour générer le header. Donc, étant donné le nombre de paquets d'un seul mot envoyés, on perd beaucoup de cycles à générer des headers.

6.2.5 La charge

Les résultats de la comparaison, pour être pertinente, nécessitent que l'on puisse caractériser le trafic. Pour mieux connaître la nature du trafic, nous avons mis des compteurs au niveau de l'interface VCI des caches. Pour le calcul des charges offerte et acceptée, on utilise les signaux $CMDVAL$ et $CMDACK$. Sur le Tableau 6.1 on définit trois variables qui nous permettront d'exprimer les valeurs des charges offerte et acceptée. La variable $N1$ indique le nombre de cycles passés à envoyer des mots. $N2$ indique le nombre de cycles d'attente : le contrôleur du cache cherche à émettre une requête vers la structure d'interconnexion mais celle-ci refuse de l'accepter. Enfin, $N3$ indique le nombre de cycles d'inactivité.

CMDVAL	CMDACK	Nbre cycles
1	1	N1
1	0	N2
0	*	N3

TAB. 6.1 – Définition des variables N1, N2 et N3

On définit la charge offerte comme un pourcentage représentant le nombre de cycles passés à attendre et envoyer des mots ; la charge acceptée est un pourcentage représentant uniquement le nombre de cycle passés à envoyer des mots. Les formules donnant les charges offerte et acceptée en fonction de $N1$, $N2$ et $N3$ sont les suivantes :

$$\textit{Charge offerte} = \frac{N1+N2}{N1+N2+N3}$$
$$\textit{Charge acceptée} = \frac{N1}{N1+N2+N3}$$

6.2.6 Déroulement de l'application

Principes de l'évaluation

On cherche à augmenter le parallélisme au sein du système en gardant toujours la même quantité de travail à effectuer ; cela revient à figer le nombre total de transferts. Lorsque l'on ajoute de nouveaux processeurs, pour que le nombre total de transferts reste le même, il faut diminuer le nombre de transferts par processeur. Concrètement, nous considérons un nombre total de 360 transferts de blocs de 1024 mots. Nous choisissons arbitrairement de faire varier le nombre de processeurs de traitement avec un pas de 6 afin de faciliter les calculs. Le tableau 6.2 donne le nombre de transferts par processeur en fonction du nombre de processeurs afin qu'un total de 360 transferts soient toujours effectués.

Nbre processeurs	Nbre transferts
6	60
12	30
18	20
24	15
30	12
36	10

TAB. 6.2 – Nombre de transferts par processeur en fonction du nombre de processeurs

Tableaux de résultats et courbes

Les tableaux 6.3 et 6.4 donnent le nombre de cycles d'exécution du programme, la charge offerte et la charge acceptée en fonction du nombre de processeurs, pour des architectures à 32 abonnés, respectivement autour d'un PI-Bus et d'un réseau SPIN.
Pour les raisons évoquées précédemment, le nombre de processeurs est un *multiple de 6 + 1*. Le processeur que l'on rajoute à chaque fois n'effectue aucun transfert ; il se contente d'exécuter les tâches d'initialisation et de fin.

6.2.7 Analyses des résultats

Les temps d'exécution

Les résultats concernant les temps d'exécution en fonction du nombre de processeurs logiques sont donnés à la Figure 6.6. La courbe concernant le PI-Bus est très légèrement

Nbre processeurs	Nbre cycles	Charge offerte	Charge acceptée
13	4 243 486	8.71	7.69
19	4 663 534	8.25	6.66
25	4 620 202	9.72	6.66
31	4 537 078	10.60	6.66
37	5 201 134	12.13	6.66

TAB. 6.3 – Résultats d'exécution de l'application autour d'un PI-Bus

Nbre processeurs	Nbre cycles	Charge offerte	Charge acceptée
13	5 621 390	9.3	3.1
19	4 188 807	11.11	3.82
25	3 155 954	15.99	5.84
31	3 246 428	16.91	6.35
37	3 257 833	17.97	6.85

TAB. 6.4 – Résultats d'exécution de l'application autour de SPIN

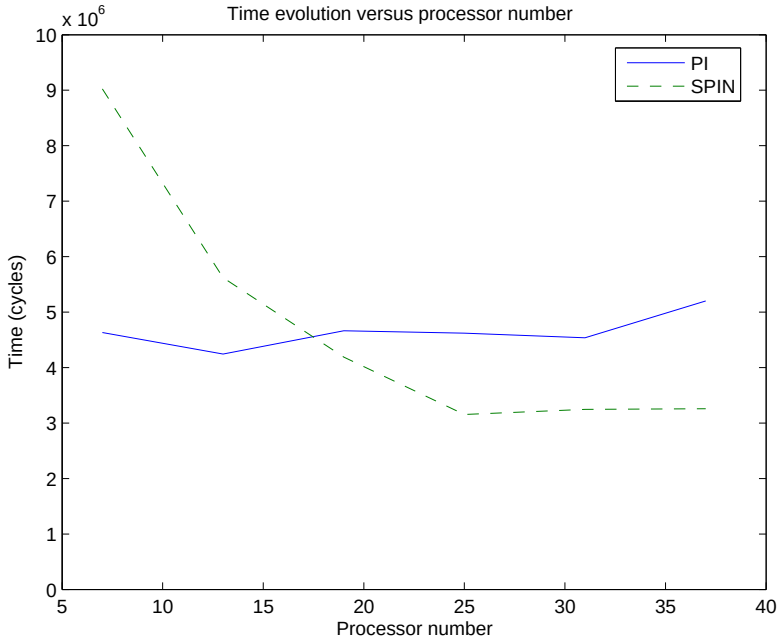

FIG. 6.6 – Evolution du temps de calcul en fonction du nombre de processeurs

croissante. Cela est dû aux appels systèmes liés à la gestion des processeurs logiques. Plus on augmente leur nombre, et plus il faut de temps (en nombre de cycles) pour les synchroniser. Le fait que la croissance ne soit pas très prononcée vient du fait que la quantité de travail (le nombre de transferts) est constante et que, même avec une augmentation du parallélisme (par le biais des processeurs logiques), on n'augmente quasiment pas le nombre de données transférées.
Le comportement est différent dans le cas du réseau SPIN. Le nombre de cycles dûs aux appels systèmes est compensé par la possibilité d'avoir plusieurs communications en parallèle. Cela explique la décroissance de la courbe. Mais naturellement, le parallélisme a des limites que l'on observe à partir de 25 processeurs logiques. On se retrouve alors dans le même cas que le PI-Bus.
Les deux courbes se coupent lorsque l'on passe le cap des 17 processeurs logiques. Dans le cas particulier de cette application et pour le système considéré, le réseau SPIN devient meilleur que le PI-Bus à partir de 17 processeurs logiques.

Les charges acceptées

L'évolution de la charge acceptée en fonction du nombre de processeurs logiques est donnée à la Figure 6.7.

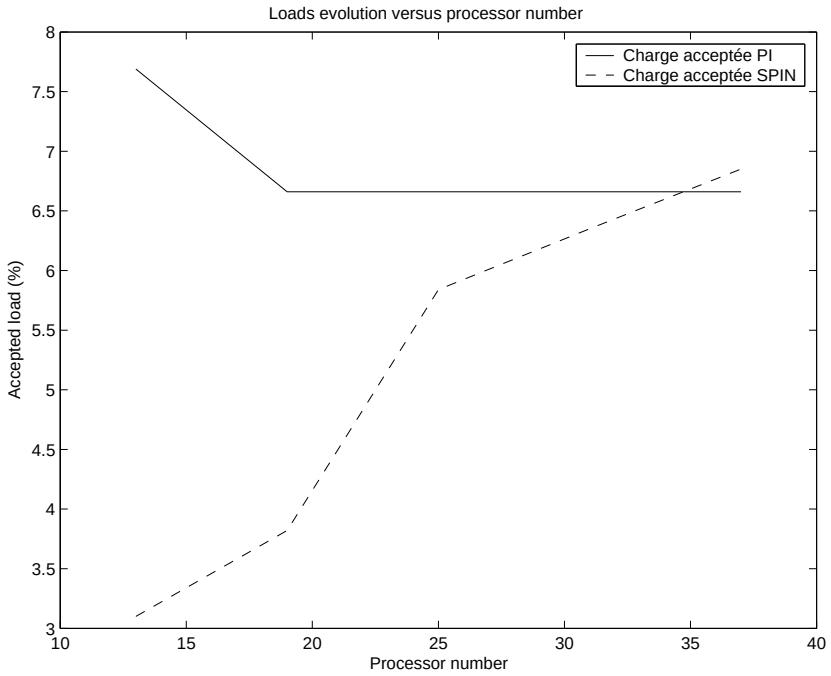

FIG. 6.7 – Evolution de la charge en fonction du nombre de processeurs

La courbe du PI-Bus est décroissante jusqu'à 19 processeurs logiques puis devient constante. La décroissance de la courbe est due au fait que lorsque l'on passe de 13 à 19 processeurs logiques, on passe de 13 à 15 initiateurs physiques utilisés. Le fait de rajouter deux initiateurs diminue la bande passante allouée à chacun d'eux. A partir de 19 processeurs logiques, l'augmentation de ce nombre ne provoque pas l'augmentation du nombre d'initiateurs physiques mais la fréquence d'accès des caches au réseau (variable $N1$). Cela entraîne une augmentation de l'attente pour l'envoi des paquets (variable $N2$). De plus, la variable $N3$ évolue dans le même sens que la variable $N1$ car l'on aura toujours après chaque requête un délai correspondant au temps d'attente de la réponse. On observe donc une autorégulation du système : les caches veulent augmenter la fréquence d'envoi des requêtes, mais ils doivent aussi attendre plus longtemps avant d'avoir la main.
Avec le réseau SPIN, du fait de la possibilité d'avoir plusieurs communications simultanées, on obtient une courbe croissante. La limitation que l'on avait avec le PI-Bus au niveau du temps d'attente des caches pour avoir la main est presque inexistante : la fréquence d'envoi des requêtes par les caches (lié à $N1$) augmente plus rapidement que le temps d'attente pour accéder au réseau (lié à $N2$). Par conséquent, la charge acceptée augmente avec le nombre de processeurs logiques.

6.2.8 Conclusion

Nous avons validé l'architecture des composants et les protocoles de communication en réalisant un test de surcharge et un test avec une application réelle. Lors du premier test, nous avons réussi avec des initiateurs particuliers (GAP) à saturer le PI-Bus et le réseau SPIN, mettant ainsi en avant le fait que le réseau SPIN résiste mieux à la charge que le PI-Bus.
Lors du deuxième test, nous avons validé le comportement des composants sur un trafic varié permettant ainsi de tester les synchronisations de processeurs, les prises de sémaphores, les accès au code système. L'autre enseignement de ce test est qu'avec des processeurs programmables, il est difficile de saturer un réseau SPIN car le système s'autorégule. Lorsqu'un cache envoie une requête il est obligé d'attendre la réponse correspondante avant d'envoyer une autre requête, et le temps d'attente d'un cache pour l'envoi d'une requête est fonction du nombre de caches voulant envoyer une requête. On vérifie cependant que la charge acceptée par le réseau SPIN augmente avec le nombre de processeurs.

Chapitre 7

Vérification formelle de l'inter-blocage

Sommaire

Ce chapitre traite de l'utilisation des techniques de vérification formelle afin de valider une hypothèse visant à supprimer les inter-blocages au sein des réseaux SPIN/VCI. Après avoir analysé un cas d'inter-blocage et présenté la solution préconisée, nous présentons la validation de cette solution sur un réseau SPIN simplifié.

7.1 Généralités

Dans un réseau à commutation de paquets véhiculant simultanément des requêtes et des réponses sur le même support, les paquets peuvent se trouver dans une configuration telle qu'aucun ne puisse arriver à destination, des paquet-réponses étant bloqués par des paquet-requêtes, eux-mêmes bloqués par des paquet-réponses. On se trouve alors en face d'un `inter-blocage` ("deadlock"). Nous analysons dans ce chapitre les conditions d'apparition d'un inter-blocage et nous proposons une solution exploitant la redondance du réseau SPIN, pour supprimer les inter-blocages.

Nous utilisons les techniques de vérification formelle pour montrer que le réseau ne séparant pas les requêtes des réponses peut présenter des inter-blocages, alors que celui qui les distingue en est exempt. Pour la modélisation nous avons utilisé le langage `ProMeLa` ; pour la vérification, nous avons choisi le système `Xspin` de Holzmann [Hol97]. Cet outil permet entre autres de rechercher d'éventuels blocages, les portions de codes jamais atteintes, de trouver les invariants, les cycles de non-progression et également de vérifier des propriétés LTL (*Linear Temporal Logic*). L'absence d'inter-blocage peut être modélisée par une propriété LTL de type *chaque initiateur pourra toujours émettre un nouveau paquet* (ce qui implique que tout paquet finit par être acquitté).

Le langage ProMeLa (*Protocol Meta Language*) permet de modéliser des systèmes distribués, parallèles et asynchrones, dont les processus s'exécutent de façon concurrente et communiquent à travers des canaux.
La logique que nous avons retenue est la logique LTL plutôt que la logique CTL (*Computational Temporal Logic*) car elle correspond plus à la propriété que l'on cherche à valider. L'une des grosses différences entre ces deux logiques est que une propriété LTL caractérise un chemin, alors qu'une propriété CTL concerne soit une arborescence, soit un ensemble de chemins. Il est important de signaler que ne disposant pas d'un outil permettant de passer aisément d'une modélisation SystemC à une modélisation ProMeLa, nous avons dû effectuer ce travail de modélisation à la main.

7.2 Inter-blocage dans un réseau SPIN à 16 ports

Nous présentons dans ce qui suit un exemple concret d'inter-blocage au sein d'un réseau SPIN à 16 ports. Le schéma de cet inter-blocage est représenté à la figure 7.1. Il fait intervenir les 8 routeurs RSPIN identifiés par $R0$, $R1$, ..., $R7$, et 8 abonnés sur les 16 connectés. Parmi ces abonnés il y a 4 initiateurs : MA, ..., MD, et 4 cibles : TA, ..., TD. Sur le schéma on distingue 5 types de traits :

- Les traits gras et continus, partant des initiateurs, représentent les requêtes ;
- Les traits gras en traits courts et interrompus, partant de cibles, représentent les réponses ;
- Les traits en pointillé léger représentent les liens non utilisés du réseau SPIN ;
- Les doubles traits courts aux extrémités des flèches des traits des réponses, symbolisent des blocages ;
- Les traits légers, en trait courts et interrompus entre deux ports inférieurs d'un même routeur, représentent les chemins devant être empruntés par les réponses.

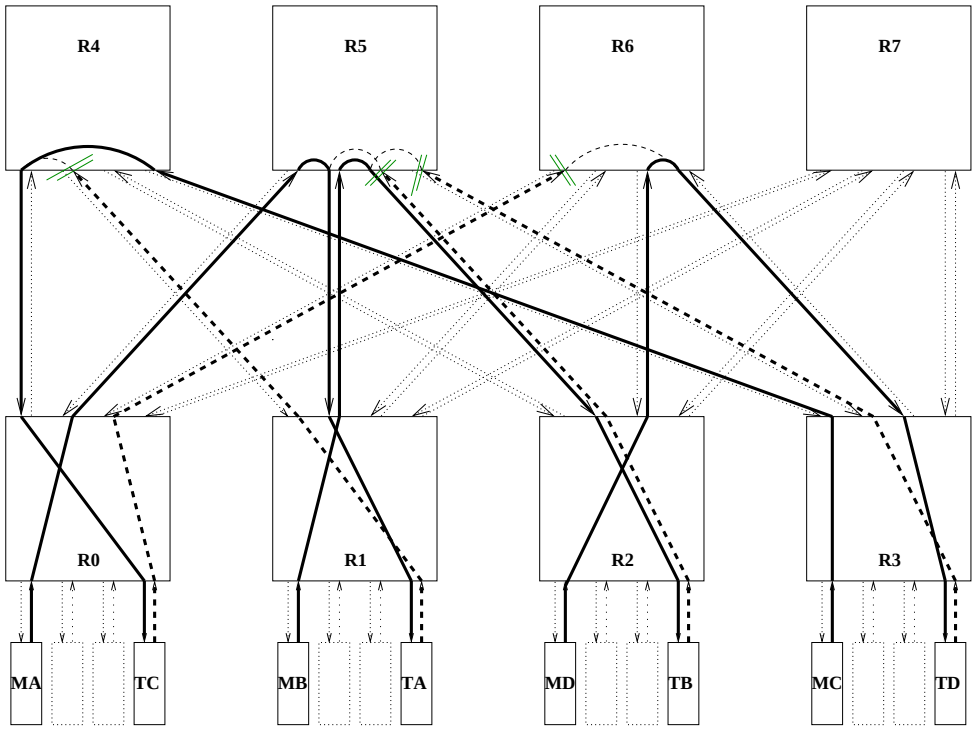

FIG. 7.1 – Schéma détaillé d'un inter-blocage dans un SPIN 16

L'analyse de ce schéma montre que toutes les réponses sont bloquées parce qu'elles doivent emprunter des chemins déjà occupés par des requêtes. Par exemple, la réponse issue de la cible TA arrivant sur le port inférieur 1 du routeur $R4$, est bloquée car elle

doit ressortir par le port inférieur 0 qui est déjà occupé par la requête issue de l'initiateur MC. Les autres réponses sont bloquées pour des raisons similaires faisant intervenir les autres requêtes.

7.3 Solution proposée face aux inter-blocages

7.3.1 Présentation de la solution

Nous avons observé que les inter-blocages proviennent à la fois du couplage requête/réponse imposé par la norme VCI et du comportement de la cible. En effet, la progression du traitement de la requête est liée à la progression de l'envoi de la réponse correspondante. Cela a pour conséquence que si le paquet réponse ne peut plus progresser dans le réseau, alors le paquet requête ne peut plus être traité par la cible. Naturellement, le blocage ne se produit que si la cible a une faible capacité de stockage. Si la cible peut absorber toute la requête avant de renvoyer la réponse, c'est à dire qu'elle a une capacité de stockage très grande, alors aucune réponse ne sera jamais bloquée par des requêtes car ces dernières pourront *toujours* progresser.
La solution proposée pour résoudre l'inter-blocage est bien connue des concepteurs de systèmes. Elle consiste à faire circuler les requêtes et les réponses sur des sous-réseaux disjoints.

7.3.2 Mise en oeuvre de la solution

La mise en oeuvre de la solution nécessite deux conditions. La première est de pouvoir différencier un paquet requête d'un paquet réponse. Cela se fait par le rajout d'un bit dans l'en-tête du paquet. Les valeurs de ce bit sont '0' s'il s'agit d'un paquet-requête et '1' si c'est un paquet-réponse. On utilise pour cela l'un des bits *réservés*.

La deuxième condition, qui est aussi la plus lourde, concerne la structure des réseaux SPIN/VCI qui doit permettre de faire circuler les paquet-requêtes et les paquet-réponses sur des sous-réseaux disjoints, c'est-à-dire sans conflit de ressources.
Ce propos est illustré par la Figure 7.2 qui montre la topologie en arbre quaternaire élargi d'un réseau SPIN à 32 ports. La séparation des requêtes et des réponses se fait dans les routeurs de premier niveau. Ils agissent comme des postes d'aiguillage servant à orienter les paquets dans un sous-réseau ou dans un autre, selon la valeur du bit requête/réponse vu précédemment. Pour ce faire chacun des ports supérieurs (UP) est attribué soit aux requêtes soit aux réponses. Une fois l'aiguillage réalisé, le paquet ne sort jamais du sous-réseau qui lui a été assigné.

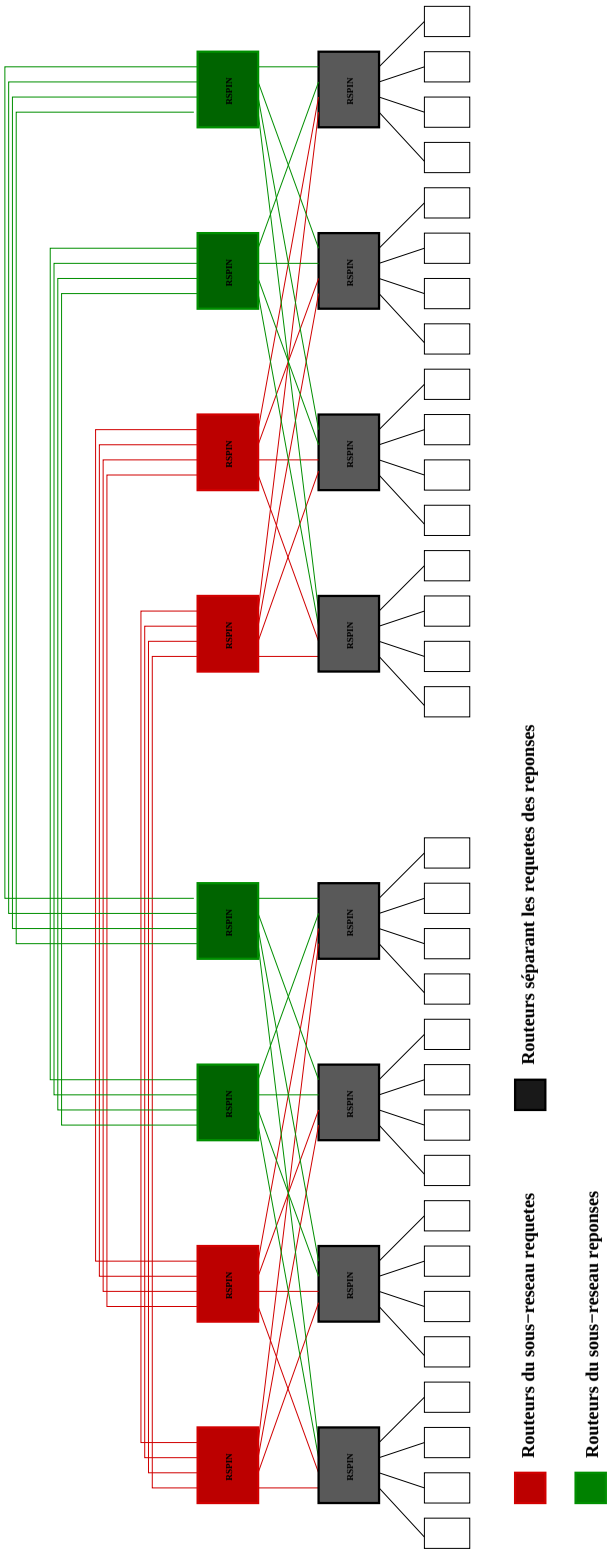

FIG. 7.2 – Topologie solution sur un réseau à 32 ports

Dans le cas du réseau de la Figure 7.2, nous avons arbitrairement assigné les deux premiers ports supérieurs des routeurs de premier niveau, aux requêtes. Les deux autres sont pour les réponses. La topologie générale des réseaux SPIN à n ports est développée au chapitre suivant.

7.4 Les simplifications architecturales en vue de la preuve

La modélisation d'un réseau SPIN complet utilisant des routeurs à 8 ports est beaucoup trop complexe pour être gérée par les outils de vérification formelle Xspin. La raison en est le phénomène d'explosion combinatoire due à l'exploration exhaustive de tous les chemins possibles du graphe d'exécution. Nous avons donc décidé de simplifier l'architecture des routeurs, du réseau et des abonnés.
Nous nous sommes assurés que les simplifications architecturales effectuées afin de réaliser la preuve formelle, conservaient bien les propriétés des réseaux SPIN. En effet, les propriétés caractéristiques d'un réseau SPIN sont :

- une architecture en arbre élargi ;
- un routage distribué, dynamique et adaptatif, de type wormhole ;
- l'acheminement de paquets constitués d'un en-tête clairement identifié suivi d'une charge utile dont le dernier paquet contient un marqueur de fin de paquet ;
- un arbitrage au sein des routeurs pour les conflits d'accès aux ressources internes des routeurs par les paquets ;
- et enfin, une communication s'effectuant selon une logique dite *postale*.

La simplification au niveau du réseau porte sur l'arité de la structure en arbre élargi. Nous passons d'une topologie en arbre quaternaire élargi à une topologie en arbre binaire élargi. Chaque routeur a deux ports supérieurs et deux ports inférieurs. On considère que la solution validée pour une topologie en arbre binaire élargi, s'applique également à une topologie en arbre quaternaire élargi.

Les fonctions de réception, transmission, et routage des routeurs et des abonnés ont été également simplifiées. Nous reviendrons sur l'implémentation ProMeLa des routeurs, des initiateurs et des cibles à la section 7.5.1. Pour l'instant nous ne présenterons que leurs grandes fonctionnalités.
Le composant initiateur est un Générateur/Analyseur de Paquets qui n'envoie des paquets qu'à une seule cible prédéfinie et peut recevoir des mots du paquet réponse sans attendre d'avoir envoyé tous les mots du paquet requête. Limiter ainsi la communication à une seule cible permet, entre autre, de réduire les cas possibles de routage. Il attend également d'avoir reçu le paquet réponse i avant d'envoyer le paquet requête $i + 1$.

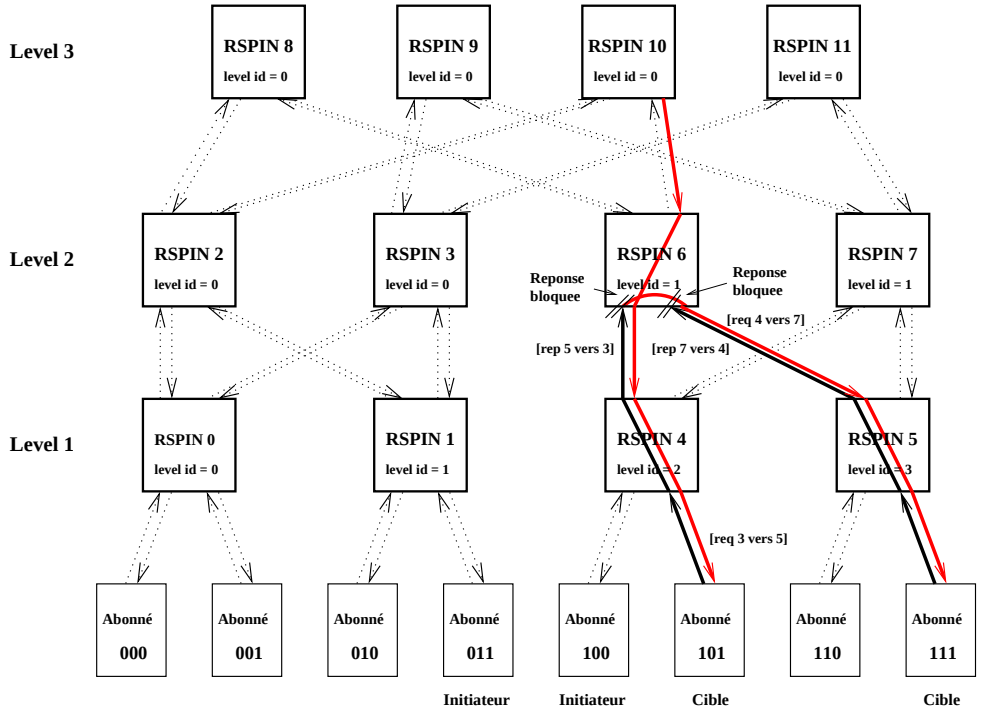

FIG. 7.3 – Schéma détaillé d'un inter-blocage dans SPIN simple à 8 ports

Le composant cible se contente de renvoyer une réponse dès qu'il reçoit une requête. La logique de routage du routeur est très simple : un paquet qui arrive par un port inférieur ne peut que monter et un paquet qui arrive par un port supérieur ne peut que descendre. Dans les deux cas, on n'a le choix qu'entre deux sorties. Le schéma de la figure 7.3 représente un cas d'inter-blocages survenant dans un réseau simplifié à 8 abonnés. Cet inter-blocage a été mis en évidence par `simulation` à l'aide de l'outil `Xspin`.

7.5 Validation d'un réseau SPIN simplifié à 2 routeurs

Un tel système à 8 abonnés demeure cependant trop complexe pour être `vérifié` avec l'outil `Xspin`. La validation d'une propriété nécessite en effet de parcourir de manière exhaustive tous les scénarios d'exécution possibles. Sur une machine quadri-processeurs (4 x 450 Mhz), contenant 4 Go et mémoire vive et 15 Go de Swap, les validations pouvaient tourner une trentaine de jours avant d'être détruites par le système pour des raisons de taille en mémoire (*Out of memory*). Nous avons donc minimisé le nombre de composants pour finalement n'avoir plus que deux routeurs et 4 abonnés. Ce système est présenté à la section suivante.

7.5.1 La modélisation ProMeLa

Cette étude s'est faite sur un réseau SPIN constitué de 2 routeurs RSPIN et de 4 abonnés. Nous rentrons légèrement dans les détails de l'implémentation ProMeLa en montrant sur la figure 7.4 le découpage du système en processus indépendants. Les rectangles rouges représentent les processus et les petits carrés verts, les points mémorisants. Afin de mettre en évidence le plus rapidement possible un deadlock, nous avons choisi de réduire au minimum les capacités de stockage des composants. Chaque port d'entrée et chaque canal de communication ne peuvent mémoriser qu'un mot.

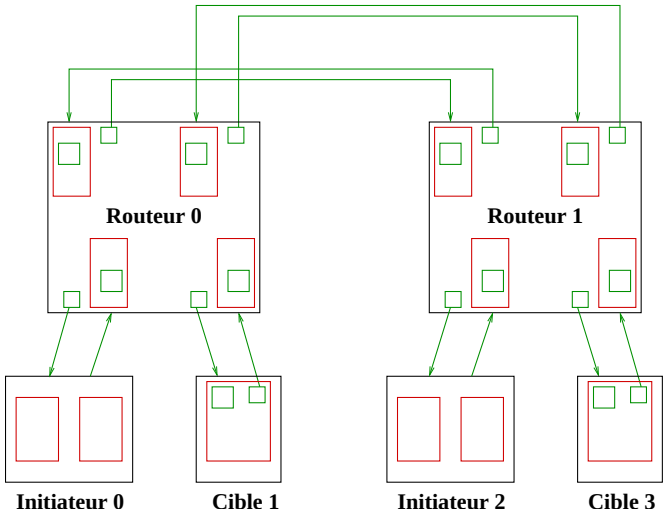

FIG. 7.4 – Découpage du système en processus et points mémorisants

Le composant initiateur se compose de deux processus indépendants, l'un pour l'envoi des requêtes, l'autre pour la réception des réponses. Le composant cible n'a qu'un seul processus qui réceptionne une requête et renvoie la réponse correspondante. Le composant routeur contient 4 processus indépendants, un pour chaque port d'entrée ; c'est là notamment que se trouve la fonction de routage. A l'arrivée d'un paquet, chaque processus cherche à prendre un jeton se situant au niveau du port de sortie désigné. Le langage ProMeLa assure la mise en attente des processus sur un jeton.

7.5.2 Analyse de l'inter-blocage

Le schéma de l'inter-blocage analysé se trouve à la Figure 7.5. Il fait intervenir deux initiateurs et deux cibles. Arbitrairement, l'initiateur 0 communique avec la cible 3, l'initiateur 2 avec la cible 1, et la taille des paquets est fixée à 6 mots. La description de l'inter-blocage est la suivante : au niveau du routeur 0, la réponse de la cible 1 vers l'initiateur 2 est bloquée par la requête de l'initiateur 0 vers la cible 3. Au niveau du routeur

1, la réponse de la cible 3 vers l'initiateur 0 est bloquée par la requête de l'initiateur 2 vers la cible 1.
Examinons plus précisément le blocage de la réponse issue de la cible 1. Lorsque la réponse issue de la cible 1 arrive sur le port inférieur 1 du routeur 0, le processus gérant ce port analyse les jetons associés aux ports de sortie pour savoir lequel des deux est libre. Dans le cas présent, les deux jetons sont disponibles ; le processus réserve donc le premier port de sortie libre qui est le port supérieur gauche. Le transfert en revanche ne peut s'effectuer parce que le canal de communication, allant du port supérieur gauche du routeur 0 au port supérieur gauche du routeur 1 n'est pas vide, puisqu'il contient encore le dernier mot de la requête de l'initiateur 0 vers la cible 2.

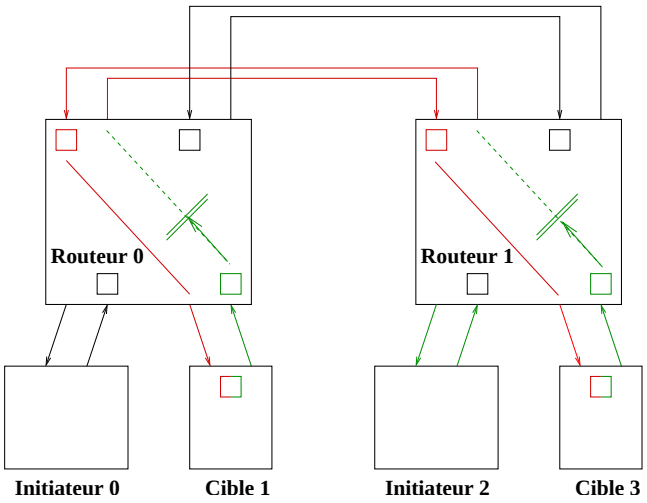

FIG. 7.5 – Schéma détaillé d'un inter-blocage

Sur cette architecture, on remarque que l'inter-blocage est dû au fait que, par hasard, les requêtes et les réponses utilisent les mêmes ports supérieurs. On cherche donc à vérifier la proposition explicitée précédemment concernant la séparation des requêtes et des réponses. Pour ce faire, nous attribuons le port supérieur gauche du routeur aux requêtes et le port supérieur droit aux réponses. La distinction requête/réponse se fait par le positionnement d'un bit à 0 ou 1 dans le mot d'en-tête du paquet. La vérification montre l'absence d'inter-blocage validant ainsi notre hypothèse dans l'architecture considérée.
Les figures 7.6 et 7.7 montrent des topologies de réseaux SPIN simplifiés à respectivement 4 et 8 abonnés, mettant en oeuvre le principe de séparation des requêtes et des réponses.

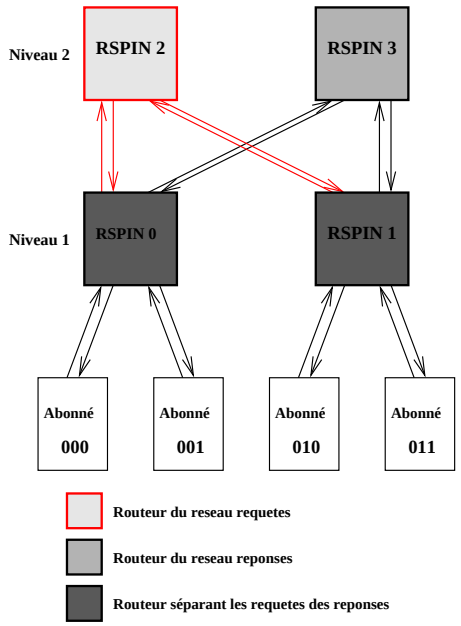

FIG. 7.6 – Topologie séparant les requêtes des réponses avec 4 abonnés

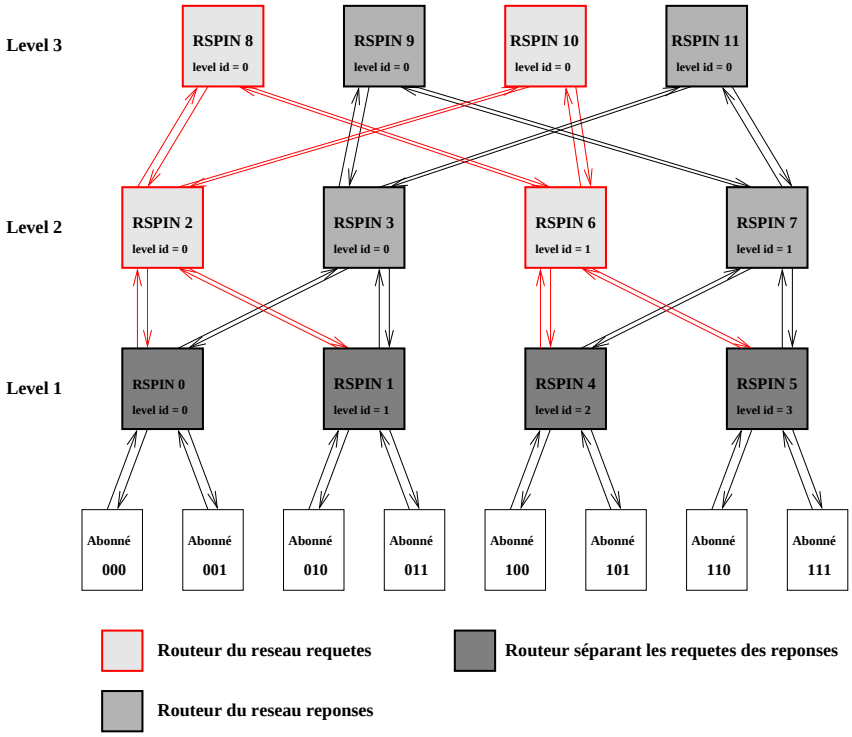

FIG. 7.7 – Topologie séparant les requêtes des réponses avec 8 abonnés

7.6 Conclusion

Nous avons présenté des cas concrets d'inter-blocages et nous avons vérifié que la séparation des requêtes et des réponses permettait de les supprimer, pour une architecture simplifiée et pour une taille de paquet fixée. Nous regrettons de ne pouvoir fournir une démonstration générale prouvant la validité de l'hypothèse pour toutes les configurations possibles d'architecture, de placement des abonnés, et des tailles de paquet. C'est d'ailleurs une des limites de la preuve formelle : du fait des risques d'explosion combinatoire, les systèmes doivent être les plus petits et les plus simples possibles.
De cette étude découlent les règles fondamentales de construction des réseaux SPIN/VCI que nous verrons au chapitre suivant sur le générateur de réseau SPIN/VCI.

Chapitre 8

Génération et configuration automatique d'un réseau SPIN

Sommaire

Ce chapitre donne une vision générale de l'architecture des réseaux SPIN. Afin de faciliter la création de ces réseaux, nous avons automatisé leur construction et leur paramétrage. Nous présentons la topologie des réseaux de 4 à 256 ports ainsi que l'algorithme de génération automatique. Dans un but de simplification, seul les réseaux dont le nombre de ports est une puissance 2 ont été étudiés.

8.1 Présentation générale

Dans un premier temps nous considérons les réseaux SPIN sans les wrappers VCI/SPIN, l'objectif étant de dégager une architecture générale des réseaux SPIN, c'est-à-dire la manière dont les routeurs sont interconnectés. Nous parlerons ensuite de l'ajout des wrappers dans la dernière section consacrée au générateur de réseaux SPIN/VCI.
Il existe plusieurs façons d'interconnecter les routeurs, mais toutes ne sont pas pertinentes. La figure 8.1 représente 3 topologies de réseaux SPIN à 32 ports. Parmi ces 3 conformations, seule la troisième est retenue. Les raisons d'un tel choix ont été détaillées dans le chapitre précédent concernant les inter-blocages. L'idée sous-jacente est de partitionner le réseau en sous-réseaux totalement disjoints au-delà des routeurs de premier niveau.
Cette idée, assez intuitive au départ sur de petits réseaux, se révèle moins triviale sur des structures d'interconnexion à 64, 128 ports ou 256 ports.

Un réseau à 2^{2i} ports (i=1, 2, 3, 4) est un arbre quaternaire élargi.
Un réseau à 2^{2i+1} ports (i=1, 2, 3) est réalisé par deux arbres quaternaires élargis tête bêche.

Les Figures 8.2 à 8.8 représentent les topologies pour des réseaux de 4 à 256 ports. Jusqu'au réseau à 32 ports nous avons explicité toutes les connexions entres les routeurs. A partir d'un réseau à 64 ports, le degré de complexité devient tel que seuls les connexions les plus parlantes ont été tracées. Les paragraphes suivants complètent le tableau 3.1 (sur le nombre de composants des réseaux SPIN) en donnant les formules permettant de calculer le nombre de routeurs et le nombre de liens.

Calcul du nombre de routeurs d'un réseau SPIN

Soit Nb_{Ports} le nombre de ports du réseau.

Le nombre d'étages Nb_{Etages} du réseau est : $Nb_{Etages} = E(\frac{\log_2(Nb_{Ports})}{2})$.
Avec E représentant la partie entière.

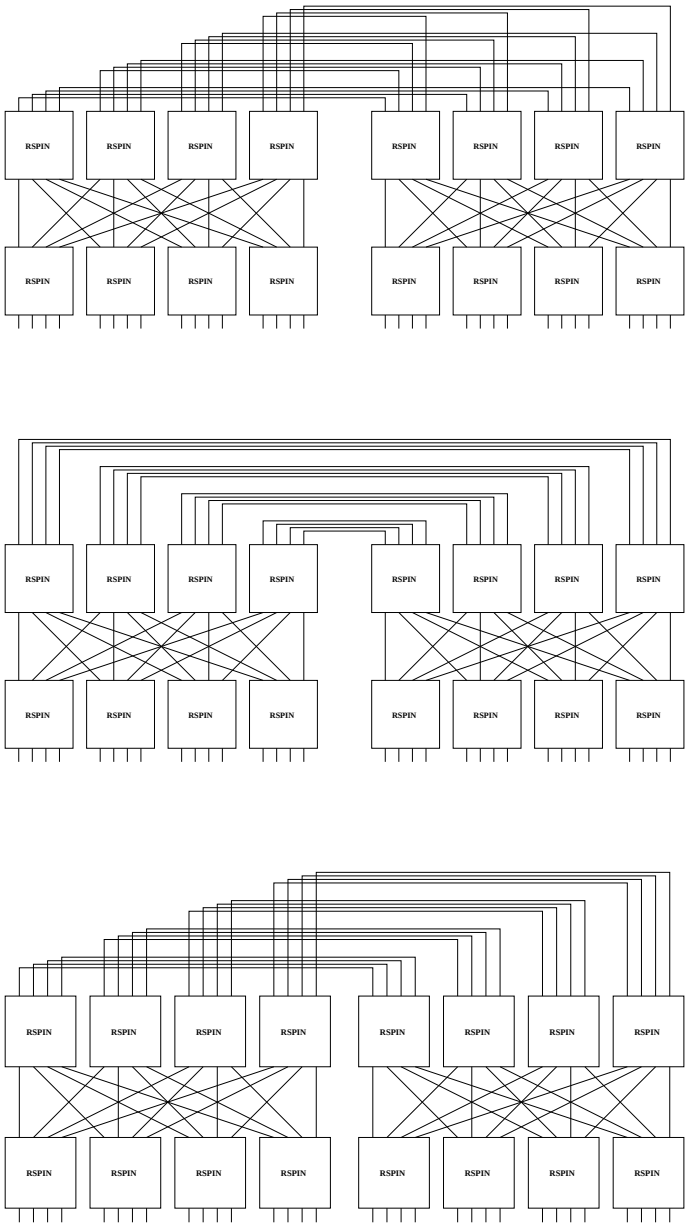

FIG. 8.1 – Réseau SPIN à 32 ports

Chaque étage possède exactement le même nombre de routeurs, donc le nombre de routeurs du 1er étage sera implicitement celui de chaque étage.
Le nombre de routeurs $Nb_{Routeurs_{1er\_etage}}$ du 1er étage est : $Nb_{Routeurs_{1er\_etage}} = \frac{Nb_{Ports}}{4}$.

Le nombre $Nb_{Routeurs}$ de routeurs du réseau est : $Nb_{Routeurs} = Nb_{Etages} * Nb_{Routeur_{1er\_etage}}$

Soit directement en fonction du nombre de ports :

$$Nb_{Routeurs} = E(\frac{\log_2(Nb_{Ports})}{2}) * \frac{Nb_{Ports}}{4}$$

Calcul du nombre de liens bidirectionnels d'un réseau SPIN

Pour le calcul du nombre de liens, posons $n = \log_2(Nb_{Ports})$.

- Si n est pair alors :

$$Nb_{Liens} = Nb_{Etages} * Nb_{Ports}$$

- Si n est impair alors :

$$Nb_{Liens} = Nb_{Etages} * Nb_{Ports} + \frac{Nb_{Ports}}{2}$$

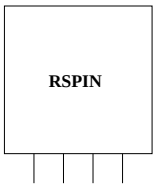

FIG. 8.2 – Réseau SPIN à 4 ports

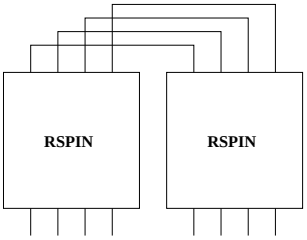

FIG. 8.3 – Réseau SPIN à 8 ports

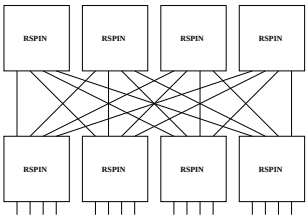

FIG. 8.4 – Réseau SPIN à 16 ports

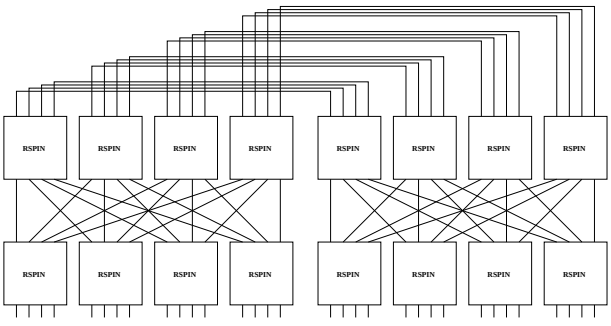

FIG. 8.5 – Réseau SPIN à 32 ports

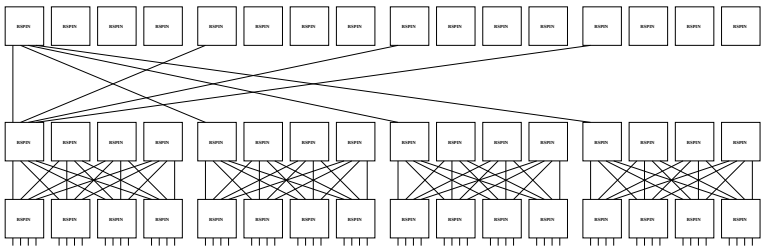

FIG. 8.6 – Réseau SPIN à 64 ports

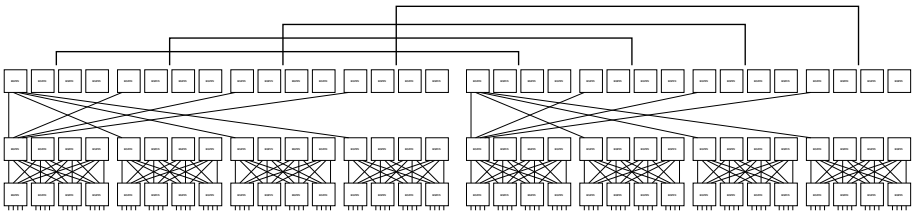

FIG. 8.7 – Réseau SPIN à 128 ports

8.2 Le générateur de réseau SPIN/VCI

8.2.1 Objectifs

L'architecture des réseaux SPIN à 2^n ports étant définie, nous nous intéressons maintenant à l'architecture des réseaux SPIN/VCI, c'est-à-dire aux réseaux SPIN dont les ports sont connectés à des wrappers VCI/SPIN initiateur ou cible.
Tout d'abord, l'étude précédente se traduit concrètement par un composant SystemC, le `SGMN`, pour *SPIN Generic Micro Network* (Cf Figure 8.9). Ce composant instancie les modèles des routeurs RSPIN, et les connecte suivant l'algorithme défini précédemment. Puis, au composant `SGMN`, nous connectons des composants wrappers, initiateur ou cible, le tout étant instancié dans un composant appelé `VHSN` (*VCI Hierarchical SPIN Network*). Ce dernier peut être identifié à un `générateur de réseau SPIN/VCI` (Cf Figure 8.9).

8.2.2 Le paramétrage

Afin de répondre aux exigences du concepteur/assembleur du SoC, le modèle du VHSN est paramétrable. Les paramètres positionnés à ce niveau seront répercutés sur les composants instanciés, c'est-à-dire le SGMN et les wrappers. Les principales informations transmises au VHSN concernent le nombre de ports du réseau et l'emplacement des wrappers initiateur et cible.

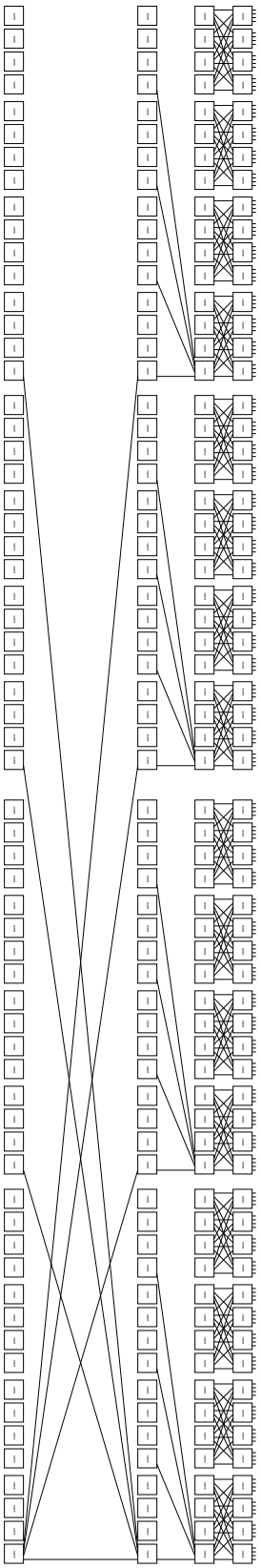

FIG. 8.8 – Réseau SPIN à 256 ports

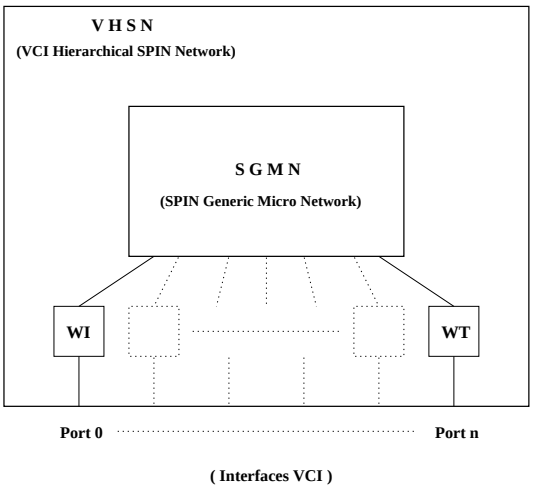

FIG. 8.9 – Structure du réseau SPIN générique à interfaces VCI (VHSN)

Le routeur lui-même possède 4 paramètres. Le paramètre `SEPARATE_PATH` est un booléen qui indique que le routeur doit traiter différemment les paquets requêtes et les paquets réponses.
Le paramètre `NB_SSR` définit le nombre de sous-réseaux requêtes et implicitement de sous-réseaux réponses. En se reportant à la figure 7.5 concernant la topologie pour un réseau SPIN/VCI à 32 ports, on observe deux sous-réseaux associés aux requêtes et deux sous-réseaux associés aux réponses. Ce nombre de sous-réseaux associés aux requêtes est directement lié au nombre de ports supérieurs attribués aux requêtes sur un routeur de premier niveau.
Selon la nature de l'application et des abonnés, on peut décider d'attribuer aux requêtes 1 ou 3 sous-réseaux, selon qu'il y a peu ou beaucoup d'abonnés initiateurs. Les cas 0 ou 4 sont écartés, car un réseau ne contenant que des cibles ou des initiateurs n'est pas pertinent. De plus, l'algorithme de routage dépendant de la position du routeur dans le réseau, chaque routeur doit connaître sa position. Celle-ci est définie par le paramètre `LEVEL` (compris entre 1 et 4) qui définit la hauteur du routeur dans l'arbre quaternaire et par le paramètre `CLUSTER` (compris entre 0 et 3) qui indique à quel sous-arbre appartient le routeur.

Le paramètre `IN ORDER DELIVERY` est un paramètre des composants wrappers VCI/SPIN. Ce paramètre indique que tous les paquets contiendront le bit qui force les routeurs à supprimer l'adaptativité du routage, c'est-à-dire que chaque port d'entrée est verrouillé sur un unique port de sortie. Dans ce mode, le nombre de sous-réseaux requêtes et le nombre de ports VCI initiateurs sont directement liés. Par exemple, si les trois premiers ports

supérieurs des routeurs de premier niveau sont pour les requêtes, aux trois premiers ports inférieurs doivent être connectés des initiateurs.
Enfin, les derniers paramètres des wrappers VCI/SPIN sont la profondeur des FIFO, le nombre de bits de poids forts à prendre en compte pour le décodage des adresses et le numéro de la cible par défaut.

8.3 Conclusion

Nous avons spécifié et implanté un modèle SystemC générique destiné à la simulation pour un réseau SPIN/VCI complet, instanciant tous les routeurs et wrappers nécessaires pour les 7 réseaux SPIN possibles (de 4 à 256 ports).
Bien que nous ne l'ayons pas fait, il est évidemment possible de s'inspirer de ce modèle structurel générique pour écrire un modèle VHDL structurel générique directement utilisable pour aller vers le silicium.

Chapitre 9

Conclusion

Nous conclurons cette thèse en résumant les principaux résultats obtenus.

Le premier point porte sur une comparaison systématique des performances d'un micro-réseau et d'un bus. En effet, le PI-Bus devient un goulot d'étranglement lorsqu'il y a trop de composants qui y sont connectés et que le fonctionnement de ces composants nécessite des accès réguliers au bus, pour des envois de paquets de tailles diverses. Dans le cas précis d'un système à 32 ports reliant 16 initiateurs et 16 cibles, le réseau SPIN supporte une charge de 30 %, alors que le bus sature pour une charge égale à 6 % de la charge maximale.
Ces résultats, obtenus pour une charge synthétique, sont confirmés par la comparaison des performances obtenues pour une application réelle, où le réseau SPIN devient meilleur à partir de 17 processeurs logiques, ces processeurs imposant au système une charge moyenne d'environ 6,5 %.

Le deuxième point porte sur l'utilisation de la preuve formelle pour analyser les problèmes d'inter-blocages. Nous rappelons qu'un inter-blocage est dû au blocage de paquets réponses par des paquets requêtes et inversement. La preuve formelle a été employée, parce qu'elle permet de valider une propriété en parcourant tous les chemins possibles du graphe d'exécution. La propriété à vérifier dans le cas qui nous intéresse est que tous les initiateurs pourront toujours envoyer un nouveau paquet. Nous avons pu valider la solution retenue pour supprimer l'inter-blocage, consistant à faire circuler des paquets requêtes et des paquets réponses sur des sous-réseaux disjoints (utilisant des ressources matérielles différentes). Cette solution se vérifie sur un réseau SPIN en arbre binaire élargi, connectant 4 abonnés (2 initiateurs et 2 cibles). La taille des paquets envoyés étant arbitrairement fixée à 6 mots.

Le troisième résultat porte sur l'automatisation de la génération et de la configuration du réseau SPIN/VCI. L'objectif était de pouvoir utiliser le modèle SystemC du réseau SPIN comme n'importe quel IP SystemC. Cet objectif est réalisé à travers deux composants SystemC. Le premier est directement visible du concepteur du système. Il prend comme paramètres le nombre de ports VCI souhaité et la nature de chaque port : *initiateur* ou *cible*. Ensuite, en fonction de la nature de chaque port, il instancie des wrappers VCI/SPIN initiateur et cible et connecte ces derniers au deuxième composant SystemC qui fournit un réseau SPIN ayant le nombre de ports souhaité. Ces deux composants sont génériques, dans la mesure où ils permettent de construire des réseaux de n'importe quelle taille avec un nombre quelconque d'abonnés.

Enfin, nous avons développé et synthétisé les modèles VHDL des wrappers VCI/SPIN initiateur et cible, afin d'obtenir une estimation fiable de la surface et de la fréquence de fonctionnement après routage.

Ce travail a contribué à la création de la bibliothèque de composants SystemC `SOCLIB`. En effet, il a nécessité l'écriture de modèles de simulation SystemC pour différents composants matériels : un MIPS R3000, une RAM multi-segments, un contrôleur de sémaphores, un écran d'affichage et les wrappers VCI/SPIN initiateur et cible. L'ensemble des modèles de simulations SystemC sont aujourd'hui disponibles dans le domaine public sous licence LGPL.

Annexe A

Différents environnements de simulation

Sommaire

Différents environnements de tests on été présentés. Nous avons voulu retracer en quelque sorte l'historique des simulations CASS et SystemC, sans pour autant faire un inventaire exhaustif. Les systèmes, d'une manière générale, vont par ordre croissant de complexité.

A.1 Les simulations CASS autour d'un PI-Bus

A.1.1 L'environnement PI-Bus de base

Cet environnement a été l'environnement de départ pour la validation des différents composants VCI. Les résultats de simulation obtenus avec lui servaient de référence.

Il se compose des éléments suivants [1] :

- une unité de contrôle du bus (BCU) ;
- un processeur MIPS R3000 à caches internes doté d'interfaces PI-Bus ;
- 5 RAM : une RAM d'instructions, une RAM d'exception, une RAM de reset, une RAM de données cachable et une RAM de données non cachable ;
- un écran d'affichage (TTY).

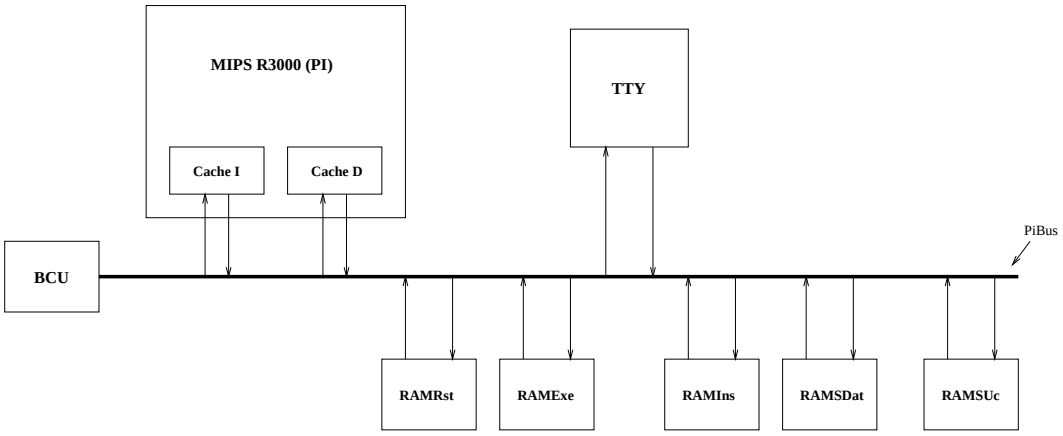

FIG. A.1 – Environnement PI-Bus de référence

A.1.2 L'environnement contenant un MIPS à caches VCI

Dans cet environnement issu du précédent, on a validé un modèle de processeur MIPS R3000 à caches externes. Ces caches d'instructions et de données sont dotés d'interfaces

[1] Cf Figure A.1 : Environnement PI-Bus de référence

VCI [2]. Pour effectuer la connexion au PI-Bus, on a rajouté des wrappers VCI/PI maîtres.

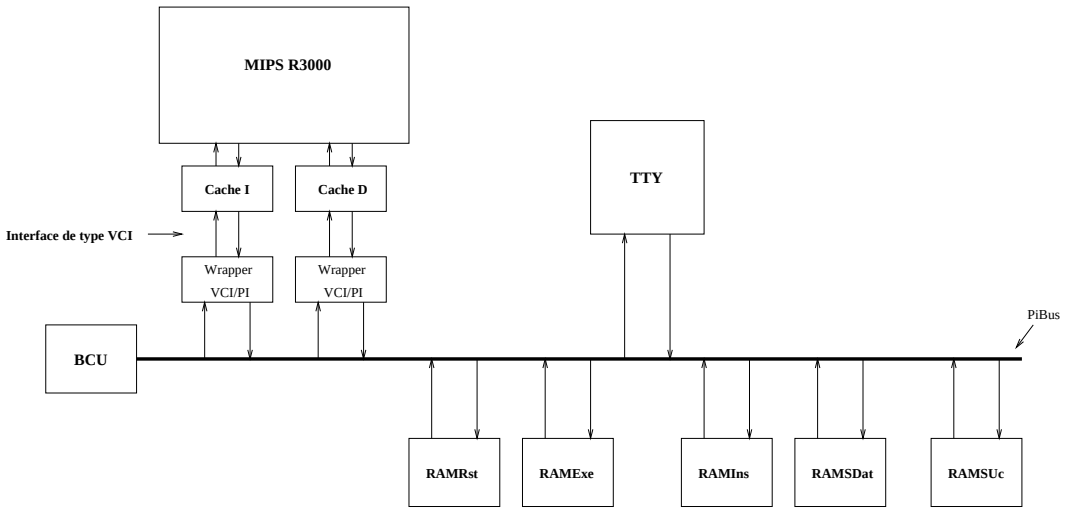

FIG. A.2 – Environnement PI-Bus avec MIPS R3000 à caches VCI externes

A.1.3 Premier environnement de mise au point des wrappers VCI/SPIN

Cet environnement a permis de développer les fonctionnalités de base des wrappers. Par rapport à l'environnement précédent, on a rajouté deux wrappers initiateur et deux wrappers cible connectés l'un à l'autre [3]. La figure A.4 représente un zoom sur la zone qui nous intéresse plus particulièrement, c'est-à-dire celle contenant les wrappers VCI/SPIN.

Les étapes intermédiaires

Ces deux étapes ont été primordiales dans la mise au points des wrappers [4]. En effet il était nécessaire, dans un premier temps, de ne faire varier qu'un petit nombre de paramètres. Par exemple, en ne connectant les wrappers qu'au cache d'instructions, nous avons validé les requêtes de lectures et les réponses associées. Ensuite, avec le cache de données, nous avons validé les requêtes et réponses d'écriture sur des mots, des demi-mots et des octets.

[2] Cf Figure A.2 : Environnement PI-Bus avec MIPS R3000 à caches VCI externes
[3] Cf Figure A.3 : Environnement PI-Bus de validation des wrappers
[4] Cf Figure A.5 : Etapes intermédiaires de validation des wrappers

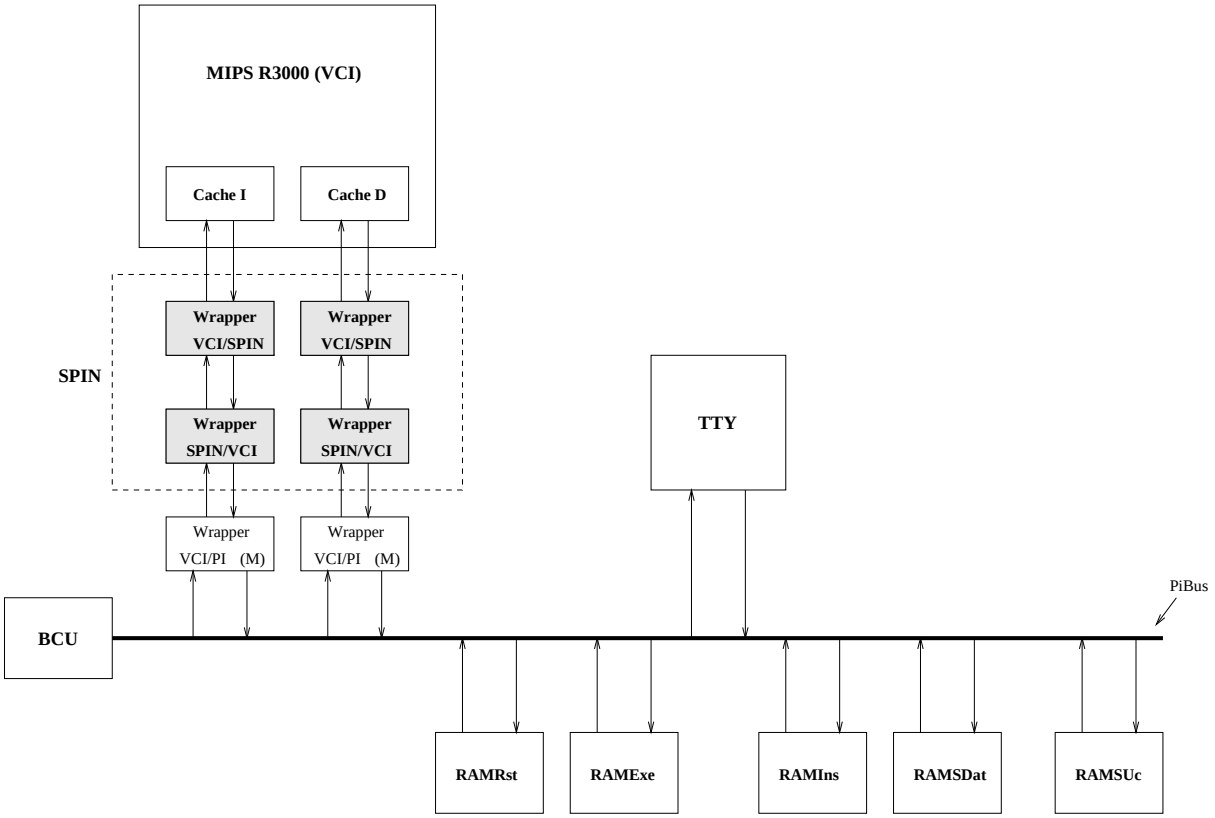

FIG. A.3 – Environnement PI-Bus de validation des wrappers

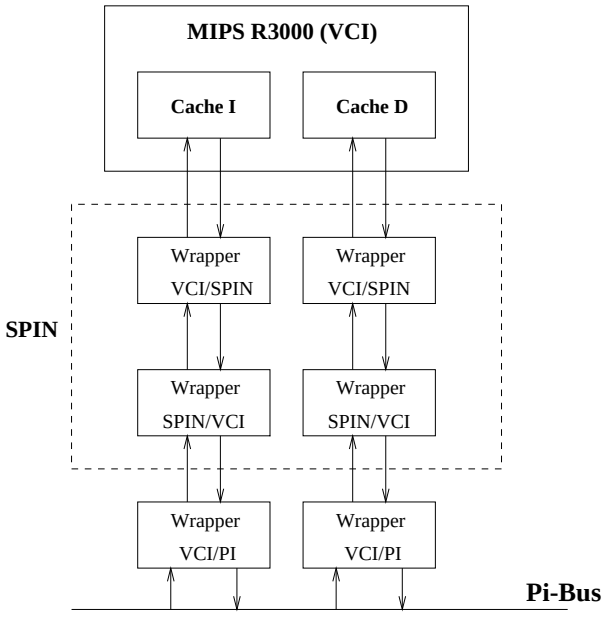

FIG. A.4 – Zoom sur la zone des wrappers

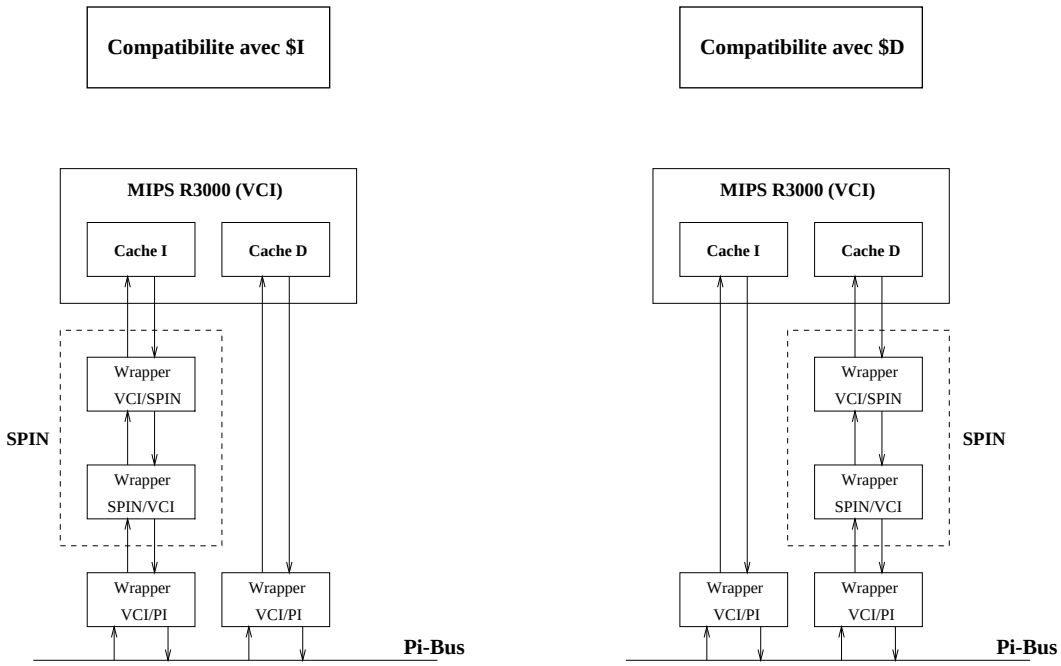

FIG. A.5 – Etapes intermédiaires de validation des wrappers

A.1.4 Environnement de validation de la RAM VCI

Enfin, la validation des RAM à interfaces VCI (Cf Figure A.6). Naturellement, le remplacement des RAM PI s'est fait progressivement de façon à pouvoir déceler aisément les erreurs de conception.

A.1.5 Environnement contenant deux PI-Bus

Cette environnement est intéressant dans la mesure où il met en oeuvre la connexion de deux PI-Bus à l'aide de wrappers jouant le rôle de `bridge` (Cf Figure A.7). Il y a en filigrane l'idée d'utilisation de techniques de `clustering` pour l'assemblage des composants.

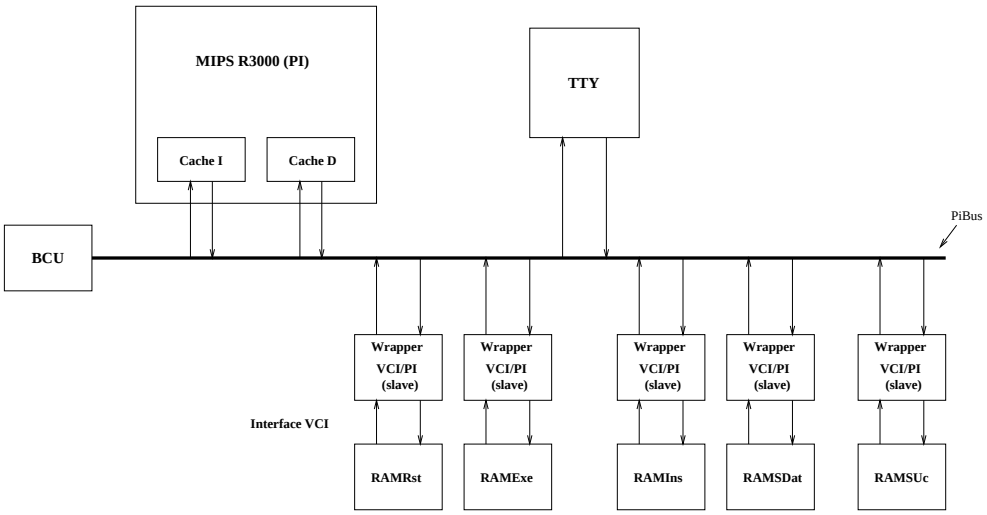

FIG. A.6 – Environnement de test de la RAM VCI simple

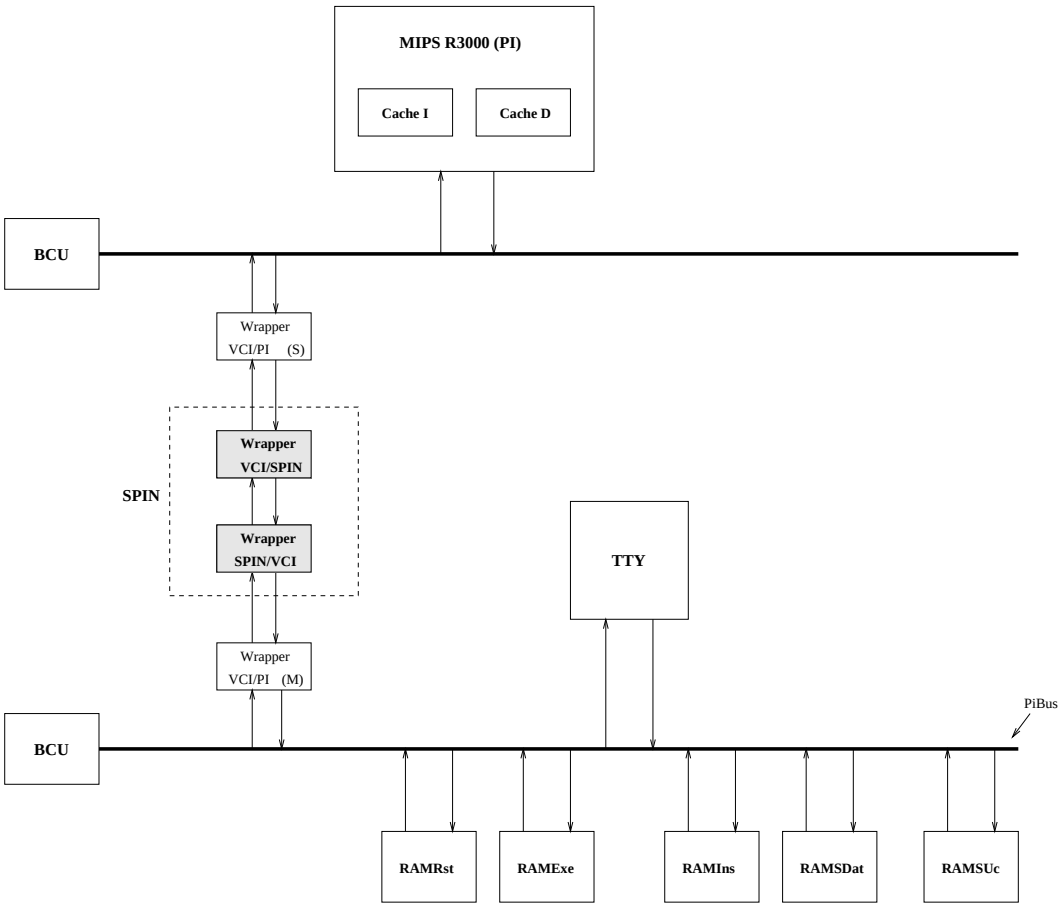

FIG. A.7 – Environnement avec 2 PI-Bus et un MIPS R3000 à caches PI internes

A.2 Les simulations CASS autour du réseau SPIN

Toutes les simulations qui suivent se sont effectuées dans des environnements ne contenant pas de PI-Bus, notre objectif étant de nous rendre le plus rapidement possible indépendant de ce système à bus.

A.2.1 Environnement simplissime

C'est l'environnement de test le plus simple que l'on a réalisé. Il se compose d'un MIPS R3000 doté de caches VCI externes et de deux RAM multi-segments. Il a été utilisé pour mettre au point les RAM multi-segments.

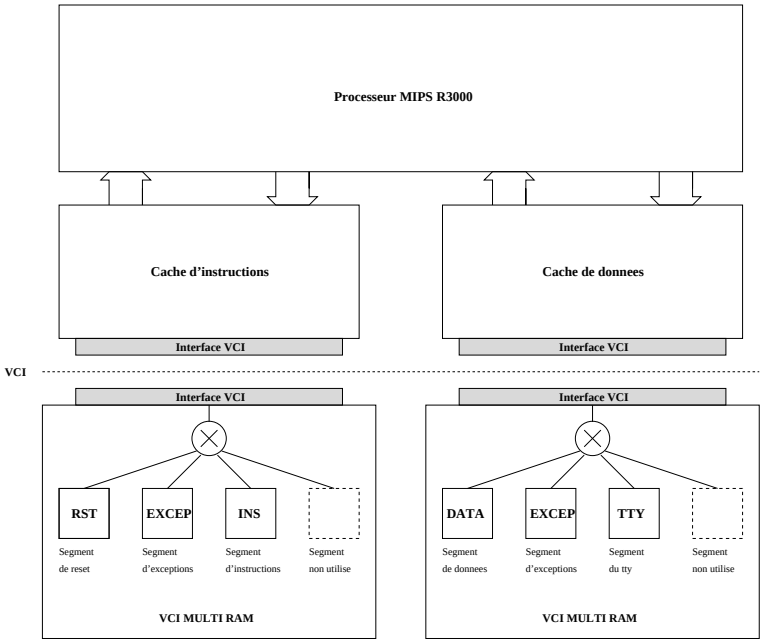

FIG. A.8 – Environnement de test des RAMs multiples sans SPIN

A.2.2 Premier environnement totalement VCI/SPIN

La Figure A.9 présente le premier environnement instanciant des wrappers VCI/SPIN et ne contenant pas de PI-Bus. On part de l'environnement précédent et on insère entre les caches et les RAM des wrappers initiateur et cible directement connectés.

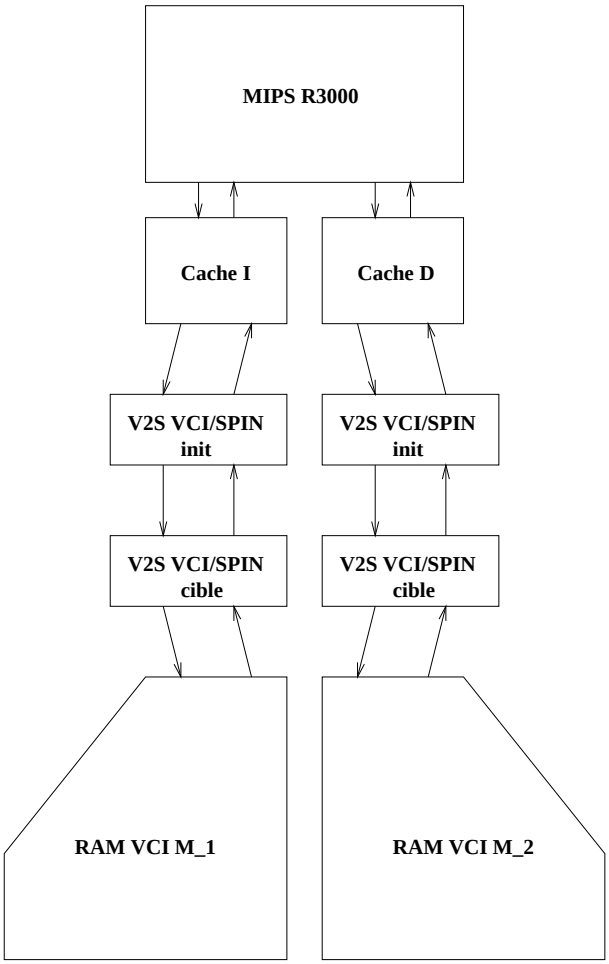

FIG. A.9 – Environnement SPIN de test des RAMs multiples

A.2.3 Environnement de tests multi-processeurs

L'environnement présenté à la Figure A.10 se compose de 44 composants répartis comme suit :

- 8 routeurs RSPIN ;
- 16 wrappers initiateurs et cibles (respectivement WI et WT) ;
- 4 processeurs de type MIPS R3000 dotés de caches à interfaces VCI ;
- 5 RAM mono-segment ;
- 1 RAM multi-segments ;
- 1 RAM des sémaphores ;
- 1 fenêtre d'affichage (TTY).

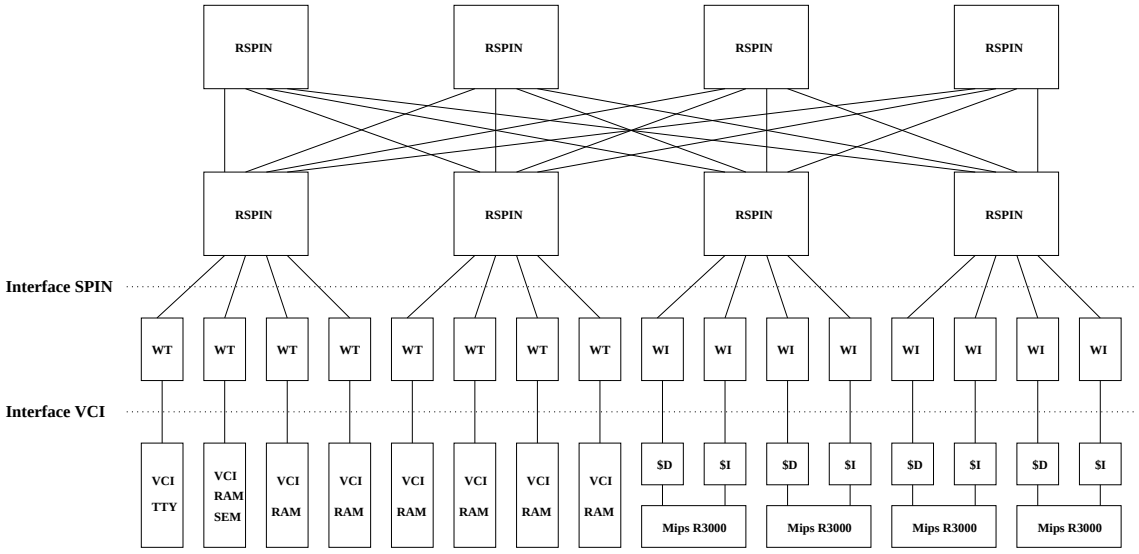

FIG. A.10 – Environnement de tests multi-processeurs

A.3 Résumé

Différents environnements de simulations ont été utilisés pour la mise au point et la validation des composants. Par environnement de simulation on entend à la fois :

- la structure d'interconnexion du système,
- les composants connectés,
- le système d'exploitation,
- le programme exécuté.

Chaque environnement a été élaboré de manière à ce qu'il puisse mettre en évidence le résultat voulu tout en gardant une certaine simplicité.

Annexe B

Architecture détaillée des wrappers

Sommaire

Cette annexe contient la description détaillée de l'architecture des wrappers VCI/SPIN. C'est en quelque sorte la suite de la spécification des wrappers commencée au chapitre 5. Nous y présentons notamment les automates et la combinatoire.

B.1 Le wrapper initiateur

B.1.1 L'automate VCI2SPIN

Description des états :

- `HEADER` : Elaboration de l'en-tête du paquet ;
- `READ` : Envoi d'un mot d'adresse dans le cas d'une lecture ;
- `WAD` : Envoi d'un mot d'adresse dans le cas d'une écriture ;
- `WDT` : Envoi d'un mot de donnée dans le cas d'une écriture.

Signification des sorties :

- `GET` : ordre de consommation sur l'interface VCI (validé par WOK) ;
- `WSET` : ordre d'enregistrement de la requête (validé par WOK et VAL) ;
- `SEL` : commande du multiplexeur trois entrées définissant la valeur écrite dans la FIFO.

B.1.2 L'automate SPIN2VCI

Description des états :

- `HEADER` : on sauvegarde le header du paquet SPIN dans le tampon ;
- `GO` : on donne l'ordre de transfert d'un mot du paquet (celui-ci n'est effectif que si la FIFO est non vide et si l'initiateur est d'accord pour recevoir).

Signification des sorties :

- `GO` : ordre de traiter une information de la fifo afin de la transmettre à l'initiateur ;
- `WH` : ordre de mémoriser le header du paquet SPIN.

B.1.3 L'automate PENDING

Les états `P` et `NP` indiquent si la requête associée à la bascule est pendante ou pas. La sortie `P` vaut respectivement '1'ou '0' selon les états précédemment cités.

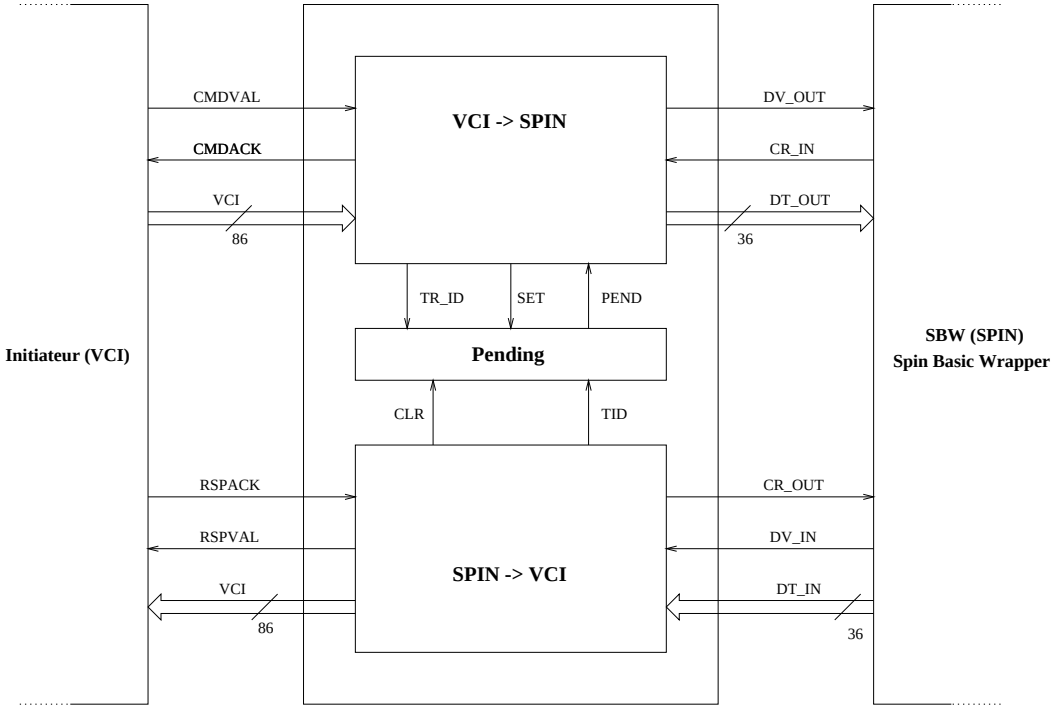

FIG. B.1 – Wrapper VCI/SPIN initiateur

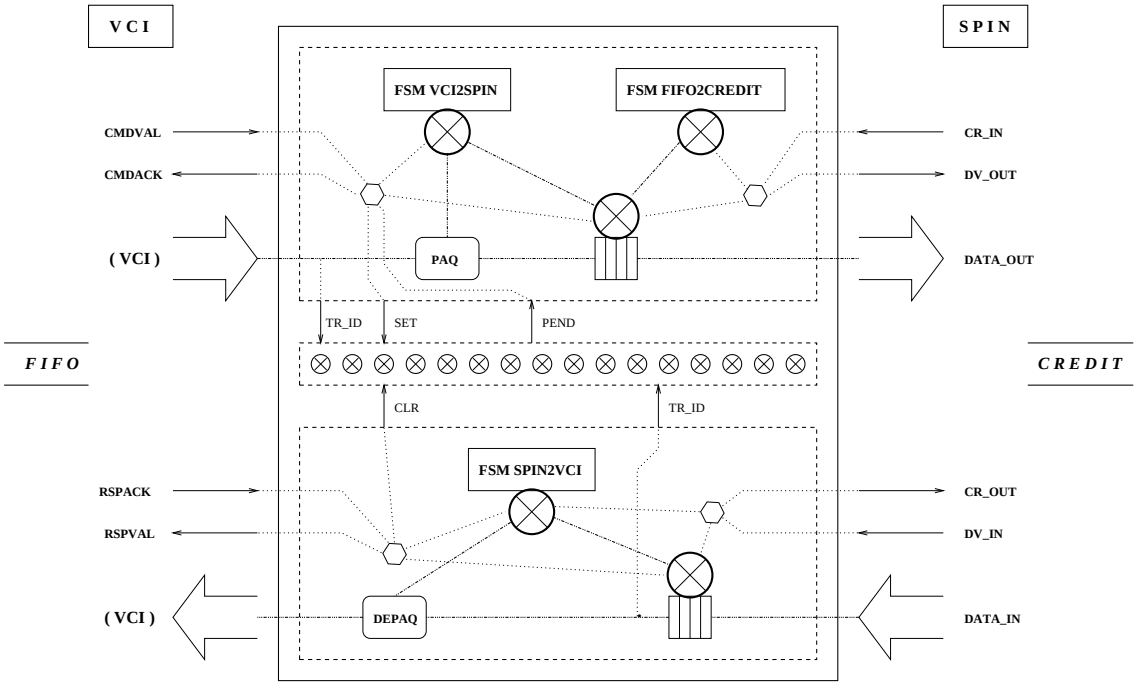

FIG. B.2 – Architecture des automates du wrapper VCI/SPIN initiateur

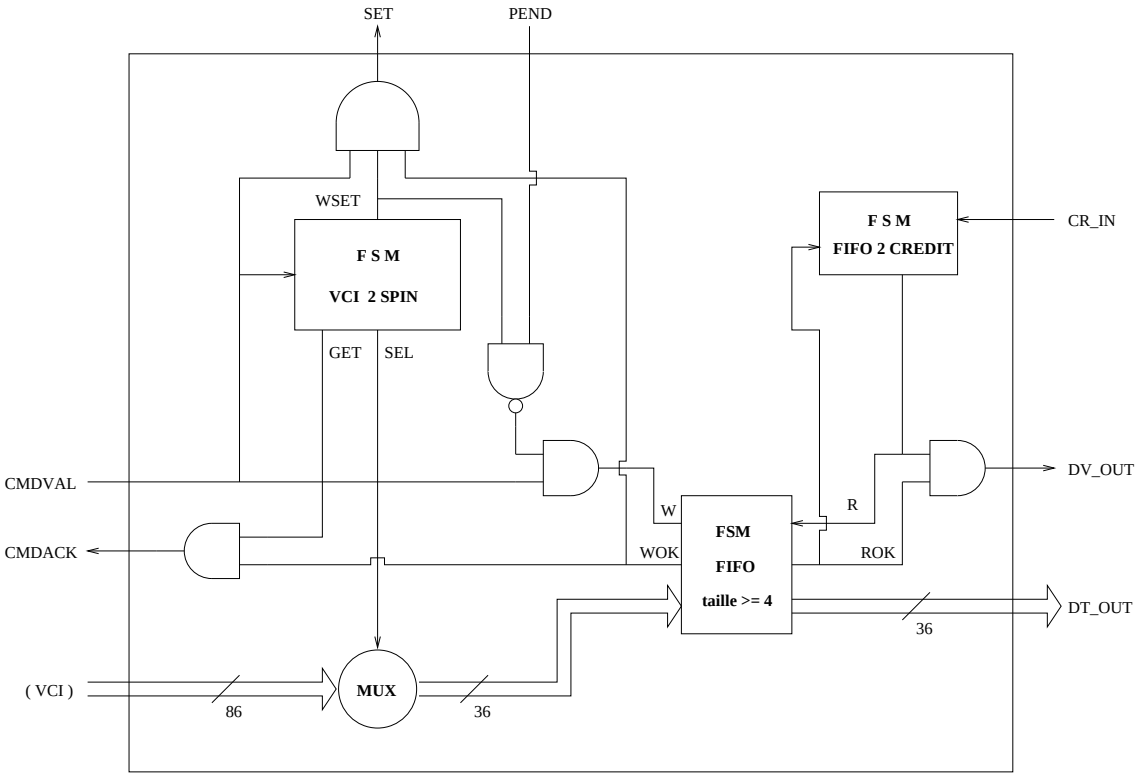

FIG. B.3 – Architecture du module VCI2SPIN du wrapper V2S init

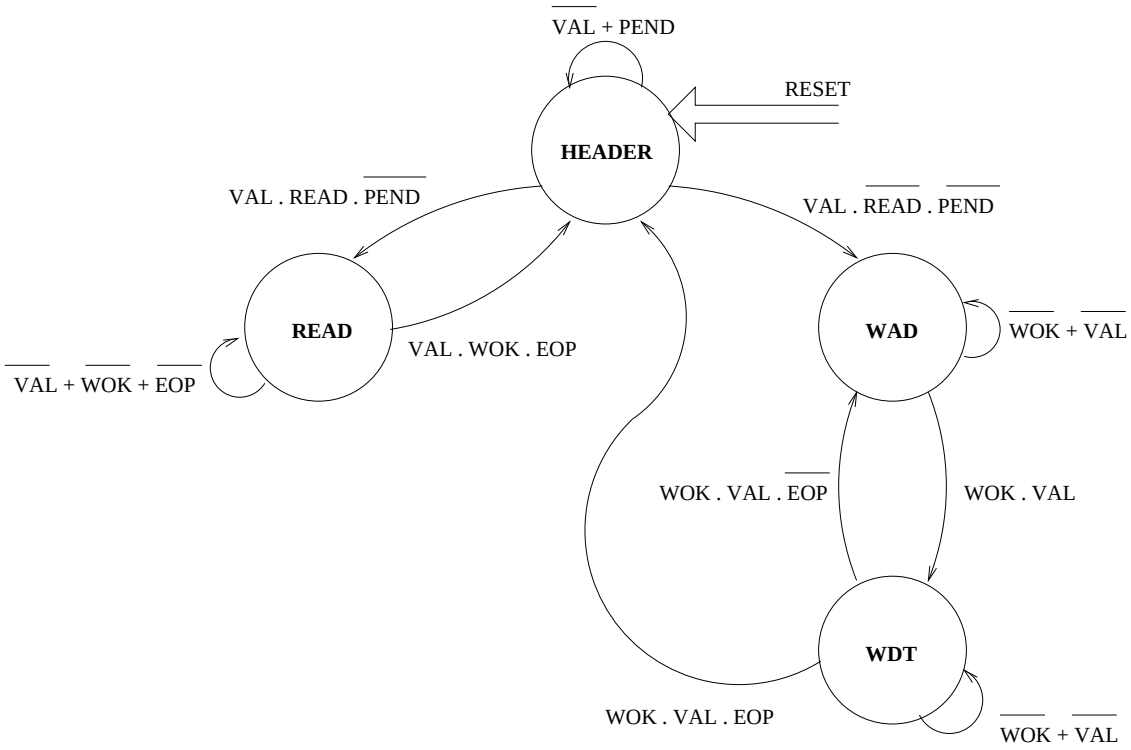

FIG. B.4 – Automate du module VCI2SPIN du wrapper V2S init

Sorties / Etats	GET	WSET	SEL
HEADER	0	1	HD
READ	1	0	ADM
WAD	0	0	ADM
WDT	1	0	DT

FIG. B.5 – Sorties de l'automate VCI2SPIN du wrapper V2S init

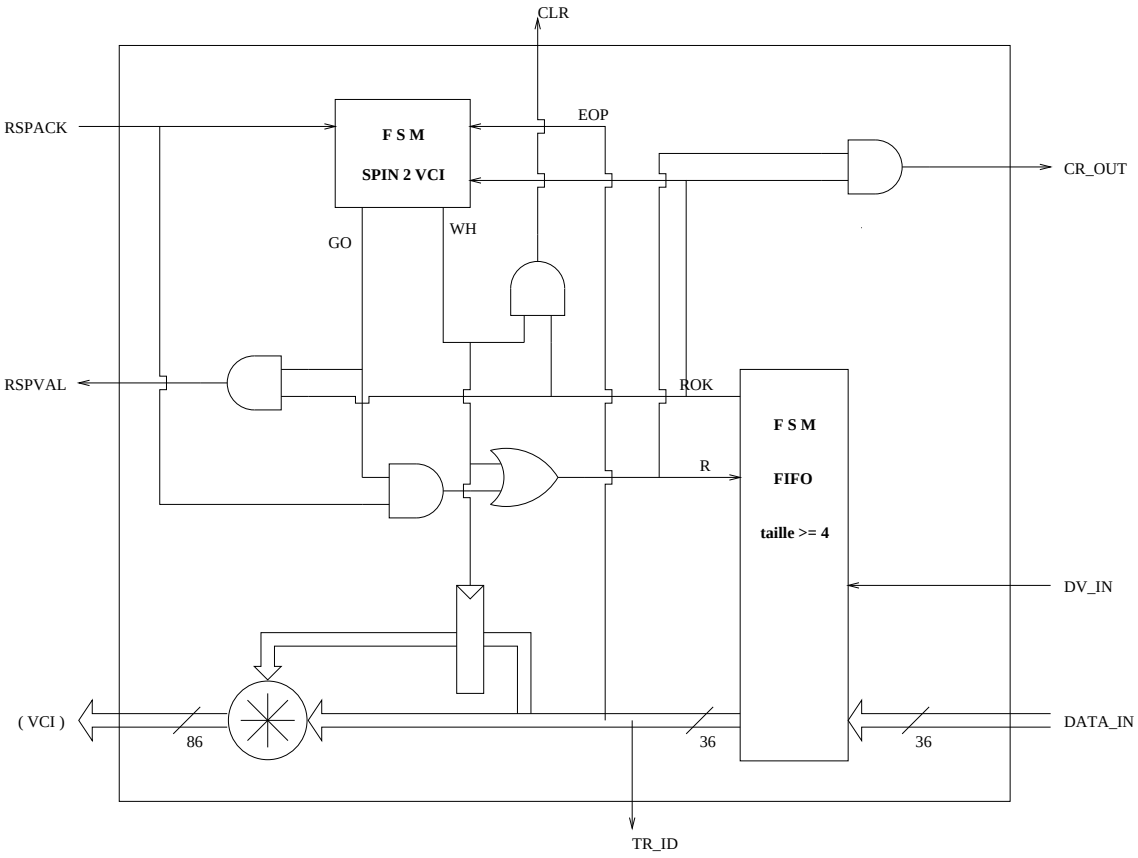

FIG. B.6 – Architecture du module SPIN2VCI du wrapper V2S init

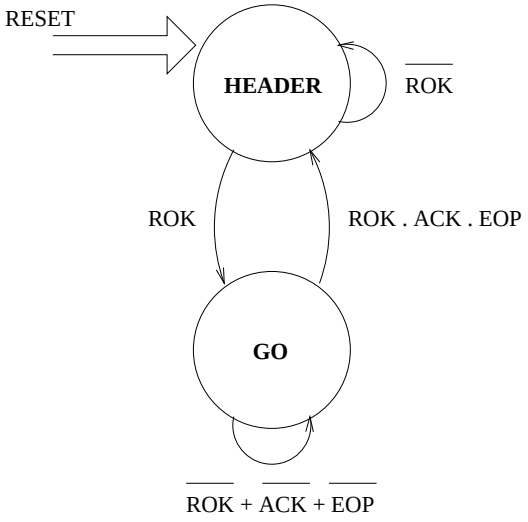

FIG. B.7 – Automate du module SPIN2VCI du wrapper V2S init

Sorties / Etats	GO	WH
HEADER	0	1
GO	1	0

FIG. B.8 – Sorties de l'automate SPIN2VCI du wrapper V2S init

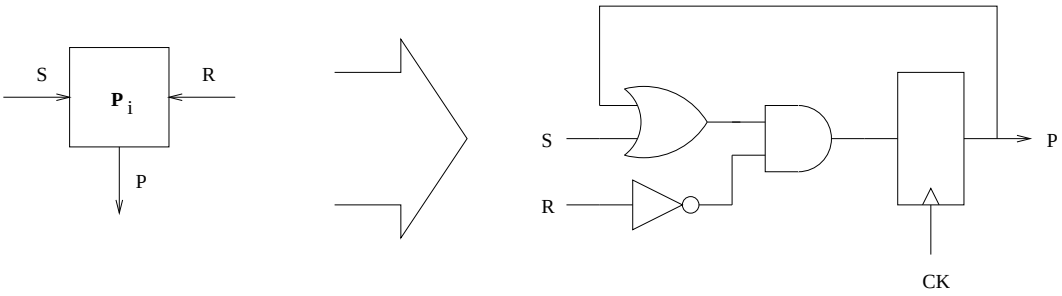

FIG. B.9 – Architecture d'une cellule Pi du module PENDING

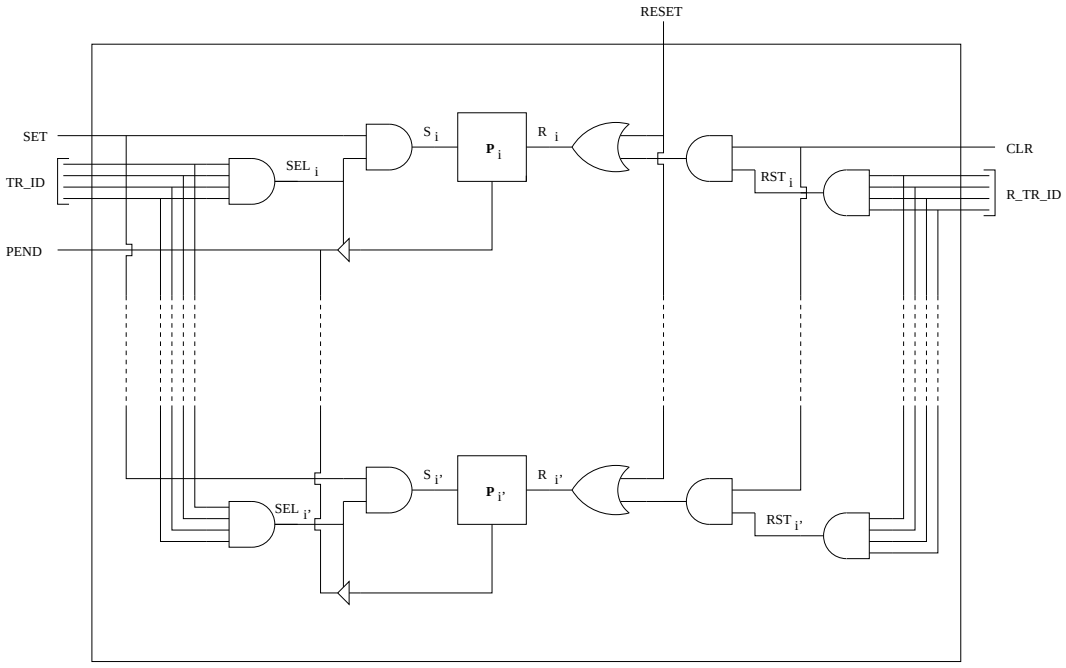

FIG. B.10 – Architecture du module PENDING du wrapper V2S init

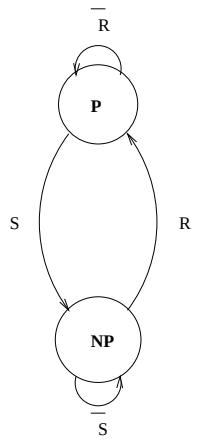

FIG. B.11 – Automate d'une cellule Pi du module PENDING

B.2 Le wrapper cible

B.2.1 L'automate SPIN2VCI

Description des états :

- `CMD_HD` : sauvegarde le header du paquet SPIN ;
- `CMD_WAD` : sauvegarde du mot d'adresse SPIN ;
- `CMD_WDT` : transfert simultané vers la cible du mot d'adresse suvegardé au cycle précédent et du mot de donnée venant de la fifo ;
- `CMD_R` : transfert vers la cible du mot de donnée venant de la fifo.

Signification des sorties :

- `WH` : ordre de sauvegarder le header venant de la fifo ;
- `WA` : ordre de sauvegarder mot d'adresse venant de la fifo ;
- `GO` : ordre de transférer simultanément vers la cible le mot d'adresse suvegardé au cycle précédent et le mot de donnée venant de la fifo.

B.2.2 L'automate VCI2SPIN

Description des états :

- `RSP_HD` : fabrication du header SPIN ;
- `RSP_DT` : traitement d'un mot VCI.

Signification de la sortie :

- `GETDT` : ordre de traiter un mot VCI, et de sauvegarder dans la fifo le mot SPIN résultant.

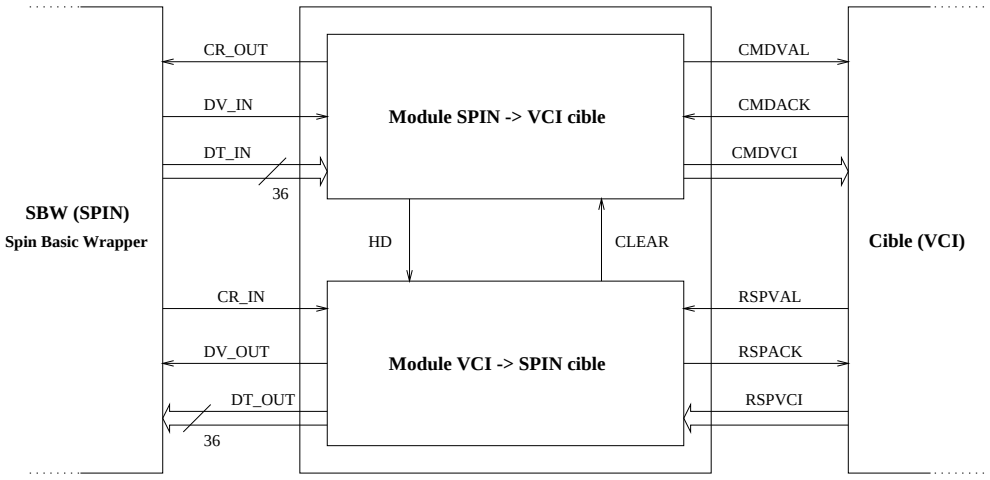

FIG. B.12 – Le wrapper VCI/SPIN cible

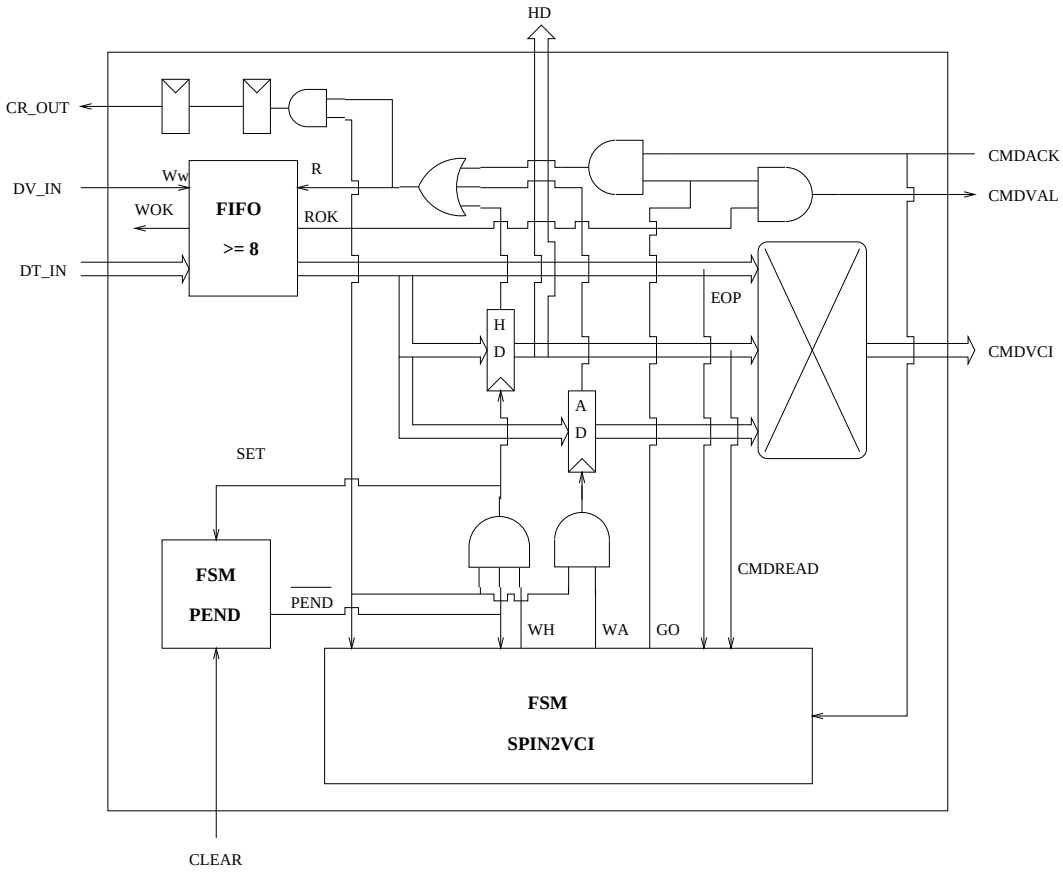

FIG. B.13 – Architecture du module SPIN2VCI du wrapper V2S cible

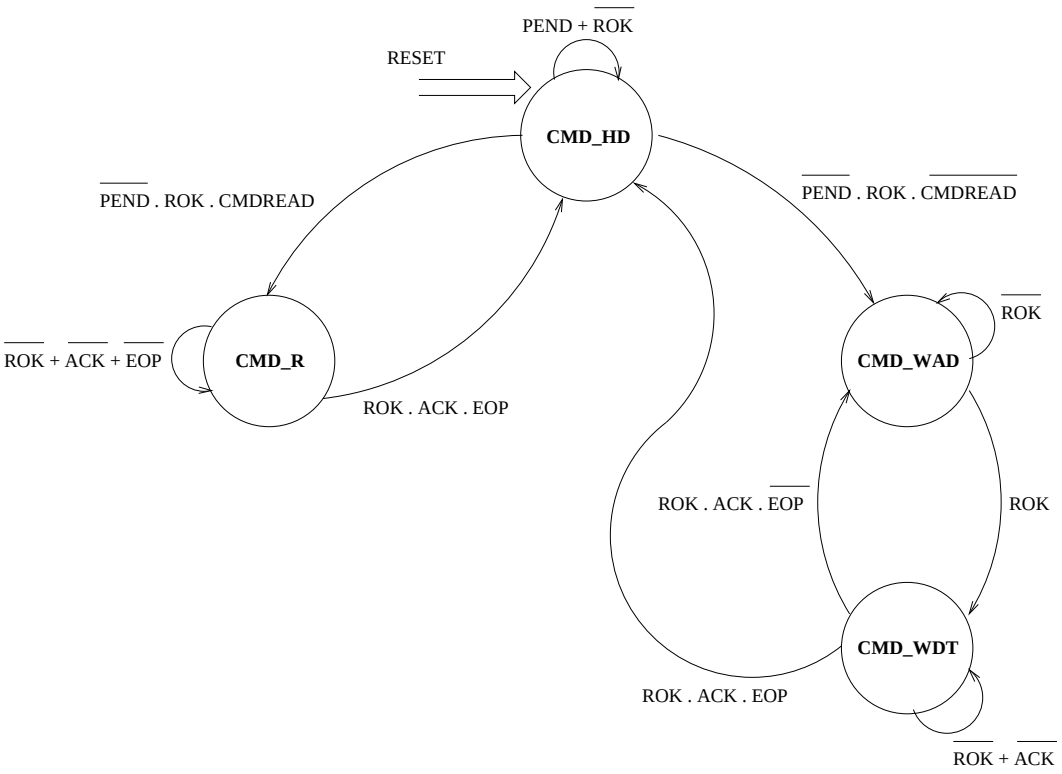

FIG. B.14 – Automate du module SPIN2VCI du wrapper V2S cible

Sorties / Etats	WH	WA	GO
CMD_HD	1	0	0
CMD_WAD	0	1	0
CMD_WDT	0	0	1
CMD_R	0	0	1

FIG. B.15 – Sorties de l'automate SPIN2VCI du wrapper V2S cible

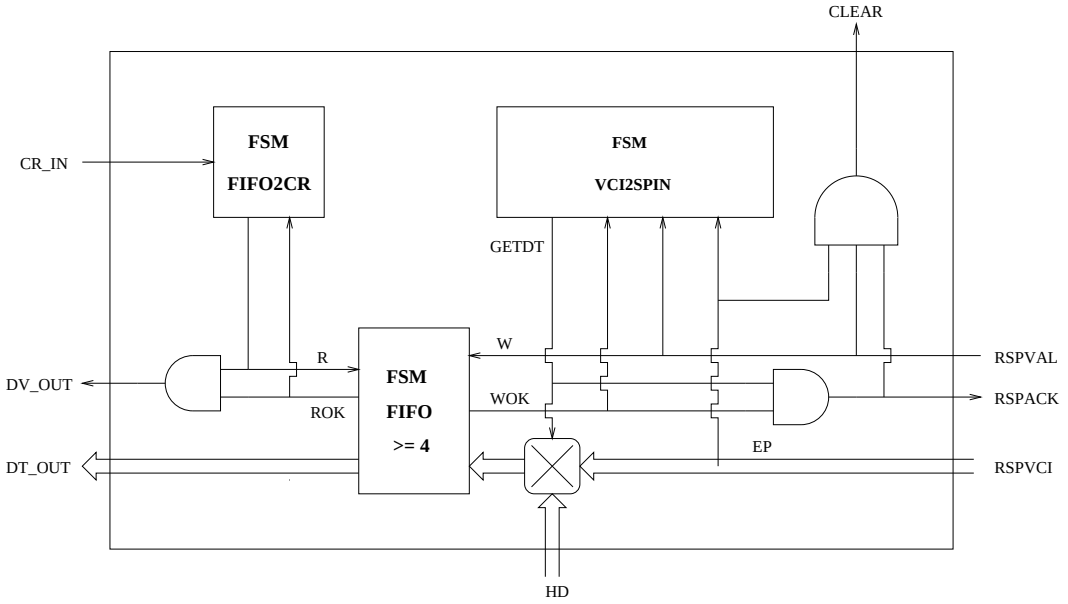

FIG. B.16 – Architecture du module VCI2SPIN du wrapper V2S cible

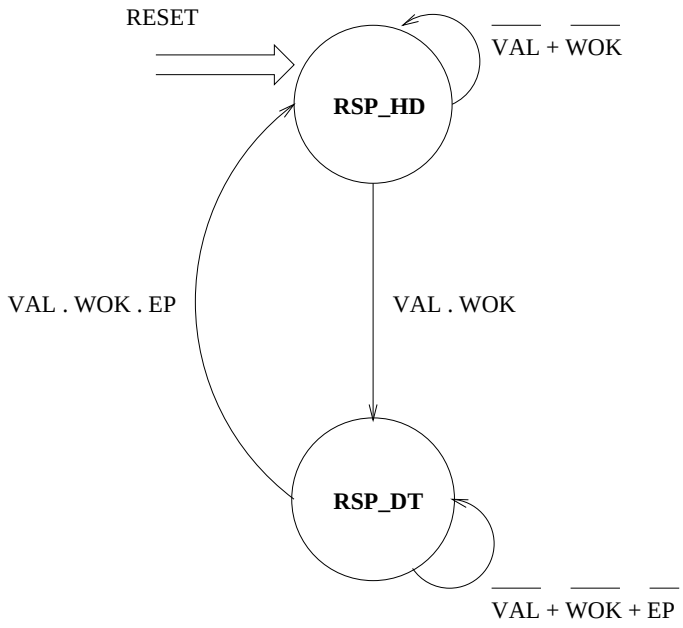

FIG. B.17 – Automate du module VCI2SPIN du wrapper V2S cible

Sortie / Etats	GETDT
RSP_HD	0
RSP_DT	1

FIG. B.18 – Sorties de l'automate VCI2SPIN du wrapper V2S cible

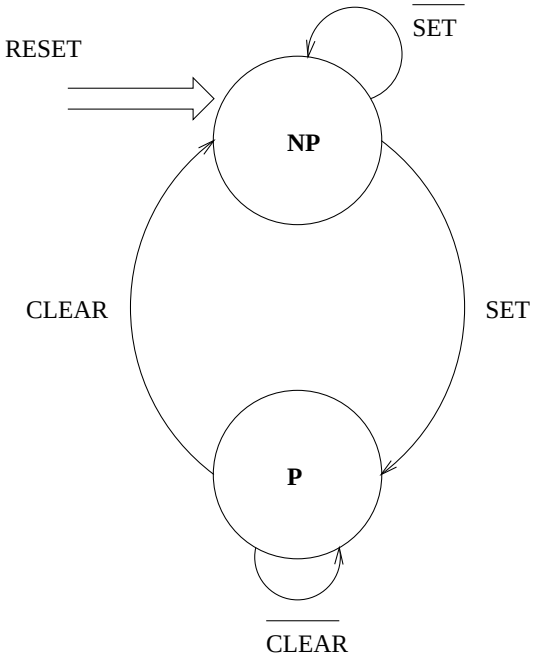

FIG. B.19 – Automate du module PENDING du wrapper V2S cible

B.3 Les contrôles de flux de type FIFO et CREDITS

Nous présentons ici les automates permettant de passer du contrôle de flux de type FIFO au contrôle de flux de type CREDITS et inversement.

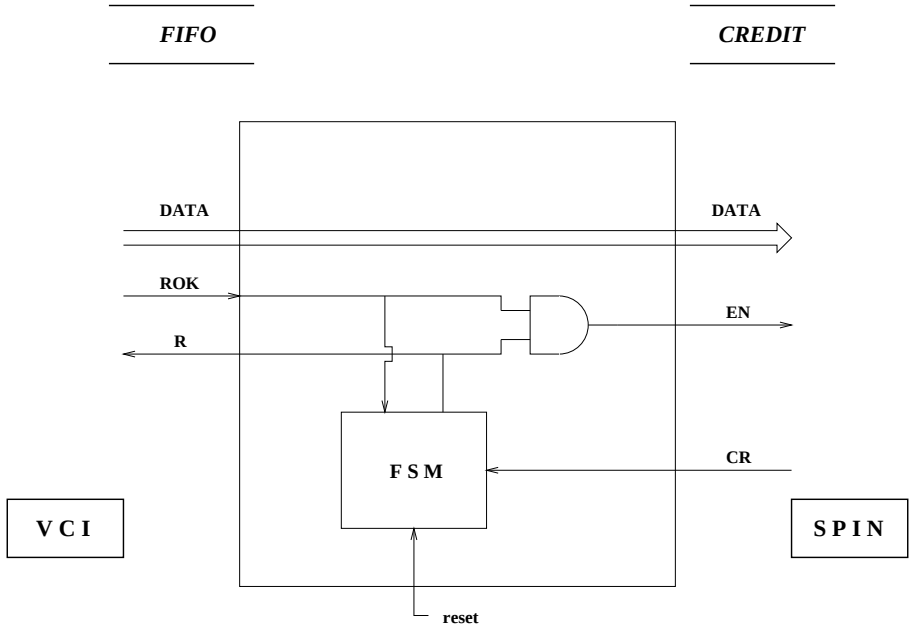

FIG. B.20 – Passage du protocole FIFO au protocole CREDITS

	Sortie
Etats	R
QUATRE	1
TROIS	1
DEUX	1
UN	1
ZERO	0

TAB. B.1 – Sorties de l'automate du bloc FIFO2CREDIT

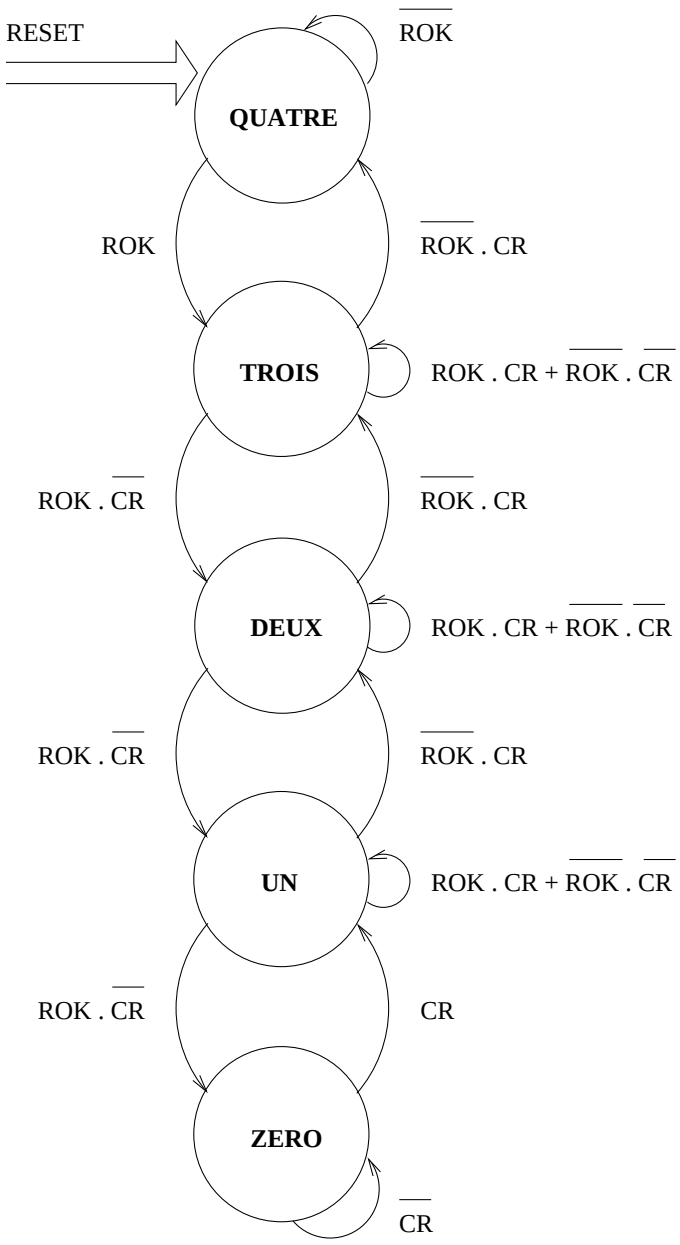

FIG. B.21 – Automate du bloc FIFO2CREDIT

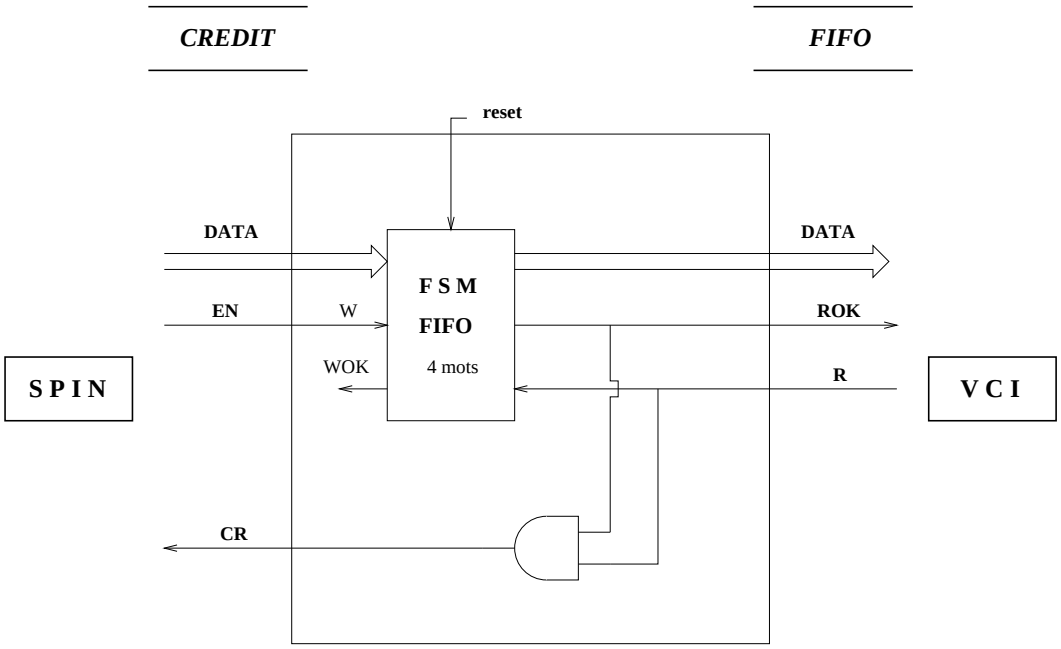

FIG. B.22 – Passage du protocole CREDITS au protocole FIFO

	Sortie	
Etats	WOK	ROK
ZERO	1	0
UN	1	1
DEUX	1	1
TROIS	1	1
QUATRE	0	1

TAB. B.2 – Sorties de l'automate du bloc CREDIT2FIFO

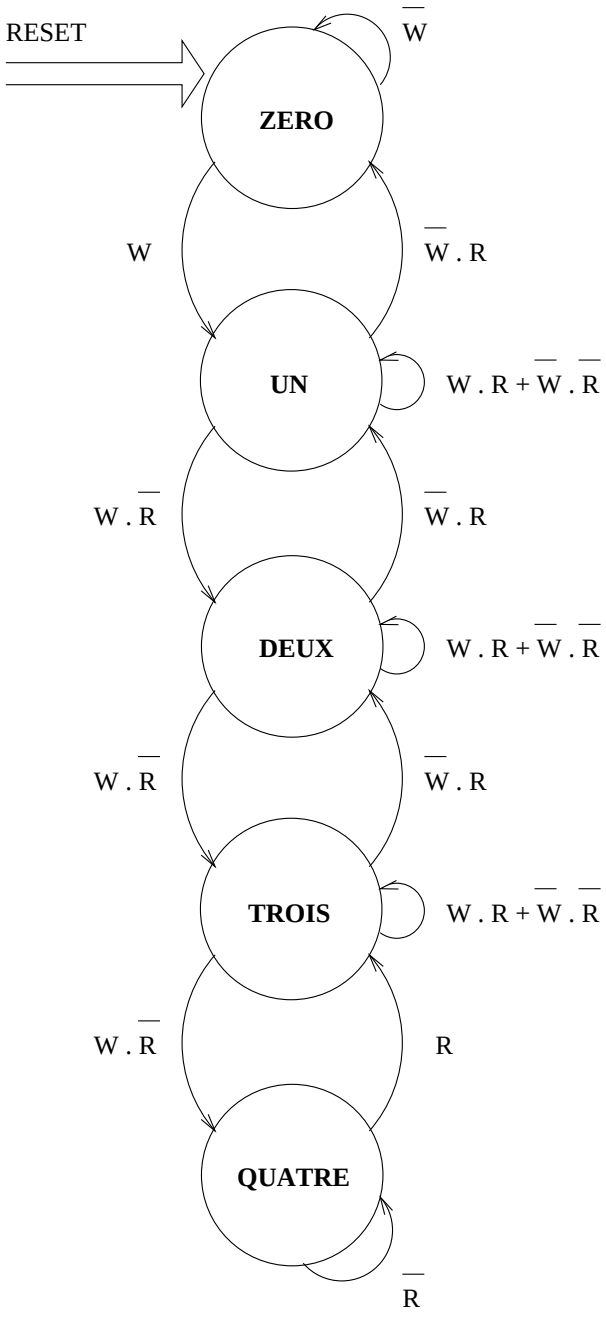

FIG. B.23 – Automate du bloc CREDIT2FIFO (idem automate FIFO)

Publications associées à ce travail

Conférences nationales

Systèmes intégrés : un micro-réseau d'interconnexion à commutation de paquets respectant la norme VCI
Herve Charlery et Alain Greiner, Troisième colloque du GDR CAO de circuits et systèmes intégrés, pp. 75-78, Paris, Mai 2002.

SPIN, un micro-réseau d'interconnexion à commutation de paquets respectant la norme VCI. Concepts généraux et validation.
Herve Charlery, Emmanuelle Encrenaz, Alain Greiner, Adrijean Andriahantenaina et Laurent Mortiez, Symposium en Architecture et Adéquation Algorithme Architecture (SympAAA 2003), pp. 337-344, La Colle sur Loup, Octobre 2003.

Conférences internationales

SPIN : a Scalable, Packet Switched, On-chip Micro-network
Adrijean Andriahantenaina, Herve Charlery, Alain Greiner, Laurent Mortiez et Cesar Albenes Zeferino, Proceeding of the Design Automation and Test in Europe Designers' Forum 2003 (DATE 2003), Munich, March 2003.

Using VCI in a on-chip system around SPIN network
Herve Charlery, Alain Greiner, Emmanuelle Encrenaz, Laurent Mortiez, Adrijean Andriahantenaina, Proceeding of the 11th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES 2004), Szczecin, Poland, June 2004, pp. 571-576.

Physical realization of the VCI wrappers
Herve Charlery, Adrijean Andriahantenaina, Alain Greiner, Proceeding of the ICM, Poster Session (ICM 2004), Tunis, December 2004.

Bibliographie

[Adr03] Adrijean Adriahantenaina and Herve Charlery and Alain Greiner and Laurent Mortiez and Cesar Albenes Zeferino. *SPIN : a Scalable, Packet Switched, On-chip Micro-network.* Proceeding of the Design Automation and Test in Europe Designers' Forum 2003 (DATE 2003), Munich, March 2003.

[All] Alliance CAD Tools. http ://www-asim.lip6.fr/recherche/alliance.

[ARM99] ARM Company Ltd. *Advanced Microprocessor Bus Architecture Specification.* http ://www.arm.com/Pro+Peripherals/AMBA/, 1997-99.

[C. 91] C. Germain-Renaud et J. P. Sansonnet. *Les ordinateurs massivement parallèles.* Armand Colin Editeur, Paris, 1991.

[Cha85] Charles Leiserson. *Fat-Trees : Universal Network for Hardware-Efficient Supercomputing.* IEEE Transactions on Computers, vol. C-34, no. 10, pp. 892-901, Octobre 1985.

[Cha92] Charles Leiserson, Richard Feynman, David Hillis et al. *The network architecture of the Connection Machine CM-5.* Thinking Machines Corporation, Cambridge, USA, Juillet 1992.

[D. 92] D. Lenoski, J. Laudon, K. Gharachorloo, W. D. Weber, A. Gupta, J. Hennessy, M. Horowitz et M. S. Lam. *The Stanford Dash Multiprocessor.* IEEE Computer, vol. 25, no. 3, March 1992.

[Den98] Denis Hommais et Frederic Petrot. *Efficient Combinational Loops Handling for Cycle Precise Simulation of System on a Chip.* IEEE Euromicro Conference, pp. 51-54, 1998.

[Fly95] M. J. Flynn. *Computer Architecture, pipeline and parallel processor design.* Jones and Bartlett Publishers, 1995.

[Fré97] Frédéric Pétrot et Hommais Denis. *Cycle-Precise Core Based Hardware/Software System Simulation with Predictable Event Propagation.* IEEE Computer Society Press, Proceedings of the 23rd Euromicro Conference, Budapest, Hongrie, pp. 182-187, Septembre 1997.

[G. 92] G. Kane, J. Heinrich. *Mips RISC Architecture (2nd Edition).* Paperback, 1992.

[Gom02] Gomez Pascal and Pétrot Frédéric and Hommais Denis. *Mutek : un noyau multi-tâches/multi-processeurs SMP pour systèmes embarqués.* Troisième Colloque du GDR CAO de circuits et systèmes intégrés, Paris, France, pp. 107-110, Mai 2002.

[Gor65] Gordon E. Moore. *Gramming more components onto integrated circuits.* Electronics, Volume 38, Number 8, pp. 114-117, April 1965.

[Gue00] P. Guerrier. *Un réseau d'interconnexion pour systèmes intégrés.* Thèse de doctorat, 10 mai 2000.

[Gun93] J. Gunther. *Structure et technologie des ordinateurs.* Collection Informatique, Edition Hermès, 1993.

[Hol97] Gerard J. Holzmann. *The Model Checker Spin.* IEEE transaction on software engineering, vol. 23, n°5, May 1997.

[Hom01] D. Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel.* Thèse de doctorat de l'Université Paris 6, 13 septembre 2001.

[Hug97] Hugo de Man. *Education for the Deep Submicron Age : Business As Usual ?* Proceedings of the 34th Design Automation Conference, Anaheim, USA, March 1997.

[IEE] IEEE P1500 Working Group. *Standard for Embedded Core Test(SECT).* http ://grouper.ieee.org/group/1500.

[J. 97] J. Pellaumail, P. Boyer et P. Leguesdron. *Autoroutes ATM.* Editions Hermès, Paris, 1997.

[J. 99] J. Hennessy, M. Heinrich, et A.Gupta. *Cache-Coherent Distributed Shared Memory : Perspectives on Its Development and Future Challenges.* Proceedings of the IEEE, vol. 83, no. 3, March 1999.

[Jac93] J. L. Jacquemin. *Informatique parallèle et systèmes multiprocesseurs.* Traité des nouvelles technologies (série informatique), Editions Hermès, Paris, 1993.

[Joh96] John Hennessy et David Patterson. *Computer Architecture, A Quantitative Approach - 2nd Edition.* Morgan Kaufmann Publishers, San Francisco, CA, USA, 1996.

[Lav90] I. Lavallée. *Algorithmique parallèle et distribuée.* Traité des nouvelles technologies (série informatique), Editions Hermès, Paris, 1990.

[M. 93] M. Cosnard et D. Trystram. *Algorithmes et architectures parallèles.* InterEdition, Paris, 1993.

[M. 02] M. Benabdenbi. *Conception en vu du test de systèmes intégrés sur silicum (SoC).* Thèse de doctorat de l'Université Paris 6, 2002.

[Mél97] J. L. Mélin. *Pratique des réseaux ATM.* Editions Eyrolles, 1997.

[Ope94] Open Microprocessor Systems Initiative. *PI-Bus Draft Standard Specification.* ftp ://www.omimo.be/ftp/standard/omi324.ps, 1994.

[Pie00] Pierre Guerrier et Alain Greiner. *A Generic Architecture for On-Chip Packet-Switched Interconnections.* Proceeding of the Design Automation and Test in Europe Conference 2000 (DATE 2000), Paris, March 2000.

[Puj99] G. Pujolle. *Les réseaux.* Editions Eyrolles, 1999.

[Syn00] Synopsys, CoWare, Frontier Design. *SystemC - User's guide.* http ://www.systemc.org, 2000.

[Vir97] Virtual Socket Interface Alliance. *Virtual Component Interface Standard - Draft Specification, v. 2.2.0.* http ://www.vsia.com (document access may be limited to members only), August 1997.

[Wil87] William Dally et Charles Seitz. *Deadlock-free Message Routing in Multiprocessor Interconnection Networks.* IEEE Transactions on Computers, vol. C-36, no. 5, pp. 547-553, Mai 1987.