Méthode de Test et Conception en Vue du Test pour les Réseaux sur Puce Asynchrones : Application au Réseau ANOC

Xuan Tu Tran

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité : « Micro et Nano Electronique »

préparée au laboratoire CEA-LETI, MINATEC

dans le cadre de **l'Ecole Doctorale *« Electronique, Electrotechnique, Automatique et Traitement du Signal »***

présentée et soutenue publiquement

par

Xuan-Tu TRAN

le 12 février 2008

Méthode de Test et Conception en Vue du Test pour les Réseaux sur Puce Asynchrones : Application au Réseau ANOC

Directeur de Thèse

Chantal ROBACH

JURY

M.	Christian LANDRAULT	, Président, Rapporteur
M.	Habib MEHREZ	, Rapporteur
Mme.	Chantal ROBACH	, Directeur de thèse
M.	Jean DURUPT	, Co-encadrant
M.	Vincent BEROULLE	, Co-encadrant
M.	Yvain THONNART	, Co-encadrant
M.	Bruno ROUZEYRE	, Examinateur
M.	Mounir BENABDENBI	, Examinateur

Pour mes parents, mon grand frère et ma petite sœur,
ma chérie et notre petite princesse Hà My !

Table des matières

Remerciements

Pour commencer ce manuscrit, je tiens à remercier toutes les personnes qui ont contribué à divers degrés au bon déroulement de ce travail de recherche.

Je remercie tout d'abord Madame Chantal Robach, Professeur à l'Institut National Polytechnique de Grenoble (INPG), Directrice de l'Ecole Supérieure d'Ingénieurs en Systèmes Industriels Avancés Rhône-Alpes (ESISAR), pour ses conseils toujours pertinents en tant que directeur de thèse. Je la remercie du temps qu'elle a pu me consacrer, ainsi que de la confiance et du soutien qu'elle a su m'accorder pour mener à bien ce manuscrit

Je remercie vivement Monsieur Jean Durupt et Monsieur Yvain Thonnart, Ingénieurs Chercheurs au CEA-LETI, Monsieur Vincent Beroulle, Maître de Conférence à l'INPG, pour leurs encadrements, leurs expertises et leurs soutiens pendant les trois années de préparation de cette thèse. Ces années passées ensemble furent toujours l'occasion de discussions formidables sur le plan spirituel aussi bien que sur le plan scientifique.

J'exprime également mes remerciements aux autres membres du jury : Monsieur Christian Landrault, Directeur de recherche au LIRMM (CNRS, Université de Montpellier II), pour m'avoir fait honneur de présider le jury et avoir accepté d'être rapporteur de mon travail ; Monsieur Habib Mehrez, Professeur au LIP6 (Université Pierre Marie Curie), pour avoir accepté d'être rapporteur de mon travail. Je les remercie également pour leurs remarques très constructives.

Je remercie aussi Monsieur Bruno Rouzeyre, Professeur au LIRMM (Université de Montpellier II), et Monsieur Mounir Benabdenbi, Maître de Conférence au LIP6 (Université Pierre Marie Curie) d'avoir participé à mon jury de thèse.

Cette thèse s'est déroulée dans le cadre du service CME (CEA-LETI, MINATEC) et je remercie Monsieur Hervé Fanet et Monsieur Jean-René Lèquepeys de m'avoir permis d'y travailler. J'exprime également ma sympathie à toutes les personnes du service, notamment à Madame Armelle De Kerleau pour sa disponibilité au secrétariat.

Je voudrais aussi remercier Monsieur François Bertrand de m'avoir accueilli dans le

laboratoire IAN, ainsi que tous les membres du laboratoire : Christian B., Didier L., Didier V., Edith B., Fabien C., Jérôme M., Pascal V., Romain L., Yves D.,. . . de m'avoir apporté un environnement de travail très agréable et amical.

Je tiens de plus à saluer tous mes collègues doctorants, stagiaires au LETI avec qui j'ai partagé cette expérience de recherche : Antoine J., Antoine S., Bettina R., Diego P., David A., Camille, Florent, Hugo, Mathieu, Michel N., Philippe G., Sylvain M.,. . . ainsi que mes amis français et vietnamiens.

J'aimerais remercier mes parents, ma petite sœur Phuong-Thao et mon grand frère Xuan-Tuan qui m'ont donné confiance tout au long de cette thèse.

Enfin, une pensée très tendre pour ma femme Hai-Son, et une immense marque d'affection pour notre petite princesse Hà-My qui a point le bout de son nez en janvier 2006.

Grenoble, Février 2008.

Bonne lecture !

Liste des publications

Voici une liste de mes publications rédigées pendant cette thèse :

X.-T. Tran, Y. Thonnart, J. Durupt, V. Beroulle, and C. Robach. A Design-for-Test Implementation of an Asynchronous Network-on-Chip Architecture and its Associated Test Pattern Generation and Application. In *Proceeding of the 2nd ACM/IEEE International Symposium on Networks-on-Chips*, NOCS 2008, (10 pages), Newcastle upon Tyne, UK, April 2008.

X.-T. Tran, J. Durupt, Y. Thonnart, F. Bertrand, V. Beroulle, and C. Robach. Implementation of a Design-for-Test Architecture for Asynchronous Networks-on-Chip. In *Proceeding of the 1st ACM/IEEE International Symposium on Networks-on-Chips*, NOCS 2007, pp. 216–216, New Jersey, USA, May 2007.

X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. How to Implement an Asynchronous Test Wrapper for Networks-on-Chip Nodes". In *Informal Proceedings of the 12th IEEE European Test Symposium*, ETS 2007, pp. 29–34, Freiburg, May 2007.

X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. A DFT Architecture for Asynchronous Networks-on-Chip. In *Proceeding of the 11th IEEE European Test Symposium*, ETS 2006, pp. 219–224, Southampton, UK, May 2006.

X-T. Tran, V. Beroulle, J. Durupt, C. Robach, and F. Bertrand. Design-for-Test of Asynchronous Networks-on-Chip. In *Proceeding of the 9th IEEE workshop on Design and Diagnostics of Electronic Circuits and Systems*, DDECS 2006, pp. 163–167, Prague, April 2006.

X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. Conception en Vue du Test pour l'Architecture d'un Réseau sur Puce Asynchrone. In *Proceeding of the 9th Journées Nationales du Réseau Doctoral en Microélectronique*, JNRDM 2006, pp. 504–507, Rennes, France, May 2006.

Introduction

Des systèmes sur puce aux réseaux sur puce

Les communications dans les systèmes sur puce (*SoC : System-on-Chip*) sont actuellement réalisées à partir de topologies de bus, standard ou propriétaire : toutes les unités de traitement devant communiquer entre elles sont reliées au même médium de communication (« bus » ou « bus hiérarchique »), et un élément central (« arbitre de bus ») arbitre l'accès au médium pour les unités afin d'éviter les conflits.

Ces architectures présentent de nombreuses limitations : limitation du débit des communications (bande passante du bus), difficulté d'adaptation des débits des échanges aux besoins des applications, latences importantes, impossibilité de bien gérer le parallélisme dans le traitement des données compte tenu des contraintes d'acheminement des informations et des éléments de contrôle...

La complexification croissante des applications multimédia, télécoms, etc., et les besoins accrus de performances conduisent les concepteurs à réaliser des SoC implémentant un nombre toujours croissant d'unités de traitement. La quantité de données échangées entre ces unités est également en constante augmentation. Les architectures de communication sur puce deviennent donc un élément important qu'il est déterminant de maîtriser lors de la conception d'un SoC complexe. Ceci a conduit à repenser les SoC pour proposer de nouvelles architectures de type « réseau sur puce » (de l'anglais « *Network-on-Chip (NoC)* »). Cette architecture consiste en une adaptation des principes des réseaux informatiques pour réaliser des structures de communication flexibles et distribuées. Cette approche présente de nombreux avantages qui font pressentir les NoC comme les successeurs des bus : augmentation des débits d'échanges et des performances globales des circuits numériques en mettant à profit les capacités d'interconnexion liées aux nouvelles générations technologiques, réduction des temps de latence entre traitements, extensibilité et flexibilité accrue de l'architecture du fait de sa régularité, gestion distribuée des arbitrages et

affectation des données sur les ressources matérielles, etc. L'architecture NoC fait ainsi l'objet de nombreux travaux de recherche dans les milieux académiques et industriels depuis quelques années.

Dans ce contexte, le Laboratoire d'Intégration des Architectures Numériques (LIAN) du LETI[1] a proposé une architecture de réseau sur puce qui s'est concrétisée au cours du projet FAUST (***F**lexible **A**rchitecture of **U**nified **S**ystems for **T**elecom*).

Motivations et Objectifs

Les NoCs permettent aux concepteurs d'intégrer de plus en plus d'unités de traitement dans un SoC. Cependant, les problèmes liés à l'ultilisation d'une horloge globale sont aussi de plus en plus présents, notamment dans les technologies avancées. Afin de surmonter ces problèmes, les structures de communication doivent désormais permettre de faire cohabiter plusieurs domaines d'horloge. Le paradigme des systèmes globalement asynchrones et localement synchrones (GALS) a donc été proposé. Le NoC est bien adapté au paradigme GALS dans la mesure où les IP peuvent avoir chacune sa propre horloge. Les NoC asynchrones, dans lesquels l'architecture de réseau est implémentée en logique asynchrone tandis que les IP sont implémentées en utilisant des méthodologies de conception synchrone standard, poussent plus loin ce paradigme avec un réseau sans horloge. L'architecture NoC du LETI, appelée ANOC, en est un exemple.

Les NoC asynchrones promettent une solution efficace pour les SoC complexes. Cependant, faute de méthodologies et d'outils de test adaptés, le test de production des réseaux sur puce asynchrones constitue un grand défi pour la mise sur le marché de ces systèmes. L'objectif de cette thèse est de proposer une nouvelle méthode de test pour les réseaux sur puce asynchrones, puis d'implémenter architecturalement cette méthode pour le réseau ANOC. Les difficultés viennent du fait que les éléments du NoC (routeurs et liens) sont profondément enfouis dans le système et l'implémentation asynchrone NoC ne permet pas d'utiliser les techniques de test répandues pour le test des circuits synchrones.

Contributions de la thèse

Afin de répondre aux objectifs de recherche présentés dans les paragraphes précédents, nous avons proposé dans cette thèse les contributions suivantes :

[1]Laboratoire d'Électronique et des Technologies de l'Information

- Une étude bibliographique avancée sur les réseaux sur puce et sur le test des systèmes sur puce nous a permis d'avoir une compréhension profonde de ces nouvelles architectures NoC, ainsi que d'analyser des méthodes de test pour assurer la testabilité pour les SoC et les circuits asynchrones.
- De ces études, nous avons proposé une nouvelle méthode de test pour les réseaux sur puce asynchrones. Dans cette méthode, nous avons développé une architecture Conçue en Vue du Test (CVT) dans laquelle chaque routeur du réseau est entouré d'un wrapper de test asynchrone qui améliore sa contrôlabilité et son observabilité. Cette architecture nous permet de tester facilement tous les éléments du réseau.
- Cette architecture CVT a été modélisée, implémentée en logique asynchrone QDI (*Quasi-Delay Insensitive*), et validée avec le réseau sur puce asynchrone ANOC développée au laboratoire. Un flot de conception à plusieurs niveaux a été développé pour faciliter ces travaux.
- La mise en œuvre de l'architecture CVT développée a été réalisée dans le cadre du circuit ALPIN (***A****synchronous* ***L****ow* ***P****ower* ***I****nnovative* ***N****oC*). Comme il n'existe aucun outil ATPG pour les circuits asynchrones, une méthode de génération des vecteurs reposant à la fois sur l'analyse des fonctionnalités et l'implémentation structurelle du réseau a été proposée. L'application de ces vecteurs de test a été étudiée, puis nous avons proposé un algorithme de test permettant de tester tous les éléments du réseau. La méthode de test complète développée dans cette thèse permet une couverture de faute de 99,86% pour le réseau ANOC en utilisant le modèle de faute de collage simple.
- Différentes utilisations alternatives de l'architecture CVT ont été proposées pour le diagnostic, la vérification du réseau sur silicium, et le test des unités de traitement en utilisant l'architecture développée.

Plan du document

Hormis cette introduction, ce document comprend quatre chapitres. Le premier chapitre présente un état de l'art sur la conception des architectures de réseaux sur puce. Il expose en détail les nouveaux concepts relatifs aux NoC et introduit le réseau ANOC que nous avons ciblé dans le cadre de cette thèse.

Le deuxième chapitre aborde les problématiques de test des systèmes intégrés et les différentes méthodologies existantes. Afin de bien comprendre la différence de problématique entre le test des circuits synchrones et asynchrones, plusieurs concepts de base de la conception asynchrone et du test de ce type de circuits sont présentés. A la fin du chapitre, la méthode de test existante pour le réseau ANOC

est également introduite et discutée.

Les chapitres suivants constituent une présentation détaillée du travail effectué au cours de cette thèse et des résultats que nous avons obtenu. Le chapitre 3 propose une méthode de test pour les architectures NoC asynchrones. Dans cette méthode, une architecture conçue en vue du test est développée. La modélisation et la réalisation de cette architecture en logique asynchrone QDI sont tout d'abord présentées, puis l'architecture développée est validée avec le réseau ANOC. Nous présentons enfin dans ce chapitre plusieurs résultats sur l'implémentation de cette architecture.

Le chapitre 4 se concentre ensuite sur la mise en œuvre de cette architecture pour le test du réseau ANOC. La génération des vecteurs de test et l'application de ces vecteurs de test pour les différents éléments du réseau sont tout d'abord présentées. On introduit ensuite un algorithme de test qui permet de tester le réseau complet, puis plusieurs résultats de test sont également présentés. Les utilisations alternatives de l'architecture CVT développées telles que le diagnostic, la vérification du réseau sur silicium, le test des IP sont aussi développées.

A l'issue de ce quatrième chapitre, nous présentons nos conclusions sur cette méthodologie de test des NoC asynchrones, aussi que des perspectives sur les suites de ces travaux.

Chapitre 1

Réseaux sur Puce

L'évolution de la technologie des circuits intégrés et les performances toujours croissantes des systèmes électroniques rendent les systèmes intégrés sur puce (*SoC : System-on-Chip*) de plus en plus complexes. Les concepteurs embarquent de plus en plus d'unités de traitement (IP : Intellectual Property) dans les systèmes pour répondre aux besoins des nouvelles applications. Ceci entraîne une augmentation des communications dans les circuits pour les échanges de données et pour le contrôle des traitements. Il est nécessaire de trouver une solution efficace pour la communication sur puce afin d'améliorer la performance globale des systèmes intégrés.

En même temps, l'entrée de la technologie silicium dans le domaine submicronique profond a aggravé le déséquilibre entre les délais des portes et les délais des interconnexions. La conception physique est devenue le goulot d'étranglement du développement des circuits. La qualité de l'architecture d'interconnexion devient prépondérante pour la performance du système.

Dans ce contexte, les solutions d'interconnexion classiques généralement basées sur des bus partagés présentent des limites. Elles ne supportent pas les débits élevés, manquent de flexibilité et ne seront pas adaptées pour les systèmes futurs implémentant plusieurs centaines d'unités de traitement.

Afin de surmonter ces problèmes, un nouveau paradigme de communication a été proposé par différents groupes de recherche dans les années 2000. Ce nouveau paradigme appelé réseau sur puce dérive des réseaux informatiques en les ramenant au niveau du système intégré (*NoC : Network-on-Chip*). Ce type d'architecture présente de réels atouts aux niveaux conception, performance, et implémentation.

L'objectif de ce premier chapitre est de présenter les architectures NoC dans le contexte des travaux de thèse. Dans une première section, nous introduisons le

réseau sur puce par rapport au contexte des SoCs et les concepts relatifs au NoC. L'état de l'art de la recherche sur les NoC au cours des dernières années sera présenté dans la deuxième section. La dernière section du chapitre présente en particulier un réseau sur puce appelé ANOC qui est développé au LETI. Ce réseau est la cible de nos travaux qui seront présentés dans les chapitres 3 et 4.

Ce chapitre cherche à donner au lecteur une large compréhension du fonctionnement d'un réseau sur puce et une vision globale de ce qui a été fait dans le domaine. Il présente également les tenants et les aboutissants de la conception d'un système intégrant un NoC.

1.1 Réseau sur puce : concepts et avantages

Dans cette première section, nous décrivons les évolutions actuelles de la conception des systèmes sur puce en termes de communications pour introduire le concept de réseau sur puce. Ensuite, nous allons étudier ses concepts de base.

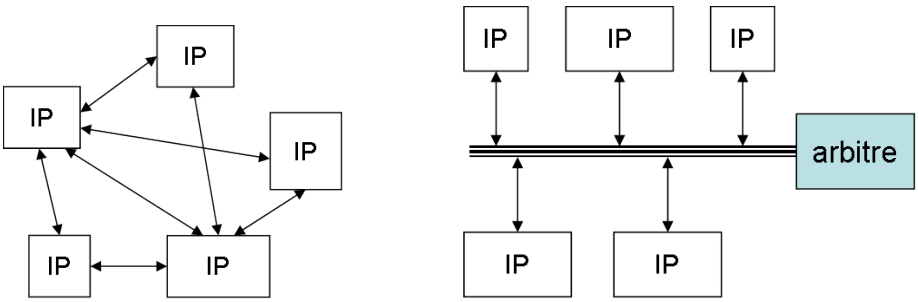
1.1.1 Interconnexions dans les systèmes sur puce

1.1.1.1 Solutions d'interconnexion actuelles et limitations

Dans les circuits intégrés, les structures d'interconnexion classiques sont souvent des connexions *ad hoc* et/ou bus partagé. Ces types d'interconnexion rencontrent beaucoup de problèmes lorsque les systèmes se complexifient. La plupart de ces problèmes augmenteront à l'avenir. Particulièrement, les problèmes tels que la limitation de débit, la consommation d'énergie, l'intégrité des signaux, la latence des signaux, et la synchronisation globale deviendront des goulots d'étranglement dans la conception des systèmes intégrés s'il n'y a aucune amélioration des méthodes d'interconnexion classiques [Guer00A, Dall01R, Beni02N, Zefe02A, Radu03C].

Avec les connexions *ad hoc*, la communication entre une source et une destination est accomplie en utilisant des liens physiques directs (liaisons point-à-point), cf. figure 1.1(a). L'avantage d'utiliser ces liens dédiés est d'avoir la meilleure performance en termes de bande passante. Cependant, ces structures d'interconnexion ont besoin d'un nombre important de liens alors que le taux d'utilisation des liens est très bas (environ 10% selon une recherche de Dally *et al.* [Dall01R]). De plus, ces structures d'interconnexion deviendront compliquées pour la conception des systèmes futurs.

Dans les systèmes sur puces complexes, le bus partagé (cf. figure 1.1(b)) est aujourd'hui le moyen de communication le plus répandu dans l'industrie. Le bus offre alors une solution maîtrisée pour connecter entre eux un ensemble de modules compatibles avec le protocole de communication. En comparaison avec les



(a) Point-à-point (b) Bus partagé

FIG. 1.1 : Architectures d'interconnexion dans les systèmes sur puce

liaisons point-à-point, les bus sont flexibles et facilement réutilisables. Cependant, ils ne permettent qu'une seule communication à la fois entre les modules qui s'y connectent. La bande passante est donc partagée et la performance diminue lorsque le nombre des modules augmente. Pour gérer les accès simultanés, ils ont besoin d'un mécanisme d'arbitrage. De plus, les bus nécessitent des longueurs de fils importantes qui consomment beaucoup d'énergie puisque les données sont diffusées sur l'ensemble du bus. Lorsqu'on augmente le nombre de modules, la taille du bus augmente, ce qui a pour conséquence d'augmenter le délai de transmission des données au point de dépasser la période d'horloge [Grec04S]. Le bus est donc par nature limité dans sa capacité à interconnecter un grand nombre d'IP.

Pour surmonter certains inconvénients rencontrés, la structure de bus hiérarchique (cf. figure 1.2) a été proposée et utilisée dans les industries (IBM Core Connect [IbmCc], ARM AMBA [Amba]). Le principe de cette solution est de relier plusieurs

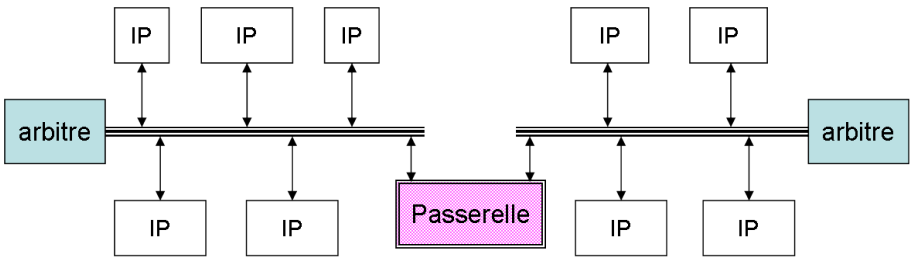


FIG. 1.2 : Bus hiérarchique.

bus entre eux par l'intermédiaire de passerelles (*bus bridge*). Avec cette idée, nous pouvons réduire le nombre des IP qui sont connectés au même bus, la taille des fils est ainsi réduite. Chaque bus possède son propre arbitre. Cependant, cette solution rencontre le problème difficile de gestion de l'adressage lorsque les informations doivent transiter par les passerelles pour passer d'un bus à l'autre. De plus, cette

solution ne peut éviter les problèmes naturels des bus partagés.

1.1.1.2 Réseaux sur puce : nouvelles architectures d'interconnexion

L'évolution des technologies d'intégration, la convergence des applications, la réduction des temps de mise sur le marché (*time-to-market*) ont entraîné de profonds changements dans les méthodologies de conception des systèmes sur puce. Les systèmes sur puce se composent de plus en plus d'IP. C'est pourquoi le concept de réutilisation d'IP (*IP reuse*) devient de plus en plus important. Les IP déjà développées peuvent être réutilisées pour d'autres applications afin de réduire le temps de conception, le nombre de concepteurs, et donc le coût de production. Les architectures de bus s'adaptent mal aux nouvelles méthodologies en raison des protocoles de bus, qui ne permettent pas d'établir une hiérarchie des services de communication. Ainsi l'ajout de nouvelles fonctionnalités induit des modifications majeures jusqu'au niveau de l'interface entre le bus et l'IP [Zefe02A].

Dans le domaine de la communication, les réseaux informatiques ont su faire face à des problèmes similaires en utilisant des structures d'interconnexion distribuées et bien hiérarchisées. Ces problèmes sont étudiés depuis des dizaines d'années aussi bien dans le contexte des réseaux locaux que dans l'interconnexion de machines parallèles. Les réseaux informatiques ont beaucoup évolué au niveau physique (Ethernet, ADSL, fibre optique, sans-fil) et au niveau des transactions (Internet, transport de la voix, de la vidéo) tout en gardant une compatibilité entre systèmes.

Par analogie avec les solutions présentées ci-dessus, l'idée d'intégrer un réseau de communication sur silicium a été proposée. L'objectif est de proposer une architecture réellement nouvelle et compétitive en termes de performance, de consommation et de surface silicium.

La recherche dans ce domaine s'est réellement intensifiée dans les années 2000 quand l'ITRS[1] a affirmé qu'il était nécessaire de proposer une nouvelle architecture de communication sur puce pour surmonter les problèmes de conception liés à l'augmentation de la densité d'intégration dans les circuits [Jant03N]. Les premières approches ont abordé le problème à la fois d'un point de vue de la conception haut niveau [Beni01P, Hema00N] et du point de vue physique [Dall01R, Ho01T]. Très rapidement, quelques contributions sur les architectures concrètes de réseau sur puce ont été proposées [Guer00A, Lian00A, Rijp01A, Kari01O].

Concernant la terminologie, le terme le plus utilisé est celui de *Réseau sur Puce* traduction directe de *Network-on-Chip (NoC)* qui dérive de *System-on-Chip (SoC)*. De plus, il existe dans les documents quelques autres termes, *On-Chip Network* ou

[1] International Technology Roadmap for Semiconductors (ITRS)

Network-on-Silicon. Cependant, ces termes sont rarement utilisés par la communauté.

La recherche sur les NoC réalise une synthèse multidisciplinaire entre les domaines du calcul distribué, des réseaux et des communications sur puce. Elle consiste en une adaptation des solutions éprouvées dans le cas des réseaux informatiques au cas spécifique de la conception de SoC. Beaucoup de groupes de recherche (centres de recherche, universités, industries) ont lancé des activités de recherche sur les NoC mais tous les aspects n'ont pas encore été abordés. Il n'existe pas de solution industrielle avec un niveau de maturité comparable à celui des bus. Il existe quelques démonstrateurs, Intel Polaris [Vang07A] d'Intel, CHAIN [Bain02C] de Silistix, FAUST [Latt07A] du LETI, mais nous sommes encore loin de solutions satisfaisantes pour envisager des produits commerciaux. Pour résumer, nous pouvons constater que les travaux sur les NoC sont encore au stade de la recherche académique ou en préparation dans les laboratoires industriels.

1.1.2 Concepts relatifs aux réseaux sur puce

La sous-section précédente a introduit le réseau sur puce comme une solution potentielle pour la communication des systèmes sur puce. Dans cette sous-section, nous allons présenter un certain nombre de concepts relatifs aux NoC. Ces différentes notions sont issues des publications qui ont été faites dans le domaine. L'objectif est de comprendre les notions principales associées aux NoC.

1.1.2.1 Topologies des réseaux sur puce

La topologie d'un réseau définit comment les routeurs sont interconnectés entre eux en utilisant les liens de réseau. Elle spécifie l'organisation physique du réseau et elle est donc souvent modélisée par un graphe. Comme pour les réseaux informatiques, de nombreuses topologies sont envisageables pour construire un NoC. La figure 1.3 introduit sélectivement les topologies les plus couramment utilisées dans la conception des NoC.

Chaque topologie possède ses avantages et inconvénients. Afin de comparer les topologies entre elles, différents paramètres physiques sont utilisés. Les paramètres suivants sont les plus couramment considérés dans les publications (dérivées de la présentation graphique) : le degré du routeur (*router degree*), le diamètre du réseau, la régularité, la symétrie, la diversité de chemins de routage, et la largeur de bissection.

Le *degré du routeur* est le nombre de liens qui connectent ce routeur avec ses voisins. Le *diamètre du réseau* est la distance maximale entre deux routeurs dans le

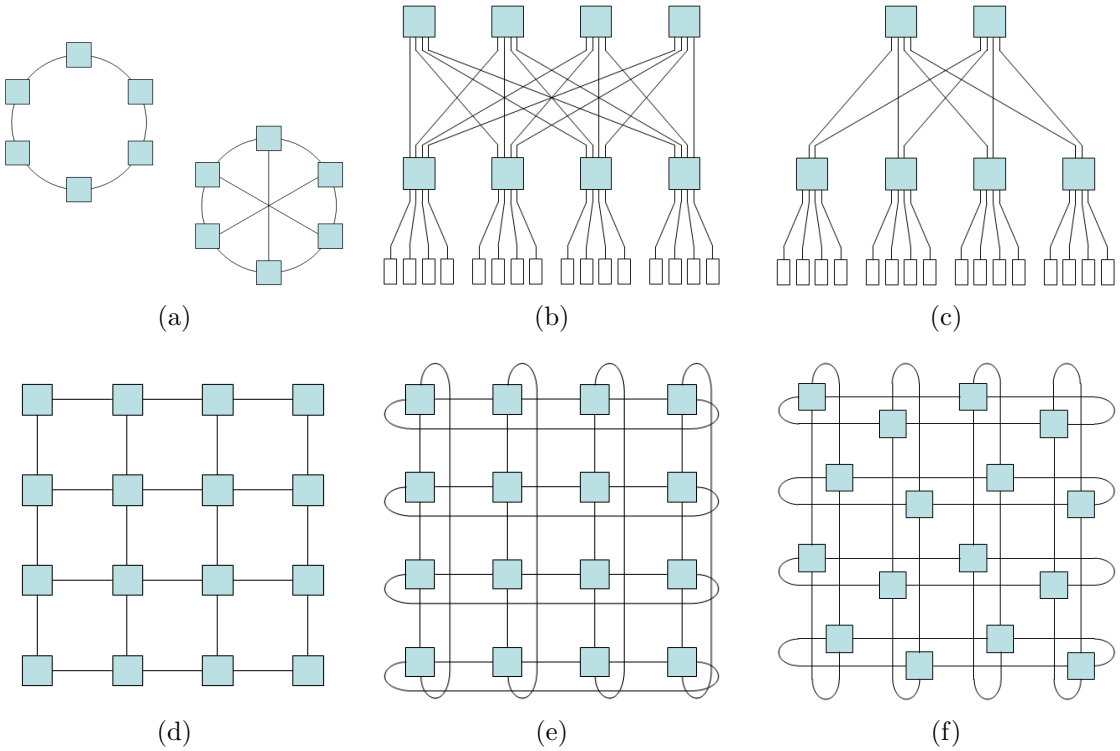


FIG. 1.3 : Topologies usuelles de réseaux sur puce : *(a) réseau en anneau avec ou sans corde ; (b) réseau en arbre élargi ; (c) réseau en arbre élargi en papillon ; (d) réseau à maillage simple à deux dimensions ; (e) réseau maillé en tore à deux dimensions ; (f) réseau maillé en tore replié à deux dimensions.*

réseau (en considérant les plus courts chemins). Un réseau est dit *régulier* lorsque tous les routeurs ont le même degré et *symétrique* lorsqu'il est de même topologie vu depuis chacun des routeurs. Un réseau a une *diversité de chemins de routage* si la plupart des couples de routeurs ont de multiples chemins de routage entre eux. La diversité de chemins de routage ajoute de la robustesse au réseau. La *largeur de bissection* est le nombre minimal de liens qu'il faut couper pour séparer le réseau en deux parties égales. Cela permet d'évaluer le coût pour transférer les données d'une moitié du réseau à l'autre. La bande passante collective de ces liens est nommée *bande passante de bissection*, et elle permet de mesurer la performance d'une topologie.

Selon les propositions d'architecture NoC, la topologie prédominante est la topologie de maillage simple à deux dimensions (*2D-mesh*). Les raisons en sont ses avantages tels que la facilité d'implémentation dans les technologies actuelles, la simplicité des stratégies de routage, et l'évolutivité du réseau. En plus de la topologie de maillage à deux dimensions, la topologie de maillage en tore à deux dimensions (*2D-torus*) est aussi utilisée afin de réduire le diamètre du réseau et d'augmenter la

bande passante. Toutefois, cette topologie possède certaines limitations. Sa structure est plus complexe à implémenter que pour un réseau maillé simple et les liens utilisés pour boucler le réseau (routeurs à la frontière) sont plus longs. Ces limitations pénalisent donc les performances des liens. Pour réduire la longueur de ces liens, la topologie de maillage en tore replié à deux dimensions (*2D folded torus*) est utilisée. Cependant, cette topologie rend les algorithmes de routage plus complexes. Toutes ces topologies orthogonales ont un problème avec la latence réseau associée.

D'autres topologies intéressantes sont la topologie d'arbre élargi (*fat-tree*) et la topologie d'arbre élargi en papillon (*butterfly fat-tree*). Ces deux topologies de réseau conduisent à un diamètre plus petit et la latence réseau associé est donc réduite. Les principaux inconvénients de ces topologies sont des problèmes de degré du routeur et la complexité d'interconnexion. La topologie *butterfly fat-tree* est utilisée pour avoir un degré inférieur à la topologie *fat-tree*. Cependant, cette topologie accroît la complexité d'interconnexion et le coût en surface en raison de sa hauteur d'arbre plus élevé. La topologie anneau avec cordes (*chordal ring*) est utilisée pour obtenir plus de performance de communication que la topologie anneau (*ring*), mais elle rend l'interconnexion plus complexe et le routage plus difficile.

En outre, les topologies hybrides qui combinent les mécanismes et les structures des bus partagés et des réseaux peuvent aussi être utilisées pour construire une architecture de communication sur puce. L'objectif est d'augmenter la bande passante par rapport aux bus partagés et de réduire la distance entre routeurs par rapport aux réseaux directs et indirects. Un exemple d'une architecture NoC basé sur une topologie hybride a été proposé par Paul Wielage dans [Wiel02N].

Au final, le choix de la topologie pour une architecture NoC est complexe. Il est fortement lié aux nombreuses contraintes de l'application et au trafic du réseau. Nous devons faire un compromis entre la performance intrinsèque de la topologie et le coût que représente son implémentation.

1.1.2.2 Mécanismes de communication et modes de commutation

Après avoir choisi une topologie, dans ce paragraphe, nous allons discuter de la manière dont les informations sont transférées sur le réseau. En fait, la communication dans un réseau est basée sur le mécanisme de commutation. Deux principaux types de commutation sont utilisés [Mora04H, Jant03N] : la commutation de circuits (*circuit-switching*) et la commutation de paquets (*packet-switching*).

La commutation de circuits : C'est le premier, le plus ancien mécanisme de commutation. Il a été utilisé dans l'ancienne génération de réseaux téléphoniques.

Dans la commutation de circuits, le chemin de communication entre deux unités de traitement doit être établi avant que toutes les transmissions puissent avoir lieu. Il existe donc un lien physique entre les deux unités de traitement pendant la communication. Les informations de contrôle (pour établir et mettre en fin à la connexion) sont séparées des données de communication.

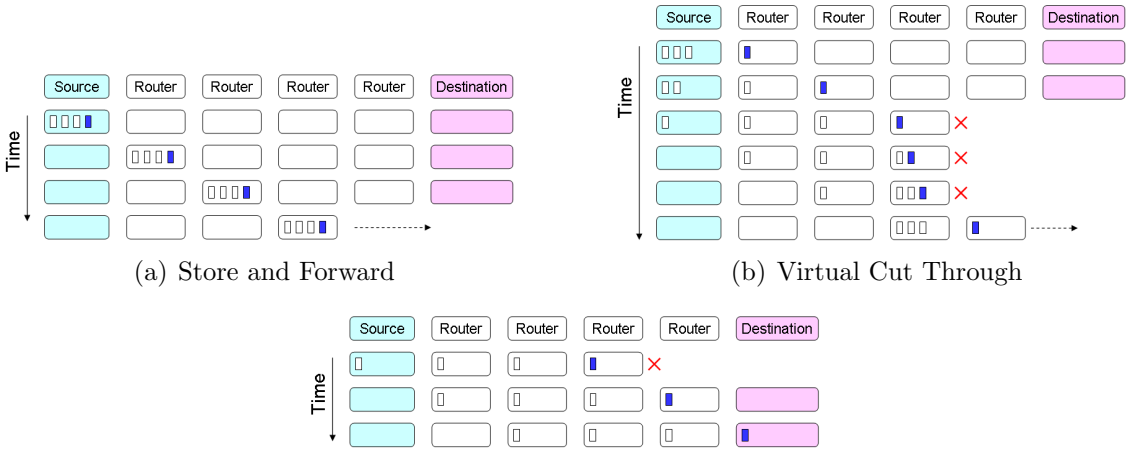
L'avantage de ce mécanisme est que l'émetteur a une bonne connaissance de la bande passante disponible et de la latence de transmission quand une connexion a été établie. Il est bien adapté lorsqu'on veut effectuer des transferts de données importantes puisqu'il faut rentabiliser le temps passé à négocier la connexion. Ce mécanisme est utilisé lorsque l'on veut des garanties de services, cf. section 1.1.2.6. En contre-partie, avec ce mécanisme, une fois la connexion établie, les éléments du réseau (routeurs, liens) sont alloués pour ce transfert de données et ne peuvent pas être utilisés par d'autres unités de traitement durant le temps de la connexion. De plus, les réseaux utilisant ce mécanisme ont besoin des contrôleurs centraux, ce qui accroît le coût en surface. Un autre inconvénient de ce mécanisme est la flexibilité du réseau. Une fois un routeur ajouté, nous devons modifier les contrôleurs centraux.

La commutation de paquets : En commutation de paquets, les données doivent être encapsulées dans des paquets en provenance de la source avant d'être envoyées à la destination. L'avantage de ce mécanisme est de donner au réseau la performance maximale possible et de partager des éléments du réseau. Le réseau est localement commuté entre deux routeurs mais pas entre deux unités de traitement comme dans la commutation de circuits. Les éléments du réseau peuvent donc être utilisés par autres transmissions.

Un autre avantage de ce mécanisme est qu'il n'y a plus besoin de contrôleur central car les paquets peuvent être individuellement routés dans le réseau. En contre-partie, les paquets doivent contenir les informations de routage : ces informations de contrôle sont envoyées en même temps que les données de communication. Les routeurs sont souvent complexes car ils doivent traiter les informations de contrôle et router les données de communication. Un autre inconvénient est que la latence est plus grande. Par exemple, les routeurs ont parfois à modifier le contenu des paquets pour mettre à jour des informations de routage. De plus, les problèmes tels que la gestion des blocages et la gestion de l'ordre des paquets sont des problèmes critiques de ce mécanisme. Il faut introduire des systèmes de gestion des blocages et des priorités.

Comme le mécanisme est complexe, dans le cas de la commutation de paquets, les paquets sont découpés en éléments de taille fixe, appelés *flits* (une unité de

contrôle de flot du réseau). C'est un élément unitaire contenant des données et aussi des contrôles qui peut être transmis sur le réseau. Il faut donc définir les modes de commutation, c'est-à-dire la façon dont les paquets vont transiter d'un routeur du réseau au suivant. Il y a trois principaux modes de commutation [Rijp03T] : *store-and-forward*, *virtual cut-through*, et *Wormhole*.



(a) Store and Forward

(b) Virtual Cut Through

(c) Wormhole

FIG. 1.4 : Modes de commutation.

– ***Store-And-Forward (SAF) :*** Dans ce mode de commutation, un paquet de donnée complet est transféré à partir d'un routeur vers le suivant, cf. figure 1.4(a). Cela implique que chaque routeur doit être en mesure de stocker la totalité d'un paquet de donnée. Lorsque la taille des paquets est grande, il faut beaucoup d'espace de mémorisation. Comme les ressources de mémorisation sont très coûteuses en termes de surface et de consommation, la taille de paquet est donc limitée. De plus, le délai des paquets augmente à chaque routeur car tous les *flits* du paquet doivent être reçus par le routeur actuel avant d'être envoyé à l'autre routeur. La latence totale est donc la multiplication entre le temps de transfert d'un paquet entre deux routeurs et le nombre de routeurs du réseau traversés par le paquet de sa source à sa destination.

– ***Virtual-Cut-Through (VCT) :*** Ce mode de commutation a été proposé afin de réduire la latence des paquets à chaque étape de routage. Dans ce mode de commutation, un paquet peut commencer à être transféré au routeur suivant avant la réception complète de ce paquet par le routeur actuel, comme illustré dans la figure 1.4(b). Le principal inconvénient de ce mode est que le routeur actuel doit pouvoir stocker le paquet dans sa totalité si le routeur suivant n'est pas disponible. La capacité de mémorisation du routeur est donc

la même que pour le mode SAF mais la latence est diminuée.

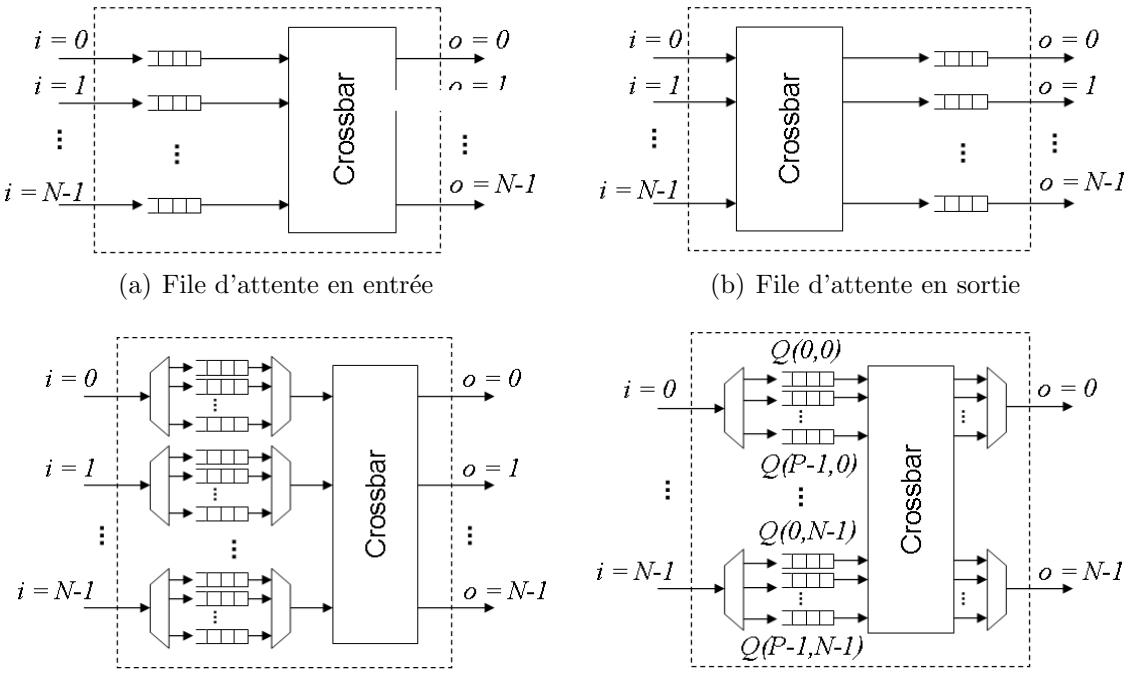
– ***Wormhole (WH) :*** Dans ce mode de commutation, les *flits* passent de routeur en routeur dès qu'il y a de la place pour un *flit* et pas nécessairement pour un paquet complet, cf. figure 1.4(c). Normalement, un paquet comprend un *flit* d'en-tête (*header flit*), qui peut être suivi par un ou plusieurs *flit(s)* de données. Le *flit* d'en-tête réserve le chemin de routage, tous les *flits* qui suivent doivent emprunter le même chemin que le *flit* d'en-tête. L'avantage de ce mode de commutation est qu'un même paquet peut être réparti sur plusieurs routeurs du réseau. Le routeur ne doit pas stocker un paquet complet, la taille des éléments de mémorisation (*buffers*) est donc diminuée. De plus, la latence est aussi réduite. En contre-partie, les risques de blocage sont plus importants car un paquet s'étend sur plusieurs routeurs.

1.1.2.3 Stratégies de stockage

Dans le paragraphe précédent, nous avons montré que dans la commutation de paquets les routeurs ont besoin d'éléments de mémorisation. Le positionnement des mémoires par rapport aux ports d'entrées et de sortie du routeur se fait selon différentes stratégies [Rijp03T]. Dans ce paragraphe, nous allons distinguer les quatre principales stratégies utilisées dans la conception de NoC : file d'attente en entrée, file d'attente en sortie, file d'attente en sortie virtuelle, et file d'attente en entrée avec canaux virtuels. On suppose que N est le nombre de ports d'entrée, et également le nombre de ports de sortie du routeur. Ces stratégies peuvent être expliqués de la manière suivante :

– **File d'attente en entrée (*Input queuing*) :** Dans cette stratégie, N files d'attente sont placées aux entrées du routeur, comme illustré dans la figure 1.5(a). Un arbitre de séquencement détermine à quel moment une file d'entrée est connectée à chaque port de sortie afin qu'aucun conflit ne survienne. Toutefois, le trafic du routeur sature à 59% [Karo87I] à cause du blocage en tête de ligne (*head-of-line blocking*) lorsque N est grand. Ce phénomène se produit lorsqu'une donnée en tête de file n'a pas accès à un port de sortie et bloque les données suivantes dans la file.

– **File d'attente en sortie (*Output queuing*) :** Dans cette stratégie, N files d'attente sont placées aux sorties du routeur, cf. figure 1.5(b). Tous les *flits* qui arrivent dans un *time slot* sont routés avant le début du prochain



(a) File d'attente en entrée

(b) File d'attente en sortie

(c) File d'attente en sortie virtuelle

(d) File d'attente en entrée avec canaux virtuels

FIG. 1.5 : Stratégies de stockage.

time slot. Par conséquent, le *crossbar* doit s'exécuter N fois plus vite que ses lignes d'entrée/sortie, même si toutes les N entrées ont des paquets destinés à la même sortie. Si k paquets arrivent pour une sortie au cours du *time slot* actuelle, une seule peut être transmise à la sortie. Les $k-1$ paquets restants iront à la file d'attente de la sortie pour transmission ultérieure dans les cycles suivants. Cette stratégie est optimale dans le sens où elle atteint un débit maximal pour chaque entrée et une meilleure performance que la stratégie précédente. L'inconvénient de cette stratégie est l'exigence d'un *crossbar* assez rapide.

– **File d'attente en sortie virtuelle (*VOQ : Virtual Output Queuing*) :** Dans cette stratégie, N^2 files d'attente sont placées aux entrées du routeur comme illustré dans la figure 1.5(c). Cette stratégie a pour but de combiner les avantages des deux précédemment citées. En principe, la stratégie VOQ est une version de la stratégie « file d'attente en entrée » avec des files d'attente en sortie virtuelles pour supprimer le blocage en tête de ligne et simuler de fonctionnement « file d'attente en sortie ». Toutes les entrées peuvent écrire à la sortie correspondante en même temps. La stratégie VOQ a donc la meilleure

performance des stratégies de stockage. Toutefois, cette stratégie a besoin d'un grand nombre de files d'attente. Cela implique une implémentation du routeur très coûteuse.

– **File d'attente en entrée avec canaux virtuels (*VCPIQ : Virtual Channel Priority Input Queuing*) :** Cette stratégie a été proposée afin d'augmenter la performance de commutation et de réduire l'inconvénient de la stratégie VOQ. Pour un canal physique, il y a P files d'attente ($P < N$) qui correspondent à des canaux virtuels et qui divisent la bande passante du canal, cf. figure 1.5(d). L'utilisation de la capacité des liens peut être augmentée par la réduction des blocages en tête de ligne. D'autre part, la latence et la complexité du routeur augmentent avec le nombre de canaux virtuels. Pande *et al.* [Pand05P] ont montré que quatre canaux virtuels ($P = 4$) constituent un bon compromis entre la vitesse et la latence.

1.1.2.4 Algorithmes de routage

L'algorithme de routage détermine le chemin emprunté par un paquet entre la source et la destination. Dans une architecture de communication, le routage joue un rôle important : le meilleur algorithme de routage donne la meilleure performance. L'algorithme de routage est donc soigneusement élaboré par les concepteurs. Il est nécessaire de faire des compromis entre une utilisation optimale des liens de communication et un algorithme simple qui peut être facilement mis en œuvre sur silicium.

Il existe plusieurs algorithmes de routage utilisé dans la conception de NoC avec des propriétés différentes. La décision de routage à chaque routeur peut être *déterministe* (i.e. statique) ou *adaptative* (i.e. dynamique).

Dans un algorithme de routage *déterministe*, les chemins permanents provenant d'une unité de traitement émettrice à une unité de traitement réceptrice sont définis et utilisés indépendamment de l'état actuel du réseau. Cet algorithme de routage ne prend pas en compte la charge actuelle des routeurs et des liens de réseau lors des décisions de routage. Au contraire, dans un algorithme de routage *adaptatif*, les décisions de routage sont prises en fonction de l'état actuel du réseau (la charge du réseau, la disponibilité des liens). Par conséquent, le trafic entre une source et une destination change ses chemins de routage avec le temps.

Le routage déterministe est plus simple à mettre en œuvre en termes de logique et de l'interaction entre les routeurs [Beni02N]. Il est plus approprié lorsque les échanges de données sont fortement prévisibles et stables. Ce routage est donc préférable pour les réseaux utilisés dans les systèmes d'application spécifique (ASIC). Au contraire,

le routage adaptatif est préférable pour les applications dans lesquelles le trafic est irrégulier et imprévisible, qui sont plus fréquentes dans les réseaux multiprocesseurs (*CMP : Chip Multi-Processors*).

Les techniques de routage (à la fois déterministe et adaptative) peuvent encore être classées en fonction de l'endroit où l'information de routage est détenue et où les décisions de routage sont prises. Le routage est de type *source* lorsque c'est l'émetteur qui définit le chemin de routage. Les informations du chemin de routage sont calculées et enregistrées dans une table de routage de l'émetteur. Lorsqu'une source transmet un paquet, elle regarde les informations de routage correspondantes à la destination du paquet, puis elle inclut ces informations dans le *flit* d'en-tête du paquet.

Au contraire, si chaque routeur choisit la destination du paquet en fonction de son adresse cible (et d'autres critères éventuels), il s'agit d'un routage *distribué*. La décision de routage est mise en œuvre dans chaque routeur soit en consultant les adresses cibles dans une table de routage ou en exécutant une fonction de routage matérielle [Bolo05E]. Avec cette méthode, chaque routeur contient une table de routage prédéfinie ou des logiques de routage dont l'entrée est l'adresse de destination du paquet et la sortie est la décision de routage. Lorsque le paquet arrive au port d'entrée du routeur, le port de sortie est cherché dans la table ou calculé par la logique de routage en fonction de l'adresse de destination.

1.1.2.5 Interblocages

Parce que les paquets de données sont transmis entre les routeurs sans contrôle de routage global, il est possible d'obtenir des blocages tels que l'interblocage statique (*deadlock*) et l'interblocage dynamique (*livelock*).

Les interblocages statiques se produisent quand un ou plusieurs paquets dans le réseau sont bloqués pour un temps indéterminé, dans l'attente d'un événement qui ne peut pas se produire. Par exemple, la figure 1.6 présente un exemple d'une situation où les deux paquets sont routés de manière circulaire dans un maillage carré de quatre routeurs. Le paquet occupant les liens du réseau L_1 et L_2 attend le lien L_3 tandis que le paquet occupant les liens L_3 et L_4 attend le lien L_1. Aucun paquet ne peut progresser à cause du manque de ressources nécessaires pour le routage : les liens sont déjà détenus par un autre paquet ne seront jamais relâchés.

Dans le routage *Wormhole*, le *flit* d'en-tête contient toutes les informations de routage et les *flits* qui suivent doivent être contigus et ne peuvent pas être intercalés avec les *flits* de l'autre paquet [Dall87D]. Si le *flit* d'en-tête est bloqué à cause d'un

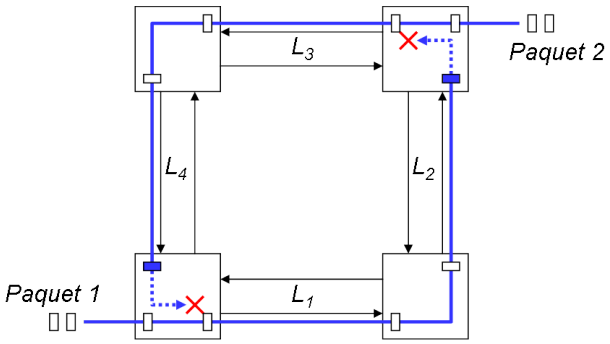


FIG. 1.6 : Exemple d'une situation d'interblocage statique.

encombrement sur les ressources de routage, tous les *flits* suivants de ce paquet seront également arrêtés, cela implique qu'ils garderont toutes les ressources qu'ils occupent en termes de liens du réseau et des *buffers*. Par conséquent, le routage *Wormhole* est très sensible à l'interblocage statique.

Pour éviter ces interblocages, il faut choisir un algorithme de routage particulier qui soit insensible à cet interblocage tel que le routage XY sur un *2D-mesh* ou qui puisse éviter la dépendance circulaire entre les paquets [Glas94T]. Le routage XY est un routage consistant à faire circuler les paquets d'abord dans la direction X (Est↔Ouest) puis dans la direction Y (Nord↔Sud). En outre, on peut également utiliser des canaux virtuels. De cette manière, un paquet temporairement bloqué et stocké sur plusieurs routeurs de réseau peut être doublé par un autre paquet, en utilisant le même canal physique mais stocké sur un autre canal virtuel.

Les interblocages dynamiques sont des situations de blocage où certains paquets ne sont pas en mesure d'atteindre leur destination, même s'ils ne sont pas bloqués en permanence. Un paquet peut circuler sur le réseau autour de son routeur destinataire mais jamais l'atteindre car les liens de réseau sont occupés par d'autres paquets. L'interblocage dynamique peut arriver dans le cas d'un routage adaptatif.

Une autre situation de blocage est appelée ***famine*** (*starvation*). C'est le cas dans lequel un paquet n'obtient jamais des ressources de routage (*buffers*, liens de réseau) car ces ressources sont toujours allouées à d'autres paquets. Cette situation se produit lorsque le trafic est intense et lorsqu'on a une mauvaise gestion de l'attribution des ressources de routage aux paquets en conflit.

1.1.2.6 Qualité de service

La *qualité de service* (*QoS : Quality of Service*) est un terme commun qui s'applique aux spécifications des services de réseau qui sont exigés et/ou donnés pour

certaines classes de trafic. Des spécifications de QoS peuvent être exprimées par les métriques de performance comme le taux de perte, le débit, la latence moyenne, la variation de latence (*jitter*), etc. qui correspondent à la qualité de transmission et à la disponibilité de services. Les publications [Goos03G, Radu03C, Rijp03T] ont proposé les notions de services ci-dessous pour les NoC :

- ***Intégrité des données :*** Les données sont transmises sans altération.
- ***Absence de perte de données :*** Aucun paquet ou *flit* n'est perdu dans le réseau pendant les transmissions. Une mauvaise gestion des flux ou des espaces mémoire dans les routeurs peut causer ces pertes.
- ***Ordonnancement des données :*** Dans le réseau, les données doivent être reçues dans l'ordre où elles ont été émises. Avec un routage déterministe, il suffit d'interdire aux paquets de se doubler. Par contre, il faut mettre en place à la réception des mécanismes de réordonnancement lorsque le routage est adaptatif.
- ***Débit :*** La quantité de données qui transitent d'une ressource à l'autre par unité de temps.
- ***Latence :*** Un service important pour la communication dans le réseau. Elle est le temps entre l'émission d'une donnée sur le réseau et sa réception par la ressource destinataire.

Pour assurer une QoS, il faut trouver un bon compromis entre les services imposés par l'application visée et le coût des ressources matérielles. Il existe deux niveaux d'implémentation de la QoS dans le NoC :

- Les réseaux à ***services garantis***. Dans ce type de réseau, des mécanismes permettent de réserver des ressources de communication pour garantir un certain débit et/ou une certaine latence quelle que soit la charge du réseau. Ce type de garantie a pour effet de surdimensionner les ressources.
- Les réseaux à services ***meilleur-effort***. Le réseau est dimensionné pour avoir de bonnes performances moyennes mais il n'offre aucune garantie de latence ou de débit dans les situations les moins favorables. Les ressources ont la même priorité et elles se partagent la capacité de trafic du réseau.

1.1.2.7 Protocole de communication

Dans un système de communication, le *protocole de communication* spécifie les principes de transmission. C'est un ensemble de règles et de méthodes qui sont nécessaires pour transférer une information de l'émetteur au récepteur. Pour simplifier l'implémentation du protocole, celui-ci est divisé en différentes couches avec

des fonctionnalités spécifiques.

Puisque le concept de NoC est emprunté aux réseaux informatiques, il est possible d'employer les mêmes protocoles de communication que ceux utilisés dans les réseaux informatiques. Cependant, les protocoles informatiques sont complexes car ils sont utilisés pour une topologie inconnue. Dans la conception de NoC, il faut donc simplifier ces protocoles afin d'obtenir le meilleur compromis entre la performance et le coût d'implémentation. Différentes propositions dans [Kuma02A, Beni02N, Mora04H, Mill04T] se concentrent sur l'adaptation de la structure du modèle de référence OSI[2], dont les sept couches sont présentées dans la figure 1.7, pour le pro-

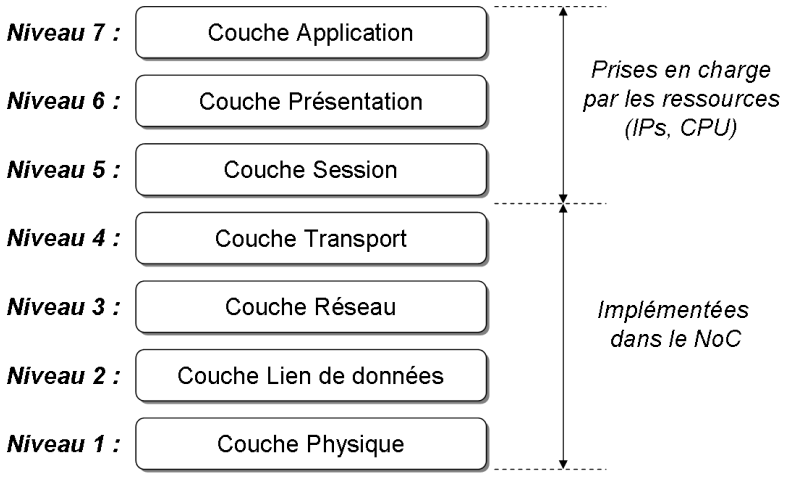


FIG. 1.7 : Le modèle OSI.

tocole de NoC. Dans la plupart des cas seules les quatre couches basses du modèle OSI sont implémentées dans les NoC. Les couches supérieures sont en général plus spécifiques à l'application et prises en charge par les ressources de calcul (IP, CPU) connectées au réseau.

Dans le reste du paragraphe, nous présentons rapidement les fonctionnalités des quatre couches implémentées dans les NoC, les trois couches restantes sont implémentées au niveau système et donc ne sont pas l'objet de ce paragraphe :

La couche physique correspond au niveau le plus bas dans le modèle OSI. Elle décrit les conditions matérielles du transfert de données : la tension et les temps caractéristiques des signaux, le nombre et la longueur des fils d'interconnexion entre les routeurs. Elle est le support de transmission de l'information au niveau bit ou mot entre deux routeurs.

La couche lien de données établit une connexion logique entre les routeurs ou entre un routeur et l'interface de réseau attachée (*hop-to-hop connection*). Elle

[2]Le modèle d'interconnexion en réseau des systèmes ouvertes de l'ISO (Organisation internationale de normalisation)

définit la façon de transmettre les informations entre deux routeurs et assure la fiabilité de communication sur les liens physiques. Donc, elle peut inclure des mécanismes de synchronisation, de contrôle de flux, et de correction d'erreur.

La couche réseau concerne l'échange des informations au niveau du paquet dans le réseau. Elle définit comment envoyer un paquet à travers le réseau d'un expéditeur à un destinataire. Elle prend en charge les mécanismes de commutation et les algorithmes de routage. Dans le cas d'une commutation *Wormhole*, elle assure le découpage des paquets en *flits*.

La couche transport est le niveau d'échange des informations entre deux unités des données. Elle découpe les blocs de données de la couche session en paquets de taille raisonnable pour le réseau. Elle prend en charge le contrôle de flux, un ensemble de mécanismes pour éviter la surcharge du réseau et régler le trafic. Les trois techniques suivantes sont souvent utilisées :

- *Technique de fenêtre :* une fenêtre glissante détermine que certains paquets peuvent circuler dans le réseau pendant une période prédéfinie.
- *Technique de contrôle de débit :* le débit d'envoi de chaque émetteur est contrôlé et limité.
- *Technique de crédit :* l'émetteur doit avoir reçu des crédits en provenance de la destination avant d'envoyer un paquet de données. Le destinataire envoie des crédits à l'émetteur en fonction de ses capacités de stockage de nouveaux paquets. A chaque paquet envoyé, l'émetteur doit baisser son compteur de crédits.

1.1.2.8 Interface réseau

Alors que le niveau d'intégration de système dans les SoC commençait à augmenter, les concepteurs de système ont fait appel à la réutilisation des unités de traitements pré-conçus et pré-vérifiés. Par conséquent, le besoin de modèles efficaces de conception standardisés (*plug-and-play*) a poussé le développement d'interfaces standardisées permettant de dissocier le développement des unités de traitement des architectures de communication [Keut00S, OcpIpSo]. Le protocole OCP (*Open Core Protocol*) [Ocp03O] a été conçu comme un moyen efficace de simplifier l'intégration en standardisant les interfaces des unités. ARM a présenté un paradigme semblable qui s'appelle AMBA AXI (*Advanced eXtensible Interface*) [Arm03A]. En fait, il définit un protocole point-à-point entre un maître et un esclave. Un autre paradigme d'interface utilisé pour l'intégration des unités dans les systèmes sur puce est DTL (*Device Transaction Level*) [Phil02D].

Dans les systèmes sur puce avec un NoC, l'interface réseau (*NI : Network Inter-*

face) est habituellement conçue comme la logique nécessaire pour adapter des unités de traitement au réseau. Elle fournit un ensemble de services à l'unité de traitement tout en masquant les mécanismes internes de fonctionnement du réseau [Mill04T]. La figure 1.8 décrit un ensemble d'une unité de traitement, son interface réseau et le routeur connecté.

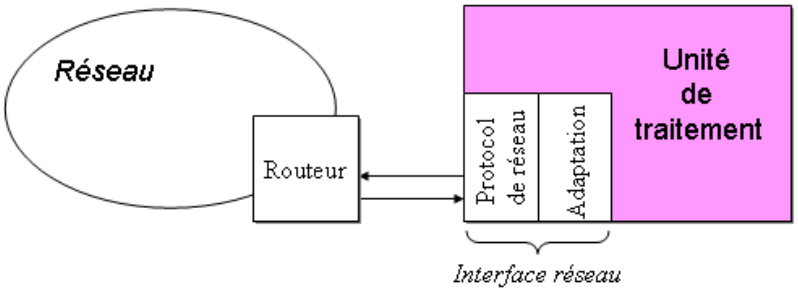


FIG. 1.8 : Interface réseau dans un système avec NoC.

L'interface réseau peut être découpée en deux parties principales. La partie connectée au réseau est indépendante de l'unité de traitement. Elle implémente les trois couches les plus basses du protocole de communication permettant la gestion des communications sur le réseau. L'autre partie connectée à l'unité de traitement implémente la couche transport. Elle réalise la conversion entre le protocole de communication au niveau réseau et les échanges de données au niveau de l'unité de traitement. Les protocoles OCP, AXI, ou DTL présentés ci-dessus peuvent être utilisés pour construire les interfaces entre les unités et les NIs.

1.2 État de l'art des architectures de réseaux sur puce

La section précédente a présenté la notion du réseau sur puce et les concepts relatifs. Dans cette section, nous allons présenter sélectivement différents travaux sur les NoC afin d'illustrer les concepts présentés et de montrer au lecteur un état de l'art sur les recherches de NoC. Ces travaux sont classés selon leurs caractéristiques en trois types suivants : *les réseaux sur puce « meilleur-effort »*, *les réseaux sur puce « qualité de service »*, et *les réseaux sur puce asynchrones*.

Les réseaux sur puce « meilleur-effort » sont des réseaux conçus dans le but d'avoir des bonnes performances moyennes. Tous les objets communicants dans le réseau se partagent la capacité de trafic du réseau, aucune priorité est accordée. Au contraire, les réseaux sur puce « qualité de service » intègrent des services garantis (cf. section 1.1.2.6) qui permettent de réserver des ressources de communication pour différents flux afin de garantir leurs débits et/ou leurs latences. Ce type de

réseau est très favorable pour les applications temps-réel telles que le traitement de la vidéo et de la voix. Finalement, les réseaux sur puce asynchrones sont des réseaux conçus et implémentés en utilisant les méthodologies de conception asynchrones, (cf. section 2.2). Il n'existe donc pas d'horloge globale dans ces réseaux. L'avantage de ce type de réseau est d'éviter des problématiques concernant à l'horloge dans la conception des SoC complexes.

1.2.1 Réseaux sur puce « Meilleur-effort »

Dans cette classe, nous présentons deux architectures de réseaux sur puce typiques, le réseau SPIN et le réseau OCTAGON.

– **SPIN (Scalable Programmable Integrated Network)**

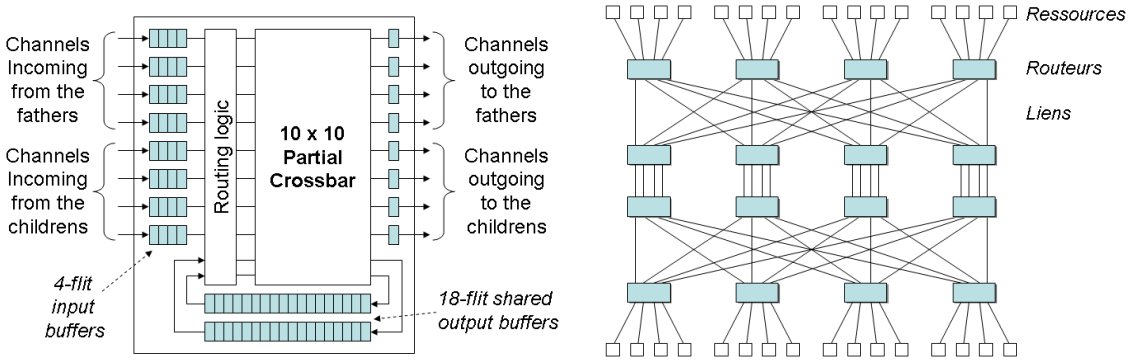
Le réseau SPIN est une architecture NoC avec une commutation de paquets développé par le laboratoire LIP6 de l'Université Pierre et Marie Curie (UPMC) [Guer00A]. Ce réseau s'appuie sur une topologie en arbre élargi afin d'obtenir une latence limitée grâce à un faible diamètre et un coût efficace pour le masque (*layout*). Par contre, cette topologie est peu flexible à intégrer et peu extensible. Le mode de commutation est de type *Wormhole* (cf. section 1.1.2.2) avec un algorithme de routage adaptatif et distribué (cf. section 1.1.2.4) ce qui impose un réordonnancement des paquets à la réception. Le contrôle de flux est assuré par un mécanisme de crédits sur des connexions point-à-point bidirectionnelles. Une connexion se compose de 32 bits de données et 4 bits de contrôle.

Le routeur (RSPIN) utilise des files d'attente en entrée ainsi que deux files centrales (une pour les paquets montants et l'autre pour les paquets descendants) pour réduire les conflits et éviter le phénomène du blocage en tête de ligne [Andr03S], cf. figure 1.9(a).

Le réseau SPIN a été réalisé en utilisant la technologie CMOS $0,13\mu m$ de STMicroelectronics [Andr03M]. Le routeur RSPIN occupe une surface de $0,24mm^2$ et un réseau SPIN32 correspondant à un réseau 16 routeurs avec 32 ports de connexion à ressources (figure 1.9(b)) nécessite $4,6mm^2$ de silicium, dont les FIFO occupent 30% de la surface.

– **OCTAGON**

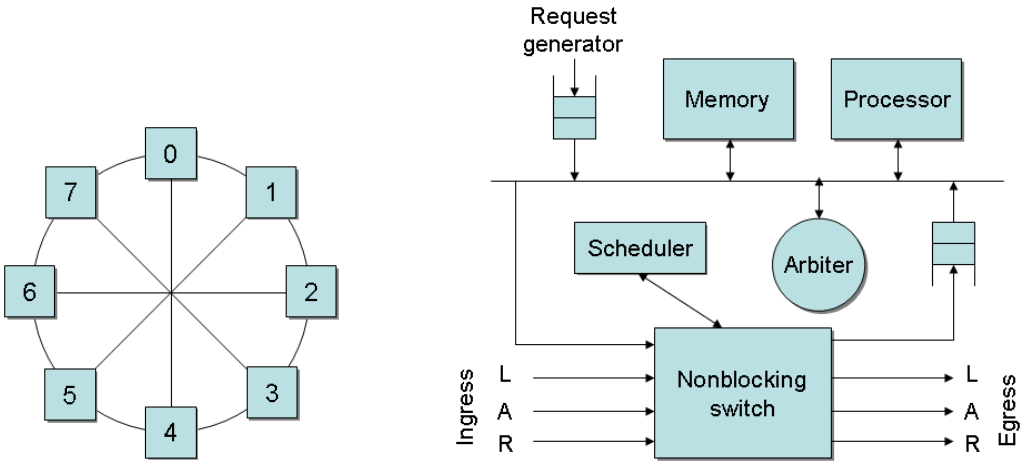
Le réseau OCTAGON a été proposé par STMicroelectronics et l'Université de Californie à San Diego en 2001 [Kari01O]. La topologie du réseau est un anneau octogonal avec des liens supplémentaires entre routeurs diamétralement opposés, cf. figure 1.10(a). La dimension d'une connexion est variable (N bits de données et



(a) (b)

FIG. 1.9 : Réseau SPIN : (a) Architecture du routeur RSPIN ; (b) Topologie du réseau SPIN à 32 ports.

3 bits de contrôle). L'architecture du réseau est destinée à répondre à des besoins en bande passante. Cette architecture offre un diamètre faible pour 8 éléments (la communication entre n'importe quelle paire de routeurs peut être exécutée après au plus deux sauts), un algorithme de routage simple, et un débit effectif supérieur à celui d'un *crossbar*.



(a) (b)

FIG. 1.10 : Réseau OCTAGON : (a) Topologie du réseau ; (b) Architecture du routeur.

Le mécanisme de commutation peut être circuit ou paquet (cf. section 1.1.2.2) suivant les choix de l'arbitre de connexion. Avec la commutation de paquets, la communication entre les ressources est exécutée en routant les paquets avec un algorithme de routage distribué et adaptatif (cf. section 1.1.2.4). Par contre, lorsque

la commutation de circuits est utilisée, un arbitre de réseau établit le chemin entier entre le routeur source et le routeur destinataire pour un certain nombre de cycles d'horloge. Plus de détails sur cette architecture et les comparaisons de cette architecture avec le modèle *crossbar* sur plusieurs caractéristiques (débit, probabilité de perte de paquets, la capacité de mettre à l'échelle de l'architecture) sont présentés dans [Kari02A].

1.2.2 Réseaux sur puce avec « Qualité de Service »

Pour montrer des architectures de type « Qualité de Service », nous présentons deux architectures de NoC typiques : QNOC et ÆTHEREAL.

– **QNOC (Quality-of-Service Network-on-Chip)**

L'architecture QNOC est proposée par l'Institut Technologie du Technion (Israël) [Bolo04Q]. La topologie du réseau est une maille à deux dimensions, elle peut être irrégulière (cf. figure 1.11(a)). L'algorithme de routage est de type distribué (cf.

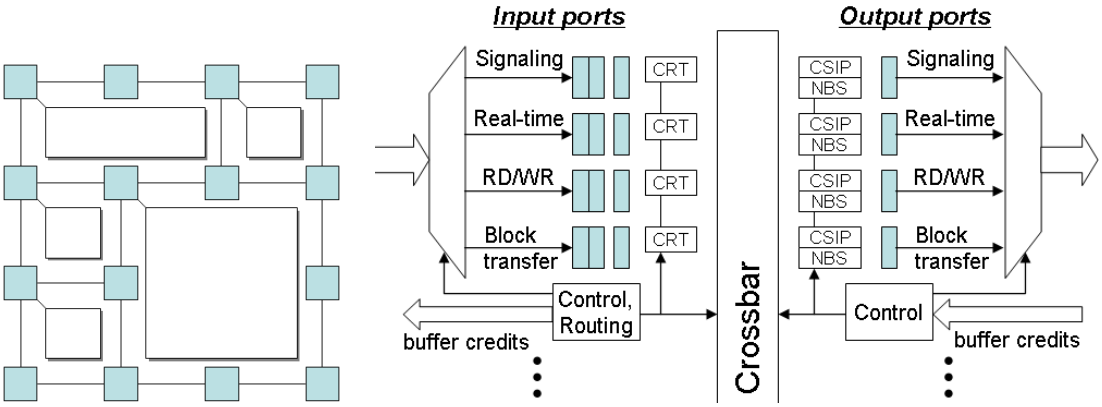


FIG. 1.11 : Réseau QNOC : (a) Topologie du réseau ; (b) Architecture du routeur.

section 1.1.2.4) en fonction des positions de l'émetteur et du récepteur de manière à ce que le chemin soit le même lors d'échanges bidirectionnels. Le contrôle de flux de message est basé sur un mécanisme de crédit.

Afin d'implémenter la QoS, le trafic est classé en quatre niveaux de services correspondant à des types de messages et des contraintes de latence et de débit souhaitées. Le niveau *Signaling* est utilisé pour les messages très courts avec la priorité la plus élevée, qui nécessitent une faible latence (par exemple, les messages de contrôle ou les messages d'interruption). Le niveau *Real-Time* garantit la bande

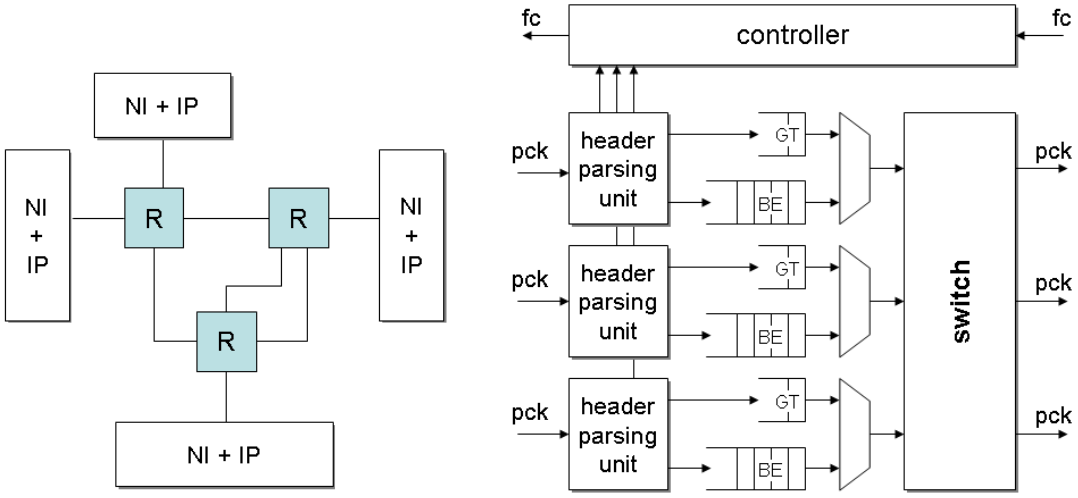
passante et la latence pour les applications temps-réel (audio, vidéo, etc.). Le niveau *Read/Write* est utilisé pour gérer des échanges de données tels qu'ils se font habituellement sur un bus. Le niveau *Block-Transfer* est employé pour transférer les messages longs et les gros blocs de données.

Pour implémenter quatre niveaux de service, le routeur QNOC inclut également quatre buffers séparés pour chaque niveau de service aux entrées et aux sorties, cf. figure 1.11(b). La taille des buffers d'entrée est variable tandis que la taille des buffers de sortie est d'une position.

– **ÆTHEREAL**

Le réseau ÆTHEREAL est proposé par les laboratoires de recherche de Philips (Eindhoven aux Pays-Bas) [Goos03G]. Le but du réseau ÆTHEREAL est de développer un réseau hétérogène avec une qualité de service garantie. L'architecture du réseau se compose donc de deux mécanismes de commutation (paquet et circuit) pour assurer à la fois un service meilleur-effort (*BE : Best-Effort*) et de proposer des services garantis, en particulier de débit (*GT : Guarantee Throughput*).

La topologie utilisée dans le réseaux ÆTHEREAL est à deux dimensions mais ne se limite pas aux réseaux maillés, cf. figure 1.12(a). L'algorithme de routage est



(a) (b)

FIG. 1.12 : Réseau AETHEREAL : (a) Un exemple de la topologie ; (b) Architecture du routeur.

déterministe et de type source, cf. section 1.1.2.4. Le service meilleur-effort est géré de façon classique par la commutation de paquets, et le mode de commutation est de type *Wormhole*. Pour avoir les services garantis, la commutation de circuits couplée au multiplexage temporel est utilisée. Ce mécanisme de commutation permet de

créer des connexions avec un débit et une latence garantis. Plus de détails de ce mécanisme sont présentés dans [Rijp03T].

Le routeur ÆTHEREAL (cf. figure 1.12(b)) a été développé et synthétisé en technologie CMOS $0,12\mu m$. Ce routeur se compose de 5 ports d'entrée/sortie, un flot *Wormhole*, une file d'attente de 8 *flits* pour chaque entrée, des *flits* de 3 mots de 32 bits et une table de 256 *time slots*. Le routeur occupe une surface de $0,26mm^2$ avec une fréquence de fonctionnement de $500MHz$.

1.2.3 Réseaux sur puce asynchrones

Nous avons évoqué précédemment (cf. section 1.1.1) que la propagation d'horloge allait être de plus en plus problématique dans les circuits intégrés. C'est pourquoi des architectures de réseaux sur puce asynchrones ont été proposées. Dans cette sous-section, nous présentons quatre exemples concrets d'architectures de NoC asynchrones : CHAIN, MANGO, NEXUS, et QNOC asynchrone.

– ***CHAIN (CHip Area INterconnect)***

Le réseau CHAIN est développé par l'Université de Manchester [Bain02C]. C'est un réseau implémenté entièrement en logique asynchrone de type insensible aux délais [Uddi86A]. La topologie du réseau est variable. Le réseau est basé sur des liens à grande vitesse insensibles aux délais utilisant le codage de données *1-of-5* (2 bits/*flit*) combiné avec un protocole de signalisation de type RTZ (*Return-to-Zero*), cf. section 2.2.1.1. Les données sont échangées en utilisant le mécanisme de commutation de paquets et l'algorithme de routage est de type source (cf. section 1.1.2.4).

Le réseau CHAIN a été implémenté en technologie CMOS $0,18\mu m$ avec un débit de $1Gbps$ par lien de réseau. Dans cette implémentation, la topologie de réseau de type Mux-Demux est utilisée, cf. figure 1.13. Les cibles de cette architecture sont les applications basse consommation. Cette architecture a permis de concevoir et tester un système pour carte à puce [Bain04T].

– ***MANGO (Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interface)***

L'architecture du réseau MANGO est proposée par l'Université Technique du Danemark. Le but est de fournir une architecture supportant du trafic meilleur-effort et également un mode connecté pour des services garantis. Le routeur se compose donc de deux parties, une partie pour les services garantis (appelé routeur GS), l'autre pour le trafic meilleur-effort (appelé routeur BE), cf. figure 1.14. La commutation de paquets est utilisée pour le routeur BE et la commutation de circuits est utilisée

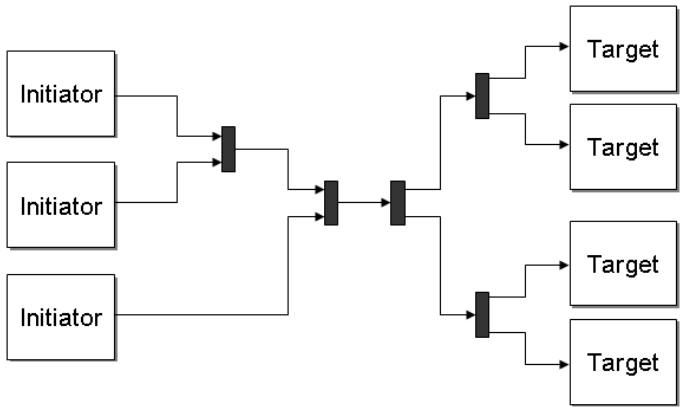


FIG. 1.13 : Topologie de type Mux-Demux utilisée dans le réseau CHAIN.

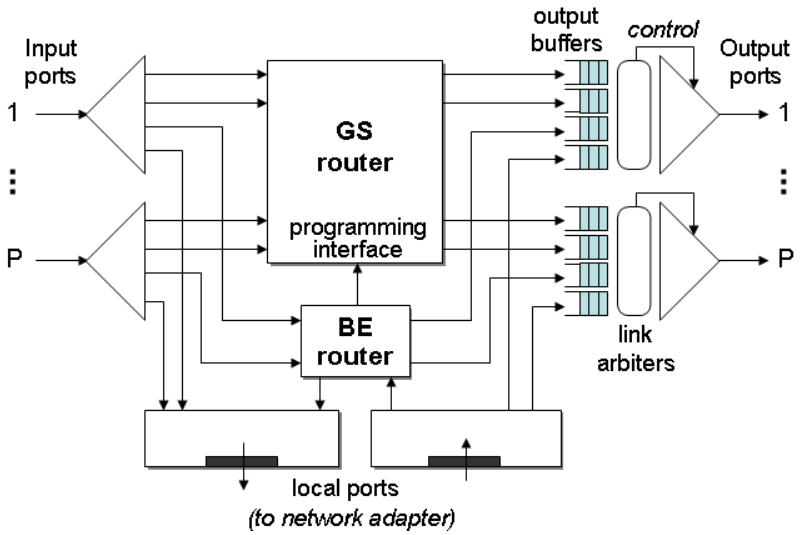


FIG. 1.14 : Architecture du routeur du réseau MANGO.

pour le routeur GS. L'algorithme de routage est de type source, déterministe.

La topologie du réseau est une maille à deux dimensions. L'architecture est implémentée en logique asynchrone afin d'éviter les problèmes liés à la propagation du signal d'horloge. Le protocole de poignée de main est de type 4-phase (cf. section 2.2.1.1). Le codage de donnée utilisé est de type « 1-of-4 » [Bain01D] qui permet de réduire la consommation d'énergie. Une implémentation du routeur est présentée en technologie $0,12\mu m$ avec une surface de $0,188mm^2$ [Bjer05A].

– ***NEXUS Interconnect***

Fulcrum Microsystems a développé une architecture asynchrone d'interconnexion appelée NEXUS [Line04A]. Cette architecture vise à résoudre des problèmes de

multi-horloges dans les grands SoCs. NEXUS est un réseau de type *crossbar* 36-bit avec 16 ports d'interconnexion. Il se relie aux ressources synchrones par convertisseurs asynchrones/synchrones.

L'architecture NEXUS a été mise en œuvre dans le processus de $130nm$ de Taiwan Semiconductor Manufaturing Company (TSMC) et elle occupe une surface de $4,15mm^2$. La vitesse maximale est $1,35GHz$ à $1,2V$.

– ***Asynchronous QNOC***

Suite à son réseau QNOC synchrone, l'Institut Technologie du Technion (Israël) a proposé une implémentation asynchrone de QNOC [Dobk05A]. La topologie du réseau est aussi une maille à deux dimensions et l'algorithme de routage est aussi de type source, déterministe.

Le routeur asynchrone a été comparé avec le routeur synchrone avec la même fonctionnalité, et la même bibliothèque de cellules. Donc, la surface de routeur asynchrone est inférieure à la surface du routeur synchrone tandis la performance est supérieure. Par exemple, un routeur asynchrone avec 4 niveaux de services occupe une surface $0,47mm^2$ et il a un débit maximal de $75,2Mflits/s$ tandis qu'un routeur synchrone avec le même nombre de services occupe $0,96mm^2$ et possède un débit maximal de $67,6Mflits/s$.

1.2.4 Synthèse de l'état de l'art sur les NoC

Les sous-sections précédentes ont présenté différents exemples de réseaux sur puce. Afin de donner une vue globale de l'état de l'art actuel sur les NoC, nous présentons dans cette sous-section un tableau synthétique d'un ensemble d'architectures de réseaux sur puce, cf. tableau 1.1.

De ce tableau, nous pouvons voir que la topologie la plus utilisée dans les architectures NoC est la maille à deux dimensions. La raison vient de ses avantages tels que la facilité de sa mise en œuvre sur silicium, la possibilité de mettre à l'échelle, les stratégies de routage simples.

La plupart des NoC utilisent le mécanisme de communication en paquet et le mode de routage prédominant est de type *Wormhole*. L'algorithme de routage est très variable mais les algorithmes de type source sont préférés car ils sont réalisées plus facilement. La QoS des NoC est garantie par une des deux solutions principales : soit le mécanisme de la commutation de circuits est utilisée, soit des canaux virtuels sont installés dans les routeurs.

TAB. 1.1 : Tableau récapitulatif des architectures NoC.

NoC	Topologie	Commutation	Routage	Liens	QoS	Référence
ÆTHEREAL	2D variable	Circuit et Paquet	Source	32 bits	Circuit	[Goos03G] [Rijp03T]
aSOC	2D maillée	Circuit	Tabulé dans les routeurs	32 bits	Circuit	[Lian00A]
CHAIN	Variable	Paquet	Source			[Bain02C]
HERMES	2D maillée	Wormhole	XY adaptatif	8 bits donnée, 2 bits contrôle		[Mora04H]
IMEC NoC	2D tore	Wormhole	XY, déterministe	16 bits donnée, 3 bits contrôle		[Mare02I]
MANGO	2D maillée	Circuit et Paquet	Source	32 bits donnée, 2 bits contrôle	Circuit	[Bjer05A]
NEXUS	2D maillée	Circuit	Source	32 bits donnée, 4 bits contrôle		[Line04A]
NOSTRUM	2D maillée	Paquet	Hot-potato	128 bits donnée, 10 bits contrôle	Canaux virtuels	[Kuma02A] [Mill04T]
OCTAGON	Anneau avec cordes	Circuit ou Paquet	Source	N bits donnée, 3 bits adresse		[Kari01O] [Kari02A]
PROTEO	Anneau variable	VCT	Source	Variable		[Saas02I] [Saas03B]
QNOC	2D maillée	Wormhole	XY	16 bits donnée, 10 bits contrôle	Canaux virtuels	[Bolo04Q]
SoCBUS	2D maillée	Circuit	Distribuée et adaptatif	16 bits donnée, 3 bits contrôle		[Wikl03S] [Sath03D]
SoCIN	2D maillée/ tore	Wormhole	XY source	N bits donnée, 4 bits contrôle		[Zefe03S] [Zefe04R]
SPIN	Arbre élargi	Wormhole	Distribuée et adaptatif	32 bits donnée, 4 bits contrôle		[Guer00A] [Andr03M]
ANOC (FAUST)	2D maillée	Wormhole	Source	32 bits donnée, 2 bits contrôle	Canaux virtuels	[Beig05A] [Latt07A]

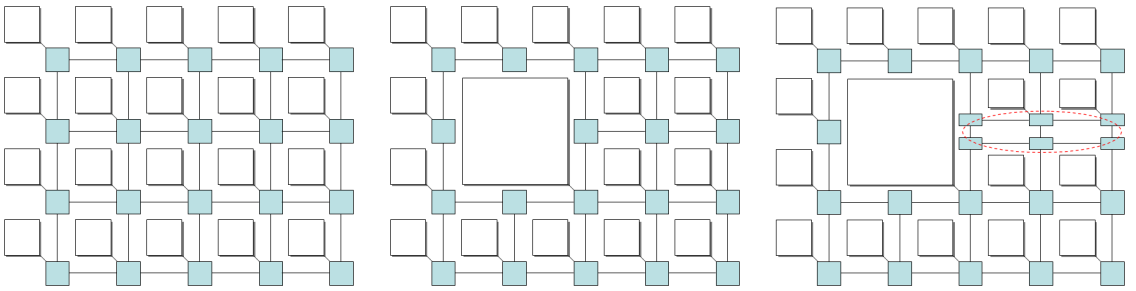
1.3 ANOC : un réseau sur puce asynchrone dans un système sur puce pour les applications de télécoms

Les sections précédentes nous ont donné un ensemble de concepts de base de NoC et une vue globale sur la recherche actuelle sur les NoC. Dans cette section, nous allons étudier tout particulièrement l'architecture du réseau sur puce asynchrone développée au LETI dans le cadre du projet FAUST (***F****lexible* ***A****rchitecture of* ***U****nified* ***S****ystem for* ***T****elecom*). Ce réseau est appelé ANOC. Son objectif est de proposer une structure de communication flexible, performante (bonne qualité de service en termes de latence et de débit) avec des mécanismes de communication assez faciles à réaliser. Le réseau ANOC est utilisé pour construire un système sur

puce qui a la possibilité de gérer les besoins des applications de télécoms (B3G[3] ou 4G[4]) de l'avenir, tels que l'augmentation de la complexité, les mécanismes de reconfiguration, et les plateformes d'intégration ouvertes et flexibles.

1.3.1 Topologie du réseau

Le réseau ANOC est une architecture d'interconnexion pour les applications de types flots de données. Il est utilisé pour établir un environnement de communication entre des ressources de traitement dans un système sur puce hétérogène. Afin de limiter la complexité de l'architecture et d'avoir de meilleures performances temporelles, la topologie du réseau ANOC est une maille à deux dimensions, cf. figure 1.15(a).



(a) Topologie régulière (b) Topologie adaptée aux ressources (c) Topologie adaptée au débit

FIG. 1.15 : La topologie pour ANOC.

Dans cette topologie, chaque routeur est relié à ses quatre voisins et à la ressource de traitement la plus proche. Comme évoqué dans la section 1.1.2.1, cette structure nous permet de développer facilement des stratégies de routage en termes de mode de routage et d'algorithme de routage, passe à l'échelle avec la taille système, et s'implémente aisément sur silicium. De plus, avec cette structure les routeurs peuvent être génériques car tous les routeurs sont identiques.

Le réseau étant utilisé dans un système hétérogène (les tailles des ressources sont variables), la topologie du réseau doit être capable de s'adapter à la taille des ressources. Dans ce cas-là, la topologie peut être remplacée par une topologie irrégulière avec certains liens supprimés, cf. figure 1.15(b).

Cependant, la suppression des liens affecte le débit global du réseau et peut créer localement des phénomènes de congestion. Pour y remédier, il est possible

[3]Beyond 3G
[4]Quatrième génération

d'augmenter la bande passante locale en doublant les routeurs et en adaptant le routage, cf. figure 1.15(c).

1.3.2 Mécanismes de communication

Le mécanisme de communication utilisé pour le réseau ANOC est de type paquet et le mode de routage est de type *Wormhole* (cf. section 1.1.2.2). Les données sont encapsulées en paquets avant d'être envoyées dans le réseau. Un paquet est toujours composé d'un *flit* d'en-tête et éventuellement d'un ou plusieurs *flits* de données. Un *flit* est une unité de donnée qui se compose de 34 bits (32 bits de donnée et 2 bits de contrôle). Les formats de ces *flits* sont décrits dans la figure 1.16.

BoP	EoP	Payload	*Path–to-Target*
33	32	31 18	17 0

(a) *Flit* d'en-tête

BoP	EoP	Payload
33	32	31 0

(b) *Flit* de donnée

FIG. 1.16 : Les formats des *flits*.

Les deux bits de poids fort de chaque *flit*, BoP *(Begin-of-Packet)* et EoP *(End-of-Packet)* spécifient le type du *flit*. Le *flit* d'en-tête est identifié par le bit BoP à l'état 1. Le dernier *flit* est repéré par le bit EoP à état 1. Si les deux bits BoP et EoP sont à 1, le paquet n'a qu'un *flit*. Si les deux bits BoP et EoP sont à 0, c'est un *flit* au milieu du paquet. Le champ *path-to-target* dans le *flit* d'en-tête spécifie le chemin de routage du paquet dans le réseau. C'est un vecteur qui contient des directions successives. Chaque direction est codée sur deux bits : 00 pour le Nord, 01 pour l'Est, 10 pour le Sud, et 11 pour l'Ouest. On note qu'un paquet n'est pas autorisé à faire demi-tour. La ressource est adressée en renvoyant le paquet dans la direction dont il provient. Par exemple, si le paquet venant sur l'entrée Ouest contient une information de routage égale à 11, il sera routé vers la ressource de traitement. La longueur actuelle du champ *path-to-target* correspond à 9 traversées de routeur, soit 18 bits.

L'algorithme de routage est déterministe et de type source (cf. section 1.1.2.4). Les chemins de routage sont pré-calculés par un algorithme de routage « Turn Model » [Glas94T] qui permet d'éviter les interblocages.

La stratégie de stockage utilisée dans le réseau ANOC est une stratégie de file d'attente avec deux canaux virtuels afin d'améliorer la latence et le débit du réseau (cf. section 1.1.2.3). Ces canaux virtuels sont utilisés pour définir deux niveaux de priorité différents. Le canal virtuel 0 (VC0) possède une priorité plus haute que le canal virtuel 1 (VC1) et les paquets envoyés sur ce canal sont donc autorisés par

l'arbitre à passer avant les paquets envoyés sur le canal virtuel 1 quand ils sont concurrents.

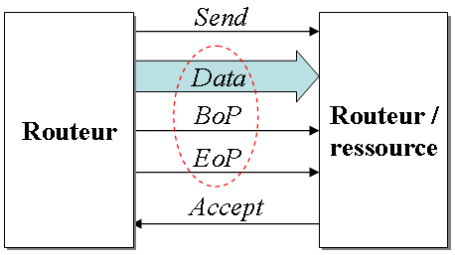
Le réseau ANOC est mis en œuvre en logique asynchrone de type quasi-insensible aux délais (cf. section 2.2.1.4). Il n'y a alors plus de signal d'horloge, les routeurs se synchronisent mutuellement entre eux par un mécanisme de poignée de main (*handshake protocol*). L'architecture est bien adaptée aux systèmes de type GALS[5]. Dans un système GALS, un certain nombre d'îles synchrones communiquent de façon asynchrone entre elles. C'est-à-dire qu'il n'existe pas de signaux globaux et l'ensemble des informations est transmis par le réseau entre les différents îlots synchrones formés par les ressources.

1.3.3 Protocole de communication

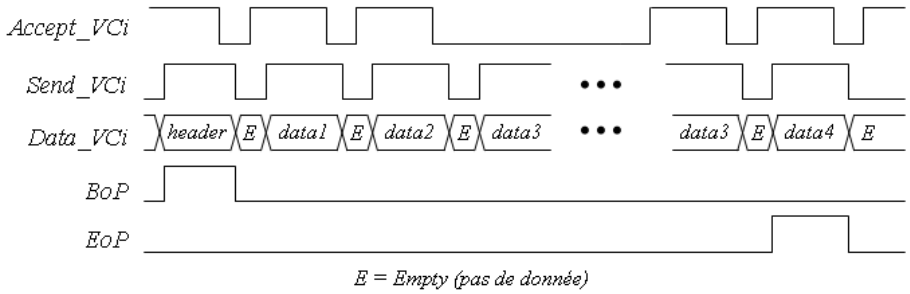
Le protocole de communication du réseau ANOC est hiérarchisé en six niveaux [Thon06N] :

- **La couche physique :** Elle correspond au *niveau bit* et elle décrit les conditions matérielles du transfert de données sur un lien. A ce niveau, le protocole asynchrone 4-phase RTZ est utilisé avec le schéma de codage de données « *m-of-n* »(cf. section 2.2.1.1). Le codage « 1-of-4 » (cf. section 2.2.1.3) est utilisé pour réaliser les flots données de 34 bits, les signaux de synchronisation sont réalisés par d'autres codages tels que le codage double-rail, single-rail.
- **La couche lien de données :** Correspondant au *niveau flit*, elle est le niveau d'échange local entre deux routeurs ou d'échange entre un routeur et une ressource d'un *flit* de donnée. Cet échange est géré par un mécanisme de synchronisation basé sur des signaux d'interface : *send* et *accept*, cf. figure 1.17(a). Ce mécanisme est donc appelé le protocole « *send-accept* ». Pour mettre en œuvre ce protocole pour deux canaux virtuels, il est nécessaire d'utiliser deux signaux *send* et deux signaux *accept*. L'expéditeur peut envoyer un nouveau *flit* sur le canal virtuel i avec $send\langle i\rangle = 1$ si le récepteur a indiqué $accept\langle i\rangle = 1$ au cycle précédent, cf. figure 1.17(b). Avec ce protocole, les transactions de *flit* sont réalisées dans plusieurs canaux virtuels avec l'assurance qu'un canal physique est libre.
- **La couche réseau :** Cette couche correspond au *niveau paquet*. Elle est le niveau d'échange des paquets à travers le réseau. Les paquets sont des messages cohérents, constitués d'une série de *flits* contigus. Dans cette couche, on définit toutes les informations nécessaires au bon acheminement du message à travers le réseau, dans le premier *flit* du paquet, appelé *flit* d'en-tête (*header flit*), cf.

[5]Globalement Asynchrone, Localement Synchrone



(a) Interface « *send-accept* ».



(b) Chronogrammes de l'échange des données sur canal virtuel i.

FIG. 1.17 : Echange des données entre deux routeurs ou entre un routeur et une ressource

section 1.3.2. Ce *flit* contient également des informations utiles aux couches supérieures.

- **La couche transport :** C'est le niveau d'échange entre deux ressources. Les blocs de données sont découpés en paquets de taille raisonnable pour le réseau. Ce niveau permet aussi de garantir que le réseau ne se bloquera pas à cause d'un engorgement. Une communication est définie entre une ressource émettrice et une ressource réceptrice. Les deux ressources s'échangent des paquets de service pour réguler la communication. La gestion de flux est faite en utilisant la technique de crédit (cf. section 1.1.2.7).
- **La couche session :** Elle synchronise les ressources au niveau des tâches à exécuter. Elle correspond au niveau bloc de données. Ce niveau assure la cohérence des traitements entre les ressources d'une même chaîne fonctionnelle, et permet de faire tourner simultanément plusieurs tâches sur le réseau.
- **La couche application :** C'est le niveau de définition des données utilisateur. A ce niveau, les ressources s'échangent des messages de différents types, permettant le transfert de flux de données, le contrôle des ressources de traitement.

Dans le réseau ANOC, toutes les couches basses jusqu'à la couche session sont matérielles. Le processeur vient configurer les couches transport et session à travers

la couche application

1.3.4 Routeur du réseau

Les routeurs sont les éléments de base de l'architecture du réseau. Ils ont cinq ports bidirectionnels de 34-bit qui se relient aux quatre routeurs voisins et à l'unité la plus proche. Les routeurs sont conçus et mis en œuvre en logique asynchrone. La figure 1.18 présente simplement l'architecture du routeur.

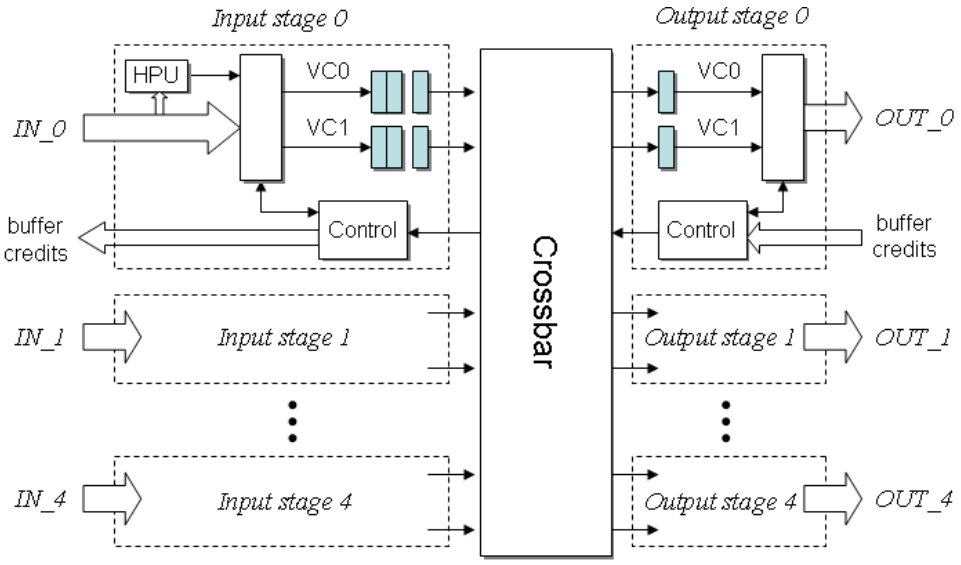


FIG. 1.18 : Architecture du routeur du réseau ANOC.

De cette figure nous pouvons voir que l'architecture du routeur se compose de cinq unités d'entrée, cinq unités de sortie, et un *crossbar*. Le trafic du système est classé en deux niveaux de priorité et implémenté en utilisant deux canaux virtuels, VC0 et VC1, cf. section 1.3.2. Chaque unité d'entrée inclut trois parties principales : un DEMUX, les *buffers* pour chaque canal virtuel, une unité d'analyse des *flits* d'en-tête (*HPU : Header Parsing Unit*) pour le routage de type source, et un bloc de contrôle. Chaque unité de sortie inclut également trois parties : un MUX, les *buffers* pour chaque canal virtuel, et un bloc de contrôle. Ces blocs de contrôle prennent en charge les activités de routage et la gestion de flux utilisant la technique de crédit. Le crossbar utilisé dans ce routeur établit des connexions point-à-point entre les unités d'entrée et les unités de sortie.

1.3.5 Intégration des unités de traitement

Les unités de traitement sont intégrées au réseau en utilisant une architecture modulaire d'interface réseau (cf. section 1.1.2.8). Dans le réseau ANOC, l'interface réseau a une double fonctionnalité. Elle gère à la fois les communications entre l'unité de traitement et le réseau mais également les configurations de routage pour les données sortantes. L'architecture de l'interface est paramétrable. Elle est constituée de différents éléments qui peuvent être assemblés et configurés en fonction des besoins de l'unité de traitement. L'interface réseau est implémentée en logique synchrone [Lema06A].

Le réseau est implémenté en logique asynchrone tandis que les unités de traitement sont synchrones, une interface asynchrone ⇔ synchrone est indispensable. C'est une architecture de synchronisation entre le domaine asynchrone et le domaine synchrone qui utilise des FIFO [Beig06D]. Elle est appelée interface SAS.

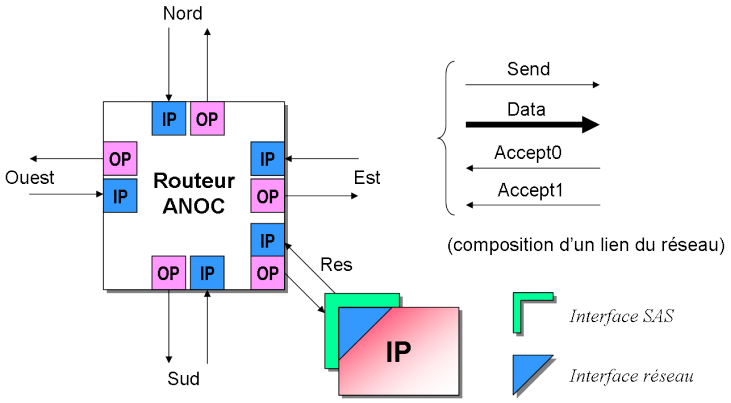


FIG. 1.19 : Interconnexion entre un routeur et une unité de traitement en utilisant l'interface réseau et l'interface SAS.

La figure 1.19 présente les interconnexions entre une unité de traitement et un routeur du réseau en utilisant les deux interfaces mentionnées, l'interface réseau et l'interface SAS.

1.3.6 Mise en œuvre du réseau ANOC : le circuit FAUST

Les concepts sur le réseau ANOC que nous venons de présenter ont été implémentés et validés dans un système sur puce conçu au LETI, appelé le circuit FAUST. Ce circuit a été réalisé dans la technologie CMOS $0,13\mu m$ de STMicroelectronics et les premiers exemplaires ont été testés en janvier 2006 [Latt07A]. La deuxième version de FAUST est en cours et elle sera implémentée dans la technologie CMOS

65nm de STMicroelectronics mi-2008. La conception du circuit par le laboratoire IAN s'est donc faite en parallèle de l'avancement de cette thèse.

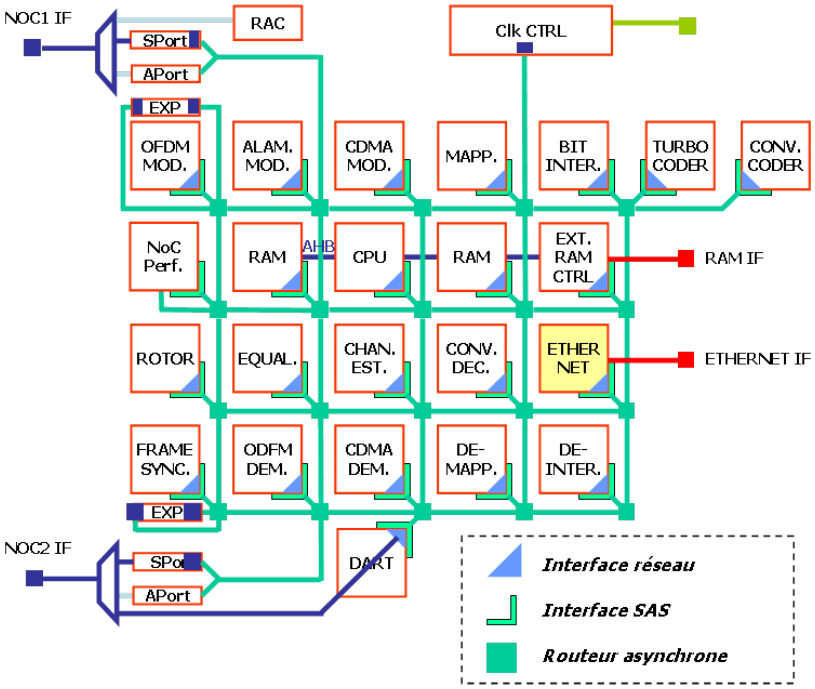


FIG. 1.20 : Le circuit FAUST.

La figure 1.20 présente simplement le circuit FAUST. Il se compose d'un réseau asynchrone de 20 routeurs et de 23 unités de traitement. Les unités de traitement ont été spécifiées et sélectionnées dans le but de réaliser les traitements correspondant à la couche physique des systèmes de télécommunication associés aux projets MATRICE [Matrice] et 4MORE [4More]. Le contrôle du circuit est réalisé par un processeur ARM946E-S [ArmES]. Le processeur dispose d'un accès direct à deux mémoires embarquées et une mémoire externe via un bus AHB[6] [Amba].

Le circuit communique avec l'extérieur grâce à un port Ethernet. Il existe également deux liens bidirectionnels du réseau qui sont connectés à des ports primaires d'entrée et de sortie. Ceci permet d'étendre le réseau à l'extérieur du circuit. Dans le cadre du projet 4MORE, une carte prototype a été réalisée incluant deux circuits FAUST et deux composants programmables (*FPGA Xilinx Virtex-4* [Virtex]).

[6]Advanced High-speed Bus

1.4 Conclusion

Dans ce premier chapitre, nous avons présenté un large panorama sur les réseaux sur puce. Après avoir identifié les problématiques des architectures de communication actuelles, nous avons introduit le paradigme NoC comme une bonne solution pour la communication dans les systèmes sur puce, puis nous avons étudié les concepts de base des NoC. Ensuite, nous avons détaillé les caractéristiques des réseaux sur puce en montrant des exemples concrets issus de travaux de recherche actuels.

Il est clair que le paradigme NoC permet de construire des architectures flexibles et extensibles de systèmes sur puce. Les communications et les traitements sont bien découplés grâce au protocole réseau. L'intégration des unités de traitement est ainsi facilitée grâce à des interfaces réseaux bien définies. Le défi est désormais de réaliser le test matériel des NoC avant de les mettre sur le marché. Ce défi est d'autant plus difficile pour le test des NoC asynchrones, pour lesquels les méthodologies de test des circuits synchrones ne s'appliquent pas. Dans le chapitre suivant, nous allons présenter le test des systèmes sur puce dans le cas général et le cas particulier des NoC, nous introduisons les principaux concepts des circuits asynchrones afin de présenter les spécificités du test de ces circuits.

Chapitre 2

Test des Systèmes sur Puce Synchrones et Asynchrones

Les systèmes sur puce doivent être testés pour s'assurer de leur bon fonctionnement avant d'être délivrés aux clients. Le test d'un système sur puce consiste à mettre en évidence son éventuel mauvais fonctionnement dû à une défaillance physique. Les systèmes peuvent être analogiques, numériques, ou mixtes. Dans le contexte du travail de cette thèse, ne sont abordés que les systèmes numériques. Pour tester un système sur puce numérique, deux types de test sont appliqués. Le premier consiste à effectuer des mesures paramétriques telles que mesures de courant, tension, temps de montée et de descente... Ce test est donc appelé *test paramétrique*. Il fait intervenir des techniques de métrologie particulières et ne concerne donc pas notre travail. Le deuxième test consiste à inspecter les données logiques manipulées par le circuit et à les comparer avec les valeurs attendues. Il est donc appelé *test logique*. Ce type de test est le cœur de notre travail.

Dans ce chapitre, la première section va aborder les concepts de test, les exigences et les défis du test des systèmes sur puce, puis différentes techniques développées pour y répondre. Ensuite, le test des systèmes sur puce intégrant un réseau est abordé et discuté à la fin de cette section.

Par ailleurs, l'architecture de réseau implémentée en logique asynchrone présente d'autres nouveaux défis. Les concepts de base de la conception asynchrone et le test des circuits asynchrones seront donc abordés dans la deuxième section. L'objectif est de fournir aux lecteurs des notions suffisantes pour suivre notre travail dans les chapitres suivants. Enfin, la dernière section abordera la méthode de test actuelle pour le réseau ANOC et ses limitations. A cause de ces limitations, nous allons

proposer dans les chapitres suivants de nouvelles méthodes de test efficaces pour les réseaux sur puce, en particulier, pour ce réseau ANOC.

Pour simplifier la rédaction, dans la suite, on utilisera le terme *réseau sur puce* ou *NoC* pour indiquer un système sur puce intégrant un réseau sur puce et le terme *réseau de communication* ou *réseau* tout court pour indiquer seulement le réseau (sans IP).

2.1 Test des systèmes sur puce

Avant d'aborder les problèmes du test des systèmes sur puce, nous présentons dans la sous section suivante plusieurs terminologies de test utilisées dans ce mémoire afin de rendre le document plus clair et compréhensible.

2.1.1 Concepts de base du test matériel

Les entrées du circuit sous test sont appelées *entrées primaires* et peuvent être contrôlées de l'extérieur. Pareillement, les sorties du circuit sous test sont appelées *sorties primaires* et peuvent être observées de l'extérieur.

La *contrôlabilité* caractérise la facilité de positionner un nœud à une valeur logique prédéfinie à partir des entrées primaires du circuit. L'*observabilité* représente la facilité de vérifier sur les sorties primaires du circuit la présence d'une valeur sur un nœud. La *testabilité* combine les notions de contrôlabilité et d'observabilité pour quantifier la facilité avec laquelle le circuit pourra être testé. Un circuit avec une haute testabilité a un degré élevé d'observabilité et de contrôlabilité.

Une *erreur* dans un circuit se produit quand le circuit dévie du comportement normal. Dans ce cas là, on peut dire que le circuit est défaillant. Le but du test est de déterminer les circuits défaillants du fait de la présence d'un ou plusieurs défauts (ou défaillances) physiques. Un *défaut physique* est un état physique qui peut empêcher le fonctionnement approprié d'un composant du circuit. Afin d'être détectés en utilisant le *test logique*, les défauts physiques sont traduits en fautes logiques. Une *faute logique* est donc la modélisation d'un défaut physique au niveau logique. Dans un modèle de circuit, c'est un état logique qui empêche l'opération d'un nœud ou un ensemble de nœuds du circuit. Une faute sera détectable si le circuit produit un résultat incorrect pour une valeur appliquée à ses entrées primaires. Cette valeur d'entrée est appelée *vecteur de test*. Le processus pour déterminer si un circuit contient une ou plusieurs fautes s'appelle la *détection de faute*.

Lors d'un test, un ensemble de vecteurs de test est appliqué au circuit sous test afin de détecter autant de fautes que possible. L'efficacité d'un test est mesurée

par la *couverture de faute*, qui est le rapport des fautes détectées par les fautes détectables dans le circuit. La *longueur du test* est le nombre de vecteurs de test. Le *temps d'application du test* est alors le temps pour appliquer tous les vecteurs de test et pour observer les résultats.

Un *modèle de faute* est une représentation abstraite employée pour décrire l'ensemble des fautes. Un bon modèle de faute a deux attributs principaux : simplicité (pour faciliter le processus de génération des vecteurs de test) et représentativité (pour permettre aux vecteurs de test générés de détecter autant de défauts que possible).

Le test est une étape indispensable pour de nombreuses raisons telles que : la qualité du circuit et le coût de production. Le test des défauts de fabrication peut être réalisé de deux façons différentes : le test structurel et le test fonctionnel.

Le *test structurel* consiste à vérifier l'existence ou non de certains défauts parmi les plus probables en utilisant la description structurelle du circuit. Ce type de test est largement utilisé pour tester les circuits intégrés et est supporté par les outils CAO (Conception-Aidée par Ordinateurs).

Le *test fonctionnel* consiste lui à valider le circuit sous test selon ses caractéristiques fonctionnelles. Ceci est étroitement lié au processus de vérification fonctionnelle qui détermine si le circuit spécifié par la description en portes logiques (*netlist*) répond aux caractéristiques fonctionnelles. Ce test fonctionnel est réalisé en supposant que le circuit a été conçu correctement. Les vecteurs de test utilisés dans l'étape de vérification fonctionnelle sont habituellement réutilisés pour le test fonctionnel. Un des problèmes du test fonctionnel est qu'il ne garantit pas un taux de couverture. De plus, il est couramment admis qu'il ne puisse pas détecter certaines défaillances. Cependant, le test fonctionnel peut parfois n'être que la seule solution envisageable, en particulier lorsque l'on ne connaît pas la structure interne du circuit.

Le coût de test est souvent évalué de 30 à 50% (peut être plus) du coût total de développement d'un circuit [Pitt84T, Ivan96D]. Pour réduire ce coût et faciliter le test, le processus de test est réalisé non seulement à la fin du cycle de production mais également considéré pendant le flot de conception du circuit [Wei97T, Mour00O]. Plusieurs techniques sont proposées et utilisées pour rendre le test des circuits économiquement viable. Ces techniques sont rassemblées sous le terme Conception en Vue du Test (CVT) (ou « *Design for Test/Testability* » en Anglais). C'est une approche de conception spécifique qui consiste à insérer de la logique supplémentaire dans le circuit afin d'améliorer sa testabilité. Les avantages des techniques CVT sont nombreux : réduire les efforts de test, réduire le coût des équipements de test, diminuer le temps d'arrivée sur le marché, augmenter la qualité du produit, etc.

Cependant, la logique additionnelle peut augmenter le coût des produits et réduire les performances temporelles des produits [Wei97T].

2.1.2 Exigences et défis du test des systèmes sur puce

Les systèmes sur puce sont de plus en plus complexes car les concepteurs intègrent de plus en plus d'unités de traitement (IP) afin de répondre aux besoins des applications. Ceci a changé les méthodologies de conception et de test. On a vu apparaître la conception des systèmes sur puce à base d'IP (*core-based SoC design*). Avec cette approche, les IP précédemment conçues peuvent être réutilisées pour les conceptions suivantes afin de réduire les temps de conception. Malheureusement, la conception à base d'IP engendre de nouveaux défis pour le test des systèmes. Pour tester ces systèmes, il faut trouver des solutions de test efficaces pour chaque IP et ensuite réaliser le test au niveau système.

De plus, pour augmenter la productivité de conception et réduire le temps d'arrivée sur le marché, la réutilisation des IP n'est pas limitée à l'intérieur d'une seule entreprise, les IP peuvent être conçues par d'autres fournisseurs. Il est prévu que dans un avenir proche, les IP embarquées provenant de l'extérieur représenteront de 40% à 60% des IP embarquées dans un système sur puce [Zori97T]. Les IP sont très variables en termes de fonctionnement, de description, de technologie, d'origine, etc.

Un système peut inclure plusieurs types d'IP, et les IP peuvent aussi inclure d'autres IP (cf. figure 2.1). Ces IP doivent être testées en utilisant différentes ap-

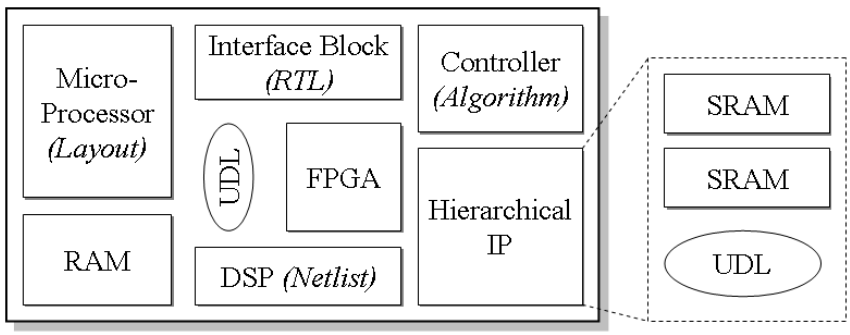


FIG. 2.1 : Un système sur puce avec plusieurs types d'IP.

proches avant que le test du système entier puisse être réalisé. Le test des interconnexions entre les IP doit aussi être pris en compte. Plusieurs publications décrivent les exigences et les défis pour le test des systèmes sur puce à base d'IP [Zori97T, Crou99D, Mari98A, Gupt97I]. Les exigences principales pour ce test peuvent être résumées comme il suit :

– Le test des IP embarquées.

– L'accès aux IP pour les tester.
– L'isolation d'une ou de plusieurs IP pour les tester en même temps.
– Le test des interconnexions entre les IP.
– Le test de la logique utilisée pour assembler les IP (*UDL : User-Defined-Logic*).

L'intégration et la coordination des capacités de test et de diagnostics dans la conception de systèmes intégrés sont très importantes. Elles permettent de faciliter le test des systèmes. Cependant, celles-ci rencontrent beaucoup de problèmes car le test des systèmes sur puce est beaucoup plus complexe que le test des systèmes sur carte. Il se compose de tests individuels tels que le test de chaque IP, le test de la logique, le test des interconnexions. De plus, il est très difficile de programmer ces tests individuels pour s'adapter aux contraintes telles que le temps total de test, la consommation d'énergie, et le coût de test en termes de la surface, etc. Il faut également répondre aux exigences de la couverture de faute, du temps d'arrivée sur le marché, etc. Ceux-ci sont les défis majeurs de l'implémentation du test pour un système sur puce.

Des exigences et défis du test des systèmes sur puce, on peut conclure que le test des IP joue un rôle très important dans le test des systèmes. Dans le paragraphe suivant, nous allons présenter comment réaliser le test d'une IP dans un système sur puce.

2.1.3 Test des IP dans un système sur puce

Pour tester une IP embarquée dans un système, il est nécessaire de trouver une solution pour accéder aux entrées de l'IP afin d'appliquer les jeux de test et pour accéder aux sorties de l'IP afin de récupérer les réponses. Zorian [Zori98T] a proposé une architecture générique pour tester des IP embarquées, illustrée dans la figure 2.2. Dans cette architecture, les IP embarquées sont entourées par une enveloppe de test, souvent appelé coquille (*wrapper*) de test, afin d'améliorer la contrôlabilité et l'observabilité de ces IP. On voit qu'il y a trois parties principales dans cette architecture : (1) le générateur de vecteurs de test et l'analyseur de la réponse, (2) le chemin d'accès de test, et (3) le *wrapper* de test.

Le *générateur de vecteurs de test (GVT)* génère des jeux de test. L'*analyseur de la réponse (AR)* compare les réponses obtenues avec les réponses prévues. Le générateur et l'analyseur peuvent être implémentés hors puce par un équipement de test automatique externe, ou sur puce grâce à un bloc spécial BIST[1], ou comme une combinaison entre ces deux techniques. Nous pouvons ainsi implémenter le générateur sur puce et l'analyseur hors puce ou inversement. Le type du générateur

[1]Built-In Self-Test

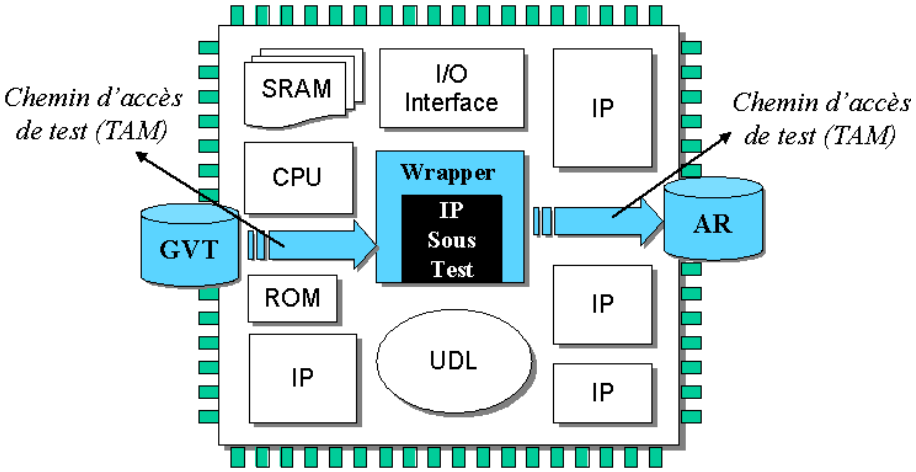


FIG. 2.2 : Architecture de test des IP.

et le type de l'analyseur dépendent du type de l'IP, du type de test prédéfini pour l'IP, et de la qualité et du coût de test [Zori98T].

Le *chemin d'accès de test*, également appelé mécanisme d'accès de test (TAM[2]), transporte les jeux de test du générateur au circuit sous test et transporte les réponses du circuit sous test à l'analyseur. Le TAM peut être formé en réutilisant des fonctionnalités existantes du système ou peut être formé par un nouveau matériel dédié. Un TAM peut également être utilisé pour plusieurs IP embarquées. Pour établir un TAM, il faut faire un compromis entre la bande passante pour le transport des données de test, la possibilité de le mettre à l'échelle du matériel, et le temps d'application de test.

Le *wrapper de test* forme l'interface entre l'IP sous test et l'environnement. Il relie les entrées/sorties de l'IP sous test aux autres blocs du système et aux mécanismes d'accès de test. Dans le mode normal, le wrapper de test est transparent au fonctionnement du système. Dans le mode test, le wrapper de test applique les vecteurs de test au circuit sous test et recueille les réponses. De plus, dans le cas nécessaire le wrapper de test peut aussi isoler son IP du reste du système (par exemple, dans le test des interconnexions).

2.1.4 Norme IEEE 1500

Les systèmes sur puce intègrent souvent différents types d'IP de différents partenaires. Chaque IP possède sa stratégie de test et ses vecteurs de test individuels. Les intégrateurs de système n'ont souvent aucune connaissance sur les structures des IP et traitent donc les IP comme des boites noires. Pour faciliter l'interopérabilité du

[2]Test Access Mechanism

test des IP de plusieurs vendeurs, il est nécessaire de standardiser le test des IP embarquées.

En septembre 1995, le Conseil Technique de la Technologie de Test (TTTC) de l'IEEE-CS[3] a lancé une action afin d'identifier les besoins pour standardiser le test des IP embarquées. Le groupe de développement P-1500 a été créé officiellement en juin 1997. L'objectif était de développer une norme qui permet de tester les IP facilement. Le travail s'est concentré sur deux aspects : développer un wrapper de test qui soit configurable pour les IP embarquées et développer un langage de test (*CTL : Core Test Language*) qui soit capable d'exprimer toutes les informations concernant le test des IP. Ce travail a été normalisé en 2005 sous la norme IEEE 1500 [IEEE1500].

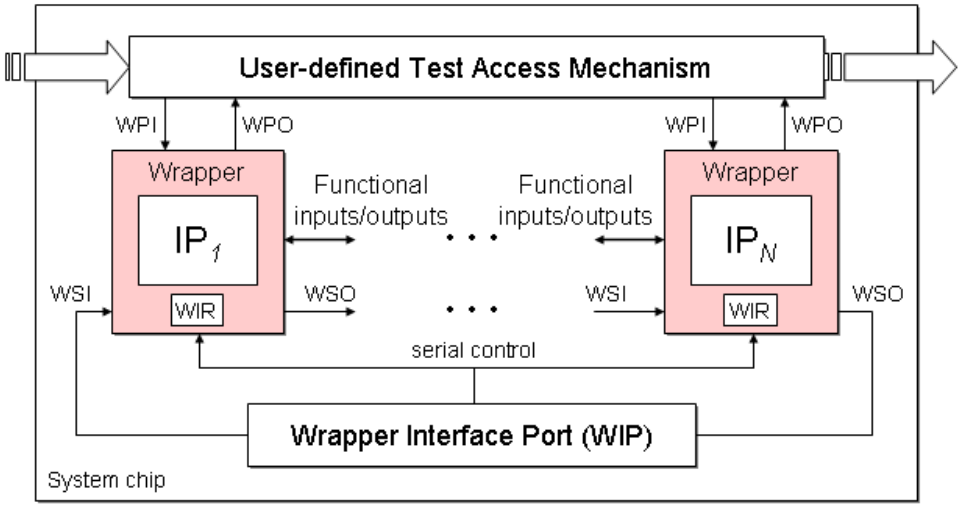


FIG. 2.3 : Architecture conceptuelle de IEEE 1500.

La figure 2.3 présente une vue conceptuelle de l'utilisation de cette norme dans un système sur puce à base d'IP. Chaque IP est entourée par un wrapper de test. Les wrappers de test sont connectés au mécanisme d'accès de test (TAM). Le TAM peut être un bus, une chaîne de bascules en série, une architecture définie par l'intégrateur du système. Les opérations des wrappers de test sont contrôlées par l'interface de wrapper (*WIP : Wrapper Interface Port*). Les lecteurs intéressés consulteront le document officiel [Ieee05I] pour plus de détails sur cette norme IEEE 1500.

[3]IEEE Computer Society

2.1.5 Test intégré

La norme IEEE 1500 présentée dans le paragraphe précédent suppose implicitement que les vecteurs de test sont appliqués sur le circuit par l'intermédiaire d'un testeur externe. L'utilisation d'un testeur externe conduit à un certain nombre de problèmes dont les plus cruciaux sont :

- le prix élevé et toujours croissant des testeurs,
- la difficulté et le temps nécessaire pour générer les séquences de test,
- le temps très élevé nécessaire pour appliquer des séquences longues,
- la perte relative de puissance des testeurs (en termes de vitesse, nombre de broches envisageables) vis-à-vis des circuits à tester.

Pour réduire ces problèmes, il peut être nécessaire d'intégrer le générateur de vecteurs de test et l'analyseur de la réponse sur le système. Cette approche est donc appelée le test intégré ou BIST (*Built-In Self-Test*). La figure 2.4 présente une architecture générale du test intégré. Comme un testeur externe, le générateur et

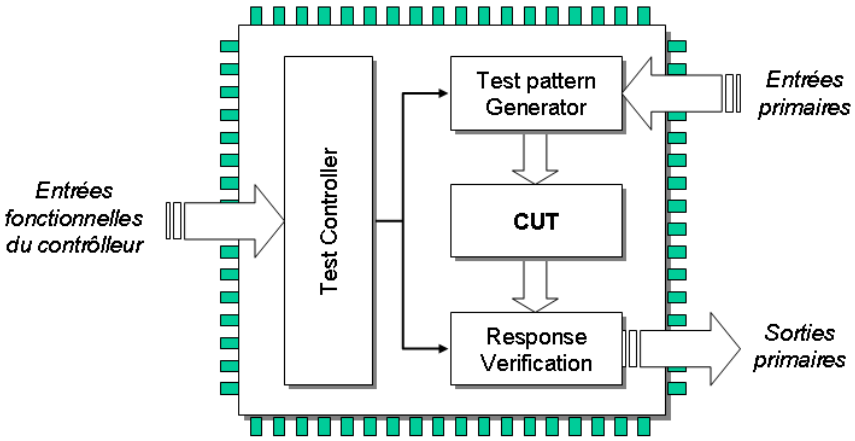


FIG. 2.4 : Architecture générale de BIST.

l'analyseur doivent être capable de générer des vecteurs de test et de comparer les réponses obtenues avec les réponses prévues. Le bloc de contrôle est utilisé pour contrôler le test.

Le générateur peut être réalisé en utilisant des LFSR (*Linear Feedback Shift Register*) ou des automates cellulaires. L'analyseur de réponses de test peut être réalisé en utilisant les techniques utilisant la parité [Cart82S], ou les techniques de comptage [Haye76T], ou les techniques utilisant des LFSR [Beno75A]. Les résumés sur la réalisation et l'implémentation des architectures de test intégré sont présentés dans [Mour00B].

Le test intégré présente de ce fait les avantages suivants :

- suppression de la nécessité d'un testeur coûteux,
- possibilité de test à haute fréquence et à la vitesse nominale de fonctionnement du circuit,
- taux de couverture élevé,
- temps de test court en utilisant à la foi le parallélisme et la hiérarchie du circuit avec une vitesse de test égale à la vitesse nominale du circuit,
- possibilité de tester le circuit en phase de maintenance mais aussi en opération en utilisant les temps de « dormance » du système (cas des systèmes temps réel en particulier).

En contre-partie, l'utilisation du test intégré conduira obligatoirement à un surcoût en matériel s'accompagnant d'une perte de performance.

2.1.6 Test des réseaux sur puce

Comme les systèmes sur puce, les réseaux sur puce (NoC) doivent être testés pour détecter d'éventuels défauts de fabrication. En raison de leurs structures régulières, la stratégie de test pour ces systèmes concerne : (*i*) le test des unités de traitement embarquées (IP) et de leurs interfaces réseaux correspondantes ; (*ii*) le test de l'infrastructure d'interconnexion se composant des routeurs et des liens du réseau entre des routeurs (i.e. le réseau de communication) ; et (*iii*) le test du système intégré entier. On rappelle que le terme NoC est utilisé pour indiquer un système sur puce avec un réseau.

2.1.6.1 Test des unités de traitements et de leurs interfaces réseaux

Pour tester les IP embarquées et leurs interfaces réseaux dans les NoC, le wrapper de test du standard IEEE 1500 (cf. section 2.1.4) et/ou d'autres approches peuvent être appliquées pour chaque IP tandis que le réseau de communication peut être réutilisé comme un TAM possédant une large bande passante pour transporter des données de test. Les vecteurs de test sont encapsulés en paquets qui sont transportés à travers le réseau en utilisant le protocole du réseau.

En fait, l'utilisation du réseau comme un TAM a d'abord été proposé par Mohsen Nahvi et André Ivanov de l'Université de British Columbia, Canada [Nahv04I]. Dans cette proposition, un réseau appelé NIMA (*Novel Indirect and Module Architecture*) a été conçu et utilisé pour transporter des données de test. Ce réseau a été intentionnellement développé pour le test, et non pas pour la communication. Peu de temps après, plusieurs travaux ont proposé de réutiliser les réseaux comme un TAM large bande passante pour tester les IP embarquées dans un NoC

[Cota04R, Kim04O, Liu05P, Amor06W, Huss07O]. La différence principale entre ces propositions réside dans la conception du wrapper de test et son adaptation au comportement du réseau.

Les avantages principaux de la réutilisation de réseau comme un TAM sont qu'il n'y a aucun coût supplémentaire nécessaire en matériel pour construire le TAM et la possibilité de réaliser plusieurs processus de test en parallèle.

2.1.6.2 Test du réseau

Pour tester une architecture réseau, nous devons adresser deux problèmes : le test des routeurs du réseau et le test des liens entre les routeurs.

Dans les réseaux synchrones, comme les routeurs se composent de *buffers* de type FIFO[4] et de parties logiques pour le routage, la plupart des discussions [Ubar03T, Verm03B, Pand05D] indiquent que le test du routeur devrait être fait séparément : utiliser un BIST pour les FIFO et une méthode traditionnelle pour les logiques de routage. Malheureusement, les FIFO sont distribués partout dans le système et cela rend difficile cette approche.

Plusieurs autres propositions [Amor05A, Hoss06A, Grec06B, Liu06R] ont été présentées mais aucune d'elles n'est complète. Ces travaux se concentrent seulement sur le test des routeurs du réseau, alors que les liens du réseau doivent également être testés. En outre, ces approches sont basées sur la méthodologie de *scan* en série qui peut mener à des temps prohibitifs de test. Notons aussi que ces approches peuvent seulement être employées pour tester les réseaux synchrones, et pas pour tester les réseaux asynchrones.

En conséquence, pour obtenir une solution de test efficace pour les NoC, il faut tenir compte de la régularité de leurs structures. Il faut également considérer la réutilisation des éléments du réseau pour construire un TAM bas coût et fonctionnant à haute vitesse.

Le chapitre 1 a présenté les NoC asynchrones et leur intérêts dans une implémentation GALS. En revanche, ce nouveau paradigme de NoC asynchrone est un nouveau champ de recherche encore très peu connu. Le test des réseaux asynchrones est plus difficile que le test des réseaux synchrones en raison du grand nombre de boucles de rétroaction (*feedback loops*) dans des circuits asynchrones, et du manque de soutien des outils de CAO pour le test. Il existe plusieurs approches CVT pour les circuits asynchrones mais leur coût en surface est inacceptable. De plus, ces travaux sont seulement appliqués pour certains types d'implémentation de circuits (cf. section 2.3). En fait, pour tester les réseaux asynchrones, nous avons connaissance

[4]First-In First-Out

d'une seule proposition dans [Efth05T]. Dans cette proposition, le test des routeurs asynchrones est fait en insérant des verrous de balayage (*scan-latch*) pour casser les boucles de rétroaction des éléments de Muller (cf. section 2.2.1.2) utilisés dans la conception des circuits asynchrones. Les résultats montrent un coût prohibitifs en surface ($\approx 100\%$) vu la proportion importante de portes de Muller dans le routeur.

Pour comprendre ces problèmes, nous présentons dans la section suivante les concepts de base et les intérêts de la conception asynchrone.

2.2 Conception des circuits asynchrones : principes fondamentaux

Les circuits synchrones correspondent à une classe restreinte de circuits qui sont contrôlés par un signal périodique uniformément distribué : l'horloge. Les problèmes d'horloge sont dans les technologies actuelles de plus en plus présents car ces technologies autorisent la conception de circuits de plus en plus complexes et de plus en plus rapides. Aujourd'hui, la conception des circuits d'horloge est devenue une question de toute première importance puis qu'elle peut directement limiter les performances du système synchrone.

Au contraire, les circuits asynchrones sont des circuits dont le contrôle est assuré par toute autre méthode que le recours à un signal d'horloge. Leur avantage évident est que tous les problèmes de conception liés à la manipulation d'horloge sont supprimés. La conception asynchrone répond de plus en plus au besoin des concepteurs. Cependant, la conception asynchrone reste encore le domaine de spécialistes à cause du manque d'outils de conception automatisés. Dans certaines applications, une solution mixte est intéressante afin de faire un compromis entre les deux types de conception. Le circuit FAUST en est un exemple, le réseau de communication est réalisé en logique asynchrone tandis que le reste du système est réalisé en logique synchrone. C'est la meilleure solution pour s'adapter à l'approche GALS. On rappelle que dans un système GALS, les IP synchrones communiquent de façon asynchrone entre elles en utilisant un environnement de communication asynchrone. Un des avantages majeurs du GALS est la capacité de gérer les différentes horloges, les différentes tensions des IP embarquées dans le système. La feuille de route (*roadmap*) de l'ITRS prédit que l'approche GALS deviendra la tendance principale dans un proche avenir.

Afin de donner au lecteur une connaissance de base sur la conception asynchrone, nous présentons dans cette section un rapide aperçu sur les concepts de base de la conception asynchrone, les styles de circuits asynchrones les plus utilisés, puis un

court résumé sur les méthodologies et les outils de conception. Les avantages et potentiels des circuits asynchrones sont aussi discutés.

2.2.1 Concepts de base

Un circuit numérique est distinct d'un circuit analogique parce qu'on suppose que la tension sur chaque fil est à l'un des deux niveaux distincts : V_{high} (*1 logique*) ou V_{low} (*0 logique*). La transmission de données sur des signaux électriques exige que ces deux niveaux soient employés pour indiquer la valeur et la séquence : la *valeur* est exigée pour donner la signification des données et la *séquence* est exigée pour indiquer l'occurrence d'une donnée et la distinguer de la suivante.

Dans les circuits synchrones, la séquence est séparée de la valeur par une horloge globale : le signal électrique est échantillonné par l'occurrence d'un front de l'horloge, le niveau de tension obtenu dénote une valeur binaire de l'ensemble {0,1}. On voit bien que l'exécution de tous les éléments d'un système synchrone sont synchronisés car ils évoluent ensemble lors de l'occurrence d'un front d'horloge. Ce mécanisme introduit une contrainte d'ordre temporel. Tous les éléments doivent respecter un temps d'exécution maximum fixé par la fréquence des occurrences des événements d'activation (condition de bon fonctionnement).

Par contre, dans les circuits asynchrones il n'existe pas d'horloge globale. La valeur et la séquence sont souvent combinées dans un protocole de communication. Le but du protocole de communication est de mettre en œuvre la transmission point-à-point des données d'un expéditeur à un récepteur à travers un canal de communication, cf. figure 2.5. Tous les éléments d'un système asynchrone évoluent

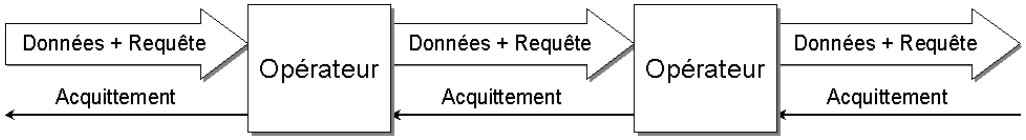


FIG. 2.5 : Communication de type requête-acquittement entre opérateurs.

de façon localement synchronisée et le déclenchement des actions dépend uniquement de la présence des données à traiter. Les opérateurs connectés se synchronisent en échangeant des informations indépendamment des autres opérateurs auxquels ils ne sont pas connectés. Ce mécanisme garantit la synchronisation et la causalité des événements au niveau local et la correction fonctionnelle du système dans son ensemble. Le contrôle local doit en conséquence remplir les fonctions suivantes : être à l'écoute des communications entrantes, déclencher le traitement localement si toutes les informations sont disponibles (*rendez-vous*) et produire des valeurs sur

les sorties. Cependant, afin d'être en mesure d'émettre de nouvelles données, les opérateurs qui se trouvent en amont doivent être informés que les données qu'ils émettent sont bien consommées. Ceci permet de savoir que le récepteur est prêt à traiter une nouvelle donnée. En conséquence, pour permettre un fonctionnement correct indépendamment du temps, le contrôle local doit prendre en charge une signalisation bidirectionnelle (une requête et un acquittement). On appelle ceci un mécanisme de type *requête-acquittement* ou *poignée de main*.

Dans le paragraphe suivant, nous allons découvrir différents protocoles de communication utilisés pour construire les circuits asynchrones.

2.2.1.1 Protocoles de communications

En effet, les protocoles de communication peuvent être classés selon deux caractéristiques distinctes, leur convention de signalisation et leur schéma de codage des données.

– ***La convention de signalisation***

La signalisation est le terme employé pour décrire une séquence d'actions suffisante pour mettre en œuvre un transfert de donnée à travers un canal de communication. Une convention de signalisation peut être un des deux types : 2-phase (*NRZ* ou *non retour à zéro*), ou 4-phase (*RTZ* ou *retour à zéro*). La signalisation 4-phase associe des actions de poignée de main aux niveaux de tension, tandis que la signalisation 2-phase associe des actions de poignée de main aux transitions entre des niveaux de tension.

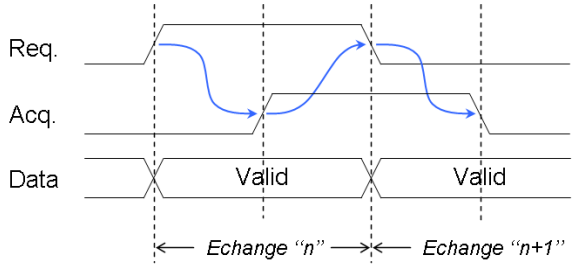


FIG. 2.6 : La signalisation 2-phase (*non retour à zéro*).

Le principe de la signalisation 2-phase, cf. figure 2.6, peut être expliqué comme suit : Phase 1, le récepteur détecte la présence de nouvelles données, il effectue le traitement et génère le signal d'acquittement ; Phase 2, l'émetteur détecte le signal d'acquittement et émet les nouvelles données si elles sont disponibles.

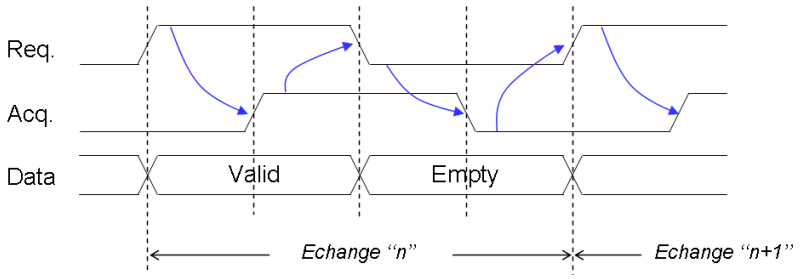


FIG. 2.7 : La signalisation 4-phase (*retour à zéro*).

Le principe de la signalisation 4-phase, cf. figure 2.7, peut être expliqué comme suit : Phase 1, le récepteur détecte la présence de nouvelles données, il effectue le traitement et génère le signal d'acquittement ; Phase 2, l'émetteur détecte le signal d'acquittement et émet des données invalides (retour à zéro) ; Phase 3, le récepteur détecte le passage des données dans l'état invalide et place le signal d'acquittement dans l'état initial (retour à zéro) ; Phase 4, l'émetteur détecte le retour à zéro de l'acquittement, il est alors prêt à émettre de nouvelles données.

Nous voyons que chaque signalisation 4-phase se compose de deux signalisations 2-phase : une signalisation d' « affirmation » (*asserting*) et une signalisation d' « effacement » (*clearing*). Dans ce sens une signalisation 4-phase peut être comparée à une combinaison d'une présence et d'une absence alternatives des données dans un canal.

– ***Le schéma de codage des données***

Il est impossible d'utiliser un seul fil par bit de donnée car cela ne permet pas de détecter que la nouvelle donnée prend un état identique à la précédente. Il faut donc adopter un codage particulier pour les données. Il y a deux types de codage des données principaux : *données-groupées*[5] ou *insensible aux délais* [Spar01P].

Dans le codage données-groupées, la requête et la valeur sont découpées en fils séparés. La valeur est codée comme dans un circuit synchrone en utilisant k fils pour dénoter une donnée de k bits par un encodage en binaire, et la requête est codée en utilisant un fil de *requête* dédié, cf. figure 2.8. Le codage données-groupées exige l'insertion explicite de délais sur le fil de requête pour assurer qu'une requête n'est jamais reçue avant que la valeur groupée ne soit valide.

Réciproquement, le codage insensible aux délais rend la valeur implicite dans la requête et aucune insertion de délais n'est donc exigée. Le codage insensible au délais peut être décrit en utilisant la notation « m-of-n » [Verh88D, Bain03D] pour indiquer

[5] bundled-data

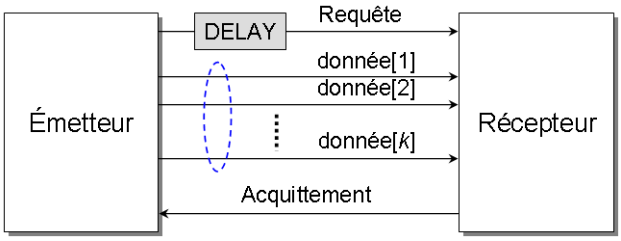


FIG. 2.8 : Encodage données-groupées.

que la transmission des données se compose de m requêtes parmi n possibles, cf. figure 2.9.

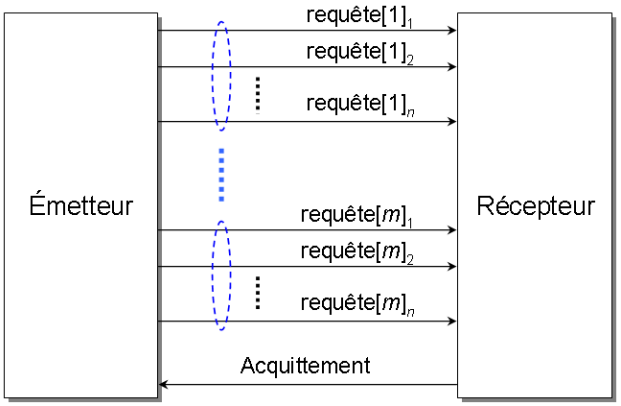


FIG. 2.9 : Encodage « m-of-n ».

Les codages insensibles aux délais les plus communs sont les codages « 1-of-n » (i.e. codage « *one-hot* »). Dans l'exemple du codage « 1-of-2 » (souvent appelé codage double-rail), deux fils sont employés pour encoder un bit de données. Donc, quatre états sont utilisables pour exprimer deux valeurs logiques (0, 1). Deux codages sont communément adoptés : l'un utilisant trois états et l'autre utilisant les quatre états, cf. figure 2.10.

Pour le codage trois états, un fil prend la valeur 1 pour coder une donnée à 1 et l'autre fil prend la valeur 1 pour coder la donnée 0. L'état « 11 » est interdit alors que l'état « 00 » représente l'invalidité d'une donnée. Avec ce codage, il faut passer par l'état d'invalidité chaque fois que l'on veut passer de la valeur 1 à la valeur 0 ou l'inverse, cf. figure 2.10(a). Ceci est bien adapté à la signalisation 4-phase.

Pour le codage quatre états, les valeur 0 et 1 d'un bit sont codées avec deux combinaisons. L'une des combinaisons est considérée comme étant de parité impaire et l'autre de parité paire, cf. figure 2.10(b). Chaque fois qu'une donnée est émise, on change sa parité. Ceci permet de passer d'une valeur logique à l'autre sans passer

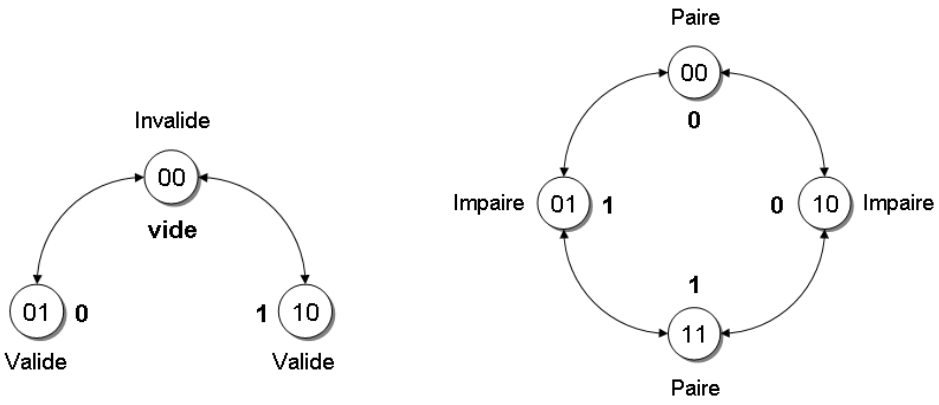


FIG. 2.10 : Diagramme de transitions : (a) Codage trois états ; (b) Codage quatre états.

par l'état d'invalidité. Il est bien adapté à la signalisation 2-phase.

2.2.1.2 Aléas et Porte de Muller (C-élément)

Dans un circuit synchrone le rôle de l'horloge est de définir des instants où les signaux sont stables et valides. Entre les fronts d'horloge, les signaux peuvent montrer des aléas et peuvent faire plusieurs transitions jusqu'à ce que les circuits combinatoires se stabilisent. Ceci ne pose pas de problème fonctionnel. Dans des circuits asynchrones la situation est différente. L'absence d'horloge indique que, dans beaucoup de circonstances, il est nécessaire que les signaux soient valides en permanence, que chaque transition de signal a une signification et, par conséquent, que des aléas (*harzards*) [Jerr02C] doivent être évités.

Pour concevoir des circuits asynchrones sans aléas, la porte de Muller (notée C) présenté dans la figure 2.11 est utilisée. C'est une porte logique à conservation d'état (*state-holding*) comme un verrou asynchrone de type SR (*Set-Reset*). Quand les deux entrées sont à 0 la sortie est mise à 0, et quand les deux entrées sont à 1 la sortie est mise à 1. Pour d'autres combinaisons d'entrées la sortie ne change pas. En conséquence, un observateur voyant la sortie changer de 0 à 1 peut conclure que les *deux* entrées sont maintenant à 1 ; et pareillement, un observateur voyant la sortie changer de 1 à 0 peut conclure que les *deux* entrées sont maintenant à 0. On peut conclure que la porte de Muller implémente un « *rendez-vous* » entre ses entrées et donc une synchronisation entre requête et acquittement.

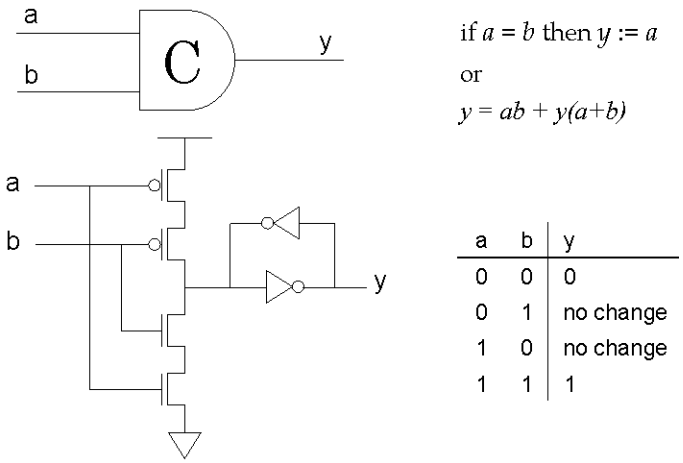


FIG. 2.11 : Porte de Muller : symbole, implémentation, spécifications.

2.2.1.3 Quelques styles d'implémentation

Il y a plusieurs styles d'implémentation des circuits asynchrones. Dans cette section, nous présentons les trois styles les plus courants pour implémenter les circuits asynchrones : le style 4-phase données-groupées, le style 4-phase double-rail, et le style 4-phase quatre-rail. Le but de cette section est d'expliquer ces styles d'implémentation. Donc, nous expliquons plus loin les fondations des pipelines construits en utilisant des verrous (*latches*) transparents simples comme éléments de stockage. Les implémentations de circuits plus optimisées peuvent être trouvées dans [Spar01P].

– ***4-phase données-groupées (4-phase bundled-data)***

Le style 4-phase données-groupées ressemble étroitement à la conception des circuits synchrones et mène normalement à des circuits plus efficaces, grâce à l'utilisation étendue des hypothèses temporelles. Le pipeline 4-phase données-groupées est particulièrement simple. Un pipeline de Muller [Suth89M] est employé pour produire des impulsions d'horloge locales. L'impulsion d'horloge produite dans une étape se recouvre avec l'impulsion produite aux étapes voisines d'une façon enclenchée. La figure 2.12 montre un pipeline avec le traitement de données des circuits combinatoires, appelé aussi blocs fonctionnels. Ces circuits combinatoires (CC) sont ajoutés entre les verrous (L). Avec le codage données-groupées, pour maintenir le comportement correct du pipeline les délais insérés (D) dans les signaux requêtes doivent être égaux ou supérieurs aux délais maximaux des circuits combinatoires correspondants (cf. section 2.2.1.1).

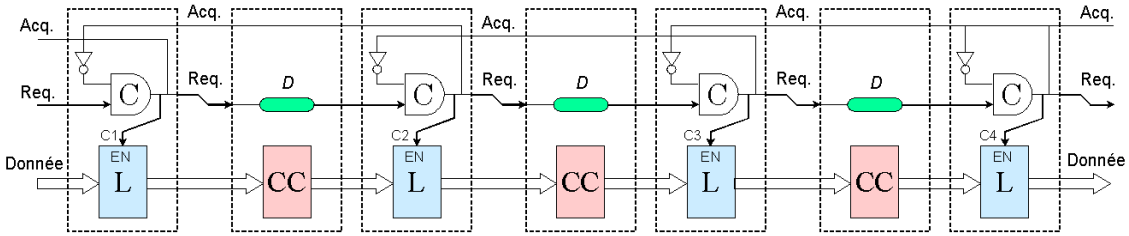


FIG. 2.12 : Un pipeline 4-phase données-groupées simple.

Le fonctionnement du pipeline peut être expliqué comme suit : quand un étage reçoit une donnée avec une requête de l'étage précédent, elle va acquitter (mettre le signal d'acquittement *Acq.* à 1) si elle est prête à consommer cette donnée (le signal d'acquittement *Acq.* venant de l'étage suivant est mis à 0).

Ce circuit peut être regardé comme un flot de donnée « synchrone » traditionnel, se composant de verrous et de circuits combinatoires qui sont synchronisés par un contrôleur d'horloge distribué (*a distributed gated-clock driver*). Ce circuit peut ainsi être regardé comme une structure flot de donnée asynchrone qui se compose de deux types de composants : les verrous et les blocs fonctionnels, comme indiqué par les éléments entourés de pointillés.

L'implémentation de pipeline représentée dans la figure 2.12 est très simple mais elle a quelques inconvénients : quand il se remplit, l'état des sorties des portes de Muller (C1, C2, C3, C4, etc) est (0, 1, 0, 1, etc.), et par conséquent seul un verrou sur deux peut stocker des données. Ce n'est pas plus mauvais que dans un circuit synchrone en utilisant des bascules maître-esclave, mais il est possible de concevoir des pipelines asynchrones qui sont meilleurs sur ce point. Un autre inconvénient est la vitesse. Le débit d'un pipeline dépend du temps qu'il prend pour accomplir un cycle de poignée de main et pour l'implémentation ci-dessus ceci implique la communication avec les deux voisins.

– ***4-phase double-rail (4-phase dual-rail)***

Le style 4-phase double-rail est l'approche inspirée du travail de David Muller dans les années 1950. Il est également basé sur le pipeline de Muller, mais d'une manière plus raffinée avec le codage combiné des données et de la requête. Si on met des pipelines de Muller en parallèle, on peut obtenir un pipeline 1-bit en utilisant un signal d'acquittement commun par étage afin de synchroniser l'opération. La figure 2.13 montre l'implémentation d'un pipeline 1-bit avec une profondeur de trois étages sans traitement de données.

Ce pipeline utilise le codage trois états, cf. section 2.2.1.1, pour coder les données. Donc, une paire de portes de Muller dans un étage de pipeline peut stocker un

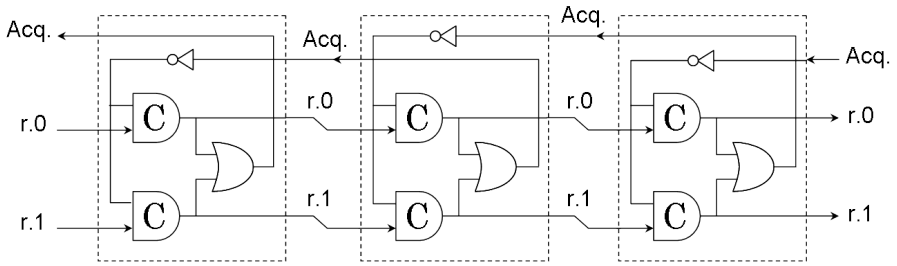


FIG. 2.13 : Un pipeline 1 bit simple avec profondeur de 3 étages.

codeword vide (*empty codeword*) $\{r.0, r.1\} = \{0, 0\}$, causant le signal d'acquittement sortie *Acq.* de cet étage à 0, ou il peut stocker un des deux *codeword* valides $\{0, 1\}$ et $\{1, 0\}$, causant le signal d'acquittement sortie *Acq.* de cet étage à 1. Vu que le *codeword* $\{1, 1\}$ n'est pas légal et puisqu'il ne se produit pas, nous pouvons donc dire que le signal d'acquittement *Acq.* généré par le port OU indique sans risque que l'état de l'étage du pipeline est « valide » ou « vide ».

Afin d'établir un pipeline N-bit, nous pouvons utiliser N pipelines 1-bit en parallèle. Parce qu'il n'est pas assuré que tous les bits arrivent en même temps, il faut donc synchroniser ces signaux avant les blocs fonctionnels. Dans un but de minimisation des fils sur le lien au détriment de la performance et de la surface, les signaux d'acquittement individuels sont combinés en utilisant une porte de Muller N-entrées pour avoir un signal d'acquittement global.

– ***4-phase quatre-rail (4-phase MR[4])***

L'implémentation 4-phase quatre-rail est similaire à l'implémentation 4-phase double-rail, sauf que le codage de donnée « 1-of-4 » est utilisé [Bain01D]. Avec ce codage, quatre rails sont utilisés pour coder une information de 2 bits. Chaque rail indique une combinaison parmi quatre possibles (« 00 », « 01 », « 10 », et « 11 »). L'avantage de ce style de circuit est la consommation : il a besoin du même nombre de rails de requête pour envoyer deux bits, mais de la moitié des transitions par rapport au style 4-phase double-rail.

La figure 2.14 montre l'implémentation d'un pipeline 2-bit 4-phase quatre-rail avec un profondeur de trois étages sans traitement de données. Quatre portes de Muller dans un étage de pipeline peuvent stocker un *codeword* vide $\{r.0, r.1, r.2, r.3\} = \{0, 0, 0, 0\}$ causant le signal d'acquittement sortie *Acq.* de cet étage à 0, ou un des quatre *codeword* valides $\{0, 0, 0, 1\}$, $\{0, 0, 1, 0\}$, $\{0, 1, 0, 0\}$, et $\{1, 0, 0, 0\}$ causant le signal d'acquittement sortie *Acq.* de cet étage à 1.

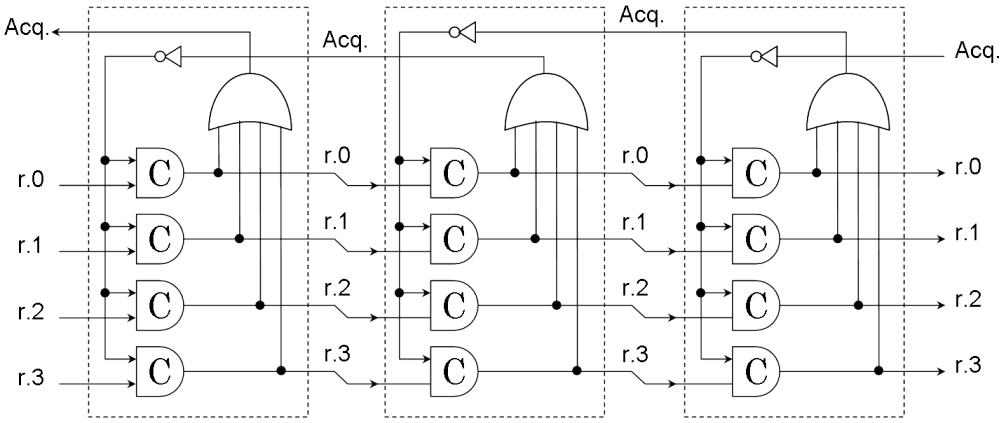


FIG. 2.14 : Un pipeline 2 bits 4-phase codage « 1-of-4 ».

2.2.1.4 Classification des circuits asynchrones

Il existe plusieurs façons de concevoir un circuit asynchrone. Les circuits asynchrones peuvent être classés selon des métriques différentes, chaque métrique qualifie certaines hypothèses de conception ou des techniques architecturales adoptées par le circuit fondamental. Dans cette section, nous présentons une classification des circuits asynchrones selon le modèle de délai représenté dans la figure 2.15. Dans

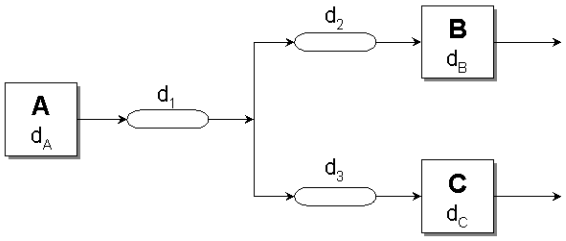


FIG. 2.15 : Modèle de délai.

cette figure, nous utilisons trois portes A, B, et C, où le signal de sortie de la porte A est relié aux entrées sur les portes B et C. Il y a donc deux types de délai dans ce modèle : le délai des composants élémentaires[6] (d_A, d_B, et d_C) et le délai de ses interconnexions[7] (d_1, d_2, et d_3).

Par définition, on dit qu'un délai est « fixe » s'il possède une valeur déterminée. Un délai est « borné » s'il possède une valeur située dans un intervalle connu. Enfin un délai est « non-borné » si sa valeur est finie mais inconnue.

[6]portes logiques
[7]fils d'interconnexion

– ***Circuits indépendants de la vitesse (SI : Speed-Independent)***

Un circuit asynchrone est un circuit indépendant de la vitesse si ses opérations ne dépendent que de l'hypothèse suivante : les délais des fils sont négligeables tandis que les délais des portes sont non-bornés. Cette classe de circuit a été présentée par Muller en 1959 [Mull59A]. En regardant la figure 2.15, cela veut dire que d_A, d_B, et d_C sont non-bornés, mais $d_1 = d_2 = d_3 = 0$. L'hypothèse délai-zéro idéal n'est pas réaliste avec les processus de fabrication des semiconducteurs d'aujourd'hui. En permettant d_1 et d_2 non-bornés et par l'exigence $d_2 = d_3$, le circuit reste théoriquement indépendant de la vitesse car les délais des fils peuvent être dispersés dans les portes.

– ***Circuits Insensibles aux Délais (DI : Delay Insensitive)***

Un circuit asynchrone est un circuit insensible aux délais si ses opérations ne dépendent pas des délais des portes, ni des délais des fils [Uddi86A]. En se référant à la figure 2.15, c'est-à-dire que d_A, d_B, d_C, d_1, d_2, et d_3 sont non-bornés. Les circuits insensibles aux délais forment une classe des circuits asynchrones la plus robuste puisque leur fonctionnement est garanti pour n'importe quel délai sur les portes et les fils. Malheureusement, la classe des circuits insensibles aux délais est très petite. Seules les circuits qui se composent de C-éléments et d'inverseurs peuvent être insensibles aux délais [Mart90T].

– ***Circuits Quasi Insensibles aux Délais (QDI : Quasi Delay Insensitive)***

Un circuit asynchrone est un circuit quasi insensible aux délais si ses opérations ne dépendent que de l'existence d'une ou plusieurs fourches isochrones [Mart90P]. Cette classe de circuit adopte les hypothèses des circuits insensibles aux délais mais y ajoute la notion de *fourche isochrone*[8].

Une fourche est par définition un fil qui connecte un émetteur à plusieurs récepteurs. On la qualifie isochrone lorsqu'on suppose que les délais entre l'émetteur et les différents récepteurs sont identiques. Martin [Mart93S] a montré que l'hypothèse temporelle de fourche isochrone est la plus faible à ajouter aux circuits insensibles aux délais pour les rendre réalisables avec des portes à plusieurs entrées et une seule sortie. Dans cette hypothèse, on peut en effet se permettre de ne tester qu'une branche d'une fourche et donc l'acquitter en supposant que le signal s'est propagé de la même façon dans les autres branches.

En pratique, l'hypothèse temporelle de fourche isochrone est assez faible et est facilement remplie par une conception soignée, en particulier au niveau du routage et des seuils de commutation. Il suffit finalement que la dispersion des temps

[8] isochronic fork

de propagation jusqu'aux extrémités de la fourche soit inférieure au délai minimum des opérateurs qui lui sont connectés et dont une sortie interagit avec la fourche [Mart90T, Berk92B]. En considérant la figure 2.15, c'est-à-dire que :

$$|(d_1 + d_2) - (d_1 + d_3)| < min(d_B, d_C).$$

2.2.2 Méthodologies et outils de conception des circuits asynchrones

Le dernier paragraphe a présenté trois classes typiques des circuits asynchrones, l'objectif de ce paragraphe est d'introduire rapidement les méthodologies et les outils de conception pour ces types de circuits asynchrones.

En effet, il existe aujoud'hui quelques outils de conception qui s'intéressent à la synthèse de circuits asynchrones mais un seul est commercialisé. Philips est le seul industriel qui possède un environnement de conception complet, « *Handshake Technology design environment* ». Les laboratoires universitaires possèdent un certain nombre d'outils mais qui souffrent d'une faible compatibilité entre eux et avec les environnements commerciaux, et qui trop souvent ne couvrent pas l'ensemble des phases de conception. Cependant, ceux-ci sont suffisants pour permettre à la communauté universitaire de concevoir des circuits fonctionnels. Voici une liste de publications sur les méthodologies de conception de circuits asynchrones qui peut être consultée : [Hauc95A, Spar01P, Mart06A, AsyOu]. Le site web [AsyOu] recense tous les outils disponibles.

2.2.2.1 Circuits indépendants de la vitesse

La conception des circuits indépendants de la vitesse est actuellement l'une des méthodologies les plus étudiées et utilisées. Pour concevoir des circuits indépendants de la vitesse, nous pouvons utiliser deux méthodologies de conception à base de graphes les plus connues : les graphes de transitions de signaux (*Signal Transition Graphs* ou STG) introduits par Chu, Leug, et Wanuga [Chu85A, Chu87S] et les diagrammes de changement de signal (*Change Diagrams* ou CD) introduits par Kishnevsky, Kondratyev, Taubin et Varshavsky [Kish92O].

Un graphe de transition de signaux (STG) spécifie un circuit à l'aide d'un formalisme proche des réseaux de Petri (*Petri-Nets*) [Mura89P] dans lequel les transitions sont étiquetées par les noms des réseaux. Ainsi, lorsqu'une transition est passée cela correspond à l'occurrence d'une transition du signal correspondant. La notation associe aux signaux le sens de la transition : un « + » signifie une transition positive et réciproquement un « - » signifie une transition négative. Il existe différents types

de STG. Les premiers STG utilisaient les graphes de transitions de signaux marqués (*Marked Graphs* ou STG/MG). Ce sont des réseaux de Petri dans lesquels chaque place possède au plus une entrée et une sortie. Ce type de graphe ne permet pas de spécifier les conflits ou les choix, ainsi une transition active ne peut être désactivée qu'en étant passée. Ceci restreint en conséquence la classe des circuits modélisables.

Afin de permettre la spécification de choix, conjointement au développement d'algorithmes de synthèse plus performants, ces graphes STG ont ensuite été étendus au modèle STG/IC (*Input-Choice STG*) ou au modèle STG/NC (*Non-input Choice STG*). Les différentes propriétés de ces graphes ont été étudiées, telles que : vivacité (*liveness*), sûreté (*safety*), persistance (*persistency*), assignation cohérente d'états (*consistent state assignement*), assignation unique d'état(*unique state assignement*), et graphes à cycle unique de transition (*single-cycle transitions*). Des algorithmes ont été développés afin de tester ces propriétés et même de transformer les graphes pour qu'elles soient remplies (voir [Hauc95A]). La principale difficulté de la synthèse logique consiste bien sûr à obtenir une implémentation et un mapping technologique sans aléa. L'outil couramment utilisé aujourd'hui par la communauté pour effectuer la synthèse de contrôleurs asynchrones à partir de STG est *Petrify* [Kond98H, Past98S].

Le diagramme de changement de signal (CD) est un modèle similaire aux STG, mais il évite quelques restrictions trouvées dans les STG. Comme STG, un CD a des nœuds marqués par des transitions, des arcs entre les transitions qui définissent les séquences autorisées des mises à feu de transition. Dans les CD, nous distinguons trois types d'arcs : *strong precedence*, *weak precedence*, et *disengageable strong precedence*. Les CD sont présentés en détail dans [Kish94C]. Malheureusement, bien que les CD aient des possibilités que les STG n'ont pas, ils n'en ont pas suffisamment pour tous les circuits indépendants de la vitesse [Hauc95A].

2.2.2.2 Circuits insensibles aux délais

Les circuits insensibles aux délais sont robustes mais ce style de conception possède des limitations imposées, cf. section 2.2.1.4. Afin de s'affranchir de ces limitations, il a été proposé de concevoir un ensemble de modules de base qui ne sont pas nécessairement insensibles aux délais en interne mais qui garantissent l'insensibilité aux délais du système une fois assemblés. Molnar, Fang et Rosenberg [Moln85S, Rose88Q] ont développé une méthodologie de conception de tels modules basée sur le formalisme des I-nets (graphes similaires aux réseaux de Petri). A partir de ces modules, Brundvand et Sporoull ont proposés un outil de synthèse utilisant le langage Occam comme formalisme d'entrée [Brun89T]. A chaque struc-

ture du langage correspond alors un module ou ensemble de module permettant son implémentation.

Une approche proposée sur la théorie des traces (*trace theory*) a été proposée par Ebergen [Eber91A]. Cette méthode propose aussi un ensemble de modules de base. Cependant le langage de spécification utilisé est le même pour décrire les modules et le circuit, ce qui le rend primitif et difficile à utiliser.

Plus récemment, des travaux de l'Université de Groningen visent à dériver des circuits insensibles aux délais à partir d'un langage formel par raffinement de programmes [Mall99A].

2.2.2.3 Circuits quasi insensibles aux délais

La conception de ces circuits a été développée à Caltech par le groupe de A. Martin. Le groupe travaille sur une méthodologie de conception de circuits asynchrones qui offre à la fois des facilités de conception dans un langage de haut niveau et une méthodologie de synthèse qui assure des circuits corrects [Mart93S].

Le langage de spécification, appelé CHP (*Communicating Hardware Processes*), est issu de langage CSP (*Communicating Sequential Processes*) de C.A.R. Hoare [Hoar78C]. Un programme CSP consiste en un ensemble de processus qui calculent en parallèle et communiquent entre eux à travers des canaux. Ce modèle de processus communicants est similaire à la synchronisation locale des circuits asynchrones et à leur mode de fonctionnement flot de données. A partir d'une spécification du circuit dans ce langage, des règles de transformations successives permettent d'obtenir des équations booléennes qui s'implémentent dans la technologie cible. Cette méthode de synthèse est éprouvée et a permis de concevoir un certain nombre de circuits d'une complexité significative qui sont certainement parmi les plus performants aujourd'hui.

Cependant, cette méthodologie de conception est éloignée des méthodes de conception classiques synchrones et ce pour plusieurs raisons. Tout d'abord, il existe actuellement aucun outil d'aide à la conception, que ce soit au niveau de la simulation du langage CHP de spécification qu'au niveau de la synthèse logique de ce type de circuits. De plus, la méthode de synthèse s'attache à concevoir uniquement des circuits dédiés optimisées (*full-custom*). Au contraire, les approches « cellules standard » couramment utilisés aujourd'hui, même si elles ne peuvent offrir les mêmes niveaux de performances que des circuits dédiés, permettent une conception plus rapide et surtout offrent des possibilités de migration technologique plus importantes. Ces deux paramètres sont prépondérants dans un contexte industriel.

2.2.3 Avantages et potentiels des circuits asynchrones

2.2.3.1 Grande vitesse de fonctionnement

Dans les circuits synchrones, la performance globale d'un système est contrainte par la performance dans le pire cas d'un bloc localement sous utilisé. Le concepteur doit porter son effort de conception vers l'optimisation des temps de calcul dans le pire cas. Au contraire, dans les circuits asynchrones la vitesse d'opération est déterminée par des latences locales réelles plutôt que par la latence du pire cas global. Chaque opérateur peut évaluer une fonction en un temps variable, compris entre une borne inférieure et une borne supérieure. Ce temps correspond au temps de traversée des données de l'entrée vers la sortie de l'opérateur, c'est la latence directe. Ce temps de traversée de l'opérateur peut aussi posséder des variations en fonction du chemin utilisé par les données et donc en fonction de leurs propres valeurs.

Comme l'opérateur asynchrone implémente un protocole de communication, la donnée est signalée dès que possible et donc utilisée à sa sortie par l'opérateur suivant au plus tôt. Ainsi, de manière naturelle les circuits asynchrones possèdent des variabilités de temps de calcul qui sont exploitables grâce aux mécanismes de synchronisation locale.

2.2.3.2 Absence d'horloge, pas de problème d'horloge skew

Les circuits asynchrones n'utilisent pas d'horloge globale. Tous les problèmes de conception liés à la manipulation d'horloges sont donc supprimés. Les éléments de synchronisation sont distribués dans l'ensemble du circuit, leur conception est plus facile à maîtriser. Avec la plupart des circuits asynchrones les problèmes d'interconnexion et de synchronisation entre blocs sont inexistants. En effet, le problème de gigue d'horloge (*clock skew)* est supprimé et le temps d'arrivée d'un signal d'un bout à l'autre du circuit n'a pas d'importance sur le fonctionnement correct du système. Le principe de synchronisation locale des circuits asynchrones constitue alors directement un outil d'aide à la conception de systèmes complexes.

2.2.3.3 Basse consommation

Dans les circuits synchrones, les lignes d'horloge sont activées pour précharger et décharger les signaux pendant les calculs. Cependant, elles sont basculées aussi dans les parties du circuit inutilisés lors de certains calculs. Selon des études, dans les circuits rapides la consommation de l'horloge et des éléments de mémorisation peut représenter jusqu'à 50% de la consommation du circuit. Cette consommation est due au chargement des circuits d'horloges et aux transitions dans les bascules.

Grâce à l'absence d'horloge, la consommation associée à l'horloge est supprimée dans les circuits asynchrones.

De plus, dans un circuit synchrone tous les blocs logiques se trouvent alimentés en données et commutent tous au moment de l'arrivée de l'horloge, que ces transitions soient utiles dans le bloc ou non. Au contraire, grâce au mécanisme de synchronisation locale des circuits asynchrones et à leur fonctionnement flot de données, seuls les blocs qui reçoivent des données et des contrôles associés consomment. La mise en veille de la logique à tous les niveaux de granularité est automatique.

Une autre propriété intéressante des circuits asynchrones pour la basse consommation est leur robustesse et leur adaptation aux conditions de fonctionnement. Comme la puissance varie avec le carré de la tension, il est aisé de réduire la tension d'alimentation pour limiter la puissance consommée. Cette réduction de la tension d'alimentation des circuits asynchrones peut se faire avec un matériel et un temps de conception minimum. Contrairement au cas synchrone, il n'y a pas besoin d'adapter et de caractériser la fréquence d'horloge aux différentes conditions de fonctionnement car la correction fonctionnelle est garantie quel que soient les délais dans les cellules élémentaires. De manière statique, nous pouvons ainsi aisément choisir entre la performance et la basse consommation dans les circuits asynchrones : le circuit consommera d'autant moins à tension réduite qu'il est performant à tension nominale.

Enfin, il est possible de faire varier dynamiquement la tension d'alimentation d'un circuit afin de réduire la consommation. Les techniques V_{DD}-hopping, DVS (*Dynamic Voltage Scaling*) et ABB (*Adaptive Body Biasing*) ont été bien adoptées dans la conception asynchrone [Labo04V].

Malgré toutes ces propriétés intéressantes, les circuits asynchrones ont besoin de plus de transitions sur le chemin de calcul que les circuits synchrones. De plus, il faut tenir compte de la consommation des parties de contrôle dans les circuits asynchrones. En conséquence, la comparaison ne peut être faite que dans un contexte particulier.

2.2.3.4 Moins d'émission de bruit électromagnétique

Dans la conception synchrone, il faut faire attention aux problèmes de pics de consommation car les activités électriques sont distribués à chaque instant prédéfini par les fronts d'horloges. Au contraire, l'activité électrique d'un circuit asynchrone est bien mieux répartie dans le temps. Ainsi, la consommation dans les lignes du circuit est distribuée dans le temps, que ce soit au niveau de la logique qu'au niveau des éléments de mémorisation. Cette propriété intrinsèque des circuits asyn-

chrones permet de limiter le bruit dans les lignes d'alimentation, et ainsi de réduire la puissance des ondes électromagnétiques émises par le circuit par rapport au cas synchrone [Spar01P]. Ceci est donc particulièrement intéressant pour des circuits qui possèdent des contraintes fortes de compatibilité électromagnétique afin d'être utilisés dans des environnements spécifiques.

2.2.3.5 Robustesse envers les variations de température, de tension d'alimentation et de paramètres de processus de fabrication

Dans la conception synchrone, le changement des conditions de fonctionnement peut faire varier de manière significative les délais logiques dans les circuits VLSI[9]. Les circuits synchrones doivent être simulés dans une gamme très large de tensions d'alimentation et de températures de fonctionnement pour assurer que la fréquence d'horloge sélectionnée garantit des opérations correctes sous toutes les conditions spécifiées.

Au contraire, les circuits asynchrones sont robustes par rapport aux performances des dispositifs élémentaires et aux conditions de fonctionnement. Cela est intéressant pour faire varier aisément la performance du système et sa consommation en adaptant la tension d'alimentation comme nous l'avons vu dans la section 2.2.3.3. Cette priorité devient aussi un avantage prépondérant pour les technologies avancées où les dispersions physiques des dispositifs élémentaires sont de plus en plus importantes.

2.2.3.6 Modularité, réutilisation

Grâce aux interfaces de poignée de main simple et à la localité du contrôle, un autre avantage de circuits asynchrones est sa modularité. Il est en effet très facile de construire une fonction - un système - complexe en connectant des modules préexistants [Clar67M]. En effet, la modularité entre blocs est facilitée par deux aspects. Premièrement, la conception du système est facilitée car il n'y a pas besoin de spécifier un bloc en fonction du nombre de cycles d'horloge pour obtenir et échanger des données avec le bloc distant. Deuxièmement, au niveau implémentation vu qu'il n'y a pas d'hypothèses temporelles qui garantissent le fonctionnement, la conception au niveau physique est rendue plus facile. Les phases de placement et routage sont moins contraintes, si ce n'est pour des questions de performances. Le fonctionnement correct du système est garanti quel que soient les délais dans les interconnexions aux interfaces.

La réutilisation est aussi un point fort de la conception asynchrone. Avec leur propriétés de modularité, d'absence d'horloge globale, l'échange de propriétés in-

[9]Very-Large-Scale Integration (VLSI)

tellectuelles (IP) dans le cadre d'implémentation de systèmes complexes est moins difficile. Cela permet de réduire le temps de conception de systèmes.

2.3 Test des circuits asynchrones

Bien que les circuits asynchrones démontrent des avantages potentiels (cf. section 2.2.3) qui leur permettent d'être employés par des applications basse consommation et hautes performances, ils peuvent seulement être exploités si des techniques de test existent pour détecter les défauts de fabrication. Les méthodes de test doivent garantir une qualité de test suffisante et un temps d'application de test acceptable pour tester le circuit sous test.

Le test de la plupart des circuits asynchrones s'apparente davantage au test des circuits synchrones. Il faut d'abord définir une séquence qui fait commuter les nœuds électriques du circuit et qui permet d'observer leur commutation. Cependant, le test des circuits asynchrones est beaucoup plus complexe que le test des circuits synchrones. Comme il existe plusieurs types de circuits asynchrones (cf. section 2.2.1.4) avec différentes caractéristiques, les méthodes de test sont donc très variables. Les facteurs principaux qui compliquent le test des circuits asynchrones sont les suivants [Mart91T, Haze92T, Hulg95T, Ronc99D] :

- Les circuits asynchrones incluent des éléments mémoires dans les portes (portes de Muller). Ceci rend la génération des vecteurs de test très difficile ou impossible.
- La difficulté de détecter les aléas (*hazards*).
- L'absence d'horloge globale rend les circuits asynchrones moins contrôlables. Il est difficile de commander le test et de contrôler l'exécution d'une séquence en pas à pas.
- Dans certains types de circuits asynchrones, on ajoute des logiques redondantes pour assurer les comportements des circuits sans aléas. Malheureusement, l'observation des fautes de collage dans ces parties redondantes est très difficile.

En revanche, grâce à l'utilisation de protocoles de poignée de main les fautes dans la plupart de circuits asynchrones sont facilement détectées car le circuit se bloque [Davi95S]. Tout ce qu'on doit faire est de fixer une borne temporelle au-delà de laquelle on considère le circuit dans un état de blocage et donc défectueux. L'exécution de la séquence sans blocage nous permet de déclarer que le circuit est bon [Mart91T, Beer92S, Khoc94T, Hulg95T]. Le test devient « comment contrôler des nœuds du circuits afin de faire apparaître les fautes ? ».

On rappelle que dans le cadre du travail de cette thèse, on aborde seulement

trois types de circuits asynchrones : DI, QDI, et SI (cf. section 2.2.1.4). Donc, on présente d'ici le test de ces types de circuits asynchrones.

2.3.1 Test des circuits DI

Pour les circuits DI, chaque transition est confirmée par une autre. Donc, s'il y a des fautes de collage, le circuit entier sera suspendu [Davi90S]. En fait, une faute de collage sur un lien est équivalente à un délai infini de propagation du signal sur ce lien. En conséquence, la transition prévue n'apparaît pas en raison d'une faute de collage. La transition est donc appelée une *transition inhibitive* [Mart91T, Haze92T]. La faute qui cause une transition inhibitive peut facilement être détectée car elle suspend le fonctionnement du circuit. Pour tester les circuits DI, nous fournissons une requête au circuit sous test. Si l'acquittement n'apparaît pas dans un temps borné nous pouvons considérer que le circuit est défectueux.

Pour expliquer ce point, on peut utiliser le protocole 4-phase (cf. figure 2.7 dans la section 2.2.1.1). Avec ce protocole, l'émetteur génère les sorties suivantes : $Req\uparrow$; $[Ack]$; $Req\downarrow$; $[\neg Ack]$ [10]. Le récepteur répond avec : $[Req]$; $Ack\uparrow$; $[\neg Req]$; $Ack\downarrow$. En conséquence, une faute de collage sur n'importe quel lien de contrôle (Req ou Ack) met en attente l'émetteur ou récepteur. Malheureusement, comme nous l'avons dit dans la section 2.2.1.1, les circuits DI sont une classe de circuits asynchrones très petite.

2.3.2 Test des circuits QDI et SI

Pour les circuits QDI ou SI, ils sont suspendus si l'on considère le modèle de faute de type « collage sur les sorties » [Beer92S]. Avec un modèle de faute de type « collage sur les entrées » il faut apporter une attention particulière aux fourches isochrones. Certaines s'avèrent suspendues, d'autres doivent faire l'objet de procédures de test particulières [Hulg95T].

En effet, les fautes de type « collage sur les entrées » dans les circuits QDI ou SI peuvent causer une *mise à feu prématurée*[11] [Mart91T, Haze92T]. C'est une transition qui se produit trop tôt selon les spécifications du circuit. Pour détecter ces fautes, il est nécessaire de faire une analyse raffinée sur la testabilité du circuit [Mart91T].

[10] C'est la *handshake expansion* [Mart90P]. $[a]$ indique une attente sur une expression booléenne a devient vrai, $b\uparrow$ et $b\downarrow$ indique un changement du signal b au niveau haut et au niveau bas, respectivement.

[11] *"premature firing"* ou *"stimulating"*

Un exemple de circuits asynchrones de type SI est le circuit D-élément, c'est un circuit d'interface de type maître-esclave très connu, proposé par Martin [Mart90P], cf. figure 2.16. Le fonctionnement du D-élément peut être décrit par la spécification

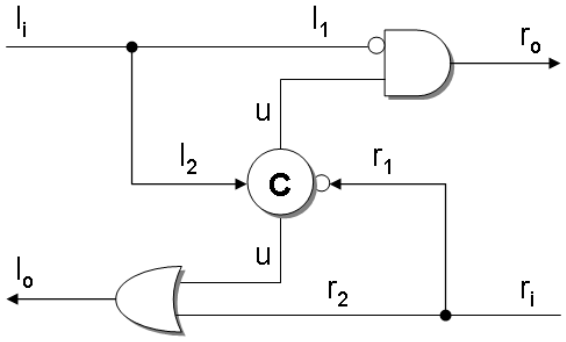


FIG. 2.16 : D-élément

suivante :

$$* [[l_i];\ u\uparrow;\ [u];\ l_o\uparrow;\quad [\neg l_i];\ r_o\uparrow;\quad [r_i];\ u\downarrow; [\neg u];\ r_o\downarrow;\quad [\neg r_i];\ l_o\downarrow] \tag{2.1}$$

ou simplement :

$$* [[l_i];\ l_o\uparrow;\quad [\neg l_i];\ r_o\uparrow;\quad [r_i];\ r_o\downarrow;\quad [\neg r_i];\ l_o\downarrow] \tag{2.2}$$

Le D-élément est un circuit SI qui a deux fourches isochrones sur les entrées l_i et r_i respectivement. On peut voir facilement que les fautes de type « collage sur les sorties » dans D-élément et les fautes collages à 1 sur les entrées entraînent le blocage du circuit.

Maintenant, nous considérons une faute de collage à 0 sur le nœud r_2 (r_2-*SA0*). Nous voyons que la sortie l_o descend après la remise de r_i. S'il y a une faute r_2-*SA0*, la sortie l_o est prématurément mise à zéro par l'événement descendant du net u (cet événement apparaît après la remise de l'entrée r_i). De même, la faute l_1-*SA0* cause une mise à feu prématurée sur la sortie r_o. Pour détecter ces fautes, il faut générer de bons patterns de test afin de faire apparaître les mises à feu prématurées et trouver une façon efficace de les observer.

En conséquence, le test des circuits QDI et SI est très difficile. Il nécessite de trouver une bonne méthode pour générer des vecteurs de test afin de positionner le circuit défectueux dans un état suspendu ou dans un état prématuré. Dans le deuxième cas, il faut trouver une façon efficace pour observer les mises à feu prématurées ou les propager aux sorties primaires.

2.3.3 Conception en vue du test pour les circuits asynchrones

Pour améliorer la testabilité, la conception en vue du test (CVT) s'applique également aux circuits asynchrones [Saxe93D, Page95D]. Cependant, il faut bien assurer que l'ajout de circuits pour le test ne doit pas introduire d'aléas dans les circuits asynchrones.

Les techniques CVT les plus simples sont les techniques *ad-hoc*. Avec ces techniques, nous ajoutons des points de test afin d'améliorer la contrôlabilité et l'observabilité du circuit sous test, cf. figure 2.17. Si une mise à feu prématurée est instable,

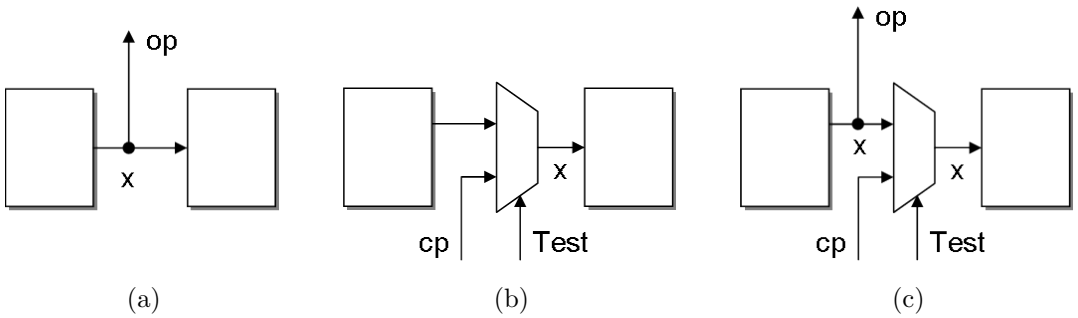


FIG. 2.17 : Ajoute les points de test : (a) point d'observation ; (b) point de contrôle ; (c) les deux types.

on ajoute un point de contrôle. Si une mise à feu prématurée n'est pas propagée à la sortie primaire, il est nécessaire d'ajouter un point d'observation. Cependant, la difficulté est de trouver les endroits où on doit ajouter ces points de test. C'est un travail très coûteux en temps.

Comme pour les circuits synchrones des méthodes de *scan* peuvent être appliquées pour les circuits asynchrones. Donc, des chaînes de *scan* peuvent être insérées dans le circuit sous test afin de supprimer les fils de rétroactions, introduire les vecteurs de test et observer les réponses. Ces chaînes de *scan* peuvent être auto-séquencées [Tris80I, Ronc94P, Petl95S, Khoc95A, Garc98S] ou peuvent être contrôlées par une horloge dédiée [Berk02A, Efth05T].

Avec la première technique, les chaînes de *scan* sont construites en utilisant les portes de Muller du circuit. Cependant, afin de rendre ces portes de Muller testables il est nécessaire de les modifier. La figure 2.18 présente une implémentation au niveau porte d'un C-élément modifié, proposée par [Khoc95A], qui peut être configué comme une porte ET ou une porte OU.

Le C-élément fonctionne comme une porte OU si l'entrée de test ajoutée (*Test*) est mise à '1'. Le C-élément fonctionne comme une porte ET en sortie de reset global (*Reset* est mis à '1' puis à '0'). En conséquence, le réseau des C-éléments modifiés

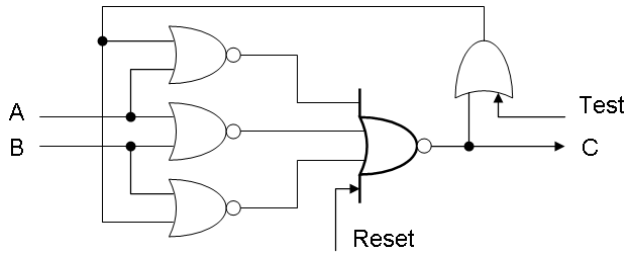


FIG. 2.18 : Implémentation au niveau porte d'un C-élément modifié.

peut être testé de la même manière que les circuits combinatoires. Cependant, cette approche rend le circuit très complexe (ajout de nombreuse portes logiques). De plus, cette technique ne permet pas d'éviter les problématiques naturelles des chaînes de *scan*, telles que le temps d'application du test important. Il est également nécessaire de garantir que les parties logiques insérées soient testables.

Avec la deuxième technique, les chaînes de *scan* sont similaires aux chaînes de *scan* utilisées dans le test des circuits synchrones. La figure 2.19 présente un registre simplifié que nous utilisons dans la technique de *scan* avec une horloge de test. Dans le mode normal ($Test = 0$), les registres de *scan* fonctionnent exactement

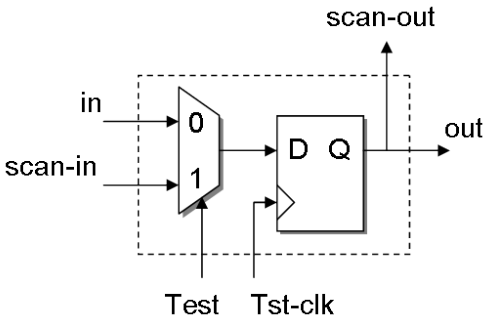


FIG. 2.19 : Registre de *scan*.

comme les registres originaux. Dans le mode test, les registres de *scan* forment un registre à décalage en reliant la sortie *scan-out* d'un registre avec l'entrée *scan-in* du registre suivant. Les patterns de test peuvent être décalés dedans la chaîne de *scan* par l'entrée *scan-in* et les données dans la chaîne de *scan* peuvent être décalées dehors par la sortie *scan-out*. Cependant, la technique de *scan* présente de nombreux problèmes : coût en surface très élevé, faible performance, temps d'application de test très long, etc.

En conclusion, la plupart des propositions se basent sur l'analyse de la structure et de la testabilité de circuits. Les modèles de fautes utilisés sont souvent les modèles

de faute de collage et les modèles de faute de transition (fautes de délai). Le coût du test est prohibitif en termes de surface et de temps de test. La génération de séquences de test est un gros problème. Il n'existe pas aujourd'hui de méthode générale pour la génération des séquences de test. Le test constitue un thème de recherche très important dans le domaine de la conception des circuits asynchrones.

2.4 Méthode de test actuelle pour le réseau ANOC et ses limitations

Le réseau ANOC (cf. section 1.3) a été testé en utilisant la même manière qu'on utilise lors de l'étape de vérification. Les données de test sont définies et générées par un programme sur un ordinateur. Ces données sont ensuite transportées et stockées dans la mémoire externe d'une carte à base de FPGA (*carte FPGA*). L'objectif d'utiliser la carte FPGA est d'implémenter une interface de communication compatible avec le circuit FAUST. Les données stockées dans cette carte FPGA seront encapsulées en paquets et envoyées à l'entrée du circuit FAUST. Selon les informations de routage intégrées dans le *flit* d'en tête de paquet, les paquets de test se propagent vers les routeurs sous test définis. Les réponses de test retournent à la porte de sortie et sont transmises à la carte FPGA afin d'être enregistrées dans la mémoire externe. Ensuite, ces données sont envoyées à un ordinateur pour être analysées. Si nous ne recevons pas les réponses de test à la carte FPGA ou si les données reçues ne sont pas correctes, il y a peut-être une ou plusieurs fautes dans le réseau ANOC sur le chemin de test. Le circuit FAUST est donc défectueux. La figure 2.20 décrit l'implémentation d'un environnement de test pour le circuit FAUST.

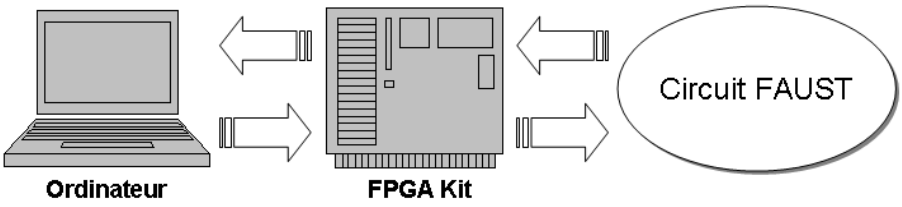
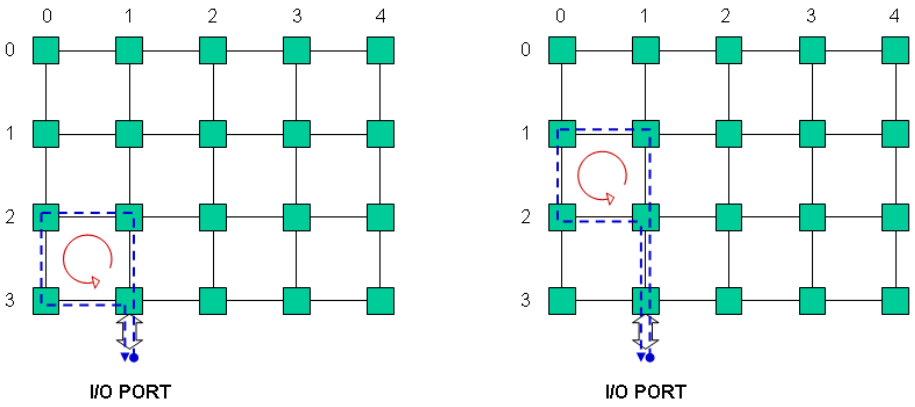


FIG. 2.20 : Architecture de test actuelle pour ANOC.

La stratégie de test pour le réseau ANOC peut être expliquée comme suit : les données de test circulent dans un groupe de quatre routeurs voisins, cf. figure 2.21. La raison de ce choix d'un groupe de quatre routeurs voisins est d'avoir le chemin de routage le plus court.

Dans le réseau ANOC (une maille de taille 5×4), il y a 12 possibilités pour créer des groupes de quatre routeurs voisins. Il y a donc 24 possibilités pour faire circuler



(a) (b)

FIG. 2.21 : Tester des routeurs et des liens en groupe de 4 routeurs.

des paquets de donnée (un chemin de chaque groupe doit être testé dans les deux directions). Nous commençons par tester les routeurs et les liens les plus proches du port d'entrée, cf. figure 2.21(a). Une fois que ces routeurs et ces liens sont testés, nous utilisons ces routeurs et ces liens comme un TAM pour tester les routeurs et les liens voisins, cf. figure 2.21(b). Selon ce principe, nous pouvons atteindre tous les routeurs et tous les liens du réseau. Notons que cette stratégie ne teste pas toutes les traversées (Nord→Sud, Sud→Nord, Ouest→Est, Est→Ouest), quelques vecteurs supplémetaires ont été recalculés.

Malheureusement, la taille d'un *flit* de donnée est limitée et donc la taille du champ « *path-to-target* » est limitée. Dans le cas du réseau ANOC, cette taille est fixée à 9 traversées de routeur. De la figure 2.21, on s'aperçoit qu'il est impossible d'atteindre tous les routeurs dans le réseau en utilisant un seul port d'entrée/sortie car le chemin de routage est limité à 9 traversées de routeur. Dans ce cas, un autre port d'entrée/sortie du réseau ANOC sera utilisée, cf. figure 2.22(a). Chaque port est utilisé pour tester plusieurs routeurs et liens de son côté. Presque tous les routeurs et les liens du réseau sont couverts avec cette technique. Cependant, il reste encore un groupe de quatre routeurs illustré dans la figure 2.22(b) qui ne peut pas être testé car il a besoin d'un chemin de routage constitué de 11 passages de routeur. Ce problème n'est pas soluble même si nous utilisons deux ports distincts en même temps, un pour rentrer les données de test et l'autre pour récupérer les réponses, voir le chemin de routage en pointillé. En effet, ce chemin de routage inclut 10 routeurs et a donc besoin d'un champ « *path-to-target* » de 10 traversées de routeur. La question est pourquoi on n'utilise pas un troisième port d'entrée/sortie pour résoudre ce problème. On note bien que l'ajout d'un port d'entrée/sortie a besoin de 194 pins d'entrée/sortie primaires. C'est un coût prohibitif.

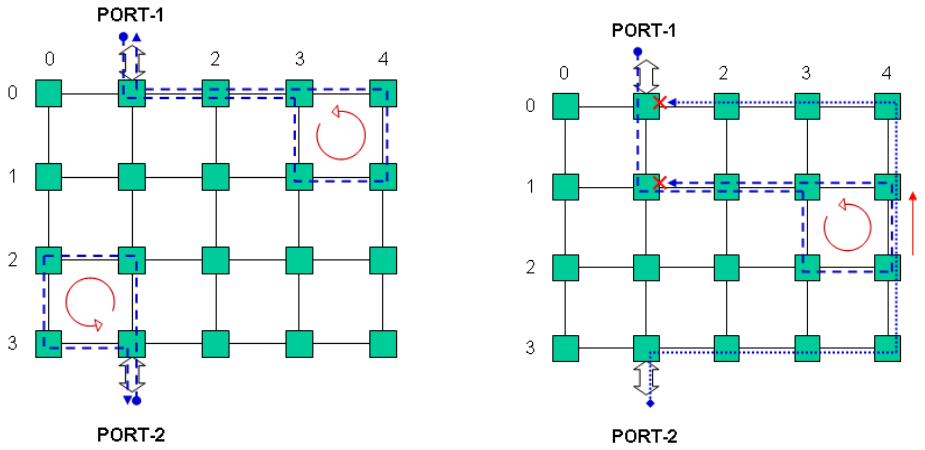


FIG. 2.22 : Configuration de chemins de test pour le réseau ANOC.

Nous pourrions certainement élargir le champ « *path-to-target* » du *flit* d'en tête de quelques traversées de routeur mais ceci est très limité par rapport à la taille d'un *flit*. Une autre solution consiste à utiliser un *flit* qui suit le *flit* d'en tête pour contenir les informations de routage supplémentaires. Ces informations de routage peuvent être déplacées au *flit* d'en tête en cas de besoin. Avec cette solution, nous pouvons atteindre tous les routeurs, tous les liens de réseau avec n'importe quelle taille. Cependant, cette solution rend l'implémentation du réseau beaucoup plus complexe.

Même si nous pouvions atteindre tous les routeurs et tous les liens dans le réseau, cette approche de test possède encore d'autres limitations. Une de ces limitations est qu'il n'est pas possible de vérifier les routages concurrents. Une faute de fabrication, par exemple une faute de délai, peut changer l'ordre de l'arbitrage du routeur. Pour vérifier l'arbitrage d'un routeur dans le cas où il y a deux ou plusieurs données en entrée simultanément, il faut garantir que les données concurrentes arrivent aux entrées du routeur sous test en même temps. Cependant, il est très difficile de maîtriser le temps de propagation de données à cause de la latence dans les réseaux, en particulier les réseaux asynchrones. On rappelle qu'il n'existe pas d'horloge dans les circuits asynchrones, il est donc impossible de contrôler des données en pas à pas jusqu'au routeur sous test. En conséquence, nous n'avons aucune manière de garantir les occurrences simultanées des données aux entrées du routeur.

De plus, même dans le cas du test d'un groupe de quatre routeurs, s'il y a une faute nous ne pouvons pas préciser où la faute se situe dans ce groupe constitué de quatre routeurs et liens. On note que le routeur défectueux ou le lien défectueux peut être isolé et qu'on peut continuer à utiliser le réseau pour certaines applications.

En conclusion, l'approche consistant à utiliser le réseau pour tester le réseau lui-même permet de supprimer l'ajout de matériel pour le mécanisme d'accès du test. En effet, le coût du TAM est nul dans cette approche. De plus, la technique de transport des données de test est la même que celle utilisée pour la communication dans le réseau. Cependant, cette approche possède plusieurs limitations. Elle ne permet pas de localiser précisément les fautes s'il y en a. La longueur du chemin de routage est limitée, cela rend l'accès aux routeurs plus difficile, notamment dans le cas où le réseau a un seul port d'entrée/sortie primaire. Enfin, l'approche n'a pas la capacité de vérifier les fonctionnalités du réseau concernant la transmission concurrente de *flits*.

2.5 Conclusion

Le test des systèmes sur puce est de plus en plus difficile car les systèmes sont de plus en plus complexes en termes d'architecture et de méthodologie de conception. Le test des SoC pour les défauts de fabrication doit être fait en utilisant plusieurs approches : test structurel, test fonctionnel, etc. et doit être considéré à plusieurs niveaux : système, blocs d'IP, porte logique, etc. La conception actuelle des systèmes SoC est basée sur la réutilisation de blocs d'IP, le test de ces systèmes doit donc être standardisé afin de s'adapter aux besoins de cette approche de conception et afin de s'adapter aux dimensions variables des systèmes. La construction de la norme de test IEEE 1500 est un exemple typique de solution à ces problèmes.

Les réseaux sur puce favorisent la réutilisation de blocs pré-conçus afin d'améliorer les temps et les coûts de conception. Celle réutilisation d'un grand nombre d'IP se répercute naturellement sur le test du circuit. En effet, le temps de test et la consommation d'énergie pendant le test sont aussi des problèmes importants pour le test des systèmes complexes tels que les NoC.

D'autre part, la conception synchrone rencontre des limitations technologiques. La conception asynchrone promet une solution alternative. De plus en plus d'applications sont construites en utilisant les circuits asynchrones. Cependant, pour faire évoluer la conception asynchrone, il reste à surmonter les problèmes suivants : (i) le manque d'outils CAO appropriés pour la conception et le test ; (ii) le surcoût en surface des techniques particulières utilisées pour éviter les aléas [Haze92T, Hulg95T].

Dans l'attente du développement d'outils CAO et de nouvelles méthodologies de conception asynchrone, une solution intermédiaire est généralement choisie par les concepteurs. Avec cette solution, les systèmes se composent de parties synchrones et de parties asynchrones. Cette approche nous permet de résoudre certains des problèmes rencontrés lors de la conception synchrone et de ne pas entrer trop pro-

fondément dans la conception asynchrone. Le paradigme GALS, et en particulier le cas de notre système sur puce basé sur un réseau asynchrone, est un exemple de ce type de solution. Afin de réaliser le transfert industriel de ces systèmes sur puce basés sur un réseau asynchrone, il faut cependant les tester. Le test des parties synchrones et asynchrones doivent être réalisés séparément en utilisant différentes méthodologies de test. Les parties synchrones dans ces systèmes peuvent être testées en utilisant les méthodologies de test classiques. Les parties asynchrones peuvent être testées en utilisant une technique *ad-hoc* ou les techniques de « *scan* » présentées dans la section 2.3. Cependant, ces techniques de test consomment beaucoup de temps et sont très coûteuses en surface. De plus, il est difficile de générer des vecteurs de test pour les circuits asynchrones.

Dans notre cas, le test d'un système sur puce basé sur un réseau asynchrone, il faut trouver une solution plus simple et plus efficace qui respectera les caractéristiques des NoC suivantes : modularité, flexibilité, réutilisation d'IP, etc. De plus, la solution doit supporter à la fois le test des parties asynchrones et le test des parties synchrones dans le système. C'est l'objectif de cette thèse. Les chapitres suivants présentent notre solution qui respecte ces exigences.

Chapitre 3

Proposition d'une Architecture CVT pour le Réseau ANOC

Dans le premier chapitre, nous avons présenté le paradigme des architectures NoC et ses avantages potentiels dans la conception des SoC futurs, notamment les réseaux sur puce asynchrones. Cependant, il faut mettre au point une méthode de test pour ces nouvelles architectures avant de les mettre sur le marché.

Dans le deuxième chapitre, nous avons abordé les différentes méthodes de test qui existent pour les systèmes sur puce classiques dans le but d'identifier des méthodes qui conviennent aux NoC. En fait, certaines de ces méthodes peuvent être adoptées pour tester les unités de traitement dans les NoC mais il est nécessaire de développer une nouvelle méthode de test pour l'architecture du réseau de communication en lui-même à cause de sa structure particulière. En effet, le test de l'architecture du réseau devient beaucoup plus difficile si l'architecture du réseau est implémentée en logique asynchrone à cause du manque d'outils CAO et de méthodologies automatisées. La conception et le test des circuits asynchrones ont été abordés au cours du second chapitre. Ensuite, nous avons présenté l'approche de test utilisée pour tester le réseau ANOC implémenté dans le circuit FAUST. Cette approche est très simple mais ne permet pas de tester toutes les fautes existantes dans le réseau.

De ce fait, nous avons proposé une nouvelle méthode de test pour les réseaux sur puce asynchrones, appliqué au réseau ANOC. L'objectif de ce chapitre est de décrire cette nouvelle approche de test et de présenter comment réaliser et implémenter cette méthode de test sur silicium. Dans une première section, on décrit la méthode générale, puis on propose une solution architecturale. Cette solution architecturale sera détaillée dans la deuxième section. La conception, la synthèse, l'optimisation, et

l'implémentation de cette architecture seront présentées dans la troisième section.

3.1 Nouvelle approche pour tester les réseaux sur puce, application au réseau ANOC

3.1.1 Description de l'approche

Selon notre discussion dans le dernier chapitre, les tests structurels existants pour les circuits asynchrones tels que les approches *ad hoc* et *scan* sont très coûteux en termes de surface et de temps d'application du test. La proposition basée sur les techniques de *scan* de Efthymiou [Efth05T] pour le réseau asynchrone CHAIN [Bain02C] en est un exemple. De plus, il manque des outils CAO pour le test des circuits asynchrones ; il n'existe à notre connaissance aucun outil de génération de vecteurs de test (ATPG) ni de simulateur de fautes adapté aux circuits asynchrones.

Les réseaux sur puce possèdent des caractéristiques qui permettent d'appliquer facilement et efficacement des vecteurs de test fonctionnels. En effet, les routeurs possèdent peu de fonctionnalités et sont tous identiques ; il est ainsi possible de couvrir facilement l'ensemble de leurs fonctionnalités avec des vecteurs de test générés manuellement. Pour ces raisons, nous avons préféré commencer par développer une méthode de test de type fonctionnel pour les réseaux sur puce asynchrones. Cette méthode a été appliquée au réseau ANOC. D'autre part, en utilisant le test fonctionnel, nous avons la possibilité de réutiliser les vecteurs de test qui sont utilisés dans l'étape de vérification. Ensuite, ces vecteurs seront complétés pour atteindre des taux de couverture de fautes de collage suffisants.

Dans le chapitre précédent, nous avons présenté une méthode de test fonctionnel appliquée au réseau ANOC de la première version du circuit FAUST. Cependant, cette méthode possède beaucoup de limitations et n'est pas suffisante pour tester le réseau, cf. section 2.4. De ce fait, des techniques de CVT doivent être appliquées pour améliorer la testabilité du réseau. Le problème est de pouvoir isoler chaque élément du réseau pour les tester indépendamment les uns des autres. La CVT doit permettre d'accéder aux éléments sous test pour insérer les vecteurs de test et observer les réponses.

L'approche proposée consiste à développer d'une part un mécanisme d'accès de test (*TAM : Test Access Mechanism*) afin de transporter les vecteurs de test vers tous les éléments du réseau, et d'autre part un mécanisme d'insertion de ces vecteurs de test aux éléments sous test et de récupération des réponses. Dans le cadre de cette approche, tous les éléments du réseau qui peuvent être réutilisés doivent l'être afin

de réduire le coût de test en termes de surface et de temps d'application.

3.1.2 Architecture CVT proposée

Pour répondre aux exigences abordées ci-dessus, nous avons proposé l'architecture CVT illustrée dans la figure 3.1. Dans cette architecture, chaque routeur du

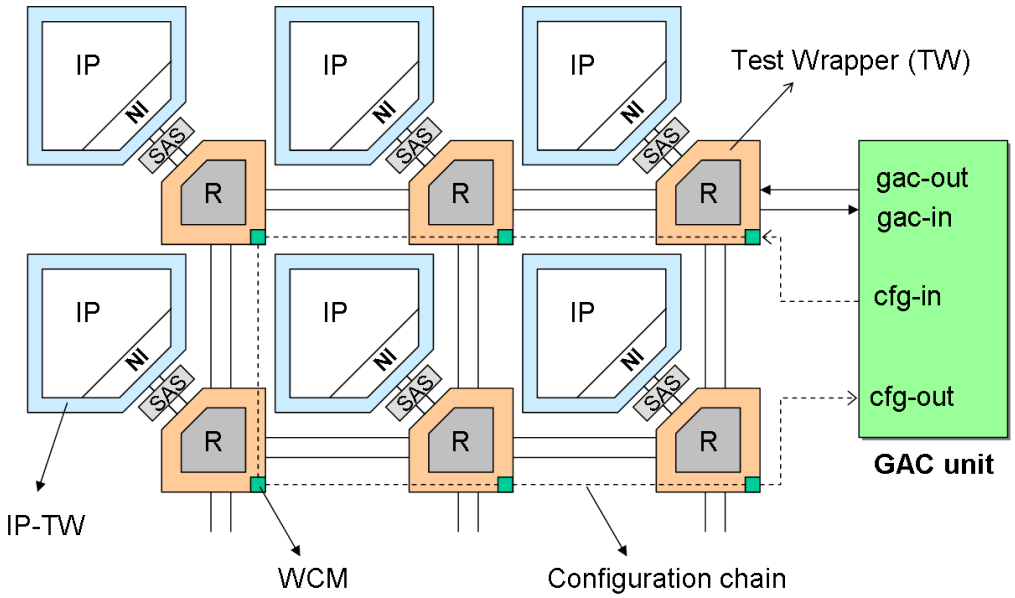


FIG. 3.1 : Architecture CVT proposée.

réseau est entouré par une enveloppe de test, appelé wrapper de test (*TW : Test Wrapper*), afin d'améliorer la contrôlabilité et l'observabilité des routeurs. Ce wrapper de test nous permet d'isoler les éléments sous test du reste du système. Les liens entre routeurs sont réutilisés pour établir un mécanisme d'accès de test.

Grâce à cette architecture CVT, nous avons la capacité d'atteindre et d'isoler tous les éléments du réseau pour les tester. Comme l'approche n'utilise pas le protocole de communication du réseau pour envoyer les données de test, le problème précédent lié à la taille du champ « *path-to-target* » pour le transport des données de test sur le réseau (cf. section 2.4) est résolu.

Ce wrapper de test joue trois rôles : (*i*) il est utilisé pour insérer les vecteurs de test aux éléments sous test (routeurs et liens du réseau), (*ii*) il est utilisé pour observer les réponses de test, et (*iii*) il est également utilisé, avec les liens du réseau, pour réaliser le mécanisme d'accès du test afin de transporter des données de test.

La réutilisation des liens du réseau nous permet de réaliser un TAM à large bande passante et de réduire le coût en surface de ce TAM. Il permet aussi d'éviter les problèmes de congestion des liens à l'étape de dessin des masques (*layout*).

Afin de minimiser le nombre des signaux de contrôle globaux et de simplifier la gestion des opérations de l'architecture, chaque wrapper de test possède un module de contrôle (*WCM : Wrapper Control Module*), chargé d'interpréter le contrôle global et de générer les signaux locaux contrôlant les opérations du wrapper. Pour contrôler le fonctionnement de l'architecture CVT entière, nous utilisons un contrôleur principal externe. Ce contrôleur envoie des configurations de test aux modules WCM sur une chaîne de 2-bits dédiée, appelé chaîne de configuration de test. Cette chaîne est établie en connectant en série tous les modules WCM des wrappers de test. On note que cette chaîne n'est pas une chaîne de données mais une chaîne de configuration. Les détails du wrapper de test et de son module de contrôle seront présentés dans les sections suivantes.

Le contrôleur principal qui contrôle le fonctionnement de cette architecture CVT est intégré dans l'unité GAC (Générateur, Analyseur, Contrôleur). C'est une unité qui permet de générer des vecteurs, de générer des configurations, et d'analyser les réponses de test. Selon l'architecture, cette unité peut être implémentée sur puce comme un bloc BIST (*Built-In Self-Test*) ou hors puce comme un testeur externe. Il s'interface avec le NoC en utilisant deux canaux de données de 34 bits (un canal de sortie *gac-out* et un canal d'entré *gac-in*), et deux extrémités de la chaîne de configuration de test (*cfg-in* et *cfg-out*).

Cette architecture CVT a été conçue, synthétisée, et implémentée en logique asynchrone pour bien s'adapter au reste du réseau sur puce asynchrone : il n'y a ainsi pas d'horloge de test, ni de coût en surface supplémentaire pour les interfaces de synchronisation associées. La logique asynchrone QDI a été utilisée pour s'interfacer avec la logique QDI du réseau ANOC.

L'architecture proposée fournit également un accès de test pour tester les unités de traitement en cas de besoin. Ce test des unités de traitement et de leurs interfaces réseaux (*NI : Network Interface*) peut être réalisé en utilisant des approches classiques telles que les chaînes de *scan* et/ou la norme IEEE 1500. Les unités de traitement sont également entourées par un wrapper de test (*IP-TW : IP Test Wrapper*). Ces wrappers doivent pouvoir communiquer avec le réseau et aussi avec le wrapper des routeurs. Le test des unités de traitement n'est pas l'objet de cette thèse. Dans les sections suivantes, on utilisera le terme « wrapper de test » (*TW : Test Wrapper*) pour désigner les wrappers de test qui entourent les routeurs du réseau. Lorsqu'on devra parler du wrapper de test pour les IP, on le précisera en écrivant « wrapper de test d'IP » ou simplement « IP-TW ».

3.2 Wrapper de test

3.2.1 Proposition d'architecture pour le wrapper de test

Comme nous l'avons abordé dans la section précédente, l'objectif du wrapper de test est d'améliorer la contrôlabilité et l'observabilité des éléments du réseau (routeurs et liens du réseau). Ainsi, le wrapper doit fournir des accès aux entrées et aux sorties du routeur sous test. Les liens du réseau peuvent être testés en utilisant les wrappers de deux routeurs. Dans la plupart des SoC actuels, le wrapper de test se compose de plusieurs cellules de test d'un bit de large, et d'une cellule pour chaque lien d'entrée ou de sortie. Ces cellules doivent être connectées au TAM afin de transporter des données de test vers ou à partir du routeur. Comme le routeur du réseau ANOC a 5×34 liens d'entrée et 5×34 liens de sortie, l'utilisation d'un TAM parallèle pour ce wrapper nécessiterait un TAM de largeur 5×34 bits, soit un coût énorme en matériel. L'utilisation d'un TAM série pour ce wrapper donnerait un TAM de 1 bit mais on rencontrerait alors un problème de temps d'application de test. En considérant les caractéristiques du réseau, on préfère réutiliser les liens du réseau pour construire le TAM et développer des cellules de test de 34 bits de large correspondant à la taille des liens du réseau. La largeur du TAM de l'architecture proposée est donc de 34 bits. Le wrapper de test se compose de 5 cellules d'entrée et de 5 cellules de sortie correspondant au nombre de ports d'entrée et de ports de sortie de chaque routeur, cf. figure 3.2.

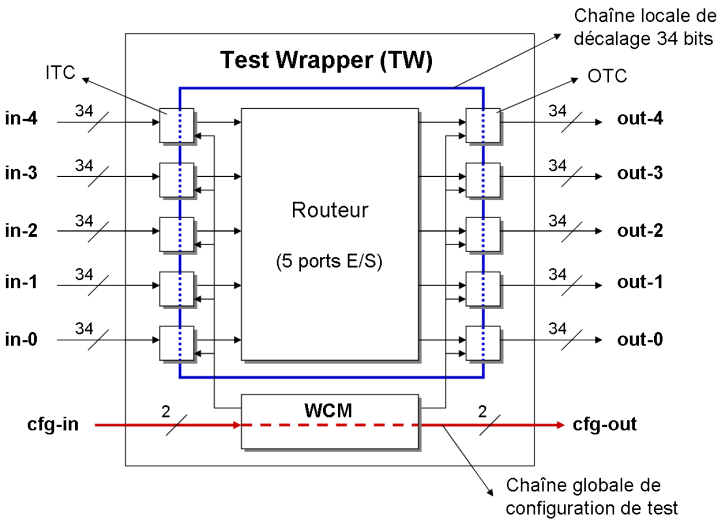


FIG. 3.2 : Wrapper de test (vue conceptuelle).

Ces cellules de test connectées en série établissent une chaîne locale de décalage

de 34 bits de large autour du routeur. Ces cellules de test sont contrôlées par le module de contrôle WCM intégré dans le wrapper. Dans les paragraphes suivants, nous allons rentrer dans le détail du fonctionnement de ces cellules de test et de ce module WCM.

3.2.2 Cellules de test

Afin de ne pas changer le fonctionnement du réseau, ces cellules de test doivent avoir la capacité lorsque le système est en fonctionnement normal de faire entrer les données depuis l'extérieur vers le routeur et de faire sortir les données du routeur vers l'extérieur. De plus, en mode test ces cellules de test doivent pouvoir faire circuler les données de test dans la chaîne de décalage pour les transporter aux différentes entrées du routeur sous test ou pour retirer des données de test à partir des différentes sorties du routeur. Pour répondre à ces exigences, la cellule du wrapper peut donc être construite comme illustré dans la figure 3.3(a).

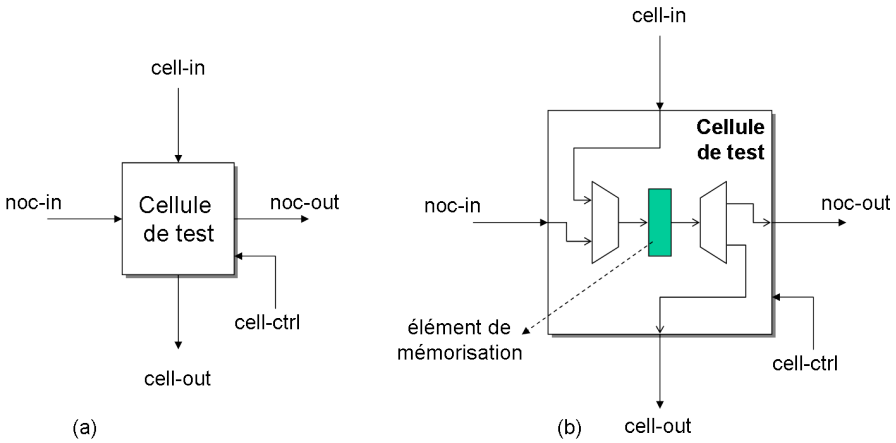


FIG. 3.3 : Architecture d'une cellule de test : (a) vue externe, (b) vue fonctionnelle interne.

L'architecture de la cellule de test comprend deux entrées de 34 bits (*noc-in* et *cell-in*), et deux sorties de 34 bits (*noc-out* et *cell-out*), et une entrée de contrôle (*cell-ctrl*). Pour les cellules d'entrée, l'entrée *noc-in* vient des liens du réseau, l'entrée *cell-in* vient de la cellule précédente, la sortie *noc-out* part vers le routeur, et la sortie *cell-out* part vers la cellule suivante. Pour les cellules de sortie, l'entrée *noc-in* vient du routeur, l'entrée *cell-in* vient de la cellule précédente, la sortie *noc-out* part vers les liens du réseau, et la sortie *cell-out* part vers la cellule suivante.

Le fonctionnement de la cellule d'entrée peut être expliqué de la manière suivante. Dans le mode normal (mode transparent), les données de communication venant du

réseau sont transportées directement vers le routeur. Dans le mode test, les données de test qui viennent du réseau ou de la cellule précédente, sont capturées dans la cellule courante. Puis ces données sont transportées : soit vers le routeur, soit vers la cellule suivante.

De façon similaire, le fonctionnement de la cellule de sortie peut être expliqué de la manière suivante. Dans le mode normal, les données de communication venant du routeur sont transportées directement vers le réseau. Dans le mode test, les données de test qui viennent du routeur sous test ou de la cellule précédente, sont capturées dans la cellule courante. Puis ces données sont transportées : soit vers le réseau, soit vers la cellule suivante.

Pour implémenter ces fonctions, une architecture simple de la cellule a été proposée, cf. figure 3.3(b). Dans cette architecture, nous avons utilisé un multiplexeur et un démultiplexeur pour choisir entre les entrées et les sorties. Les choix des entrées et des sorties sont décidés par la valeur du signal de contrôle *cell-ctrl* fourni par le module de contrôle WCM. Pour avoir la capacité d'enregistrer les données dans le wrapper, les cellules incluent un élément mémoire de la taille d'un *flit* 34 bits.

3.2.3 Module de contrôle du wrapper (WCM)

Le module de contrôle du wrapper, appelé module WCM (*Wrapper Control Module*) est le contrôleur local de chaque wrapper de test. Il reçoit les configurations de test du contrôleur principal (intégré dans l'unité GAC), analyse ces configurations et distribue les signaux de contrôle locaux vers les cellules de test du wrapper. Cette architecture permet de minimiser le nombre de contrôles globaux et de simplifier la gestion des opérations de l'architecture entière. La figure 3.4 présente une structure d'interconnexion en série des modules WCM, qui établit une chaîne de configuration de test. Cette structure d'interconnexion permet de réduire le nombre d'interconnexions entre l'unité GAC et les modules WCM.

Avec cette structure d'interconnexion, chaque configuration de test, appelée TCF (*Test Configuration Frame*), est envoyée à la chaîne de configuration avec un identifiant (ID). Les modules WCM reçoivent ces TCF et en vérifient l'ID. Si une TCF reçue possède un ID correspondant à celle du wrapper de test, il exécute cette TCF et envoie les signaux de contrôle correspondants aux cellules de test du wrapper. Sinon, cette TCF sera décalée au wrapper suivant.

3.2.4 Interconnexions des cellules de test dans le wrapper

Dans la figure 3.2 (section 3.2.1), nous avons présenté une architecture simple du wrapper de test sans considérer les échanges des données entre les cellules du

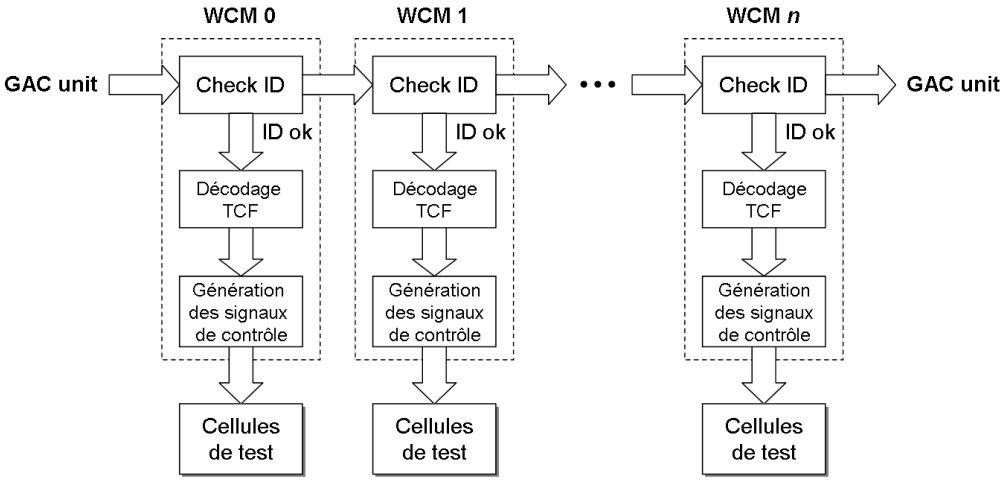


FIG. 3.4 : Interconnexion des modules de contrôle des wrappers.

wrapper. Ici, nous allons présenter différentes manières d'interconnecter les cellules de test. Ces solutions seront évaluées de manière à déterminer la solution optimale par rapport au temps d'application du test.

Dans la première solution, on suppose que toutes les cellules sont connectées en série pour établir une chaîne de décalage autour du routeur en regroupant les cellules d'entrée sur un côté et les cellules de sortie sur l'autre côté, cf. figure 3.5. Cette architecture est une architecture classique que l'on utilise souvent pour le test

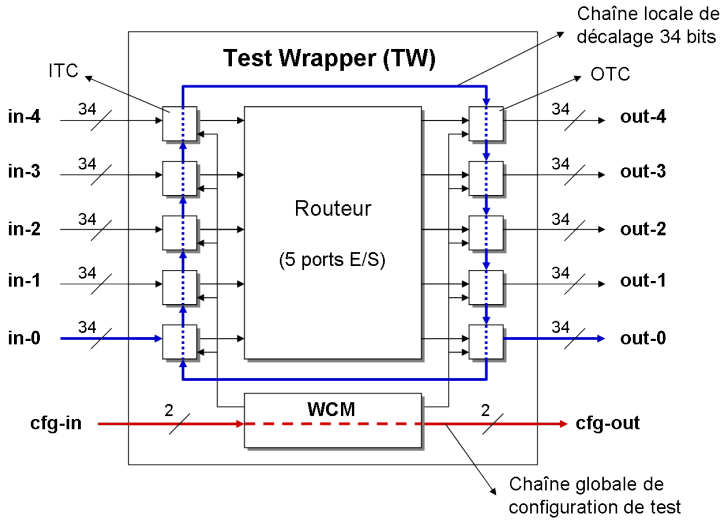


FIG. 3.5 : Architecture du wrapper en vue de conception classique.

des circuits synchrones (en particulier, elle est utilisée dans la norme IEEE 1500). En fait, dans les approches synchrones il faut imposer une valeur sur chaque entrée

puis récupérer les valeurs sur les sorties. De ce fait, l'agencement des entrées puis des sorties dans le wrapper est naturel.

Par exemple, pour étudier ce type d'interconnexion des cellules de test, considérons un réseau composé de quatre routeurs, cf. figure 3.6. Pour simplifier, on ne

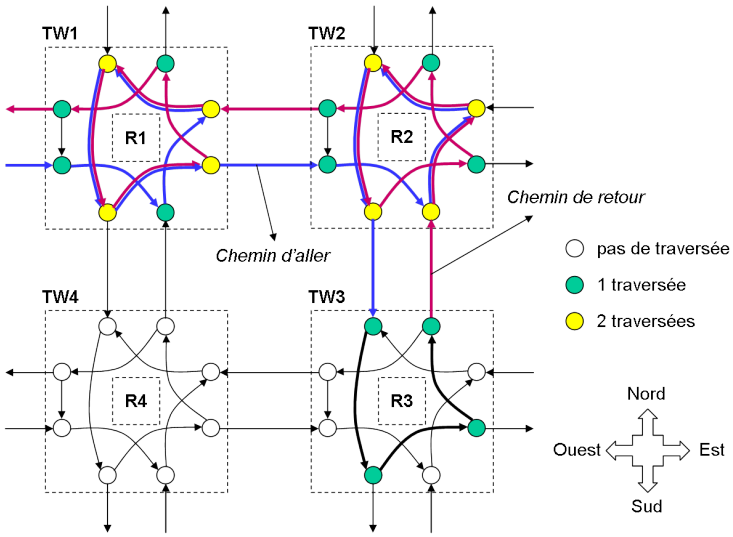


FIG. 3.6 : S1 - Interconnexions entre les cellules en regroupant des entrées puis des sorties.

représente pas dans cet exemple les connexions entre la ressource de traitement et le réseau, ni les interconnexions entre le wrapper et le routeur.

Dans le mode normal, les cellules sont transparentes, le réseau fonctionne comme s'il n'y avait pas de wrappers de test autour des routeurs. Maintenant, considérons les chemins des données de test dans le mode test. On suppose que l'unité GAC est connectée à l'Ouest du wrapper de test TW1 et que l'on veut tester le routeur R3. Les données de test rentrant de l'Ouest du wrapper TW1 doivent traverser les wrappers TW1 et TW2 pour arriver au Nord du wrapper TW3 (chemin aller). Les données de test sortant du Nord du wrapper TW3 doivent également traverser ces wrappers pour retourner à l'Ouest du wrapper TW1 (chemin retour). Nous voyons que les données de test passent deux fois par certaines cellules. Le module WCM doit donc envoyer deux TCF pour chaque wrapper TW1 et TW2 : une TCF pour configurer le chemin aller et l'autre pour configurer le chemin retour. On remarquera que le temps de contrôle est beaucoup plus long que le temps de traversée des cellules, cf. section 3.3.5.

Pour éviter ces problèmes, nous allons étudier dans ce qui suit l'interconnexion des cellules en tenant compte de leurs positions : les cellules d'entrée et les cellules de sortie sont interconnectées alternativement, cf. figure 3.7. Le sens de décalage

dans le wrapper est anti-horaire.

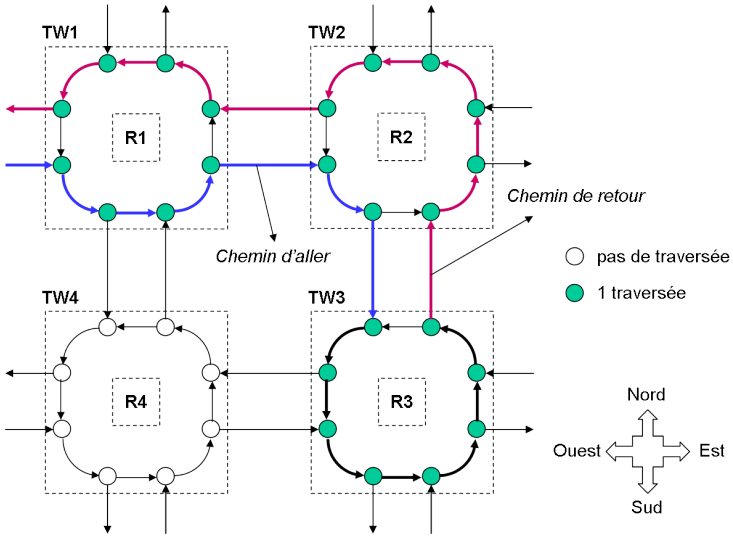


FIG. 3.7 : S2 - Interconnexions entre les cellules en tenant compte leurs positions.

Avec cette nouvelle architecture, les données de test entrant à l'Ouest du wrapper TW1 doivent aussi traverser les wrappers TW1 et TW2 pour arriver au wrapper TW3. Les données de test sortant du wrapper TW3 doivent aussi traverser ces wrappers pour retourner à l'Ouest du wrapper TW1. Cependant, le chemin aller et le chemin retour ne se croisent jamais. Les données de test traversent seulement une fois chaque cellule. Ceci permet de minimiser les TCF du WCM et donc permet de réduire le temps de contrôle.

Dans la figure 3.8 ci-dessous, nous présentons la même solution de positionnement alternative entre les cellules d'entrée et les cellules de sortie mais le sens de décalage des chaînes de décalage est l'inverse de la solution précédente. C'est-à-dire que le décalage est en sens horaire.

Nous voyons que cette solution permet de faire un demi-tour très rapide dans le cas où on veut tester les interconnexions entre deux routeurs. Dans la figure 3.8, les données ne doivent traverser que deux cellules de test du wrapper TW3 pour tester les liens entre les routeurs R2 et R3. On passe de l'entrée d'une direction à la sortie de cette même direction. Cependant, le chemin aller et le chemin retour (par TW1 et TW2) se croisent. Ceci double le temps de contrôle car on a besoin de deux TCF pour chaque wrapper de test.

Le tableau 3.1 résume le nombre de TCF, le nombre de fois que les cellules sont traversées, et le temps de test pour les trois solutions présentées. Les trois premières lignes présentent les résultats du test des liens du réseau entre R2 et R3. Les trois dernières lignes présentent les résultats du test du routeur R3.

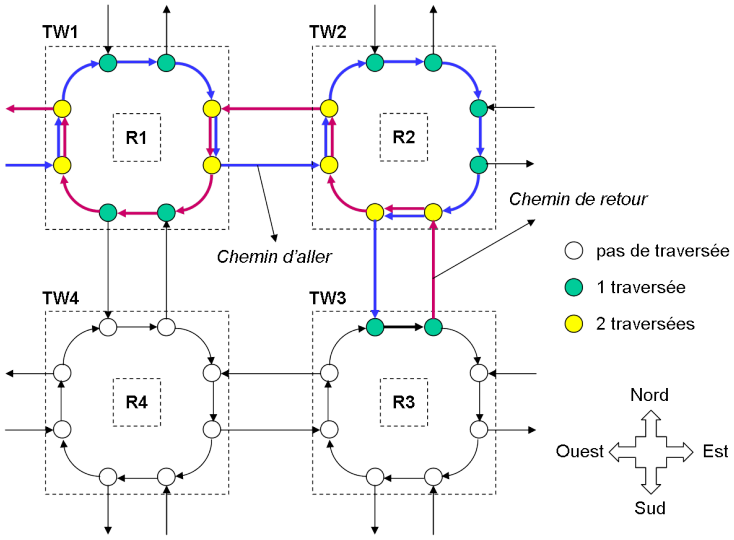


FIG. 3.8 : S3 - Interconnexion identique à la figure précédente mais décalage dans le sens horaire.

TAB. 3.1 : Nombre des configuration de test, nombre de traversées des cellules, et temps de test calculés

Test	**Solution**	**Nombre de TCF**	**Nombre de traversées**	**Temps de test**
Liens R2-R3	S1	$5N$	$28N$	$(5T_{TCF} + 28T_C)N$
	S2	$3N$	$24N$	$(3T_{TCF} + 24T_C)N$
	S3	$5N$	$26N$	$(5T_{TCF} + 26T_C)N$
Routeur R3	S1	$6M$	$(24 + 2k)M$	$(6T_{TCF} + (24 + 2k)T_C)M$
	S2	$4M$	$(16 + 2(2k - 1))M$	$(4T_{TCF} + (16 + 2(2k - 1))T_C)M$
	S3	$6M$	$(24 + 2(2k - 1))M$	$(6T_{TCF} + (24 + 2(2k - 1))T_C)M$

N est le nombre de vecteurs de test pour tester les liens du réseau et M est le nombre de vecteurs de test pour tester le routeur. T_{TCF} est le temps pour envoyer une configuration de test complète et T_C est le temps pour qu'un vecteur traverse une cellule.

Dans le test des liens du réseau entre R2 et R3, le nombre de traversées de cellules afin d'insérer un vecteur de test et de récupérer la réponse est égal à 28 pour la solution S1, égal à 24 pour la solution S2, et égal à 26 pour la solution S3. Le nombre des T_{TCF} utilisés pour ces solutions est 5, 3, 5 ; respectivement. Dans le test du routeur R3, le nombre de traversées de cellules afin d'insérer un vecteur de test et de récupérer la réponse est égal à $24 + 2k$ pour la solution S1, égal à

$16 + 2(2k - 1)$ pour la solution S2, et égal à $24 + 2(2k - 1)$ pour la solution S3. Le nombre des T_{TCF} utilisés pour ces solutions est 6, 4, 6 ; respectivement. On note que $1 \leq k \leq 5$ et que $T_C << T_{TCF}$ (cf. section 3.3.7.5). En conclusion, la deuxième solution S2 est choisie pour construire le wrapper de test afin d'optimiser le temps d'application du test. Dans la section suivante, nous allons présenter la conception détaillée, l'implémentation et l'optimisation de ce wrapper de test.

3.3 Conception, synthèse, optimisation et implémentation

3.3.1 Méthode de conception et d'implémentation

Pour modéliser les circuits asynchrones, nous pouvons utiliser plusieurs langages de description de matériel basés sur des algèbres de processus tels que CHP (*Communicating Hardware Processes*) [Mart86C], HASTE (autrefois TANGRAM) [Peet06H], ou BALSA [Edwa02B]. Ces langages permettent de décrire des processus concurrents communiquant par des synchronisations de type « poignée de main », et sont interprétés par des outils de synthèse qui conduisent à l'implémentation de ces processus.

Dans un premier temps, pour prototyper et valider rapidement notre architecture proposée ci-dessus, nous avons utilisé le langage SystemC [OSCI]. Il s'agit d'une bibliothèque C++ qui permet de décrire les comportements des circuits électroniques. Cependant, SystemC supporte pour l'instant principalement la modélisation des circuits synchrones. Pour modéliser des circuits asynchrones, nous avons utilisé les éléments « sc-fifo » avec quelques modifications afin de les adapter aux comportements des circuits asynchrones.

Malheureusement, l'utilisation de SystemC présente certains inconvénients. Premièrement, la capacité de mémorisation d'un canal de communication est au moins d'une place dans les FIFOs SystemC tandis que la capacité de mémorisation d'un canal de communication peut être 1/2 (WCHB), voire 0 (combinatoire) en CHP [Mart86C]. Ceci rend le circuit modélisé en SystemC plus pipeliné que nécessaire. Cependant, ceci convient avec parfois quelques adaptations pour simuler les comportements des circuits. Deuxièmement, avec SystemC nous ne pouvons modéliser que les canaux où la communication est à l'initiative de l'émetteur (*Push*). Finalement, nous ne pouvons pas voir la visualisation de deux transactions de données successives avec des valeurs identiques dans les traces VCD (*Value Change Dump*) car il n'existe pas de référence de temps.

Nous avons évalué notre approche en SystemC avec différentes solutions d'interconnexion des cellules de test afin de trouver la modélisation la plus appropriée. L'architecture proposée a été validée avec le réseau ANOC développé dans notre laboratoire. Le debug a été fait en consultant les fichiers *.vcd* et *.log*, et les fichiers de données.

Ensuite, pour décrire l'architecture candidate à plus bas niveau, nous avons utilisé le langage CHP. Cette architecture a été modélisée et validée en utilisant conjointement à ce langage l'outil TAST[1]. TAST est un outil construit par le laboratoire TIMA qui permet de simuler les circuits modélisés en CHP par des réseaux de Petri [Mura89P] et de synthétiser ces circuits. Le debug a été fait en consultant les réseaux de Petri et les fichiers de données.

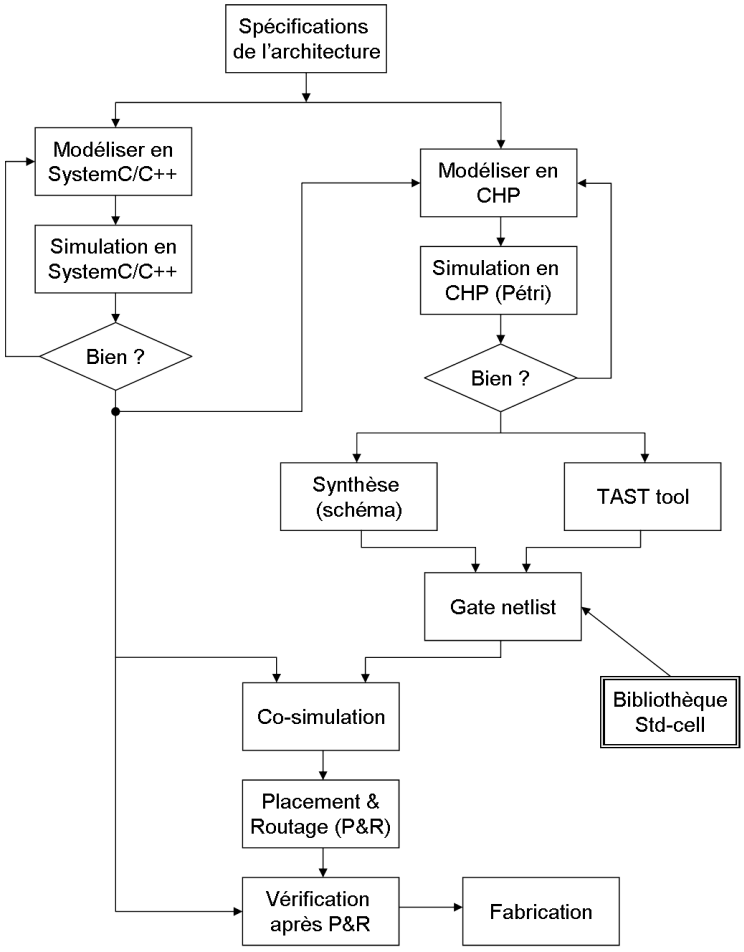


FIG. 3.9 : Flot de conception de l'architecture CVT proposée.

Finalement, l'architecture validée a été mise en œuvre en utilisant le design kit

[1]TIMA Asynchronous Synthesis Tool (TAST)

CMOS 65nm de STMicroelectronics (CMOS065LP) et la bibliothèque asynchrone 65nm du TIMA (TAL065). L'aspect synthèse de l'outil TAST étant encore peu mature, la synthèse a été effectuée à la main en utilisant l'outil *Cadence Virtuoso Schematic Editor*. La figure 3.9 présente un résumé du flot de conception pour l'architecture CVT proposée.

Les sous-sections suivantes présenteront la réalisation de l'architecture proposée. Le wrapper de test a été conçu et implémenté en utilisant un modèle de conception de logique asynchrone de type QDI WCHB. Nous présentons d'abord une implémentation d'un *buffer* asynchrone QDI afin de montrer le principe de mémorisation des données dans un pipeline QDI, qui sera utilisé dans le wrapper de test. Ensuite, nous allons présenter comment réaliser les différentes parties du wrapper de test. La synthèse, l'optimisation et les résultats de l'implémentation seront également présentés.

3.3.2 Buffer asynchrone QDI

Afin de montrer les principes qui permettent de stocker les données sur un canal asynchrone, on présente dans cette sous-section le comportement d'un *buffer* asynchrone, fait par la juxtaposition de deux processus asynchrones WCHB (deux *half-buffers*), cf. section 2.2.1.3, illustré dans la figure 3.10. Un processus asynchrone WCHB est parfois désigné sous le nom d'étage de pipeline asynchrone.

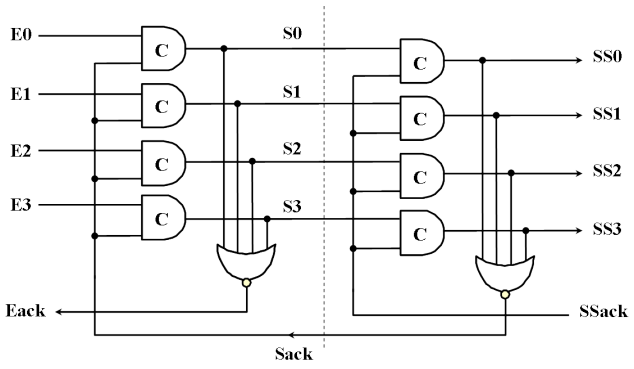


FIG. 3.10 : Un *buffer* de 2 bits en juxtapositionnant deux half-buffers asynchrones.

On note que ce *buffer* est implémenté en utilisant le protocole 4 phases avec remise à zéro (RTZ) et le schéma de codage de type « 1-of-4 » (MR[4]), cf. sections 2.2.1.1. Donc, son canal d'entrée contient 4 rails (*E0*, *E1*, *E2*, et *E3*) et un acquittement (*Eack*), et son canal de sortie contient 4 rails (*SS0*, *SS1*, *SS2*, et *SS3*) et un acquittement (*SSack*).

Le fonctionnement de ce *buffer* peut être expliqué de manière suivante : d'abord, on suppose que le canal de communication entre deux *half-buffers* est libre, le signal *Eack* se met à 1, et on suppose qu'une donnée entre sur le canal d'entrée du premier *half-buffer*. Ensuite, on suppose que le canal de sortie du deuxième *half-buffer* est connecté à un récepteur. En conséquence, la donnée entrant peut traverser le premier *half-buffer* si le canal de sortie du deuxième *half-buffer* est libre (le signal *Sack* est à 1). Cette donnée est stockée sur le canal de communication entre les *half-buffers* si le signal *SSack* est à 0 (le récepteur n'est pas prêt pour recevoir une nouvelle donnée). Dans ce cas-là, on ne peut pas faire entrer une autre donnée sur le canal d'entrée du premier *half-buffer* car le signal *Eack* est égal à 0. Quand le récepteur devient prêt pour recevoir une nouvelle donnée, il met le signal *SSack* à 1 et la donnée stockée sur le canal entre les *half-buffers* va traverser le deuxième *half-buffer* et sera lue par le récepteur. Le canal entre les *half-buffers* est libre et le signal *Eack* se met à 1. Maintenant, on peut faire entrer une nouvelle donnée.

Dans cet exemple, nous avons montré : (1) que les données peuvent être stockées sur un canal de communication entre des *half-buffers* dans les circuits asynchrones ; et (2) qu'on ne peut pas faire entrer une nouvelle donnée sur le canal d'entrée tant que le *buffer* est occupé. La capacité de mémorisation des données sur un canal de communication asynchrone donne la possibilité de stocker les données dans le wrapper de test sans besoin d'éléments de mémorisation classiques tels que des bascules D. De plus, l'utilisation d'un pipeline asynchrone pour construire les cellules de test permet de décaler les données de test sur le wrapper avec une vitesse maximale.

3.3.3 Conception, synthèse et optimisation d'une cellule de test

Dans la suite nous allons décrire les deux versions de la cellule de test. La première version dite « *slack-1* » permet de mémoriser un vecteur de test complet dans chaque cellule de test. La deuxième version dite « *slack-1/2* », plus adaptée au besoin du wrapper avec la solution d'interconnexion entre cellules retenues (cf. section 3.2.4) a besoin de deux cellules pour mémoriser un vecteur de test.

3.3.3.1 Conception d'une cellule de test – version « slack-1 »

La cellule de test est un composant essentiel du wrapper de test. Elle permet de faire l'interface entre le routeur et le reste de l'environnement. Comme nous l'avons présenté dans la section 3.2.2, dans le mode normal les données de communication venues du réseau doivent être transportées directement vers le routeur et les données venues du routeur doivent être transportées vers le réseau. Les cellules de test sont donc transparentes dans ce mode. Dans le mode test, au contraire les cellules de

test d'entrée (*ITC : Input Test Cell*) doivent stocker les vecteurs de test du réseau, puis les transporter vers le routeur sous test. De même, les réponses du routeur sont récupérées et transportées vers le réseau par les cellules de test de sortie (*OTC : Output Test Cell*). Pour implémenter ces fonctions, nous avons développé l'architecture de la cellule de test illustrée dans la figure 3.11. On considère dans cette première version que chaque cellule de test doit pouvoir stocker une donnée complète.

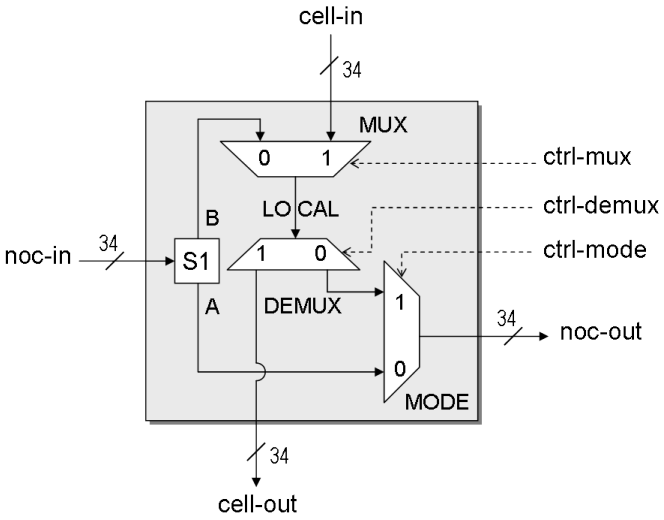


FIG. 3.11 : Architecture générale de la cellule de test.

Cette architecture comprend deux multiplexeurs (MUX et MODE), un démultiplexeur (DEMUX), et un *split* (S1). Le port *noc-in* est connecté à une entrée du multiplexeur MODE. Ceci permet de transporter des données plus vite dans le mode normal car les données ne doivent pas être enregistrées dans les cellules avant d'être envoyées au routeur par les cellules ITC ou au réseau par les cellules OTC. La performance du réseau n'est donc pas trop modifiée.

S1 est un bloc qui permet de relier le port *noc-in* aux deux entrées de deux multiplexeurs : MUX et MODE. Le but est de permettre aux deux processus MUX et MODE de partager la même donnée en entrée. En effet, en logique asynchrone, les canaux sont toujours point-à-point et ne peuvent être partagés qu'avec un bloc de duplication ou de séparation. Cependant, les données de cette entrée doivent être transportées soit à MUX, soit à MODE (de manière mutuellement exclusive), donc dans ce cas le bloc S1 est un processus de séparation (*split*) sans contrôle, pas un processus de duplication (*fork*). Ce processus de séparation peut être construit en logique combinatoire, ce qui sera expliqué dans la section suivante (section 3.3.3.2).

Le processus MUX permet de choisir entre les deux entrées *noc-in* et *cell-in*, le processus DEMUX permet d'envoyer les données soit au processus MODE, soit à

la sortie *cell-out*. Les données peuvent être stockées sur le canal LOCAL entre les processus MUX et DEMUX, cf. section 3.3.2. Le processus MODE permet de choisir les données venues de *noc-in* ou les données stockées sur le canal LOCAL.

En conséquence, dans le mode normal les données de communication sont transportées directement de *noc-in* à *noc-out* par le multiplexeur MODE. Dans le mode test, les données qui viennent de *noc-in* ou de *cell-in* sont stockées sur le canal LOCAL. Puis, les données stockées sont transportées vers *noc-out* ou *cell-out*. Ces opérations sont contrôlées par les signaux qui viennent du module de contrôle du wrapper WCM : *ctrl-mux*, *ctrl-demux*, et *ctrl-mode*. Le tableau 3.2 présente les fonctionnements de la cellule selon ces signaux de contrôle.

TAB. 3.2 : Description des signaux de contrôle de la cellule.

ctrl-mux	***ctrl-demux***	***ctrl-mode***	**Description du fonctionnement de la cellule**
-	-	0	Mode normal
0	-	-	Recevoir les données de test de *noc-in* et les stocker dans la cellule
1	-	-	Recevoir les données de test de *cell-in* et les stocker dans la cellule
-	1	-	Transporter les données stockées à *cell-out*
-	0	1	Transporter les données stockées à *noc-out*
1	1	-	Décaler les données de test de *cell-in* à *cell-out*

***Note** : (-) signifie que ce signal de contrôle n'est pas concerné.*

Pour bien s'adapter au réseau, tous les blocs dans la cellule sont conçus et implémentés en logique asynchrone QDI. La largeur de chaque cellule est la même que la largeur d'un canal de communication du réseau, soit 34 bits. Cette cellule a été modélisée et validée en SystemC, puis en CHP.

3.3.3.2 Synthèse de plusieurs blocs de la cellule de test

Ce paragraphe présente la synthèse de quelques blocs de la cellule de test. Pour simplifier la présentation, on présente ici la description fonctionnelle du bloc en utilisant la syntaxe du langage CHP, sa représentation par l'expansion des communications (*HSE : HandShake Expansion*) [Mart90P], et puis le schéma correspondant. Nous rappelons que les blocs présentés dans cette section n'incluent pas les signaux de gestion, tels que « *send* » et « *accept* », des canaux virtuels et du flot des données. De plus, les chemins de données sont des chemins de 2-bit implémentés en utilisant

le protocole QDI 4-phase avec un encodage de type « 1-of-4 ».

– ***Multiplexeur à deux entrées***

Soit le multiplexeur suivant qui transmet une entrée A ou B sur la sortie S selon la valeur de contrôle C.

```
Processus P1 : [ C ? c ;
  @[
    c = 0 => EA ? data ; S ! data ; break //si c=0 alors S=EA
    c = 1 => EB ? data ; S ! data ; break //si c=1 alors S=EB
  ];
loop];
```

L'expansion des communications de ce processus en WCHB est donc :

```
[C0^EA0^Sa] ; S0+, EAa-, Ca- ; [/C0^/EA0^/Sa] ; S0-, EAa-, Ca+
[C0^EA1^Sa] ; S1+, EAa-, Ca- ; [/C0^/EA1^/Sa] ; S1-, EAa-, Ca+
[C0^EA2^Sa] ; S2+, EAa-, Ca- ; [/C0^/EA2^/Sa] ; S2-, EAa-, Ca+
[C0^EA3^Sa] ; S3+, EAa-, Ca- ; [/C0^/EA3^/Sa] ; S3-, EAa-, Ca+
[C1^EB0^Sa] ; S0+, EBa-, Ca- ; [/C1^/EB0^/Sa] ; S0-, EBa-, Ca+
[C1^EB1^Sa] ; S1+, EBa-, Ca- ; [/C1^/EB1^/Sa] ; S1-, EBa-, Ca+
[C1^EB2^Sa] ; S2+, EBa-, Ca- ; [/C1^/EB2^/Sa] ; S2-, EBa-, Ca+
[C1^EB3^Sa] ; S3+, EBa-, Ca- ; [/C1^/EB3^/Sa] ; S3-, EBa-, Ca+
```

Ces règles de production déterminent pour chacune des sorties *S0*, *S1*, *S2*, *S3*, deux possibilités mutuellement exclusives. Chaque entrée est gardée par une porte de Muller et des acquittements sont réalisés par les portes NON-OU. Les alternatives pour S étant mutuellement exclusives, des portes OU suffisent à joindre les deux canaux gardés *SA* et *SB*.

Pour les signaux d'acquittement, seul le canal qui a été lu (*EA* ou *EB*) doit être acquitté. Cette propriété permet d'acquitter facilement le canal de contrôle C, son acquittement est le ET des deux acquittements *EAa* et *EBa*.

Le canal *EA* est acquitté lorsque l'un des quatre états *S0+*, *S1+*, *S2+*, et *S3+* est généré pour la condition $C = 0$, réciproquement pour B lorsque $C = 1$. Ainsi, les différentes portes peuvent être regroupées avec des *half-buffer* commandés, cf. figure 3.12.

– ***Démultiplexeur à deux entrées***

Soit le démultiplexeur suivant qui transmet une entrée E sur l'une des sorties *SA* ou *SB* selon la valeur de contrôle C.

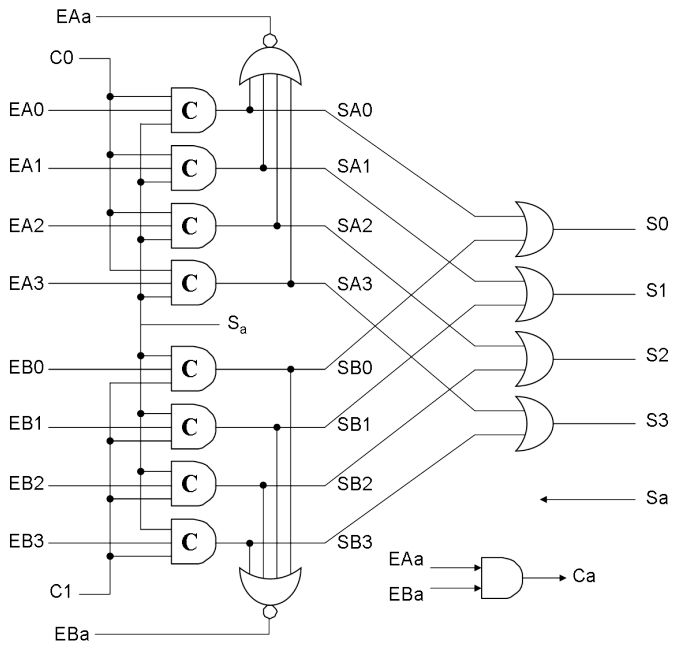


FIG. 3.12 : Implémentation d'un multiplexeur simple QDI 2-bit.

```
Processus P2 :  [ C ? c ;
  @[
    c = 0 => E ? data ; SA ! data ; break // si c=0 alors SA=E
    c = 1 => E ? data ; SB ! data ; break // si c=1 alors SB=E
  ] ;
loop] ;
```

La synthèse de ce processus en WCHB est donc :

```
[C0^E0^SAa] ; SA0+, Ea-, Ca- ; [/C0^/E0^/SAa] ; SA0-, Ea+, Ca+
[C0^E1^SAa] ; SA1+, Ea-, Ca- ; [/C0^/E1^/SAa] ; SA1-, Ea+, Ca+
[C0^E2^SAa] ; SA2+, Ea-, Ca- ; [/C0^/E2^/SAa] ; SA2-, Ea+, Ca+
[C0^E3^SAa] ; SA3+, Ea-, Ca- ; [/C0^/E3^/SAa] ; SA3-, Ea+, Ca+
[C1^E0^SBa] ; SB0+, Ea-, Ca- ; [/C1^/E0^/SBa] ; SB0-, Ea+, Ca+
[C1^E1^SBa] ; SB1+, Ea-, Ca- ; [/C1^/E1^/SBa] ; SB1-, Ea+, Ca+
[C1^E2^SBa] ; SB2+, Ea-, Ca- ; [/C1^/E2^/SBa] ; SB2-, Ea+, Ca+
[C1^E3^SBa] ; SB3+, Ea-, Ca- ; [/C1^/E3^/SBa] ; SB3-, Ea+, Ca+
```

Les règles de production donnent de manière évidente des portes de Muller pour les rails de données *SA0*, *SA1*, *SA2*, *SA3*, *SB0*, *SB1*, *SB2*, *SB3* à trois entrées. Les signaux d'acquittement *Ea* et *Ca* sont égaux, en effet les canaux d'entrées *E* et *C* sont toujours lus ensembles. *Ea* est donné par le NON-OU à huit entrées (*SA0*,

SA1, *SA2*, *SA3*, *SB0*, *SB1*, *SB2*, *SB3*), ou deux portes NON-OU à quatre entrées et une porte ET à deux entrées. Ainsi, les différentes portes peuvent ensuite être regroupées avec des *half-buffer* commandés, cf. figure 3.13.

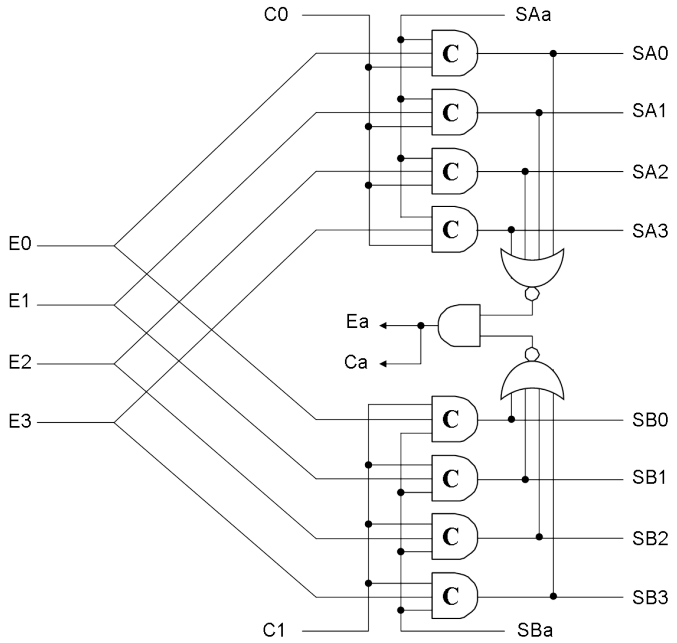


FIG. 3.13 : Implémentation d'un démultiplexeur QDI 2-bit.

– ***Split (processus de séparation)***
Soit le processus P3 suivant :

```
Processus P3 :  [  (A# or B#) => E ? data
  @[
    A# => A ! data ; break // si requête sur A alors A=E
    B# => B ! data ; break // si requête sur B alors B=E
  ] ;
loop] ;
```

Le processus de séparation (P3) permet de propager une entrée E sur l'une des deux sorties A ou B de manière mutuellement exclusive. Afin de construire ce processus, on peut utiliser les deux *half-buffers* des deux multiplexeurs MODE et MUX, cf. figure 3.14, pour optimiser le coût d'implémentation. Ceci rend une architecture similaire au démultiplexeur à deux entrées (P2) sauf que nous utilisons deux signaux de contrôle : *ctrl-mode* pour la partie de sortie S-MODE ; *ctrl-mux* pour la partie de sortie S-MUX.

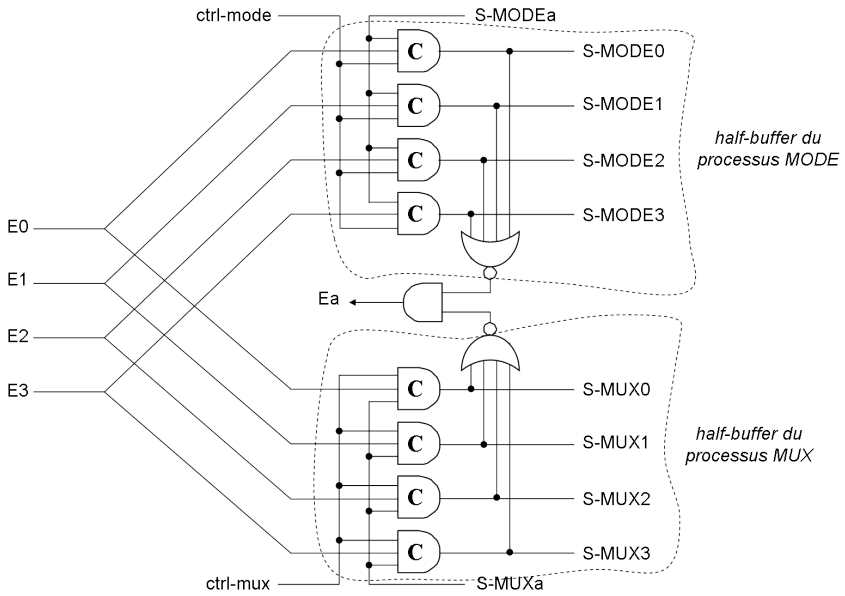


FIG. 3.14 : Implémentation d'un *split* en utilisant les *half-buffers* des multiplexeurs MODE et MUX.

Si on ne considère pas les *half-buffers* des multiplexeurs MODE et MUX, l'implémentation du processus de séparation est combinatoire, comme illustrée dans la figure 3.15. On constate en effet qu'il n'y a pas d'élément de mémorisation, et pas d'interaction entre la logique directe et la logique retour. La donnée est stockée entre le processus source et les processus destination.

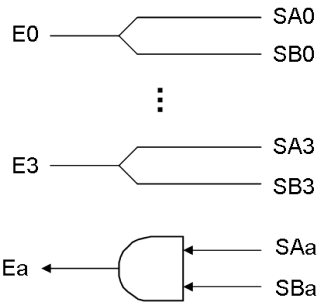


FIG. 3.15 : Implémentation d'un *split* réduit pour la cellule de test proposée.

– ***Multiplexeur à deux entrées avec la fonction probe pour le bloc MODE***

Soit le processus P4 suivant :

```
Processus P4 : [  C# =>
```

```
  @[
    C# = 0 => A ? data ; S ! data ; break //si c=0 alors S=A
    C# = 1 => C ? c , B ? data ; S ! data ; break
                              //si c=1 alors S=A et consommer c.
  ] ;
loop] ;
```

Le processus P4 (MODE) est similaire au processus P1 (MUX) sauf que le signal de contrôle C ne doit pas être consommé si $C0 = 1$. Dans ce cas-là, le processus est équivalent à un *half-buffer*. L'acquittement Ca est donc égal à l'acquittement EBa. Au contraire, si $C1 = 1$ le processus est équivalent à un *half-buffer* gardé. On note que les portes de Muller utilisées dans le canal EA sont des portes asymétriques de Muller bloquantes si $C0 = 0$ et passantes si $C0 = 1$, cf. figure 3.16.

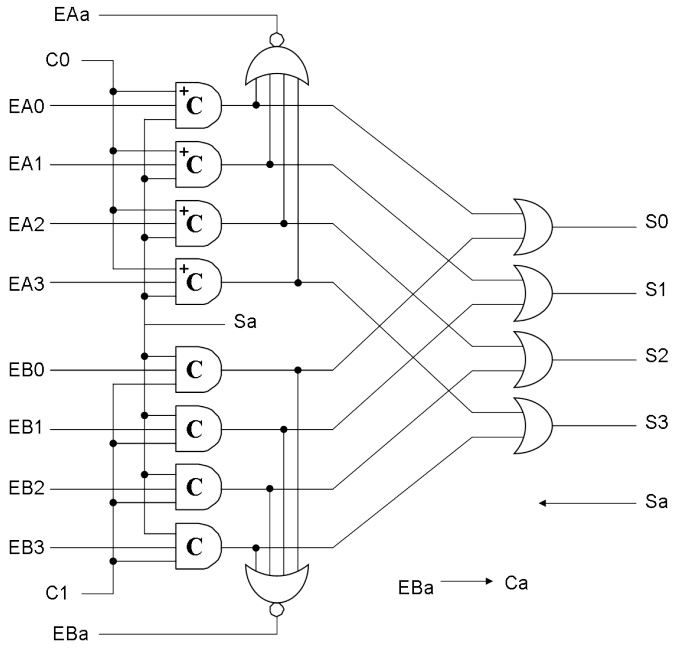


FIG. 3.16 : Implémentation d'un multiplexeur simple QDI 2-bit (pour le cas de MODE).

– ***Implémentation du choix des modes d'opération***

Afin de définir le mode initial du circuit, on utilise un plot d'entrée TM (*Test mode*). Tous les wrappers de test doivent être contrôlés par ce signal. Les wrappers de test peuvent être configurés en mode normal ou en mode test quand la valeur de la broche $TM = 1$ tandis qu'ils sont forcément en mode normal quand $TM = 0$.

En regardant l'architecture de la cellule de test dans la section 3.3.3.1, on voit que le canal de contrôle *ctrl-mode* peut être remplacé par une combinaison entre

ctrl-mode et le signal *TM*. La combinaison peut être représentée par les formules suivantes :

```
NC1 = C1 AND TM ;
NC0 = C0 OR NOT(TM) ;
```

Avec ces formules, la valeur *NC0* est forcément à 1 quand $TM = 0$ (mode normal). Quand $TM = 1$ (mode test), les valeurs de *NC0* et de *NC1* dépendent des configurations de test. Le schéma de cette fonction est décrit dans la figure 3.17.

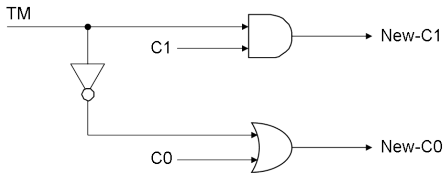


FIG. 3.17 : Implémentation de la broche *TM*

3.3.3.3 Optimisation – version « slack-1/2 »

En utilisant la cellule présentée dans la figure 3.11 pour construire le wrapper de test, nous voyons qu'il est possible d'optimiser l'architecture des cellules pour gagner en surface grâce au positionnement alternatif entre les cellules d'entrée et les cellules de sortie. Chaque cellule de test ne mémorise qu'1/2 donnée (une cellule d'entrée + une cellule de sortie mémorisent une donnée). Le démultiplexeur DEMUX de la cellule peut donc être remplacé par un bloc combinatoire identique à S1, appelé S2. L'architecture optimisée de cette cellule est illustrée dans la figure 3.18. Les données de test peuvent être stockées en utilisant les acquittements de la cellule suivante. C'est-à-dire que les données peuvent être stockées entre le multiplexeur MUX de la cellule courante et le multiplexeur MUX de la cellule suivante, cf. figure 3.19. Dans cette architecture optimisée, le canal LOCAL traverse le processus combinatoire S2 comme expliqué dans la figure 3.15 et relie le processus MUX au processus MODE et au processus MUX de la cellule suivante. Les données de test sont maintenant stockées entre le MUX de la cellule courante et le MUX de la cellule suivante pour décalage eventuel et stockées entre le MUX et le MODE de la cellule courante avant d'être éventuellement transportées au *noc-out*.

Si on revient à la section 3.2.4, on constate que la solution d'interconnexion des cellules dans le wrapper S1 (l'agencement des cellules d'entrée puis des cellules de sortie) ne permet pas de réaliser cette optimisation car on ne peut pas faire rentrer une nouvelle donnée sur le canal d'entrée si le *buffer* est occupé, cf. section 3.3.2.

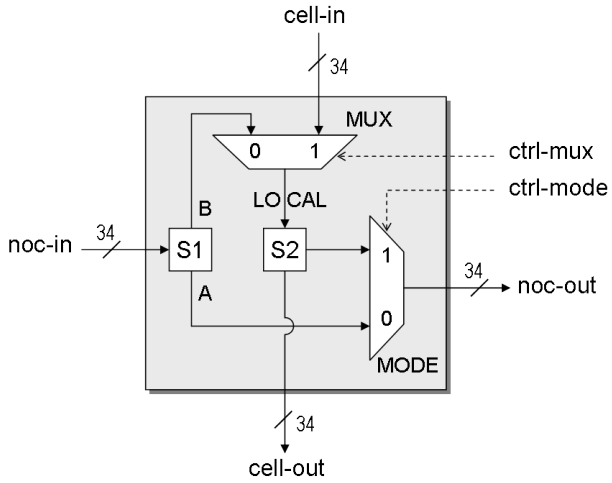


FIG. 3.18 : Architecture de la cellule optimisée.

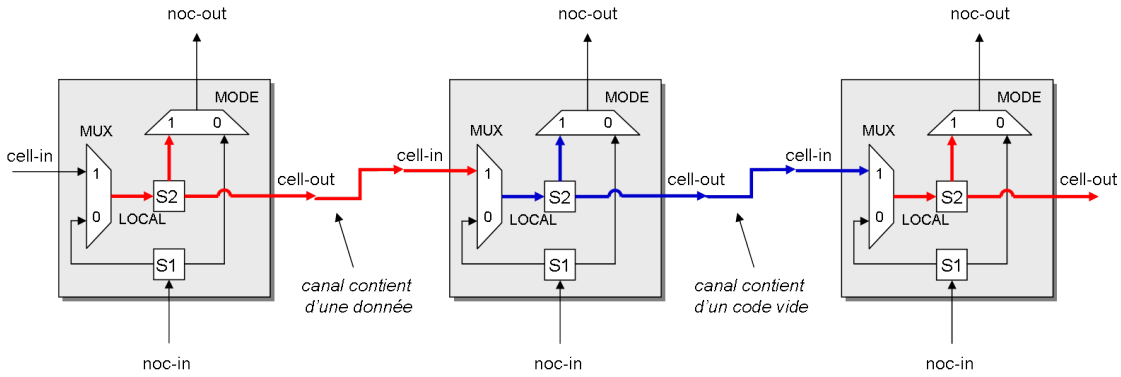


FIG. 3.19 : Chaîne des cellules et la mémorisation des données sur la chaîne

En remplaçant le démultiplexeur DEMUX par un splitter S2, le coût en surface de la cellule est réduit (environ 27%). On note que les processus de séparation S1 et S2 sont des blocs combinatoires implémentés par le ET des signaux d'acquittement. De plus, le nombre des signaux de contrôle générés par le module WCM pour la cellule est également réduit, les signaux *ctrl-demux* n'existent plus. Le tableau 3.3 décrit le fonctionnement de la cellule optimisée selon ses signaux de contrôle.

Dans tous les paragraphes précédents, nous avons présenté comment construire et réaliser une cellule de test simpliste, sans les signaux de contrôle. Afin de s'adapter au réseau ANOC, les cellules doivent inclure tous les signaux de contrôle tels que *send*, *accept0*, *accept1*, etc. L'intégration de ces signaux de contrôle rend l'architecture de la cellule plus complexe. L'architecture d'une cellule complète est présentée dans la figure 3.20.

Les signaux ajoutés à la cellule concernent les canaux virtuels et le contrôle de

TAB. 3.3 : Description du fonctionnement de la cellule optimisée

ctrl-mux	*ctrl-mode*	**Description du fonctionnement de la cellule**
-	0	Mode normal
0	-	Recevoir les données de test de *noc-in* et les stocker dans la cellule et sur *cell-out*
1	-	Recevoir les données de test de *cell-in* et les stocker dans la cellule et sur *cell-out*
-	1	Transporter les données stockées à *noc-out*

***Note** : (-) signifie que ce signal de contrôle n'est pas concerné.*

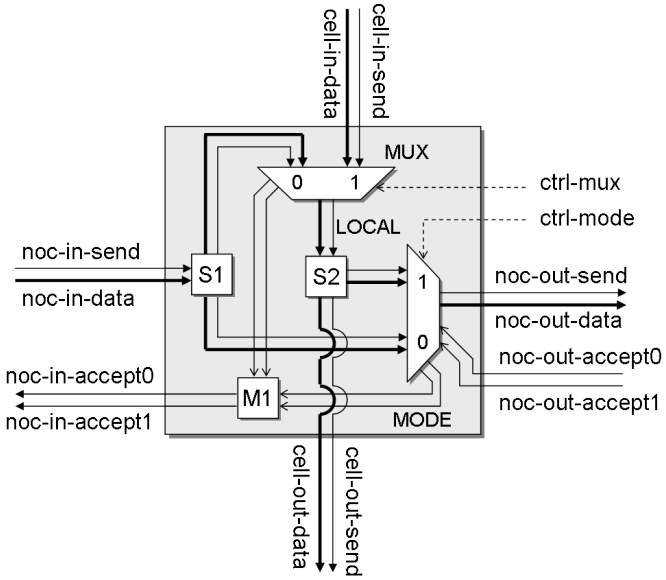


FIG. 3.20 : Architecture de la cellule avec ses signaux de contrôle (*send*, *accept*)

flots de données utilisé dans le réseau. Ils sont également implémentés en logique asynchrone en utilisant le protocole 4-phases RTZ avec les codages de données multi-rail, double-rail, single-rail.

3.3.4 Cellules de test avec fonction bypass

Comme nous le savons, dans une architecture CVT pour un système sur puce on préfère toujours la fonction bypass. Cette fonction permet de réduire le temps d'application du test et de n'activer que les éléments sous test. Pour implémenter la fonction bypass dans notre wrapper de test, nous avons proposé de nouvelles architectures pour les cellules d'entrée et les cellules de sortie, comme illustrées

dans la figure 3.21.

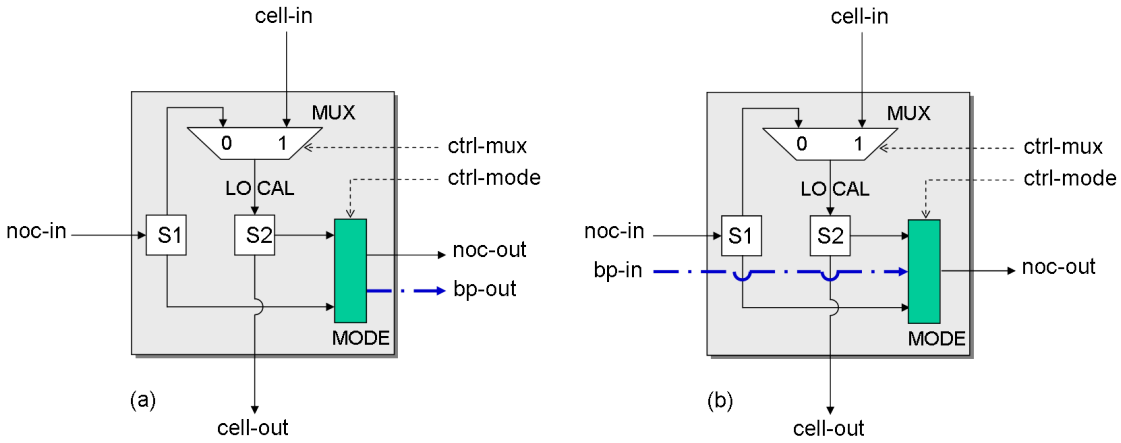


FIG. 3.21 : Cellules de test avec bypass.

Les modifications majeures sont réalisées sur les multiplexeurs MODE. Pour les cellules d'entrée (ITC), on ajoute un port de sortie pour le bypass, appelé *bp-out*. Ce port est une deuxième sortie du multiplexeur MODE. Pour les cellules de sortie, on ajoute un port d'entrée pour le bypass, appelé *bp-in*. Ce port est une troisième entrée du multiplexeur MODE. Le signal de contrôle a maintenant besoin de trois valeurs pour contrôler trois modes (normal, test, et bypass). Dans le mode bypass, les données arrivant au port *noc-in* de la cellule d'entrée seront transportées vers le port *bp-out*, et les données arrivant au port *bp-in* de la cellule de sortie seront transportées vers le port *noc-out*. On note que le port *bp-out* de la cellule d'entrée est connecté au port *bp-in* de la cellule de sortie pour établir le canal de bypass. Le tableau 3.3 devient le suivant, cf. tableau 3.4 :

TAB. 3.4 : Signaux de contrôle incluant la fonction bypass

ctrl-mux	***ctrl-mode***	**Description du fonctionnement de la cellule**
-	0	Mode normal
0	-	Recevoir les données de test de *noc-in* et les stocker dans la cellule et sur *cell-out*
1	-	Recevoir les données de test de *cell-in* et les stocker dans la cellule et sur *cell-out*
-	1	Transporter les données stockées à *noc-out*
-	2	Mode bypass

***Note** : (-) signifie que ce signal de contrôle n'est pas concerné.*

La figure 3.22 présente un wrapper complet de 5 cellules d'entrée (ITC) et 5

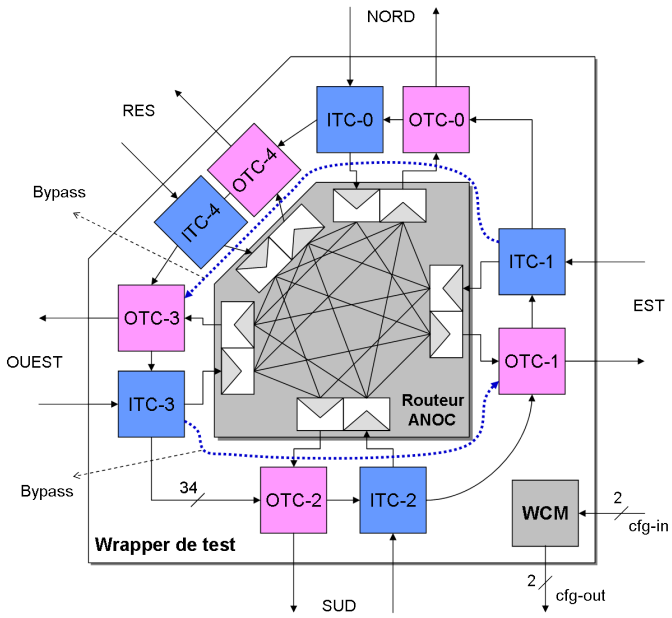


FIG. 3.22 : Architecture du wrapper de test avec deux canaux bypass entre EST ⇔ OUEST.

cellules de sortie (OTC) avec les deux canaux bypass entre les entrées/sorties Ouest et Est (bypass dans les deux sens).

La direction des bypass sont différentes dans le réseau suivant l'ordre de la chaîne de configuration des WCM des routeurs. Ceci pour que les routeurs puissent être testés avant d'être mis en mode bypass pour le test des routeurs suivants. On a une chaîne de bypass qui suit la chaîne de configuration de test, cf. section 4.2.4.

3.3.5 Conception et synthèse du module WCM

Les opérations des cellules dans le wrapper sont définies par un vecteur de configuration de test, appelée TCF (*Test Configuration Frame*). On note que la TCF peut être comprise comme une instruction de test. Le module de contrôle principal de test intégré dans le GAC génère des configurations de test selon la stratégie de test. Ensuite, ces configurations de test sont envoyées aux modules de contrôle WCM des wrappers. Afin de réduire le nombre des interconnexions entre le GAC et les WCM, la TCF est coupée en plusieurs éléments. Dans notre cas, chaque élément se compose de deux bits et est donc codé par une donnée MR[4] envoyés aux WCM en série en utilisant la chaîne de configuration.

Dans notre implémentation, une TCF se compose de 25 digits MR[4] (cf. figure 3.23). Le premier MR[4] (M) de la TCF spécifie le mode de fonctionnement du

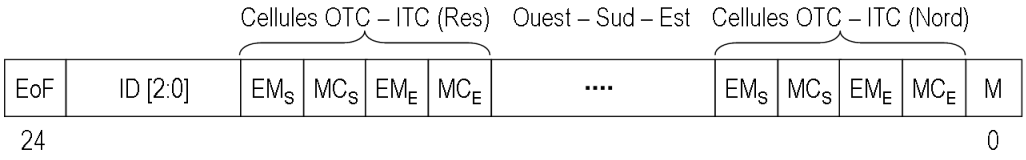


FIG. 3.23 : Vecteur de configuration de test (TCF).

wrapper affecté. Le dernier MR[4] (EoF) spécifie la fin de la TCF. Les trois MR[4] avant le dernier MR[4] de la TCF codent l'identifiant (ID) du wrapper concerné.

Le rôle du module WCM est de recueillir les MR[4] successifs envoyés sur la chaîne de configuration jusqu'à ce qu'il forme une TCF complète pour son wrapper. Quand il a reçu une TCF complète (en détectant une indication « EoF » comme dernier MR[4] de la TCF), le module WCM doit déterminer si la TCF est destinée à son wrapper ou pas. Ceci est fait au moyen du champ d'identification (ID) adressant les wrappers de test successifs sur la chaîne de configuration. Si le champ d'identification correspond à celui de son wrapper de test, le module WCM du wrapper concerné va générer les signaux de contrôle pour les cellules de son wrapper en consultant les valeurs de la TCF.

Pour implémenter ce fonctionnement, nous avons proposé une architecture pour le module WCM dans la figure 3.24. Le module WCM se compose donc d'un bloc

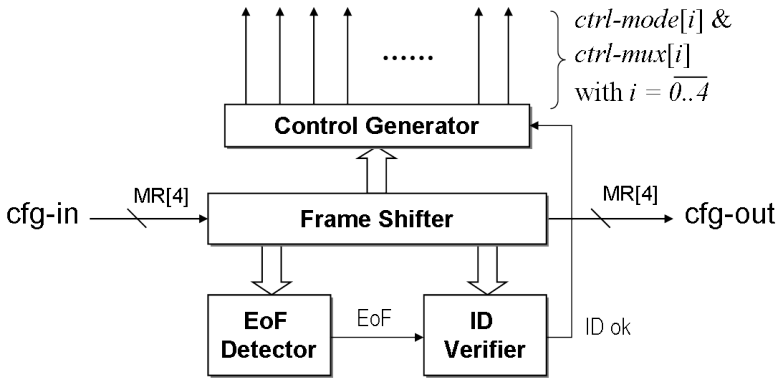


FIG. 3.24 : Module de contrôle du wrapper (WCM).

de décalage appelé « Frame Shifter », d'un détecteur de fin de la TCF appelé « EoF Detector », d'un bloc de vérification d'identifiant appelé « ID Verifier », et d'un bloc de génération des signaux de contrôle appelé « Control Generator ».

Le bloc « Frame Shifter » nous permet de recevoir un nouveau digit MR[4] de la chaîne de configuration via l'entrée *cfg-in*, de décaler à droite d'une position les digits MR[4], et de faire sortir le MR[4] de poids faible de la chaîne de configuration

via la sortie *cfg-out*. Le détecteur « EoF Detector » détecte la fin de la TCF. Une fois que la TCF est complète, il active le bloc « ID Verifier ». Le bloc « ID Verifier » vérifie l'identifiant de la TCF. Si l'identifiant correspond à celui du wrapper, le bloc « Control Generator » est activé et il envoie les commandes aux cellules du wrapper. L'explication d'une trame de configuration de test est présentée dans le tableau 3.5.

TAB. 3.5 : Détails d'une trame de configuration

EoF	Valeur spéciale indiquant la fin d'une configuration
ID[2 :0]	Identifiant du wrapper de test
$EM_S[i]$	Activer le signal *ctrl-mode* pour la cellule OTC-i
$MC_S[i]$	Définir la valeur du signal *ctrl-mux* pour la cellule OTC-i
$EM_E[i]$	Activer le signal *ctrl-mode* pour la cellule ITC-i
$MC_E[i]$	Définir la valeur du signal *ctrl-mux* pour la cellule ITC-i
M	Mode du wrapper de test affecté par la configuration

La valeur de chaque MR[4] peut être "0", "1", "2", ou "3" (respectivement {00}, {01}, {10}, {11} en codage binaire). La valeur "3" est réservée pour l'indication EoF. Les autres MR[4] de configuration peuvent donc seulement être "0", "1" ou "2". Dans la trame de configuration de test, présentée dans la figure 3.23, il y a trois MR[4] de configuration utilisés pour encoder les IDs des wrappers de test. Cette trame de configuration de test peut donc être utilisée pour un réseau sur puce de 27 routeurs maximum (3^3). Pour les réseaux plus larges, il est nécessaire d'ajouter d'autres MR[4] de configuration pour ce champ d'identification.

La valeur de M peut être "0" (mode normal), "1" (mode test), ou "2" (mode bypass). La valeur des signaux de contrôle *ctrl-mode* pour toutes les cellules est la valeur indiquée par le digit M. Cependant, ces valeurs ne seront transférées qu'aux cellules de test activées par les digits EM (***E**nable ctrl-**M**ode*). L'écriture sur le canal *ctrl-mode*$[i]$ de la cellule ITC-i est activée quand $EM_E[i]$ = "1". De façon similaire, l'écriture sur le canal *ctrl-mode*$[i]$ de la cellule OTC-i est activée quand $EM_S[i]$ = "1". La valeur écrite sur la canal *ctrl-mux*$[i]$ de la cellule ITC-i est à '1' quand $MC_E[i]$ est égal à "2", et la valeur écrite sur la canal *ctrl-mux*$[i]$ de la cellule ITC-i est à '0' quand $MC_E[i]$ est égal à "1". De façon similaire, la valeur écrite sur le canal *ctrl-mux*$[i]$ de la cellule OTC-i est à '1' quand $MC_E[i]$ est égal à "2", et la valeur écrite sur le canal *ctrl-mux*$[i]$ de la cellule OTC-i est à '0' quand $MC_E[i]$ est égal à "1".

Ce qui peut se résumer par les équations suivantes :

– $MC_E[i]$ = "1" $\Rightarrow$ *ctrl-mux* de la cellule ITC-i est écrit par la valeur '0'

– $MC_E[i]$ = "2" ⇒ *ctrl-mux* de la cellule ITC-i est écrit par la valeur '1'
– $MC_S[i]$ = "1" ⇒ *ctrl-mux* de la cellule OTC-i est écrit par la valeur '0'
– $MC_S[i]$ = "2" ⇒ *ctrl-mux* de la cellule OTC-i est écrit par la valeur '1'
– $EM_E[i]$ = "1" ⇒ *ctrl-mode* de la cellule ITC-i peut être écrit par la valeur M
– $EM_S[i]$ = "1" ⇒ *ctrl-mode* de la cellule OTC-i peut être écrit par la valeur M

Les tableaux 3.6 et 3.7 suivants résument les commandes pour une cellule ITC et pour une cellule OTC.

TAB. 3.6 : Commandes pour une cellule ITC

M	$EM_E[i]$,$MC_E[i]$	***ctrl-mode***	***ctrl-mux***	**Description**
0	XX	0	-	Mode normal
1	00	-	-	Pas de commande
1	01	-	0	Insertion dans la chaîne
1	02	-	1	Décalage de la chaîne
1	12	1	1	Extraction de la chaîne et injection dans le routeur
1	11	1	0	Identique au mode normal mais en passant par la chaîne de décalage
2	XX	2	-	Mode bypass

***Note** : (-) signifie que ce signal de contrôle n'est pas concerné.*

TAB. 3.7 : Commandes pour une cellule OTC

M	$EM_S[i]$,$MC_S[i]$	***ctrl-mode***	***ctrl-mux***	**Description**
0	XX	0	-	Mode normal
1	00	-	-	Pas de commande
1	01	-	0	Insertion dans la chaîne
1	02	-	1	Décalage de la chaîne
1	12	1	1	Extraction de la chaîne et transfert au réseau
1	11	1	0	Identique au mode normal mais en passant par la chaîne de décalage
2	XX	2	-	Mode bypass

***Note** : (-) signifie que ce signal de contrôle n'est pas concerné.*

Par exemple, considérons le test d'un routeur pour le chemin de routage du Nord vers le Sud, cf. figure 3.25. Pour configurer le wrapper de test afin d'appliquer les vecteurs de test à ce routeur par l'entrée Nord et récupérer les réponses par la sortie Sud, nous utilisons la configuration décrite dans le tableau 3.8.

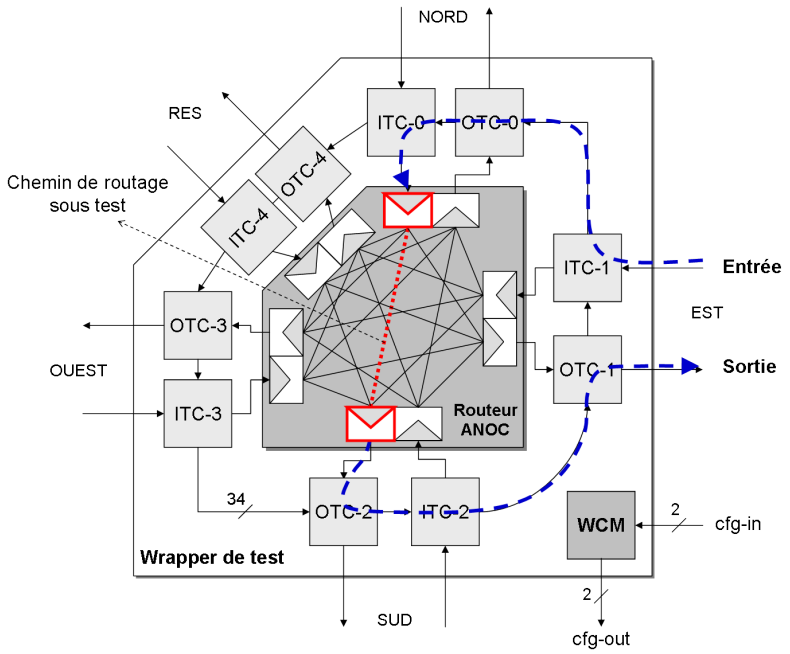


FIG. 3.25 : Test du chemin de routage Nord ⇒ Sud.

TAB. 3.8 : Exemple d'une configuration de test pour le chemin de routage Nord ⇒ Sud.

EoF	**ID**	$R_S - R_E$	$O_S - O_E$	$S_S - S_E$	$E_S - E_E$	$N_S - N_E$	**M**
3	002	00-00	00-00	01-02	12-01	02-12	1

Cette configuration peut être expliquée comme suit : la valeur de EoF égale à "3" implique la fin de cette configuration. Le champ ID égale à "002" implique que le wrapper numéro 2 est affecté par cette configuration de test. La valeur de M égale à "1" place le wrapper en mode test. La valeur E_E = "01" implique que la cellule ITC-1 (Est) reçoit les données de test à partir du TAM et les envoie à la cellule OTC-0 (Nord). La valeur N_S = "02" implique que la cellule OTC-0 est en mode décalage, les données de test reçues sont donc décalées jusqu'à la cellule ITC-0 (Nord). Puis, ces données de test sont appliquées au routeur par cette cellule ITC-0 car N_E est égale à "12".

Après l'application de ce test au routeur, la valeur S_S = "01" commande à la cellule OTC-2 (Sud) de recevoir les réponses et de les transporter vers la cellule ITC-2 (Sud). Cette cellule est en mode décalage car S_E = "02". Les réponses reçues sont donc décalées à la cellule OTC-1 (Est). Puis, grâce à la valeur E_S égale à "12" la cellule OTC-1 reçoit ces réponses et les envoie à l'analyseur (intégré dans le GAC) via le TAM.

On rappelle que le TAM est construit en utilisant les liens du réseau entre les routeurs et les autres wrappers de test. On note aussi que les autres cellules dans le wrapper telles que OTC-4 et ITC-4 (Ressource), OTC-3 et ITC-3 (Ouest) ne sont pas concernées dans ce test et donc ne sont pas contrôlées par la configuration de test. Ainsi, les valeurs de R_S, R_E, O_S, et O_E sont mises à "00".

3.3.6 Plateforme de validation

L'architecture CVT proposée a été modélisée et validée à plusieurs niveaux avec différents langages de conception comme nous l'avons abordé dans la section précédente. Puis, cette architecture a été synthétisée et implémentée en utilisant *Cadence Virtuoso Schematic Editor* avec les librairies CMOS065LP de STMicroelectronics et TAL065 de TIMA. Pour valider à plusieurs niveaux de la conception cette architecture, nous avons développé une plateforme de co-simulation SystemC/VHDL/Verilog, illustrée dans la figure 3.26. Le but de ce développement est

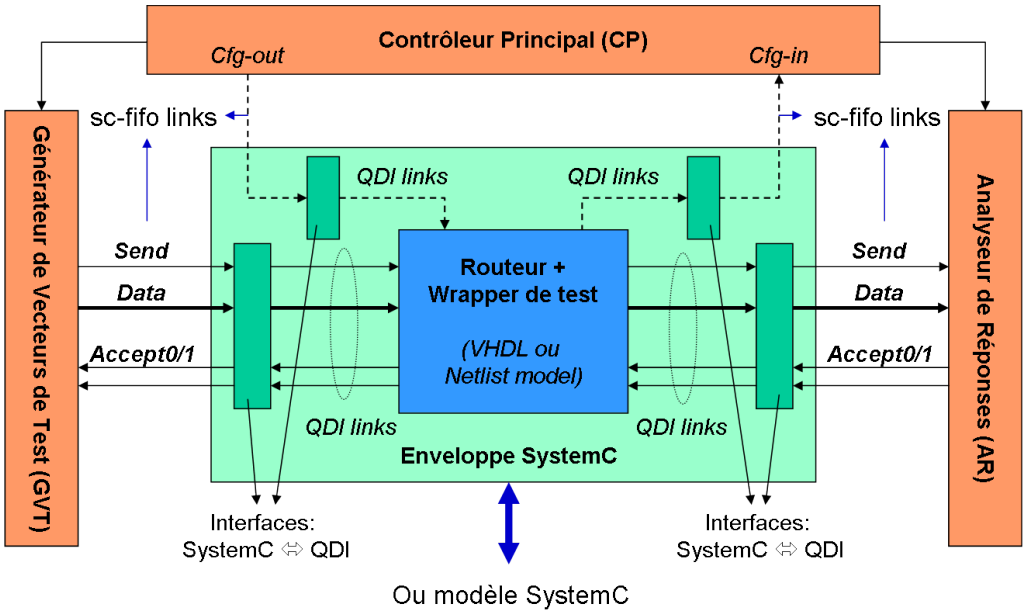


FIG. 3.26 : Plateforme de validation.

de construire une plateforme de vérification unique pour tous les niveaux du flot de conception pour notre travail. L'avantage de cette approche est d'économiser le temps de conception et de minimiser les erreurs ajoutées à chaque raffinement de niveau de conception.

Dans cette plateforme, les routeurs et les wrappers de test sont modélisés en SystemC, en VHDL, ou en Verilog netlist. Le bloc GAC (incluant les sous blocs

GVT, AR, CP précisés sur la figure) et l'environnement de simulation sont construits en SystemC uniquement. Afin de communiquer avec les blocs SystemC, chaque ensemble routeur et wrapper associé est entouré par une enveloppe d'adaptation en SystemC permettant de réaliser la co-simulation entre les blocs SystemC et les blocs VHDL ou Verilog *netlist*. Cette enveloppe inclut différentes interfaces SystemC sc-fifo ⇔ QDI. La co-simulation a été réalisée principalement avec l'outil ModelSim de Mentor Graphics [MenGra].

3.3.7 Implémentation et résultats

L'architecture CVT modélisée et validée a été mise en œuvre sur silicium en utilisant la technologie CMOS 65nm de STMicroelectronics avec la bibliothèque des portes asynchrones TAL065 du TIMA. La suite de cette section présente plusieurs résultats d'implémentation tels que le coût en surface, le débit de test, la latence ajoutée dans le système, etc.

3.3.7.1 Implémentation de l'identifiant et de la direction de bypass

L'identifiant et la direction de bypass d'un wrapper de test dépendent de sa position dans le réseau. Nous pouvons implémenter ces caractéristiques de manière individuelle ou de manière générique. Lorsqu'on utilise la manière individuelle, ces caractéristiques sont définies pour chaque couple wrapper-routeur. L'implémentation du réseau entier consiste ensuite à assembler ces différents couples. Lorsqu'on utilise la manière générique, on utilise une seule description mais cette fois paramétrée de façon spécifique pour tous les couples wrapper-routeur du réseau. Dans le reste de ce paragraphe, on présente comment implémenter ces couples de manière générique.

L'identifiant d'un wrapper est codé par trois digits MR[3] qui permettent de coder trois valeurs "0", "1", et "2" (cf. section 3.3.5). Afin d'utiliser une macro unique wrapper-routeur lors de l'implémentation du réseau, ces identifiants doivent être définis au plus haut niveau de la hiérarchie. Pour cela, le wrapper de test doit avoir un port d'entrée de 9 fils (3×MR[3]) pour son identifiant. L'implémentation de l'identifiant de chaque wrapper est donc réalisée par une valeur câblée sur ce port d'entrée supplémentaire.

De la même manière, les directions de bypass doivent être implémentées au plus haut niveau de la hiérarchie. Pour faire cela, toutes les cellules ITC doivent avoir un port *bp-out*, et toutes les cellules OTC doivent avoir trois ports *bp-in* pour trois entrées différentes. Comme il y a six directions de bypass possibles (Nord ⇔ Sud, Est ⇔ Ouest, Nord ⇔ Ouest, Ouest ⇔ Sud, Sud ⇔ Est, Est ⇔Nord), il est nécessaire d'ajouter 6 fils d'entrée pour configurer ces bypass. On note que cette configuration

peut être réalisée de façon définitive pendant l'étape d'implémentation au niveau système ou être changée dynamiquement par le GAC.

Afin de réaliser cette implémentation, nous utilisons le schéma d'implémentation suivant pour tous les wrappers, cf. figure 3.27. Selon ce schéma, nous avons besoin

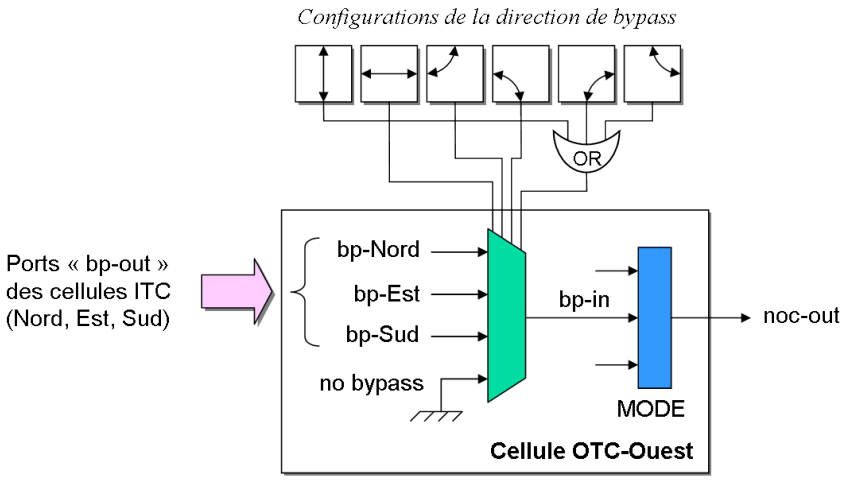


FIG. 3.27 : Implémentation de la direction de bypass du routeur au niveau de la hiérarchie.

de 70 MUX à quatre entrées (68 MUX pour 68 rails de données et 2 MUX pour les 2 rails du signal « *send* »), et d'une porte OU à trois entrées pour chaque cellule OTC. On note que les MUX à quatre entrées sont des MUX combinatoires car le canal de bypass est établi un fois pour toutes (soit au niveau matériel à l'implémentation, soit par configuration jusqu'au Reset suivant comme pour le mode normal).

3.3.7.2 Coût en surface

La surface d'une cellule de test est de $8.560\mu m^2$ et la surface du bloc WCM est de $10.400\mu m^2$. En conséquence, un wrapper de test qui se compose de 5 cellules d'entrée, de 5 cellules de sortie, et d'un bloc WCM occupe une surface de $96.000\mu m^2$ (équivalent à 9.600 portes logiques), i.e. 32,7% de la surface totale d'un routeur asynchrone testable. La figure 3.28 présente la proportion des surfaces de ce routeur.

Comme nous en avons discuté, l'architecture proposée est utilisée pour tester tous les éléments du réseau de communication tels que les routeurs et les liens du réseau, et elle est aussi utilisée comme un TAM pour tester les unités de traitement. Le coût en surface de l'architecture proposée doit ainsi être comparée avec la surface totale du système sous test. Le coût en surface pour l'architecture CVT portée sur

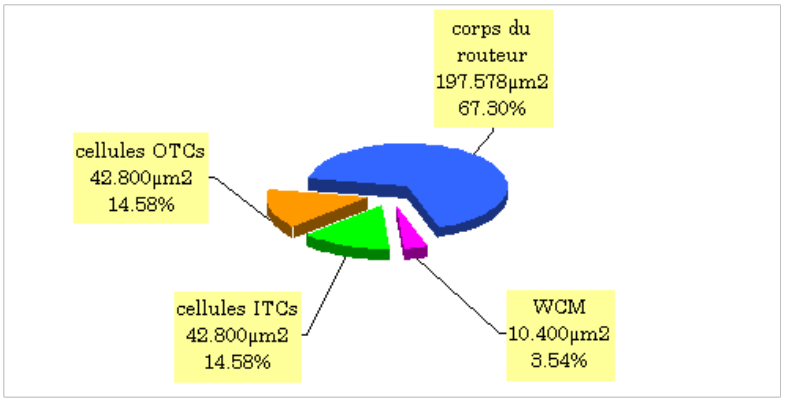


FIG. 3.28 : Proportion des surfaces du routeur testable.

un circuit applicatif complet tel que le circuit FAUST (avec un réseau ANOC de 20 routeurs) serait d'environ $2mm^2$, soit 4,8% de la surface totale du système.

3.3.7.3 Latence ajoutée de l'architecture

Les wrappers de test sont ajoutés afin d'améliorer l'accessibilité et la testabilité des architectures basées sur un réseau. Malheureusement, ils impliquent une latence supplémentaire qui réduit la performance du système en mode normal. Dans notre cas, la latence ajoutée d'un wrapper (en considérant à la fois la latence en entrée et en sortie) est de $0,34ns$; latence qui doit être comparée à la latence de $2,00ns$ du routeur. Les mesures sont faites au niveau masque (*layout*) avec une fréquence d'opération de $500MHz$, cf. figure 3.29. Cette latence ajoutée correspond à un délai de deux portes de Muller asymétriques et de deux portes ET.

3.3.7.4 Débit de test conservé

On note que dans un circuit asynchrone QDI la performance est déterminée par la plus longue boucle de poignée de main. Or, les boucles de poignée de main aux entrées/sorties du routeur ne sont pas les boucles les plus critiques, donc il n'y a pas de dégradation du débit de communication du réseau. Ainsi, le débit du réseau reste à $500Mflits/s$ en mode normal.

3.3.7.5 Bande passante de test

En utilisant les liens du réseau comme mécanisme d'accès de test, l'architecture CVT proposée utilise des chemins de test haute bande passante. Cependant, nous devons envoyer des configurations de test (TCF) afin de contrôler l'injection et la col-

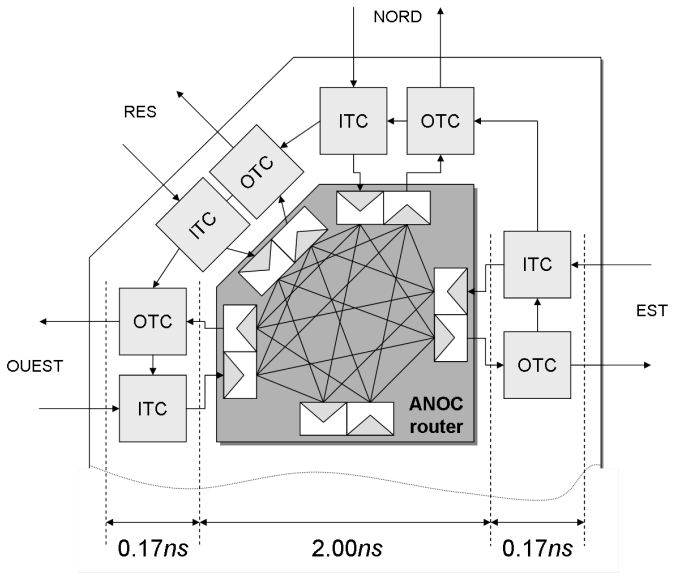


FIG. 3.29 : Latence ajoutée par le wrapper de test.

lection des vecteurs de test. Le temps pour envoyer une configuration TCF complète est d'environ $50ns$, soit 25 digits à $2ns$/digit (débit de la chaîne de configuration de $500MHz$).

Pour le test des routeurs, dans plusieurs cas, nous avons besoin d'une seule TCF pour insérer un vecteur de test et pour observer le résultat, et la vitesse maximale est donc d'environ $20Mvecteurs/s$; cependant, afin de simplifier la génération des TCF, nous utilisons en général deux TCF par vecteur de test lors du test des routeurs : une TCF pour insérer le vecteur de test et l'autre pour observer le résultat. Lors du test des liens du réseau, nous avons besoin de deux TCF par vecteur de test pour insérer un vecteur de test et pour observer le résultat car deux wrappers de test sont utilisés. Ceci implique une vitesse de test d'environ $10Mvecteurs/s$.

3.3.7.6 Prototype utilisant l'architecture CVT

L'architecture CVT proposée a été mise en œuvre dans le cadre de projet ALPIN (***A**synchronous **L**ow **P**ower **I**nnovative **N**oC*) au laboratoire IAN du LETI. L'objectif du projet ALPIN est de valider un ensemble de nouvelles techniques pour le projet FAUST-2 (le projet FAUST deuxième version) telles que les techniques basse consommation (DVS/DFS, Vdd-Hopping), l'architecture CVT pour tester le réseau de communication, la mesure de la performance du réseau, etc. Plus de détails sur ce projet est présenté dans [Beig06A].

Le circuit développé dans ce projet s'appelle circuit ALPIN. C'est un système sur puce basé sur un réseau asynchrone de 9 routeurs, cf. figure 3.30. Ce circuit a été

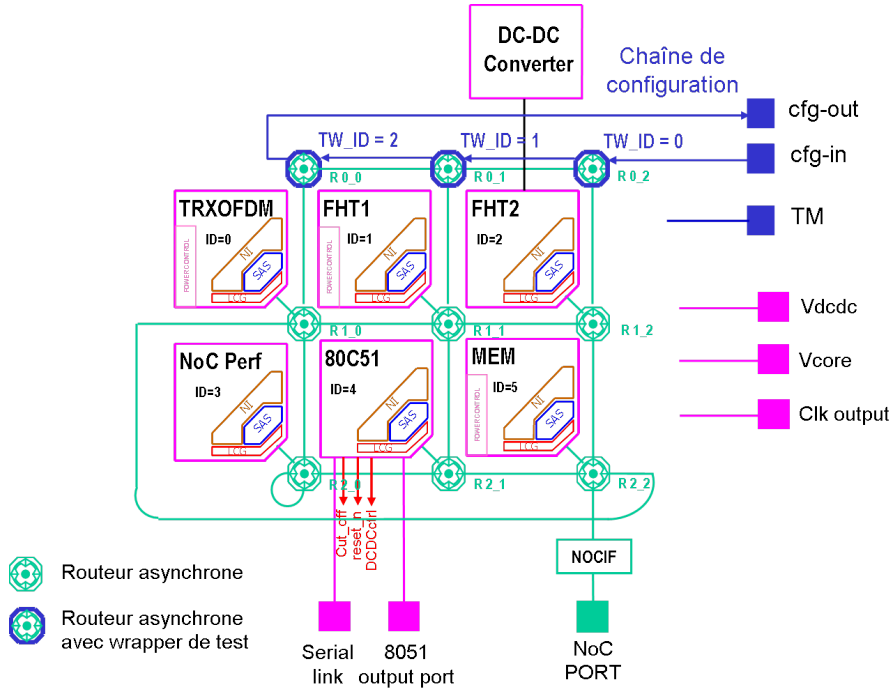


FIG. 3.30 : Architecture générale du circuit ALPIN.

développé et implémenté en utilisant la technologie CMOS 65*nm* basse consommation de STMicroelectronics (CMOS065LP). Il se compose d'un contrôleur MC8051, d'un bloc de transmission TrxOFDM, de deux blocs FHT (*Fast Hartley Transform*), d'un bloc NoC-Perf pour mesurer la performance du réseau, de blocs de mémoire, etc. L'architecture CVT proposée a été implémentée dans ce circuit et se compose de 3 wrappers de test qui entourent trois routeurs.

Donc, nous pouvons mettre en pratique les processus de test définis dans la section 4.2.2 (chapitre 4) pour ces routeurs. Les tests des autres routeurs peuvent être réalisés en utilisant ces wrappers avec les routeurs sans wrappers. La figure 3.31 présente le masque (*layout*) du circuit ALPIN.

On voit bien que l'architecture a besoin seulement d'un port d'entrée et d'un port de sortie pour la chaîne de configuration de test. En effet, les données de test peuvent être transportées en utilisant les ports d'entrée et de sortie du système (via l'interface NOC-IF).

3.4 Conclusion

Dans ce chapitre, nous avons présenté notre approche de test pour les architectures de réseaux asynchrones. Suivant cette approche, une architecture matérielle

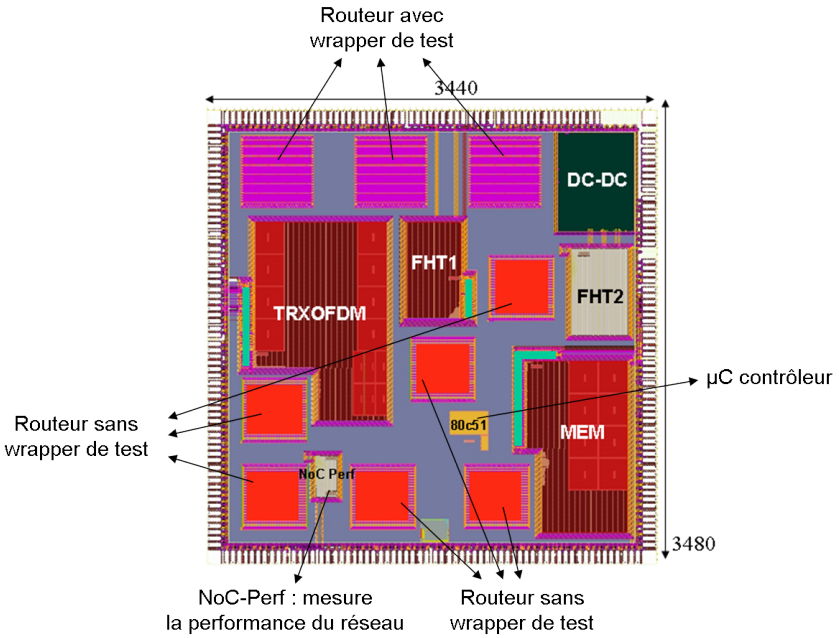


FIG. 3.31 : Circuit APLIN - vue de *layout*.

CVT a été proposée et développée afin d'améliorer la testabilité des NoC asynchrones. Selon cette architecture, chaque routeur du réseau est entouré par un wrapper de test asynchrone.

Les wrappers de test sont les éléments de base de l'architecture proposée. De ce fait, nous avons présenté différentes solutions architecturales afin de choisir la solution la mieux adaptée pour construire ces wrappers. Les avantages et les inconvénients de ces solutions ont donc été discutés et comparés.

Ainsi, suite à la sélection de l'architecture du wrapper, nous avons détaillé la conception, l'optimisation et l'implémentation de ce wrapper. Plusieurs résultats de l'implémentation de l'architecture CVT ont été présentés. Finalement, nous avons intégré cette architecture CVT dans le réseau ANOC du projet ALPIN. La mise en œuvre de cette architecture CVT sera présenté dans le chapitre suivant.

Chapitre 4

Mise en Œuvre de l'Architecture CVT

Dans le chapitre précédent, nous avons proposé une méthode de test pour laquelle une architecture CVT a été développée. Cette architecture a été modélisée, implémentée, et puis intégrée dans le réseau ANOC. Dans ce chapitre, nous allons décrire comment est mise en œuvre cette architecture afin de tester le réseau ANOC.

Comme présenté dans le chapitre 2, il n'existe aucun outil ATPG pour les circuits asynchrones. Dans une première section de ce chapitre, nous allons présenter une méthode de génération des vecteurs de test pour tous les éléments du réseau. Cette méthode est basée sur une analyse des fonctionnalités du réseau et son implémentation structurelle. Dans la deuxième section, nous allons montrer comment appliquer ces vecteurs de test en utilisant l'architecture CVT afin de tester tous les éléments du réseau ANOC. Une analyse de la couverture de fautes structurelles (fautes de collage) pour ces vecteurs sera réalisée afin de vérifier leur efficacité pour le test matériel. Ainsi, à la fois la stratégie et les résultats de test seront présentés dans cette section. Finalement, une étude sur les utilisations alternatives de cette architecture sera réalisée.

4.1 Génération des vecteurs de test

Comme discuté dans le chapitre 2, il n'existe aucun outil ATPG pour les circuits asynchrones, et nous proposons donc d'analyser les fonctionnalités du réseau afin de générer des vecteurs de test. Cette section présente comment générer des vecteurs de test pour les liens et les routeurs du réseau en analysant leur fonctionnalité et

leur implémentation structurelle.

4.1.1 Génération des vecteurs de test pour les liens du réseau

Les liens du réseau sont implémentés en logique asynchrone QDI avec le protocole 4-phases (cf. section 2.2), un lien du réseau se compose donc de 17 canaux MR[4], un canal double-rails (DR), et deux canaux single-rail (SR), cf. section 1.3. Les canaux MR[4] sont utilisés pour transporter les données de communication, le canal DR (« *send* ») est utilisé pour indiquer le canal virtuel de ces données, et les canaux SR (« *accept0* » et « *accept1* ») sont utilisés pour indiquer l'acceptation des données sur les canaux virtuels. Un canal MR[4] comprend 4 rails de requête et 1 rail d'acquittement, illustré dans la figure 4.1. Un canal DR comprend 2 rails de

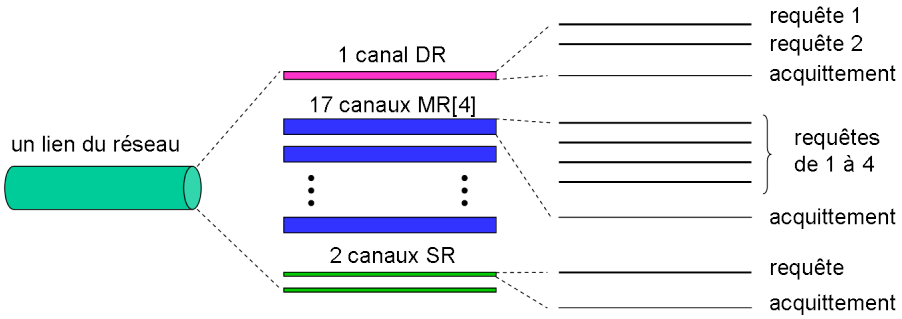


FIG. 4.1 : Structure d'un lien du réseau.

requête et 1 rail d'acquittement. Un canal SR comprend un rail de requête et un rail d'acquittement. Considérons le fonctionnement du réseau, les liens du réseau n'ont aucun contrôle : ils transportent des données entrantes sur leur entrées vers leurs sorties. Le test d'un lien du réseau est équivalent à 17 tests individuels de 17 canaux MR[4] plus le test des canaux DR et SR.

Pour tester un canal MR[4], il suffit donc de déclencher aux entrées et d'observer aux sorties des transitions montantes et descendantes sur chaque rail du canal. Le protocole 4-phases utilisé dans l'implémentation du canal assure des transitions montantes et descendantes sur un rail de requête et le rail d'acquittement. En conséquence, il est nécessaire d'utiliser les valeurs exhaustives d'un digit « 1-of-4 » pour tester un canal MR[4], soit quatre valeurs possibles (« 1 », « 2 », « 3 », et « 4 »). De même manière, pour tester un canal DR il est nécessaire d'utiliser les valeurs exhaustives d'un digit « 1-of-2 », soit deux valeurs possibles « 0 » et « 1 ». Un canal SR n'a qu'une requête possible, et est testé par le protocole 4-phases sur cette requête. Au final, les 4 vecteurs (« 0.0.0...0 » ; « 1.1.1...1 » ; « 2.2.2...2 » ; « 3.3.3...3 ») constituent donc un jeu complet de vecteurs de test pour la partie

« *data* » d'un lien du réseau. Pour tester un lien du réseau complet (incluant aussi les signaux de contrôle), ces 4 vecteurs de test sont combinés avec deux valeurs du test (« 0 » et « 1 ») du canal DR (« *send* »). Ces vecteurs sont envoyés à partir d'un wrapper de test d'un routeur vers le wrapper de test du routeur voisin, cf. section 4.2.3. Les canaux SR « *accept* » correspondent, au niveau protocolaire « *send–accept* » (cf. section 1.3.3) implémenté sur le deuxième wrapper, à l'acceptation d'une transaction sur le canal DR « *send* ». Ils sont naturellement testés par l'émission de deux valeurs sur chacun des rails de requête du canal « *send* ». Le tableau 1 présente l'ensemble de ces vecteurs de test.

TAB. 4.1 : Jeu de test complet pour un lien du réseau.

Data *(tous les 17 digits « 1-of-4 »)*	**Send**
« 0.0.0. . . . . 0 »	« 0 »
« 1.1.1. . . . . 1 »	« 1 »
« 2.2.2. . . . . 2 »	« 0 »
« 3.3.3. . . . . 3 »	« 1 »

Finalement, nous avons besoin de seulement quatre vecteurs de test pour tester un lien du réseau ANOC. Comme tous les liens sont identiques, ce jeu des vecteurs de test est utilisé pour tester tous les liens du réseau.

4.1.2 Génération des vecteurs de test pour les routeurs du réseau

Comme les liens du réseau, les routeurs du réseau sont également implémentés en logique asynchrone QDI avec le protocole 4-phases. L'architecture du routeur ANOC a été présentée dans le chapitre 1, section 1.3.4. Selon cette présentation, le routeur se compose de 5 unités d'entrées et de 5 unités de sortie. En analysant la structure du routeur, on peut voir que la partie contrôle du routeur occupe environ 10% des portes logiques de la totalité du routeur et que le nombre des portes logiques d'une unité d'entrée est à peu près le même que celui d'une unité de sortie. En dehors de la partie contrôle, chaque unité d'entrée et de sortie peut être dissociée en trois sous-parties équivalentes en nombre de portes logiques, cf. figure 4.2.

Dans cette figure, on voit que l'unité d'entrée comprend un bloc IN qui reçoit des données sur l'entrée et les démultiplexe vers les deux canaux virtuels dans les blocs VC0 et VC1. De même, une unité de sortie comprend deux blocs VC0 et VC1 utilisés pour deux canaux virtuels et un bloc OUT qui permet de multiplexer les données à partir des canaux virtuels et de les transporter à la sortie.

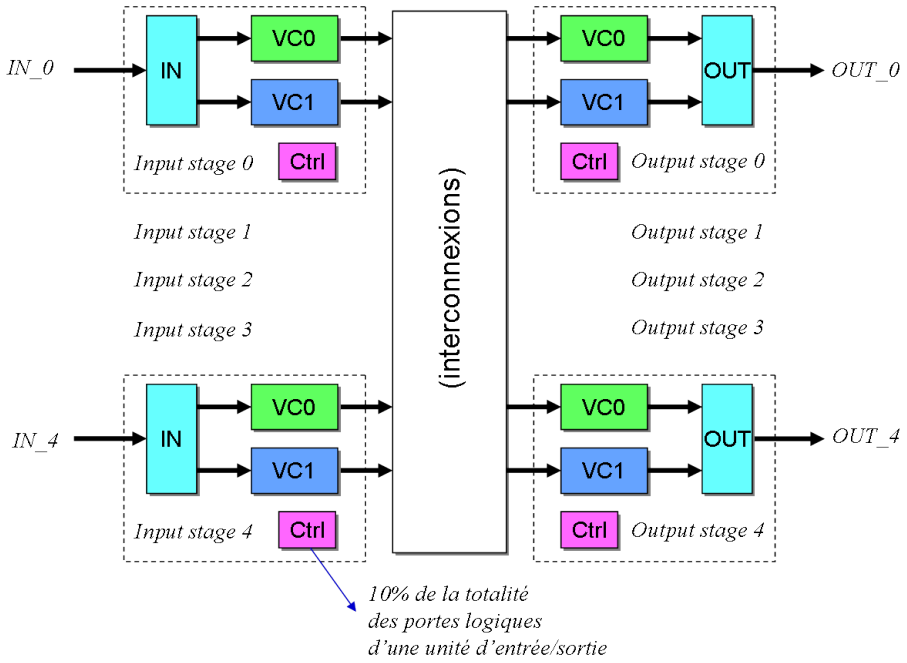


FIG. 4.2 : Analyse de la structure d'un routeur ANOC.

Le test du routeur peut être réalisé de la manière suivante : les données de test sont insérées à un des ports d'entrée et sont observées aux ports de sortie. Pour amener les données de test au routeur, ces données de test doivent respecter le format des paquets de données, décomposés en plusieurs *flits*, comme présenté dans la section 1.3.2. Les deux bits BoP et EoP doivent être remplis afin de pouvoir indiquer le début et la fin d'un paquet de test. Les deux derniers bits (i.e. le dernier digit « 1-of-4 ») du *flit* d'en-tête d'un paquet doivent contenir la direction de routage du paquet. Ceci permet de router les paquets de test entrants sur une unité d'entrée vers les unités de sortie des 4 autres directions. L'injection des paquets de données dans les différentes unités d'entrée est réalisée par le wrapper en 5 phases identiques, cf. section 4.2.2. Au vu de ces signaux de contrôle nécessaires à la couverture complète du routeur, le format d'un paquet de test est donc le suivant, cf. figure 4.3.

De plus, afin d'être routé dans différents canaux virtuels (pour tester les blocs VC0, VC1) de chaque unité d'entrée/sortie, les paquets de test doivent être accompagnés des signaux qui indiquent les canaux virtuels sur le canal « *send* ». La structure d'un port d'entrée/sortie du routeur est identique à celle d'un lien du réseau, comme présenté dans la sous-section précédente, et comprend 17 canaux asynchrones MR[4] pour les données, un canal DR et deux canaux SR pour les signaux « *send–accept* ». Nous pouvons donc utiliser la même manière de générer des vecteurs de test que pour les liens du réseau. Tous les digits « 1-of-4 » d'un *flit*

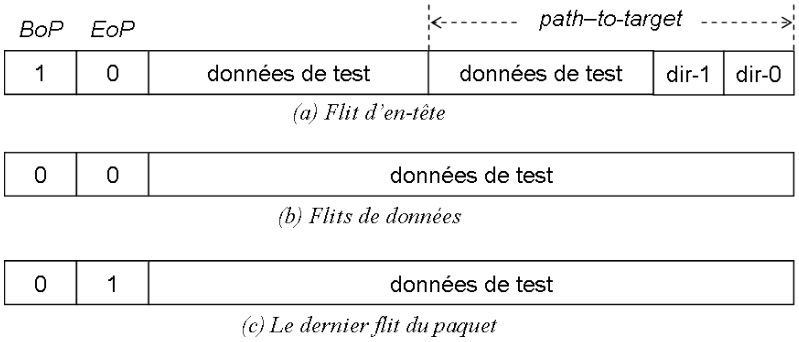


FIG. 4.3 : Format d'un paquet de test.

de test qui ne participent pas au contrôle doivent être forcés par toutes les valeurs possibles (« 0 », « 1 », « 2 », et « 3 »), soit 15 digits, cf. tableau 4.2.

TAB. 4.2 : Vecteurs de test pour un triplet « entrée/sortie/canal virtuel » du routeur

Data			**Send**
BoP/EoP	*15 digits « 1-of-4 »*	*Direction*	*Canal virtuel*
2	« 0.0.0. . . . . 0 »	« dir. »	« vc. »
0	« 0.0.0. . . . . 0 »	« 0 »	« vc. »
0	« 1.1.1. . . . . 1 »	« 1 »	« vc. »
0	« 2.2.2. . . . . 2 »	« 2 »	« vc. »
1	« 3.3.3. . . . . 3 »	« 3 »	« vc. »

On voit que ces vecteurs de test permettent de tester la partie « data » des différents blocs (IN, VCvc) de l'unité d'entrée définie par le wrapper et des différents blocs (VCvc, OUT) de l'unité de sortie définie par la direction de routage (dir.). La couverture des cinq unités d'entrée ne correspond pas à des bits de contrôle dans le paquet, mais à cinq phases de fonctionnement du wrapper pour injecter les mêmes données de test aux différentes unités d'entrée. Pour tester toutes les unités de sortie et tous les sous-blocs VC0 et VC1, les digits « dir. » et « vc. » doivent être forcés à toutes les valeurs possibles.

Cependant, pour couvrir la partie contrôle de chaque unité d'entrée/sortie, toutes les combinaisons possibles des digits de contrôle doivent être considérées, notamment le digit BoP/EoP, et tous les digits dans le champ *path-to-target*. Ils doivent aussi être forcés à toutes les valeurs possibles. Pour faire cela, nous avons défini encore 3 paquets d'un seul *flit*, décrits dans le tableau 4.3 suivant.

Au final, pour tester tous les blocs d'un triplet unité d'entrée, unité de sortie, canal virtuel, nous avons besoin de 8 vecteurs de test. Comme un routeur a 5 entrées,

TAB. 4.3 : Vecteurs de test pour un triplet « entrée/sortie/canal virtuel » du routeur

Data			**Send**
BoP/EoP	*15 digits « 1-of-4 »*	*Direction*	*Canal virtuel*
3	« 1.1.1. . . . . 1 »	« dir. »	« vc. »
3	« 2.2.2. . . . . 2 »	« dir. »	« vc. »
3	« 3.3.3. . . . . 3 »	« dir. »	« vc. »

que chaque entrée est connectée à 4 sorties, et qu'il y a deux canaux virtuels le nombre total de vecteurs de test nécessaires pour tester un routeur complet est donc $8 * 5 * 4 * 2 = 320$ vecteurs. Ces vecteurs de test sont insérés dans le routeur sous test en utilisant le wrapper de test présenté dans le chapitre précédent.

Pour insérer un vecteur de test puis récupérer le résultat, dans le cas général nous avons besoin de deux configurations de test (cf. section 3.3.7.5), le nombre de configurations de test nécessaire au test complet d'un routeur est donc 640 TCFs. Ces configurations de test sont calculées de manière automatique par un programme *ad hoc*.

La couverture de faute de ces vecteurs de test pour les fautes de collage sera présentée dans la section 4.2.5. Dans la section suivante, nous allons présenter l'utilisation de l'architecture développée pour tester tous les éléments du réseau ANOC.

4.2 Application des vecteurs de test et taux de couverture

Dans cette section, on présente d'abord comment tester l'architecture CVT elle-même, puis comment utiliser cette architecture pour tester les routeurs et les liens du réseau. Ensuite, nous introduisons une stratégie de test pour un réseau ANOC complet. Finalement, les résultats du test seront présentés.

4.2.1 Test de l'architecture CVT

Le test de l'architecture CVT peut être réalisé par les deux étapes suivantes : le test de la chaîne de configuration de test, et le test des cellules de test dans les wrappers.

La chaîne de configuration est une chaîne de 2 bits implémentée par un canal MR[4]. Pour tester cette chaîne de configuration, il est suffisant d'envoyer toutes les valeurs d'un digit « 1-of-4 ». Cependant, ceci doit être fait en respectant la condition

qu'il ne reforme pas une TCF qui puisse affecter un des wrappers dans le réseau. En effet, si une valeur envoyée configure un wrapper alors ce wrapper sera en attente d'une donnée de test. Tant que cette donnée n'est pas attribuée à ce wrapper la chaîne de configuration sera pertubée. Or la valeur « 3 » indique la fin d'une TCF, et déclenche la lecture de l'identifiant (cf. section 3.3.5). Ainsi, il doit toujours rester un identifiant libre pour n'adresser aucun wrapper. Dans notre cas, le réseau ANOC se compose de 20 routeurs donc, il n'existe aucun identifiant qui soit supérieur à 20. Les valeurs possibles d'un digit « 1-of-4 » peuvent donc être envoyées sur la chaîne de configuration en l'ordre suivant : « 0 », puis « 1 », puis « 2 », puis « 3 » de sorte que la TCF constituée est « 3.2.1.0.0.0. . . . . 0 ». L'identifiant « 2.1.0 » n'affecte aucun wrapper dans le réseau (il n'existe que 20 wrappers tandis que l'identifiant de la TCF indique le wrapper numéro 22). Dans le cas général, un ID est laissé libre, et la TCF transmise est « 3.**Id-libre**.2.1.0. . . . . 0 ».

Lorsque la chaîne de configuration a été testée, nous pouvons tester la chaîne de décalage qui est construite par des cellules de test des wrappers. Pour établir cette chaîne de décalage, nous devons configurer tous les wrappers en mode traversée, sauf le dernier en mode demi-tour (cf. figure 4.4). Puis on applique les vecteurs de test

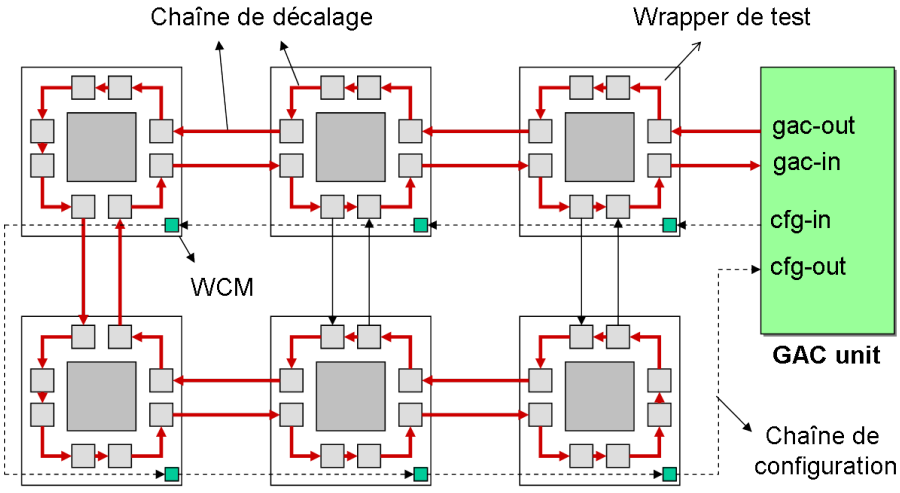


FIG. 4.4 : Chaîne de décalage pour tester les cellules elles-mêmes

sur cette chaîne de décalage. Les vecteurs pour tester cette chaîne de décalage sont les mêmes que ceux qu'on utilise pour tester des liens du réseau, soit 4 vecteurs de 17 digits « 1-of-4 » : « 0.0.0. . . . . 0 », « 1.1.1. . . . . 1 », « 2.2.2. . . . . 2 », et « 3.3.3. . . . . 3 ». Ce test permet de trouver des défauts dans les cellules de test ou sur les liens qui font la partie de la chaîne de décalage.

Une fois que l'architecture CVT a été testée, nous pouvons l'utiliser pour tester les routeurs et les liens du réseau. Ces tests sont les objectifs des sous-sections

suivantes.

4.2.2 Test des routeurs du réseau

Afin de tester un routeur, le wrapper de test de ce routeur doit être configuré pour qu'il puisse recevoir les vecteurs de test à partir du mécanisme d'accès de test (TAM) constitué par les wrappers des autres routeurs, et les insérer dans le routeur. Puis, le wrapper de test doit être configuré pour qu'il puisse récupérer les réponses à partir des sorties du routeur sous test et les envoyer au testeur par le TAM. Les wrappers utilisés pour construire le TAM peuvent être configurés en mode bypass ou en mode traversée. On note que le mode bypass est préféré pour minimiser le temps de contrôle car il n'exige qu'une seule configuration TCF par wrapper pour toute la suite du test. Cependant, ce mode est établi une fois pour toutes et une réinitialisation (*reset*) est nécessaire pour en sortir.

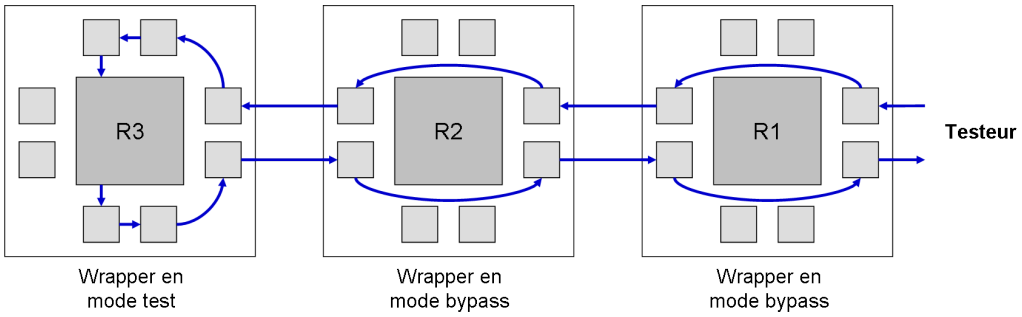


FIG. 4.5 : Un exemple du test du routeur.

Dans la figure 4.5, nous présentons un exemple du test pour le routeur 3 (R3). On suppose que le testeur est connecté aux ports d'entrée/sortie Est du routeur R1. Pour construire un TAM afin de tester le routeur R3, nous devons donc configurer les wrappers des routeurs R1 et R2 en mode bypass. Ceci ne prend que deux configurations de test. Ensuite, on réalise le test du routeur R3 en envoyant l'ensemble des vecteurs de test générés (cf. section 4.1.2) avec leurs configurations de test correspondantes (cf. section 3.3.5). Chaque vecteur de test s'accompagne de deux configurations de test : une pour insérer le vecteur de test au routeur sous test, l'autre pour récupérer la réponse du routeur et la transporter au testeur. Ces configurations de test sont générées par un programme *ad hoc*.

4.2.3 Test des liens du réseau entre les routeurs

Pour tester les liens entre deux routeurs, nous devons utiliser deux wrappers de test de ces routeurs. Ces wrappers doivent être configurés afin d'insérer les vecteurs

de test, récupérer les réponses et les envoyer au testeur. Pour faire cela, le premier wrapper doit être configuré en mode traversée et le deuxième wrapper doit être configuré en mode demi-tour (rebouclage), cf. figure 4.6. Les wrappers précédents peuvent être configurés en mode bypass.

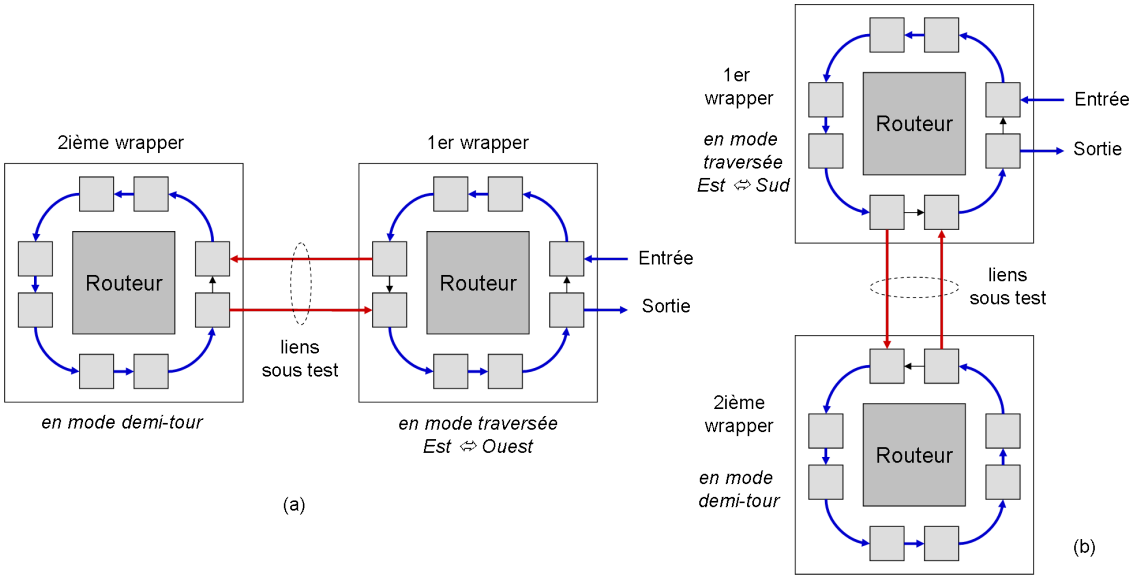


FIG. 4.6 : Configuration des wrappers pour tester des liens du réseau.

La figure 4.6(a) présente un cas où on veut tester les liens du réseau en direction horizontale. La figure 4.6(b) présente un autre cas où on veut tester les liens du réseau en direction verticale. Les premiers wrappers dans ces deux cas sont configurés en mode traversée. Lorsqu'un wrapper est en mode traversée, il relie les entrées et les sorties de deux directions les unes aux autres, aiguillant ainsi les vecteurs de test vers le deuxième wrapper configurés en mode demi-tour (rebouclage).

L'exemple suivant présente le test des liens entre les routeurs R3 et R8, cf. figure 4.7.

On suppose que le testeur est connecté aux ports d'entrée/sortie Est du routeur R1. Pour construire un TAM afin de tester les liens entre le routeur R3 et le routeur R8, nous devons donc configurer les wrappers des routeurs R1 et R2 en mode bypass. Ceci prend deux configurations de test. Ensuite, on réalise le test de ces liens en envoyant l'ensemble des vecteurs de test générés (cf. section 4.1.1) avec leurs configurations de test (cf. section 3.3.5). On note que chaque vecteur de test s'accompagne de deux configurations de test : une configuration est utilisée pour chaque wrapper. Les configurations associées aux vecteurs de test sont les mêmes pour chaque test d'un lien du réseau. Ces configurations de test sont générées de manière automatique par un programme *ad hoc*.

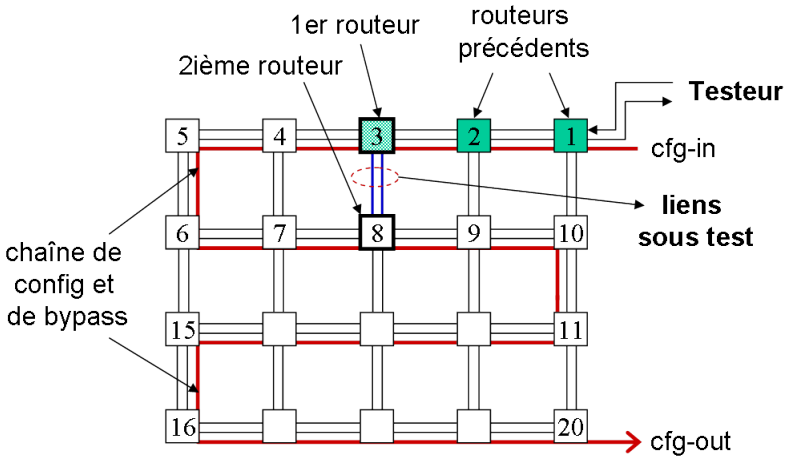


FIG. 4.7 : Un exemple du test des liens entre les routeurs R3 et R8.

4.2.4 Algorithme de test pour le réseau ANOC entier

Les sous-sections précédentes ont présenté comment utiliser l'architecture développée pour tester chaque routeur du réseau et chaque paire de liens entre deux routeurs. Dans cette sous-section, on présente une stratégie globale qui permet de tester tous les éléments du réseau ANOC (i.e. tester le réseau entier). Avec cette stratégie, seul un routeur est testé à la fois, puis on fait le test des liens du réseau qui sont connectées à ce routeur. Quand le test du routeur courant et de ses liens au réseau est fini, nous configurons son wrapper de test en mode bypass et nous pouvons tester le routeur suivant et ses liens au réseau. Le flot de test est défini par la chaîne de configuration et indiqué par le trait gras, cf. figure 4.8.

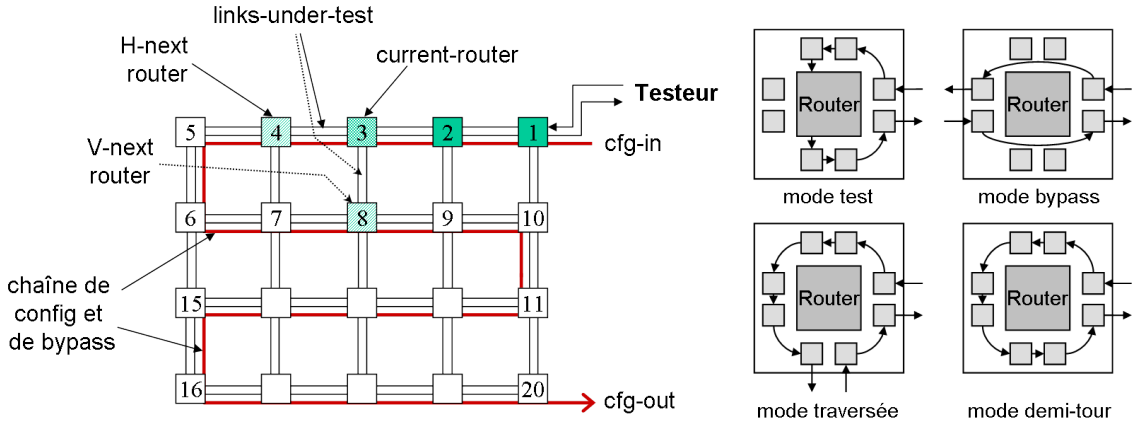


FIG. 4.8 : Stratégie de test globale pour le réseau ANOC.

Avec ce flot du test, on a proposé un algorithme de test pour le réseau ANOC

comme ci-dessous :

ALGORITHME DE TEST POUR LE RESEAU ANOC

```
Set current-router = 1;
While current-router = N do
   /* N is number of network routers */

   /***** Router test *****/
   Set current-router in test mode;
   /* see figure 4.8 for different modes used in test process */
   Apply all router test vectors (320 vectors) to the router-under-test;
   /* A test vector is applied with its 2 TCF */

   /***** Network Link test *****/
   If current-router is not in the last row of the network
   then
      Set current-router in transfer mode;
      /* Transfer direction towards SOUTH (V-next router) */
      Set V-next-router in loop-back mode;
      /* because of loop-back, a single vector tests both */
      /* directions of a bidirectional link */
      Apply all link test vectors (4 vectors) to
      the vertical links-under-test;
      /* links-under-test is the vertical links between */
      /* current router and V-next router */
   End if;
   If the current-router is not the last router of the row-under-test
   then
      Set current-router in transfer mode;
      /* Transfer direction towards H-next router */
      Set H-next-router in loop-back mode;
      Apply all link test vectors (4 vectors) to
      the horizontal links-under-test;
      /* links-under-test is the horizontal links */
      /* between current router and H-next router */
   End if;
```

```
   /***** Prepare next test iteration *****/
   Set current-router in bypass mode;
   /* Set current-router in bypass mode to go to the next-router. */
   /* Note that the bypass direction is defined by the test flow*/
   current-router = current-router + 1;
End while;
```

4.2.5 Résultats du test

4.2.5.1 Temps d'application du test

Dans la section 3.3.7.5, nous avons montré que la vitesse du test dans le cas général est $10Mvecteurs$ par seconde. En utilisant la stratégie de test et les vecteurs de test présentés dans les sections précédentes, le temps du test pour chaque routeur est $32\mu s$; et le temps du test pour chaque paire de liens entre deux routeurs est $0,4\mu s$. En conséquence, le temps d'application du test pour un réseau ANOC de 20 routeurs est d'environ $0,7ms$. Le tableau 4.4 présente le temps d'application du test pour des réseaux de différentes tailles.

TAB. 4.4 : Temps de test pour des réseaux ANOC de différentes tailles

Taille du réseau sous test	**Temps de test (μs)**		
	seulement les routeurs	*seulement les liens*	*le réseau entier*
3×3	288,40	5,20	293,20
3×4	384,55	7,35	391,35
4×4	512,75	10,35	522,35
4×5	640,95	13,35	653,35

4.2.5.2 Couverture de faute

Pour évaluer l'efficacité de notre approche de test, nous utilisons le modèle de fautes de collage simple (*SSAF : Single Stuck-At Fault*). Comme il n'existe aucun outil qui permet de calculer la couverture de faute pour les circuits asynchrones, on a développé un programme basé sur des scripts afin d'insérer de manière automatique des fautes de collage sur les entrées et les sorties de chaque porte logique du routeur (une seule faute à la fois), puis simuler et calculer les fautes détectées. Une faute est détectée si le circuit est suspendu (plus de transitions sur les sorties ou sur les signaux d'acquittement) avant la fin attendue du test.

Avec les vecteurs de test générés dans la section 4.1, notre méthode de test présente une couverture de faute de 99,86% pour le test du routeur du réseau et une couverture de faute de 100% pour le test des liens du réseau en utilisant le modèle de faute de collage simple. Cette bonne couverture de faute obtenue est possible grâce à l'architecture de type flot de données du routeur du réseau. En effet, le routeur n'est pas une unité de traitement, mais une structure des pipelines avec des multiplexeurs et démultiplexeurs. De plus, le réseau ANOC est réalisé en utilisant le style de conception basé sur des jetons (token-based). En étudiant les fautes non-détectées, nous nous sommes aperçu que ces fautes sont les fautes qui se localisaient avant les entrées d'un petit nombre de portes de Muller asymétriques utilisées dans le routeur afin de vérifier les conditions aléas sur les jetons de contrôle utilisés pour certain jetons de données. Ces fautes causent des mises à feu prématurées, cf. section 2.3.2. Pour tester ces fautes, il faut trouver des bons vecteurs de test afin de rendre le circuit dans un état prématuré, puis trouver un moyen d'observer cette mise à feu prématurée.

Le tableau 4.5 détaille le nombre de fautes injectées, le nombre de fautes détectées, et la couverture de faute de chaque partie du routeur, puis du routeur entier.

TAB. 4.5 : Rapport de la couverture de fautes du test du routeur

Circuit sous test	**Unité d'entrée**	**Unité de sortie**	**Routeur entier**
SSAF sur les sorties	3178/3178 (100%)	5185/5198 (99,75%)	41815/41880 (99,84%)
SSAF sur les entrées	6016/6016 (100%)	8925/8944 (99,79%)	74705/74800 (99,87%)
SSAF sur les entrées et les sorties	9194/9194 (100%)	14110/14142 (99,77%)	**116520/116680 (99,86%)**

La couverture de faute du test de l'architecture CVT elle-même est 99,75% en utilisant tous les vecteurs de test possibles, et 76,81% en utilisant seules les vecteurs de test qui ne stimulent pas le routeur. Le tableau 4.6 détaille le nombre de fautes injectées, le nombre de fautes détectées, et la couverture de faute de chaque partie du wrapper, puis du wrapper entier. Les fautes non-détectées sont les fautes qui

TAB. 4.6 : Rapport de la couverture de fautes du test du wrapper

Circuit sous test	**Cellule de test**	**WCM**	**Wrapper entier**
SSAF sur les entrées et sorties	3380/3382 (99,94%)	3400/3472 (97,93%)	**37200/37292 (99,75%)**

se localisaient sur les signaux *ctrl-mode* dans les cellules de test (avant les entrées des portes de Muller asymétriques) et sur les signaux concernant la vérification de l'identifiant (ID) du wrapper de test.

4.3 Utilisations alternatives de l'architecture CVT

L'architecture CVT est développée pour tester le réseau sur puce. Cependant, elle permet également d'autres utilisations. Dans cette section, nous allons présenter comment utiliser cette architecture pour faire le diagnostic, pour réaliser la vérification sur silicium (debug), pour tester les IP.

4.3.1 Utilisation de l'architecture CVT pour le diagnostic

L'architecture CVT proposée permet non seulement de détecter les fautes de collage, mais également de déterminer la localisation des défauts générant ces fautes. C'est-à-dire que nous pouvons réaliser un diagnostic avec l'architecture proposée. Le diagnostic permet de localiser les fautes dans le circuit sous test et puis de trouver les problèmes de fabrication qui créent ces fautes. En conséquence, il permet d'améliorer les procédures de fabrication afin d'éviter ces fautes dans la suite. De plus, le diagnostic est un point très intéressant pour le test des réseaux sur puce grâce à la multiplicité des trajets dans le réseau. On peut re-router les paquets et faire fonctionner tout de même certaine application.

Pour ces scénarios, notre approche ne nous permet pas de dire exactement dans quelle porte une faute se localise mais elle permet de déterminer quel lien du réseau et quel routeur du réseau sont défectueux. Plus précisément, elle permet de déterminer pour les liens le rail exact où se situe la faute, et pour les routeurs, la sous-partie d'une unité d'entrée ou de sortie où se situe la faute, cf. figure 4.9. En conséquence, nous pouvons isoler un morceau de 1/30 du routeur qui contient les fautes, soit environ 600 portes logiques, et ce en regardant uniquement la bonne progression du test (capacité d'injecter de nouvelles valeurs et d'observer un résultat). Une analyse des digits « 1-of-4 » reçus permet en outre de réduire ce chiffre à 40 portes logiques. En effet, soit aucune donnée ne sort et le défaut est situé sur une partie de génération du contrôle (concerne 10% de la logique de chaque unité d'entrée/sortie, soit 60 portes logiques), soit un digit unique ne sort pas et le défaut est situé sur un des 15 digits de « *payload* » (soit 40 portes logiques).

4.3.2 Utilisation de l'architecture pour le déverminage du premier prototype (la vérification sur silicium)

L'architecture CVT peut également aider les concepteurs à vérifier les fonctionnalités du routeur sur silicium. Les vérifications simples qui ont pu être réalisées par simulation pendant les étapes de conception peuvent être aisément reproduites, par exemple, la vérification des directions de routage et des canaux virtuels.

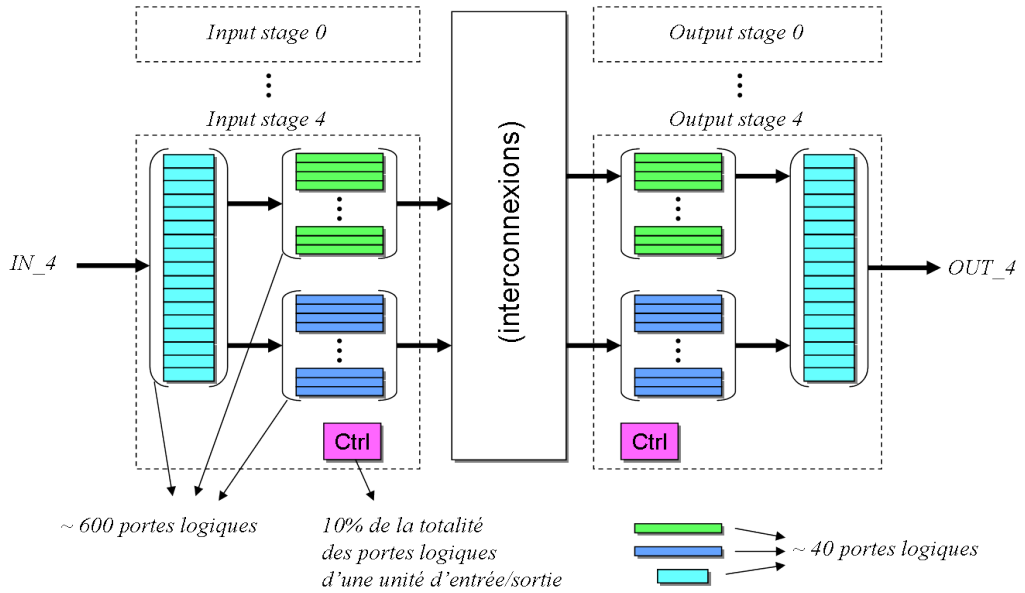


FIG. 4.9 : Partitionnement des différentes unités du routeur pour le diagnostic.

Dans cette sous-section, nous présentons l'utilisation de l'architecture pour réaliser les vérifications que nous n'avons pas pu réaliser pendant la conception à cause du manque des paramètres physiques de l'implémentation. Par exemple, dans la conception on n'a pas pu vérifier l'arbitrage du routeur des paquets concurrents parce qu'on ne sait pas les délais d'implémentation des fils physiques. On note que les paquets concurrents sont des paquets entrant aux différents ports d'entrée du routeur et destinés à la même sortie du routeur.

Pour vérifier ce problème, nous devons envoyer deux ou plusieurs paquets concurrents au routeur en même temps. Ceci peut être garanti par le wrapper. Nous devons configurer le wrapper pour stocker différents paquets concurrents dans les cellules qui correspondent aux ports d'entrée ciblées du routeur, puis configurer le wrapper pour envoyer ces paquets au routeur en même temps. Par exemple, la figure 4.10 présente un cas où il y a deux paquets de données entrant aux différentes entrées du routeur (l'entrée Ouest et l'entrée Sud) qui sont destinés à la sortie Est. Ces paquets ont le même niveau de la priorité. En observant sur la sortie Est, on peut vérifier que les paquets ne se perturbent pas l'un l'autre : ils arrivent avec tous les *flits* de l'un puis tous les *flits* de l'autre dans l'ordre.

La figure 4.11 montre un autre exemple : il y a aussi deux paquets de données entrant aux différentes entrées du routeur (l'entrée Ouest et l'entrée Sud) qui sont destinés à la sortie Est du routeur. Ces paquets ont différents niveaux de priorité. Le paquet entrant sur l'entrée Ouest (P1) est plus prioritaire que le paquet entrant sur

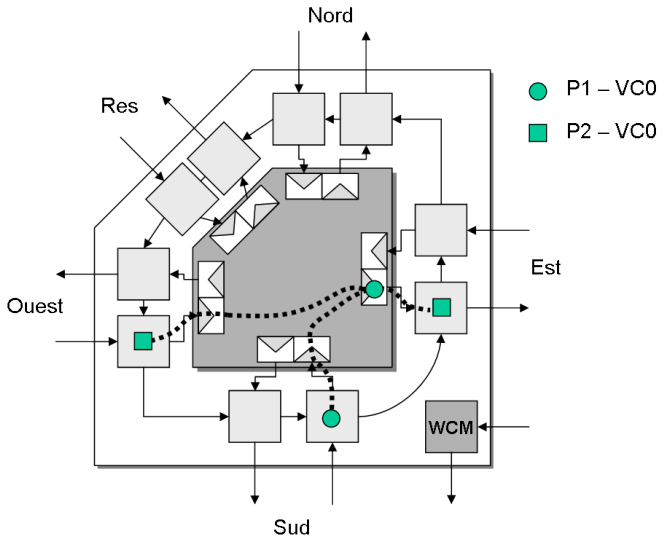


FIG. 4.10 : Vérification de l'ordre de l'arbitrage des paquets concurrents (même niveau de priorité).

l'entrée Sud (P2). Les paquets doivent conserver leurs canaux virtuels respectifs : aucun flit ne doit changer de canal virtuel.

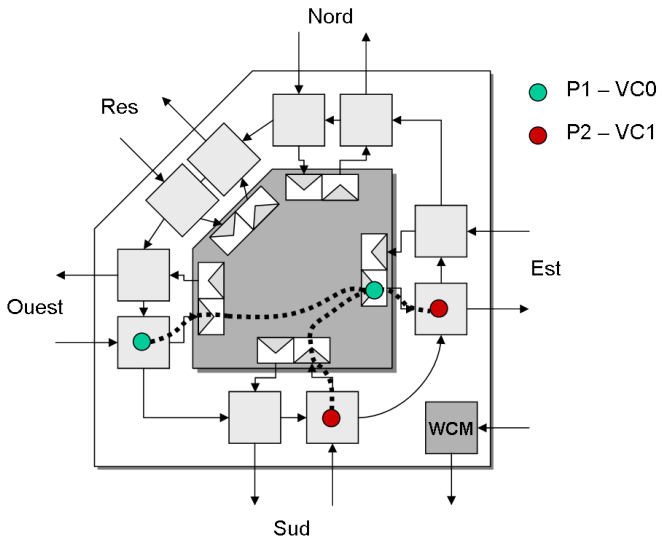


FIG. 4.11 : Vérification la priorité de l'arbitrage des paquets concurrents.

Les deux exemples précédents ont montré comment on peut vérifier l'arbitrage du routeur quand il y a des paquets de données concurrents. Dans la suite, nous allons présenter une autre vérification qui concerne l'influence de deux paquets successifs sur une même entrée l'un sur l'autre. Ceci n'a pas pu être réalisé pendant

la conception à cause du manque des paramètres physiques de l'implémentation du routeur.

Pour vérifier ce problème, nous devons faire rentrer très vite deux paquets différents (qui sont destinés aux différentes sorties) au même port d'entrée du routeur. Ceci peut être réalisé en configurant des différents wrappers des routeurs. La figure 4.12 présente comment réaliser cette vérification. D'abord, nous devons confi-

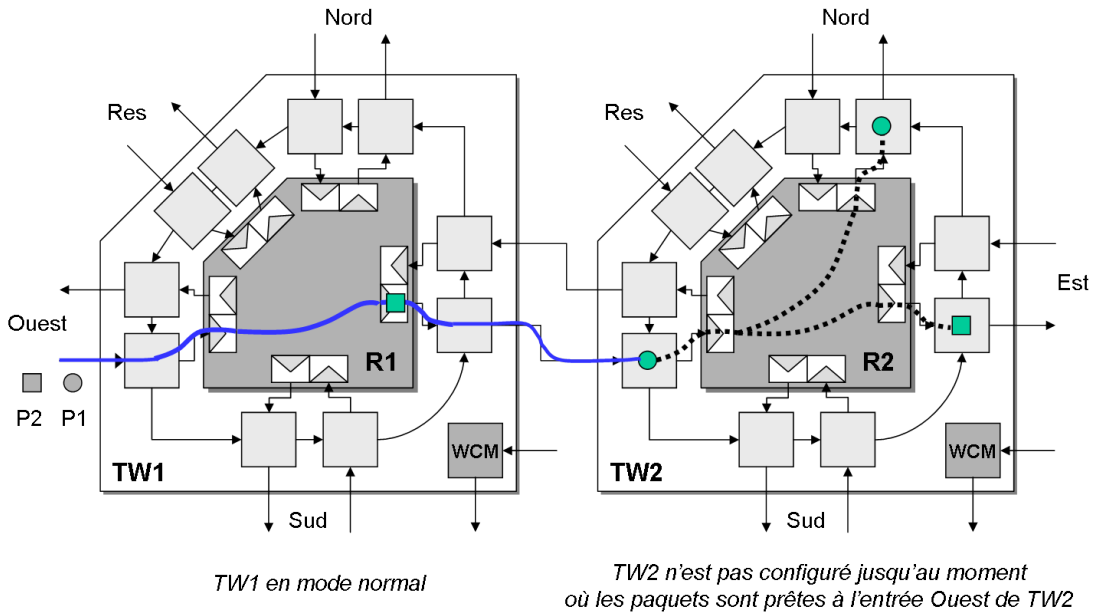


FIG. 4.12 : Vérification la direction de routage d'un paquet en présence d'un autre paquet.

gurer le wrapper TW1 en mode normal et ne pas configurer le wrapper TW2. Les différents paquets de données entrants sur l'entrée Ouest de routeur R1 sont transportés vers la sortie Est du routeur R1. Comme le wrapper TW2 n'est pas encore configuré, ces paquets sont bloqués à la cellule d'entrée Ouest du wrapper TW2. Dans la figure, il y a deux paquets P1 et P2 : P1 est stocké dans la cellule d'entrée Ouest du wrapper TW2, P2 est stocké à l'unité de sortie Est du routeur R1. Ensuite, on configure le wrapper TW2 en mode normal, ces paquets sont immédiatement envoyés au routeur R2, puis transportés aux sorties prévues. Pour observer ces paquets aux sorties de routeur R2, il faut utiliser les wrappers voisins correspondants. De la même manière, on peut également vérifier ce problème avec deux paquets qui ont différents niveaux de priorité.

4.3.3 Utilisation de l'architecture CVT développée pour tester les IP

L'architecture CVT développée peut être utilisée comme un mécanisme d'accès du test (TAM) pour tester les unités de traitement (IP). Dans cette section, on présente comment cette architecture peut être utilisée pour tester les IP et quels avantages nous pouvons obtenir dans ce cas-là. Notons bien qu'on ne rentre pas dans le détail du test des IP.

4.3.3.1 Test des IP en utilisant le réseau de communication comme un mécanisme d'accès de test

Comme nous avons discuté dans la section 2.1.6, les unités de traitement peuvent être testées en utilisant les approches classiques telles que la chaîne de *scan* ou la norme IEEE 1500, et le réseau de communication peut être utilisé comme un mécanisme d'accès de test. Dans le cas de FAUST, l'approche chaîne de *scan* est utilisée pour tester les unités de traitement et leurs interfaces réseaux. Afin de réutiliser le réseau comme un TAM, un wrapper de test d'IP (IP-TW) a été développé qui puisse réaliser l'interfaçage entre le réseau et les chaînes de *scan*. La figure 4.13 présente la conception de ce wrapper IP-TW.

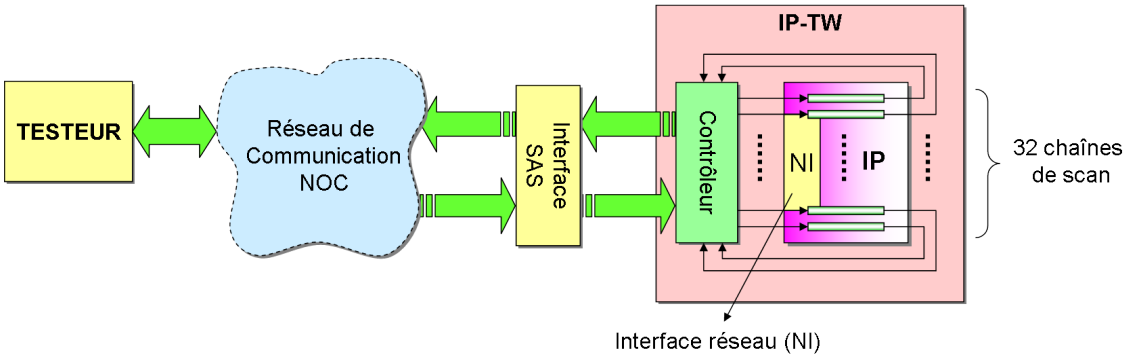


FIG. 4.13 : Conception d'un wrapper de test d'IP (IP-TW) en utilisant le réseau comme un TAM.

Dans cette figure, le test d'une IP avec son interface réseau est réalisé en utilisant 32 chaînes de *scan*. Les opérations de ces chaînes sont contrôlées par un bloc de contrôle du wrapper IP-TW. Ce contrôleur est en charge aussi pour le protocole réseau. Le fonctionnement du wrapper IP-TW peut être expliqué de la manière suivante : les vecteurs de test sont encapsulés en paquet afin d'être envoyés au wrapper IP-TW par le réseau. Le format d'un paquet de vecteurs de test se compose de deux *flits* de contrôle et d'un ou plusieurs *flits* de données de test, cf. figure 4.14.

Le premier *flit* de contrôle contient les informations du chemin de routage aller

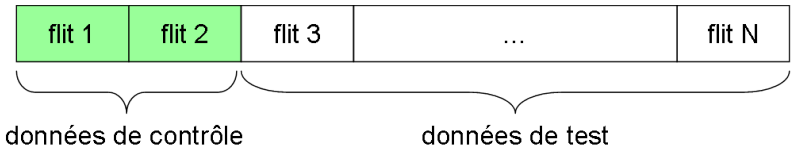


FIG. 4.14 : Format d'un paquet de test pour les IP.

et les configurations des chaînes de *scan*, le deuxième *flit* de contrôle contient les informations du chemin de routage de retour pour le paquet de réponse. Les *flits* suivants sont les *flits* de données de test. Les formats de ces *flits* sont les mêmes formats que des *flits* de données décrits dans la section 1.3.2.

On voit bien que l'utilisation du réseau de communication comme un TAM permet de réduire le coût du TAM. Cependant, il faut implémenter le protocole du réseau dans l'architecture du wrapper IP-TW. Par ailleurs, si le réseau contient un défaut, on peut vouloir tester les unités à des fins de diagnostic. A cause de cela, nous allons présenter dans la sous-section suivante l'utilisation de l'architecture CVT développée comme un TAM alternatif pour tester des IP.

4.3.3.2 Test des IP en utilisant l'architecture CVT développée comme un mécanisme d'accès de test alternatif

Le mécanisme d'accès de test peut être établi par les wrappers des routeurs et les liens du réseau. Il n'y a alors plus besoin d'intégrer le protocole de réseau dans l'IP-TW. Cependant, il faut configurer les wrappers des routeurs pour transporter les vecteurs de test. La figure 4.15 présente l'implémentation du test d'une IP avec son interface réseau en utilisant l'architecture CVT proposée comme un TAM. Dans cette implémentation, afin d'établir un TAM pour tester une IP, les wrappers des routeurs sont configurés de la manière suivante : le wrapper du routeur qui est connecté à l'IP sous test est configuré en mode traversée et les wrappers des routeurs précédents sont configurés en mode bypass.

Le wrapper IP-TW présenté dans la figure 4.13 peut être utilisé mais notons qu'il peut être optimisé (supprimer la partie qui implémente le protocole du réseau) pour réduire le coût en surface.

Dans notre prototype, on utilise le même wrapper IP-TW, c'est-à-dire qu'on ne supprime pas le bloc de routage qui implémente le protocole du réseau (inclu dans le contrôleur de la figure 4.13) dans le wrapper IP-TW. Ainsi, les données concernant le chemin de retour contenu dans le flit 1 de la figure 4.14 seront ignorés puisque sans intérêt pour les wrappersde test des routeurs. Finalement, seules les informations

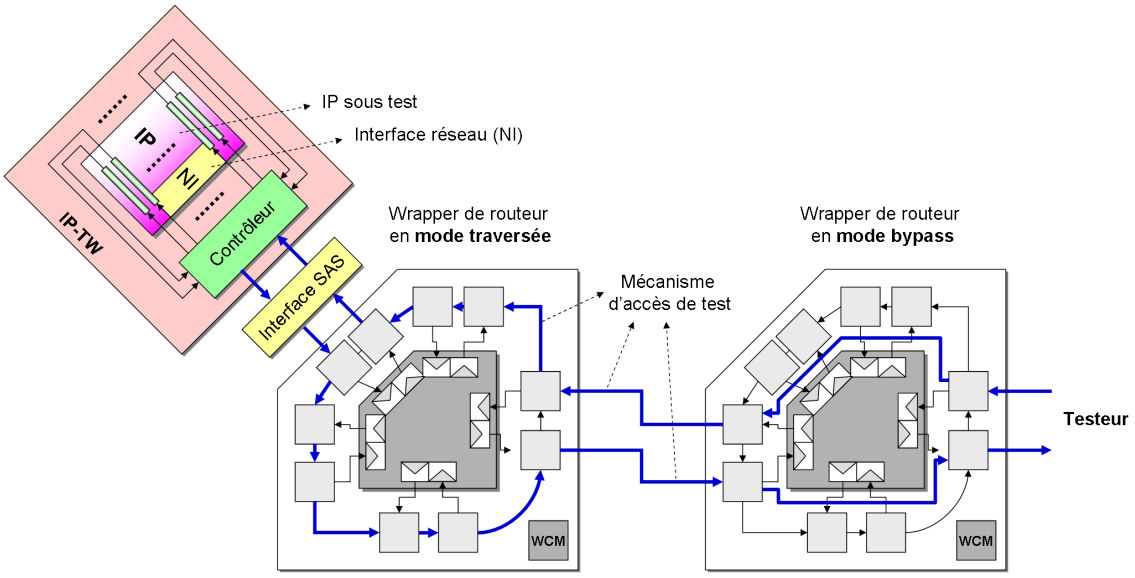


FIG. 4.15 : Implémentation du test d'une IP en utilisant l'architecture CVT comme un TAM.

contenues dans le flit 2 de la figure 4.14 concernant la configuration des chaînes de *scan* de l'IP seront utilisées.

4.3.3.3 Stratégie du test pour les IP

L'objectif de cette sous-section est de présenter une stratégie globale qui permet de tester toutes les unités de traitement dans un système sur puce basé sur le réseau ANOC. Comme les IP sont connectées aux routeurs, la stratégie du test des routeurs peut être adoptée. On définit un flot du test correspondant à la chaîne de configuration (i.e. chaîne de bypass), cf. figure 4.16.

Avec ce flot du test, pour tester tous les IP d'un système basé sur le réseau ANOC, un seul IP à la fois, nous avons proposé un algorithme de test comme ci-dessous :

ALGORITHME DE TEST POUR LES IP

```
Set current-router = 1;
While current-router = N do
   /* N is number of network routers */
   Set current-router in transfer mode;
   Apply all IP test vectors to the IP-under-test;
```

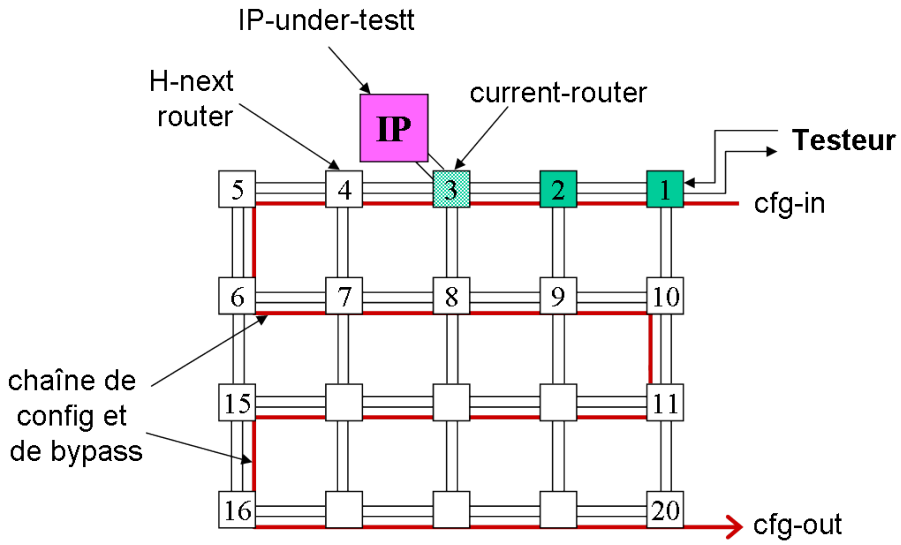


FIG. 4.16 : Flot de test pour les unités de traitement.

```
   /* A test vector is applied with its TCF */

   /***** Prepare next test iteration *****/
   Set current-router in bypass mode;
   /* Set current-router in bypass mode to go to the next-router */
   /* Note that the bypass direction is defined by the test flow */
   current-router = current-router + 1;
End while;
```

4.3.3.4 Résultats du test

– ***Bande passante du test***

En utilisant l'architecture CVT proposée pour transporter un vecteur de test vers l'IP et récupérer un vecteur de réponse, nous devons utiliser une configuration de test (une TCF). Considérons la section 3.3.7.5, la bande passante du test maximum est donc environ $20 Mvecteurs/s$.

– ***Temps d'application du test***

Considérons le format d'un paquet de test (cf. figure 4.14), il est nécessaire d'envoyer deux *flits* de contrôle avant les vecteurs de test pour tester une IP. Le temps de test pour une IP peut être calculé de la manière suivante :

(N_{PDT} * L_{CDS} + 2 vecteurs de contrôle) * 1/Bande passante du test + T_{CFG}

Où : N_{PDT} est le nombre des patterns de test ; L_{CDS} est la longueur des chaînes de *scan* ; T_{CFG} est le temps de configuration des wrappers précédents en mode bypass. On rappelle qu'il faut deux vecteurs de contrôle pour contrôler le wrapper IP-TW, cf. section 4.3.3.1.

Par exemple, on considère le test d'un bloc FHT qui est connecté au routeur R3 du réseau, cf. figure 4.17. Pour tester ce bloc, les wrappers des routeurs R1 et

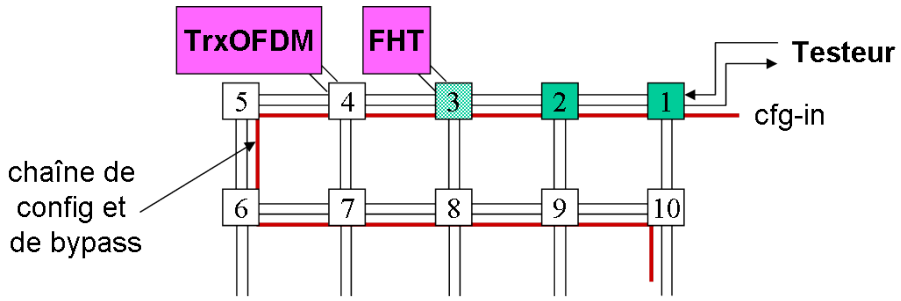


FIG. 4.17 : Un exemple du test des IP.

R2 sont configurés en mode bypass. Cela coûte deux TCF, et le temps T_{CFG} est donc $100ns$. Le test de ce bloc FHT est réalisé par 32 chaînes de *scan* longues de 209 bascules chacune. Il faut donc 209 *flits* de données pour positionner ces chaînes de *scan*. Les patterns de test pour ce bloc sont générés de manière automatique par l'outil FASTSCAN de Mentor Graphics [MenGra], soit 493 patterns de test. En conséquence, le temps d'application du test pour ce bloc est donc environ $5,152ms$.

Similairement, considérons le bloc TrxOFDM qui est connecté au routeur R4 du réseau, cf. figure 4.17. Pour tester ce bloc, les wrappers des routeurs R1, R2, et R3 sont configurés en mode bypass. Le temps de configuration des wrappers R1, R2, et R3 en mode bypass est $150ns$ (3 TCF). Le test de ce bloc est réalisé par 32 chaînes de *scan* longues de 271 bascules chacune. Les patterns de test pour ce bloc sont aussi générés par l'outil FASTSCAN, soit 525 patterns de test. Le temps d'application du test pour ce bloc est donc environ $7,114ms$. Le tableau 4.7 présente un résumé sur les résultats du test pour ces deux exemples : le test du bloc FHT et le test du bloc TrxOFDM. Les couvertures de fautes sont calculées avec un ATPG.

4.4 Conclusion

Dans ce chapitre, nous venons de présenter la mise en œuvre de l'architecture CVT proposée pour le réseau ANOC ainsi que les résultats obtenus. Le but était

TAB. 4.7 : Résultats de test pour le bloc FHT et le bloc TrxOFDM

IP sous test	**Longueur de chaîne**	**Nombre de patterns**	**T_{CFG} des wrappers précédents**	**Temps de test**	**Couverture de test**
FHT	209	493	$100ns$	$5,152ms$	96,28%
TrxOFDM	271	525	$150ns$	$7,114ms$	94,42%

de présenter comment utiliser l'architecture CVT développée pour tester tous les éléments du réseau et de mesurer l'efficacité de notre méthode de test. Comme il n'existe aucun outil pour générer des vecteurs de test, nous avons proposé une méthode de génération des vecteurs de test. Un outil de génération automatique des vecteurs de test reposant sur cette méthode a aussi pu être développé. Grâce à ces vecteurs de test, l'architecture CVT permet d'atteindre une bonne couverture de faute pour le réseau ANOC avec un temps de test relativement faible.

L'architecture CVT est développée pour tester le réseau mais elle peut également être utilisée pour d'autres buts. Dans ce chapitre, nous avons présenté différentes utilisations de cette architecture. Par exemple, nous avons montré comment utiliser cette architecture pour faire le diagnostic afin de localiser et trouver les défauts physiques existants dans le réseau. Ceci permet entre autre, d'améliorer le processus de fabrication. Les autres utilisations de cette architecture telles que la vérification fonctionnelle du réseau sur silicium et le test des unités de traitement ont aussi été présentées. La vérification sur silicium permet aux concepteurs de trouver les erreurs (*bugs*) qu'ils ne peuvent pas trouver pendant la conception à cause du manque de précision des modèles avant d'implémentation. L'utilisation de cette architecture pour tester des unités de traitements permet d'éviter d'utiliser le réseau de communication comme un TAM et donc de supprimer une partie matérielle gérant le protocole réseau au niveau de l'IP-TW. Deux exemples de test des unités de traitement en utilisant l'architecture CVT comme un TAM ont été présentés. Dans ces cas, le temps du test est plus long que celui où on utilise le réseau comme un TAM à cause du temps de configuration des wrappers pour établir le TAM. Ce problème peut être résolu et il fait l'objet d'une de nos perspectives.

Conclusions et Perspectives

Aujourd'hui, pour répondre aux besoins de nouvelles applications, les concepteurs intègrent de plus en plus d'unités de traitement dans les systèmes sur puce. La quantité de données échangées dans les systèmes est ainsi augmentée. Les architectures de communication sur puce deviennent donc des éléments très importants qui déterminent la réussite d'un système sur puce. Dans ce contexte, les paradigmes NoC et GALS sont devenus des solutions privilégiées pour la communication dans les SoC complexes. Ils ont conduit à la création des réseaux sur puce asynchrones. Cependant, faute de méthodologies et d'outils de test adaptés, le test matériel de ces réseaux sur puce asynchrones constitue un défi majeur pour l'industrialisation de ces systèmes.

Dans cette thèse, nous avons précisé plusieurs problématiques liées au test des NoC, en particulier celles pour les NoC asynchrones. Puis, nous avons développé une nouvelle méthode de test permettant de répondre à ces problèmes. Cette méthode repose sur une architecture CVT ; qui a été proposée et développée afin d'améliorer la testabilité des réseaux sur puce asynchrones. Dans cette architecture, chaque routeur du réseau est entouré par un wrapper de test. Cette architecture CVT a été modélisée et réalisée en logique asynchrone QDI. Ce dernier point permet de plus facilement l'interfacer aux réseaux asynchrones : il n'y a en effet pas d'horloge de test, ni de coût en surface supplémentaire pour les interfaces de synchronisation associés. Afin d'arriver à cette solution architecturale, différentes propositions ont été étudiées et modélisées à plusieurs niveaux d'abstraction en utilisant différents langages tels que SystemC, CHP, et VHDL.

L'implémentation de cette architecture a finalement été réalisée avec la technologie CMOS $65nm$ de STMicroelectronique. Le coût en surface additionnelle est alors seulement de 3 à 5% de la surface totale du système. Par exemple, il est de 4,8% dans le cas particulier du circuit FAUST (un réseau ANOC qui comprend 20 routeurs). La latence ajoutée au réseau ANOC par le wrapper de test de l'archi-

tecture CVT est relativement faible, soit environ de $0,34ns$. Elle correspond à un délai de deux portes de Muller asymétrique et de deux portes ET. De plus, il n'y a aucune dégradation du débit de communication sur le réseau ; le débit reste donc égal à $500Mflits/s$. La vitesse maximale du test est de $20Mvecteurs/s$ et la vitesse moyenne de $10Mvecteurs/s$.

L'architecture CVT a finalement été intégrée dans le circuit ALPIN. Le but de cette intégration est de valider la méthode de test proposé sur silicium avant de l'utiliser dans de futurs circuits.

Suite au développement de l'architecture CVT, nous avons également proposé une méthode de test mettant en œuvre cette architecture. A cause de l'absence d'ATPG pour les circuits asynchrones, une méthode de génération des vecteurs de test a été proposée. Cette méthode est basée à la fois sur l'analyse des fonctionnalités et l'implémentation structurelle du réseau afin d'obtenir une bonne couverture de faute. Un outil de génération automatique des vecteurs de test a ensuite été développé. Afin d'appliquer ces vecteurs de test à tous les éléments du réseau, nous avons proposé un algorithme de test. Grâce à cet algorithme, le temps d'application du test pour un réseau ANOC de 20 routeurs est d'environ $0,7ms$. De plus, la méthode de test proposée présente une couverture de faute de 99,86% pour le test du routeur et une couverture de test de 100% pour le test des liens du réseau en utilsant le modèle de faute de collage simple.

Finalement, différentes utilisations alternatives de l'architecture CVT ont été étudiées dans cette thèse. En effet, cette architecture a permis de réaliser le diagnostic des fautes pouvant apparaître sur tous les éléments du réseau. L'intérêt du diagnostic est de localiser les défauts dans les premiers prototypes du circuit, puis de résoudre les problèmes de fabrication liés à ces défauts. Ceci permet d'améliorer les procédures de fabrication afin d'éviter ces défauts dans la suite. Selon nos études, l'architecture CVT permet de localiser les défauts à des blocs d'environ 40 portes. De plus, cette architecture nous a également permis de réaliser la vérification fonctionnelle du réseau sur silicium, en particulier pour trouver les erreurs qui n'ont pas pu être trouvées lors des étapes de conception. Une dernière utilisation possible de cette architecture CVT est le test des IP. Dans ce cas, l'architecture CVT est utilisée comme un mécanisme d'accès de test pour tester les IP. L'avantage de cette approche est qu'elle permet d'éviter le surcoût en matériel pour implémenter les protocoles du réseau dans les wrappers de test des IP.

Ce travail de thèse est une avancée pour la mise en œuvre des architectures de réseaux sur puce dédiés aux nouvelles applications (multimédia, télécom, etc.). Notre volonté initiale était de répondre mieux aux besoins du test matériel pour les NoC asynchrones.

Dans la suite de nos travaux, nous souhaitons dans un premier temps valider notre implémentation physique sur le circuit ALPIN. Ceci permet de valider notre méthode et architecture directement sur silicium.

De plus, comme nous l'avons abordé dans ce document le module GAC (Générateur, Analyseur, Contrôleur) peut être implémenté sur puce ou hors puce. Dans notre implémentation, ce module a été réalisé hors puce afin d'avoir la possibilité de le modifier afin de s'adapter aux différentes stratégies de test. Cependant, une fois que nous aurons validé notre stratégie de test, ce module pourrait être implémenté sur puce afin d'obtenir un circuit auto-testable. Ceci permettrait d'éviter l'utilisation de testeurs et permettrait d'obtenir une vitesse de test maximum.

Une autre perspective pour nos travaux est l'optimisation du temps d'application du test en considérant comme contrainte le coût de la surface additionnelle de l'architecture de test. On a vu que, pour la méthode de test proposé, ce temps de test est fixé par le temps de configuration des wrappers. Pour réduire ce temps, il pourrait être possible soit d'ajouter des bypass reconfigurables, soit de configurer parallèlement plusieurs wrappers de test.

La première proposition peut être appliquée pour le test des IP. Comme nous avons vu qu'il faut contrôler le wrapper lié à l'IP-sous-test chaque fois qu'on veut transférer un vecteur de test. Ce problème serait résolu si nous implémentions un bypass vers l'IP-sous-test. Afin d'éviter le surcoût en surface d'un vrai bypass, nous pouvons réaliser ce bypass de manière reconfigurable. C'est-à-dire que le wrapper aura un seul bypass à la fois mais la direction de ce bypass pourra être reconfigurable.

Avec la deuxième proposition, la chaîne de configuration pourrait être coupée en plusieurs chaînes indépendantes. Dans ce cas, une même configuration de test pourrait être utilisée pour configurer plusieurs wrappers de test. Ceci permet de tester plusieurs routeurs (ou IP) en même temps.

Bibliographie

[4More] IST. 4MORE Project Website. *www.ist-4more.org*.

[Amba] ARM. ARM AMBA Bus Architecture. *www.arm.com*.

[Amor05A] A.M. Amory, E. Brião, É. Cota, M. Lubaszewski, and F.G. Moraes. A Scalable Test Strategy for Network-on-Chip Routers. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 591–599, Manchester, UK, October 2005.

[Amor06W] A.M. Amory, K. Goossens, E.J. Marinissen, M. Lubaszewski, and F. Moraes. Wrapper Design for the Reuse of Networks-on-Chip as Test Access Mechanism. In *Proceedings of the IEEE European Test Symposium (ETS)*, pages 225–230, Southampton, UK, May 2006.

[Andr03M] A. Andriahantenaina and A. Greiner. Micro-Network for SoC : Implementation of a 32-port SPIN Network. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 1128–1129, Munich, Germany, March 2003.

[Andr03S] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C.A. Zeferino. SPIN : a Scalable, Packet Switched, On-Chip Micro-Network. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 1530–1591, Munich, Germany, March 2003.

[Arm03A] ARM. *AMBA AXI Protocol*. ARM, 2003.

[ArmES] ARM. ARM946E-S. *www.arm.com*, 2006.

[AsyOu] The Advanced Processor Technologies Group. Asynchronous Tools. *www.cs.manchester.ac.uk/apt/async/tools/index.html*, 2007. The School of Computer Science, The University of Manchester.

[Bain01D] W.J. Bainbridge and S.B. Furber. Delay Insensitive System-on-Chip Interconnect Using 1-of-4 Data Encoding. In *Proceedings of the 7th IEEE Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 118–126, Utah, USA, March 2001.

[Bain02C] J. Bainbridge and S. Furber. CHAIN : a Delay-Insensitive Chip Area Interconnect. *IEEE Micro*, 22(5) :16–23, September-October 2002.

[Bain03D] W.J. Bainbridge, W.B. Toms, D.A. Edwards, and S.B. Furber. Delay-Insensitive, Point-to-Point Interconnect Using m-of-n Codes. In *Proceedings of the 9th IEEE Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 132–140, Vancouver, BC, Canada, May 2003.

[Bain04T] J. Bainbridge, L.A. Plana, and S. Furber. The Design and Test of a Smartcard Chip Using a CHAIN Self-timed Network-on-Chip. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 274–279, February 2004.

[Beer92S] P.A. Beerel and T.H.Y. Meng. Semi-Modularity and Testability of Speed-Independent Circuits. *Integration, The VLSI Journal*, 13(3) :301–322, September 1992.

[Beig05A] E. Beigné, F. Clermidy, P. Vivet, A Clouard, and M. Renaudin. An Asynchronous NoC Architecture Providing Low Latency Service and its Multi-level Design Framework. In *Proceedings of the 11th IEEE Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 54–63, New York, USA, March 2005.

[Beig06A] E. Beigné and et al. *ALPIN : General Architecture*. CEA-LETI/ DCIS/ SCME, 2006. Internal report.

[Beig06D] E. Beigné and P. Vivet. Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture. In *Proceedings of the 12th IEEE Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 54–63, Grenoble, France, March 2006.

[Beni01P] L. Benini and G. De Michel. Powering Networks on Chips. In *Proceedings of the 14th IEEE/ACM Int'l Symposium on Systems Synthesis (ISSS)*, pages 33–38, Quebec, Canada, September 2001.

[Beni02N] L. Benini and G. De Michel. Networks on Chips : A New SoC Paradigm. *IEEE Computer Journal*, 35(1) :70–78, January 2002.

[Beno75A] N. Benowitz, D.F. Calhoun, G.E. Alderson, J.E. Bauer, and C.T. Joeckel. An Advanced Fault Isolation System for Digital Logic. *IEEE Transactions on Computers*, C-24(5) :489–497, May 1975.

[Berk02A] K. van Berkel, A. Peeters, and F. te Beest. Adding Synchronous and LSSD Modes to Asynchronous Circuits. In *Proceedings of the Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 161–170, 2002.

[Berk92B] K. van Berkel. Beware the Isochronic Fork. *Integration, the VLSI Journal*, 13 :103–128, 1992.

[Bjer05A] T. Bjerregaard and J. Sparso. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 1226–1231, Munich, Germany, March 2005.

[Bolo04Q] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNOC : QoS Architecture and Design Process for Network on Chip. *Journal of System Architecture : The Euromicro Journal*, 50(2–3) :105–128, February 2004.

[Bolo05E] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Efficient Routing in Irregular Topology NoCs. *CCIT Technical Report*, 554, September 2005.

[Brun89T] E. Brundvand and R. Sproull. Translating Concurrent Programs into Delay-Insensitive Circuits. In *Proceedings of the Int'l Conference on Computer-Aided Design (ICCAD)*, pages 262–265, 1989.

[Cart82S] W.C. Carter. Signature Testing with Guaranteed Bounds for Fault Coverage. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 75–82, Philadelphia, PA, USA, November 1982.

[Chu85A] T.A. Chu, C.K.C. Leung, and T.S. Wanuga. A Design Methodology for Concurrent VLSI Systems. In IEEE Computer Society Press, editor, *Proceedings of the Int'l Conference on Computer Design (ICCD)*, pages 407–410, 1985.

[Chu87S] T.A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-Theoretic Specifications*. Rep. MIT/LCS/TR-393. M.I.T. Tech., June 1987.

[Clar67M] W.A. Clark. Macromodular Computer Systems. In *Proceedings of the Spring Joint Computer Conference*, pages 335–336, Atlantic City, NJ, April 1967. AFIPS.

[Cota04R] É. Cota, L. Carro, F. Wagner, and M. Lubaszewski. Reusing an On-Chip Network for the Test of Core-Based Systems. *ACM Transactions on Design Automation of Electronic Systems*, 9(4) :471–499, October 2004.

[Crou99D] A.L. Crouch. *Design-for-Test for Digital IC's and Embedded Core Systems*. Printice Hall, 1999. Chapter 5.

[Dall01R] W.J. Dally and B. Towles. Route Packets, Not Wires : On-chip Interconnection Networks. In *Proceedings of the Design Automation Conference (DAC)*, pages 648–689, Las Vegas, NV, June 2001.

[Dall87D] W.J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, 36(5) :547–553, May 1987.

[Davi90S] I. David, R. Ginosar, and M. Yoeli. *Self-Timed is Seld-Diagnostic*. Department of Computer Science, University of Utah, Salt Lake City, UT, USA, 1990.

[Davi95S] I. David, R. Ginosar, and M. Yoeli. Self-timed is Self-checking. *Journal of Electronic Testing : Theory and Applications*, 6(2) :219–228, Avril 1995.

[Dobk05A] R. Dobkin, V. Vishnyakov, E. Friendman, and R. Ginosar. An Asynchronous Router for Multiple Service Levels Networks on Chip. In *Proceedings of the IEEE Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 44–53, 2005.

[Eber91A] J. Ebergen. A Formal Approach to Designing Delay-Insensitive Circuits. *Distributed Computing*, 5(3) :107–119, July 1991.

[Edwa02B] D. Edwards and A. Bardsley. Balsa : An Asynchronous Hardware Synthesis Language. *The Computer Journal*, 45(1) :12–18, 2002.

[Efth05T] A. Efthymiou, J. Bainbridge, and D. Edwards. Test Pattern Generation and Partial-Scan Methodology for an Asynchronous SoC Interconnect. *IEEE Transactions on VLSI systems*, 13(12) :1384–1393, December 2005.

[Garc98S] T.A. Garcia, A.J. Acosta, J.M. Mora, J.M. Ramos, and J.L. Huertas. Self-timed Boundary-Scan Cells for Multi-Chip Module Test. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 92–97, April 1998.

[Glas94T] C.J. Glass and Ni L.M. The Turn Model for Adaptive Routing. *Journal of the Association for Computing Machinery*, 41 :875–902, 1994.

[Goos03G] K. Goossens, J. Dielissen, J. van Meerbergen, P. Poplavko, A. Radulescu, E. Rijpkema, E. Waterlander, and P. Wielage. Guaranteeing The Quality of Services in Networks on Chip. In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, pages 61–82. Kluwer Academic Publisher, 2003.

[Grec04S] C. Grecu, P.P. Pande, A. Ivanov, and R. Saleh. Structured Interconnect Architecture : A Solution for the Non-Scalability of Bus-based SoCs. In *Proceedings of the Great Lakes Symposium VLSI*, pages 192–195, April 2004.

[Grec06B] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. BIST for Network-on-Chip Interconnection Infrastructures. In *Proceedings of the 24th IEEE VLSI Test Symposium (VTS)*, 2006.

[Guer00A] P. Guerrier and A. Greiner. A Generic Architecture for On-chip Packet-Switch Interconnections. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 250–256, March 2000.

[Gupt97I] R.K. Gupta and Y. Zorian. Introducing Core-Based System Design. *IEEE Design & Test of Computers*, 14(4) :15–25, October-November 1997.

[Hauc95A] S. Hauck. Asynchronous Design Methodologies : An Overview. *Proceedings of the IEEE*, 83(1) :69–93, Jan. 1995.

[Haye76T] J. Hayes. Transition Count Testing of Combinational Logic Circuits. *IEEE Transactions on Computers*, C-25(6) :613–620, June 1976.

[Haze92T] P.J. Hazewindus. *Testing Delay-Insensitive Circuits.* Caltech-CS-TR-92-14. California Institute of Technology, 1992. PhD Thesis.

[Hema00N] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on Chip : An Architecture for Billion Transistor Era. In *Proceedings on the IEEE NorChip Conference*, Turku, Finland, November 2000.

[Ho01T] R. Ho, K.W. Mai, and M.A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4) :490–504, April 2001.

[Hoar78C] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM 21*, 8 :666–677, August 1978.

[Hoss06A] M. Hosseinabady, A. Banaiyan, M.N. Bojnordi, and Navabi Z. A Concurrent Testing Method for NoC Switches. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Messe Munich, Germany, March 2006.

[Hulg95T] H. Hulgaard, S.M. Burns, and G. Borriello. Testing Asynchronous Circuits : A Survey. *Integration, the VLSI Journal*, 19(3) :111–131, November 1995.

[Huss07O] F.A. Hussin, T. Yoneda, and H. Fujiwara. Optimization of NoC Wrapper Design under Bandwidth and Test Time Constraints. In *Proceedings of the IEEE European Test Symposium (ETS)*, pages 35–42, Freiburg, Germany, May 2007.

[IEEE1500] IEEE 1500 Group. IEEE 1500 Standard for Embedded Core Test. *http ://grouper.ieee.org/groups/1500/.*

[IbmCc] IBM CoreConnect. CoreConect Bus Archiecture. *www-03.ibm.com/chips/products/coreconnect/index.html.*

[Ieee05I] IEEE 1500 Standard Group. *IEEE 1500 Standard Testability Method for Embedded Core-Base Integrated Circuits.* IEEE Computer Society, August 2005.

[Ivan96D] A. Ivanov. Design for Testability and Built-In Self-Test of Integrated Circuits and Systems : How These Can Add Value to Your Product. In *Proceedings of the Midwest Symposium on Circuits and Systems*, pages 712–717, 1996.

[Jant03N] A. Jantsch and H. Tenhunen (Eds). *Networks on Chip*. ISBN 1-4020-7392-5. Kluwer Academic Publisher, February 2003.

[Jerr02C] A.A. Jerraya. *Conception logique et physique des systèmes monopuces*. ISBN 2-7462-0434-7. Lavoisieur, 2002.

[Kari01O] F. Karim, A. Nguyen, S. Dey, and R. Rao. On-Chip Communication Architecture for 0C-768 Network Processors. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 678–683, Las Vegas, NV, USA, June 2001.

[Kari02A] F. Karim, A. Nguyen, and S. Dey. An Interconnect Architecture for Networking Systems-on-Chip. *IEEE Micro*, 22(5) :36–45, September-October 2002.

[Karo87I] M Karol, Hluchyj. M, and S Morgan. Input versus Output Queuing on a Space-Division Packet Switch. *IEEE Transactions on Communications*, 35(12) :1347–1356, December 1987.

[Keut00S] K. Keutzer, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-Level Design : Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systèmes*, 19(12) :1523–1543, December 2000.

[Khoc94T] A. Khoche and E. Brunvand. Testing Micropipelines. In *Proceedings of the Int'l Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 239–246, November 1994.

[Khoc95A] A. Khoche and E. Brunvand. A Partial Scan Methodology for Testing Self-Time Circuits. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 283–289, Princeton, New Jersey, USA, May 1995.

[Kim04O] J.-S. Kim, M.-S. Hwang, S. Roh, J.-Y. Lee, K. Lee, S.-J. Lee, and H.-J. Yoo. On-Chip Network Based Embedded Core Testing. In *Proceeding of IEEE Int'l SoC Conference (SOCC)*, pages 223–226, September 2004.

[Kish92O] M.A. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. On Self-Timed Behavior Verification. In *Proceedings of the ACM TAU 92*, March 1992.

[Kish94C] M.A. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware : The Theory and Practice of Self-Timed Design*. Series in Parallel Computing. John Wiley & Sons, 1994.

[Kond98H] A. Kondratyev. Hazard-Free Implementation of Speed-Independent Circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 17(9), September 1998.

[Kuma02A] S. Kumar, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. O berg, K. Tiensyrja, and A. Hemani. A Network on Chip Architecture and Design Methodology. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 105–112, Pittsburgh, PA, USA, April 2002.

[Labo04V] E. Labonne, G. Sicard, and M. Renaudin. *Dynamic Voltage Scaling and Adaptive Body Biasing Study for Asynchronous Design.* SRN : TIMA-RR–04/06-01–FR. 2004. TIMA Research Report.

[Latt07A] D. Lattard and *et al.* A Telecom Baseband Circuit based on an Asynchronous Network-on-Chip. In *Proceedings of the IEEE Int'l Solid-State Circuits Conference (ISSCC)*, pages 9–11, San Fransisco, USA, February 2007.

[Lema06A] R. Lemaire, Y. Durand, D. Lattard, and A. Jerraya. A Semi-distributed Control System for Application Management in a NoC-based Architecture. In *Proceedings of the 24th Norchip Conference*, pages 175–178, 2006.

[Lian00A] J. Liang, S. Swaminathan, and R. Tessier. aSOC : A Scalable, Single-Chip Communications Architecture. In *Proceedings of the IEEE Int'l Conference on Parallel Architectures and Compilation Techniques*, pages 37–46, October 2000.

[Line04A] A. Lines. Asynchronous Interconnect for Synchronous SoC Design. *IEEE Micro*, 24(1) :32–41, January-February 2004.

[Liu05P] C. Liu, V. Iyengar, J Shi, and É Cota. Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 349–354, May 2005.

[Liu06R] C. Liu, Z. Link, and D.K. Pradhan. Reuse-Based Test Access and Integrated Test Scheduling for Network-on-Chip. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Messe Munich, Germany, March 2006.

[Mall99A] W. Mallon, J.T. Udding, and T. Verhoeff. Analysis and Application of the XDI Model. In *Proceedings of the Int'l Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 231–242, Barcelona, Spain, April 1999.

[Mare02I] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs.

In *Proceedings of the 12th Int'l Conference on Field-Programmable Logic and Applications*, pages 795–805, September 2002.

[Mari98A] E.J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters. A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 284–293, October 1998.

[Mart06A] A.J. Martin and M. Nystrom. Asynchronous Techniques for System-on-Chip Design. *Proceedings of the IEEE*, 94(6) :1089–1120, June 2006.

[Mart86C] A.J. Martin. Compiling Communication Processes into Delay-Insensitive VLSI Circuits. *Distributed Computing*, 1(4) :226–234, 1986.

[Mart90P] A.J. Martin. Programming in VLSI : From Communicating Processes to Delay-Insensitive Circuits. In C.A.R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[Mart90T] A.J. Martin. The Limitations to Delay Insensitive in Asynchronous Circuits. In *Proceedings of the* 6^{th} *MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[Mart91T] A.J. Martin and P.J. Hazewindus. Testing delay-insensitive circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI*, pages 118–132. MIT Press, 1991.

[Mart93S] A.J. Martin. Synthesis of Asynchronous VLSI Circuits. In California Institute of Technology, editor, *Caltech-CS-TR-93-28*, 1993.

[Matrice] IST. MATRICE : MC-CDMA Transmission Techniques for Integrated Broadband Cellular Systems. *www.ist-matrice.org*, 2006.

[MenGra] Mentor. Mentor Graphics. *www.mentor.com*.

[Mill04T] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip. In *Proceedings of the 17th Int'l Conference on VLSI Design (VLSID)*, pages 693–696, 2004.

[Moln85S] C.E. Molnar, T.P. Fang, and F.U. Rosenberg. Synthesis of Delay-Insensitive Modules. In *Chapel Hill Conference on VLSI*, pages 67–85. Computer Science Press, 1985.

[Mora04H] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost. HERMES : an Infrastructure for Low Area Overhead Packet-Switching Networks on Chip. *Integration, the VLSI Journal*, 38(1) :69–93, October 2004.

[Mour00B] S. Mourad and Y. Zorian. Built-In Self-Test. In *Principles of Testing Electronic Systems*, chapter 11, pages 261–293. Wiley Inter-Science, 2000.

[Mour00O] S. Mourad and Y. Zorian. Overview of Testing. In *Principles of Testing Electronic Systems*, chapter 1, pages 3–25. Wiley Inter-Science, 2000.

[Mull59A] D.E. Muller and W. S. Bartky. A Theory of Asynchronous Circuits. In Harvard University Press, editor, *Proceedings of the Int'l Symposium on the Theory of Switching*, pages 204–243, 1959.

[Mura89P] T. Murata. Petri Nets : Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4) :541–580, 1989.

[Nahv04I] M. Nahvi and A. Ivanov. Indirect Test Architecture for SoC Testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7) :1128–1142, July 2004.

[OSCI] OSCI. Open SystemC Initiative. *www.systemc.org*.

[Ocp03O] OCP-IP. Open Core Protocol Specification. *www.ocpip.org*, September 2003. Version 2.0.

[OcpIpSo] OCPIP white paper. The Importance of Sockets on SoC Design. *www.ocpip.org/socket/whitepapers/*.

[Page95D] S. Pagey, A. Khoche, and E. Brunvand. DfT for Fast Testing of Self-Timed Control Circuits. In *Proceedings of the Asian Test Symposium (ATS)*, pages 382–386, 1995.

[Pand05D] P.P. Pande, C. Grecu, A. Ivanov, R. Saleh, and G. De-Micheli. Design, Synthesis, and Test of Networks on Chip. *IEEE Design & Test of Computers*, pages 404–413, September-October 2005.

[Pand05P] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance Evaluation and Design Trade-offs for Network-on-Chip Interconnect Architectures. *IEEE Transactions on Computers*, 54(8) :1025–1040, August 2005.

[Past98S] E. Pastor, J. Cortadella, A. Kondratyev, and O. Roig. Structural Methods for Synthesis of Speed-Independent Circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 17(11), November 1998.

[Peet06H] A. Peeters and M. de Wit. *Haste Manual, Version 3.0*. Handshake Solutions, 2006.

[Petl95S] O.A. Petlin and S.B. Furber. Scan Testing of Micropipelines. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 296–301, Mai 1995.

[Phil02D] Philips Semiconductors. Device Transaction Level (DTL) Protocol Specification. *Version 2.2*, July 2002.

[Pitt84T] J.S. Pittmant and B.C. Bruce. Test Logic Economic Considerations in a Commercial VLSI Chip Environment. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, 1984.

[Radu03C] A. Radulescu and K. Goossens. Communication Services for Network on Chip. In *Domain-Specific Processors : Systems, Architectures, Modeling, and Simulation*, ISBN : 0-8247-4711-9, pages 275–299. Marcel Dekker, 2003.

[Rijp01A] E. Rijpkema, K. Goossens, and P. Wielage. A Router Architecture for Networks on Silicon. In *Proceedings of 2nd Workshop on Embedded Systems*, pages 181–188, November 2001.

[Rijp03T] E. Rijpkema, K. Goossen, A. Radulescu, J. Dielissen, P. van Meerbergen, J. Wielage, and E. Waterlander. Trade Offs in the Design of a Router with both Guaranteed and Best-Effort Services for Networks on Chip. *IEE Proceedings on Computers and Digital Techniques*, 150(5) :294–302, September 2003.

[Ronc94P] M. Roncken. Partial Scan Test for Asynchronous Circuits Illustrated on a DCC Error Corrector. In *Proceedings of the Int'l Symposium on Advanced Research in Asynchronous Circuits and Design (ASYNC)*, pages 247–256, November 1994.

[Ronc99D] M. Roncken. Defect-Oriented Testability for Asynchronous ICs. *Proceedings of the IEEE*, 87(2) :363–375, 1999.

[Rose88Q] F.U. Rosenberg, C.E. Molnar, T.J. Chaney, and T.P. Fang. Q-Modules : Internaly Clocked Delay-Insensitive Modules. *IEEE Transaction on Computers*, 37 :1005–1018, September 1988.

[Saas02I] I. Saastamoinen, S. Tortosa, and J. Nurmi. Interconnect IP Node for Future System-on-Chip Designs. In *Proceedings of the 1st Int'l Workshop on Electronic Design, Test and Applications (DELTA)*, pages 116–122, Christchurch, New Zealand, January 2002.

[Saas03B] I. Saastamoinen, M. Alho, and J. Nurmi. Buffer Implementation for Proteo Network-on-Chip. In *Proceedings of the Int'l Symposium on Circuits and Systems (ISCAS)*, pages II.113–II.116, May 2003.

[Sath03D] S. Sathe, D. Wiklund, and D. Liu. Design of a Switching Node (Router) for On-Chip Networks. In *Proceeding of the 5th Int'l Conference on ASIC*, pages 75–78, Beijing, China, October 2003.

[Saxe93D] J. Saxena and D.K. Pradhan. Design for Testability of Asynchronous Sequential Circuits. In *Proceedings of the Int'l Conference on Computer Design (ICCD)*, pages 518–522, October 1993.

[Spar01P] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design – A Systems Perspective*. ISBN : 0-792-37613-7. Kluwer Academic Publisher, December 2001. *Part I : Tutorial*.

[Suth89M] I. Sutherland. Micropipelines. *Communication of the ACM*, 32(6) :720–738, June 1989.

[Thon06N] Y. Thonnart and et al. *Network-on-Chip Communication Model*. CEA-LETI/DCIS/SCME, 2006. Internal report.

[Tran06A] X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. A DFT Architecture for Asynchronous Networks-on-Chip. In *Proceedings of the 11th IEEE European Test Symposium (ETS)*, pages 219–224, Southampton, UK, May 2006.

[Tran06D] X.-T. Tran, V. Beroulle, J. Durupt, C. Robach, and F. Bertrand. Design-for-Test of Asynchronous Networks-on-Chip. In *Proceedings of the 9th IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems (DDECS'06)*, pages 163–167, Prague, Czech, April 2006.

[Tran07H] X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. How to Implement an Asynchronous Test Wrapper for Network-on-Chip Nodes. In *Informal Proceedings of the 12th IEEE European Test Symposium (ETS)*, pages 29–34, Freiburg, Germany, May 2007.

[Tran07I] X.-T. Tran, J. Durupt, Y. Thonnart, F. Bertrand, V. Beroulle, and C. Robach. Implementation of a Design-for-Test Architecture for Asynchronous Networks-on-Chip. In *Proceedings of the first ACM/IEEE Int'l Symposium on Networks-on-Chips (NOCS)*, pages 216–216, New Jersey, USA, May 2007.

[Tran08A] X.-T. Tran, Y. Thonnart, J. Durupt, V. Beroulle, and C. Robach. A Design-for-Test Implementation of an Asynchronous Network-on-Chip Architecture and its Associated Test Pattern Generation and Application. In *Proceedings of the 2nd ACM/IEEE Int'l Symposium on Networks-on-Chips (NOCS)*, New Type Upon, UK, April 2008.

[Tris80I] E. Trischler. Incomplete Scan Path with an Automatic Test Generation Methodology. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 153–162, 1980.

[Ubar03T] R. Ubar and J Raik. Testing Strategies for Network on Chip. In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, chapter 7, pages 131–152. Kluwer Academic Publisher, 2003.

[Uddi86A] J.T. Udding. A Formal Model for Defining and Classifying Delay-Insensitive Circuits. *Distributed Computing*, 1(4) :197–204, 1986.

[Vang07A] S. Vangal and *et al.* An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *Proceedings of the IEEE Int'l Solid-State Circuits Conference (ISSCC)*, pages 98–99, San Fransisco, USA, February 2007.

[Verh88D] Tom Verhoeff. Delay-Insensitive Codes – An Overview. *Distributed Computing (Springer)*, 3(1) :1–8, 1988.

[Verm03B] B. Vermeulent, J. Dielissen, K. Goossens, and C. Ciordas. Bringing Communication Networks on a Chip : Test and Verification Implications. *IEEE Communication Magazine*, pages 74–81, September 2003.

[Virtex] Xilinx. Virtex-4 Multi-Platform FPGA. *www.xilinx.com*, 2006.

[Wei97T] S. Wei, P.K. Nag, R.D. Blanton, A. Gattiker, and W. Maly. To DFT or not to DFT. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 557–566, November 1997.

[Wiel02N] P. Wielage and K. Goossens. Networks on Silicon : Blessing or Nightmare ? In *Proceedings of the Euromicro Symposium on Digital System Design (DSD)*, pages 196–200, Dortmund, Germany, September 2002.

[Wikl03S] D. Wiklund and D. Liu. SoCBUS : Switched Network on Chip for Hard Real Time Embedded Systems. In *Proceedings of the Int'l Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003.

[Zefe02A] C.A. Zeferino, M.E. Kreutz, L. Carro, and A.A. Susin. A Study on Communication Issues for Systems-on-Chip. In *Proceedings of the 15th Symposium on Integrated Circuits and System Design*, pages 121–126, September 2002.

[Zefe03S] C.A. Zeferino and A.A. Susin. SoCIN : A Parametric and Scalable Network-on-Chip. In *Proceedings of the 16th Symposium on Integrated Circuits and System Design*, pages 169–174, September 2003.

[Zefe04R] C.A. Zeferino, M.E. Kreutz, and A.A. Susin. RASoC : A Router Soft-Core for Network-on-Chip. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, pages 198–203, February 2004.

[Zori97T] Y. Zorian. Test Requirements for Embedded Core-Based Systems and IEEE P-1500. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 191–199, Washington, DC, USA, November 1997.

[Zori98T] Y. Zorian. Test Embedded Core-Based System Chips. In *Proceedings of the IEEE Int'l Test Conference (ITC)*, pages 130–140, Washington, DC, USA, October 1998.

Liste des Abbréviations

Abréviation	Description	Définition
ABB	Adaptive Body Biasing	
ALPIN	Asynchronous Low Power Innovative NoC	
AR	Analyseur de réponse	
ATPG	Automation Test Pattern Generation	
AXI	Advanced eXtensible Interface	
BE	Best-effort	
BIST	Built-In Self-Test	
CAO	Conception Aidée par Ordinateur	
CD	Change Diagram	
CHP	Communicating Hardware Processes	
CMP	Chip Multiprocessors	
CPU	Central Processing Unit	
CSP	Communicating Sequential Processes	
CTL	Core Test Language	
CVT	Conception en Vue du Test	
DfT	Design-for-Test/Testability	
DI	Delay-Insensitive	
DR	Dual-rail ou Double-rail	
DTL	Device Transaction Level	
DSM	Deep Submicron	
DVS	Dynamic Voltage Scaling	
FAUST	Flexible Architecture of Unified System for Telecom	
FFT	Fast Fourrier Transform	
FHT	Fast Hartley Transform	
FIFO	First-in First-out	
FPGA	Field Programmable Gate Array	

Abréviation	Description	Définition
GAC	Générateur, Analyseur, Contrôleur	
GALS	Globally Asynchronous, Locally Synchronous	
GS	Guarantee Service	
GVT	Générateur de Vecteurs de Test	
HPU	Header Parsing Unit	
HSE	HandShake Expansion	
IP	Intellectual Property	
IP-TW	IP Test Wrapper	
ISO	International Standard Organization	
ITC	Input Test Cells	
ITRS	International Technology Roadmap for Semiconductors	
LFSR	Linear Feedback Shift Register	
MG	Marked Graph	
MR	Multi-rails	
NI	Network Interface	
NoC	Network-on-Chip	
NRZ	Not Return to Zero	
OCP	Open Core Protocole	
OSI	Open System Interconnection	
OTC	Output Test Cell	
QDI	Quasi-Delay Insensitive	
QoS	Quality of Service	
RTZ	Return to Zero	
SAF	Store-and-Forward	
SI	Speed-Independent	
SoC	System-on-Chip	
SR	Set-Reset	
STG	Signal Transition Graphs	
TAM	Test Access Mechanism	
TAST	TIMA Asynchronous Synthesis Tool	
TCF	Test Configuration Frame	
TTTC	Test Technology Technical Conseil	
TW	Test Wrapper	
UDL	User-Defined-Logic	
VC	Virtual Channel	
VCD	Value Change Dump	
VCPIQ	Virtual Channel Priority Input Queuing	
VCT	Virtual-Cut-Throught	

Abréviation	Description	Définition
VOQ	Virtual Output Queuing	
WCM	Wrapper Control Module	
WH	Wormhole	
WIP	Wrapper Interface Port	

Table des figures

Liste des tableaux

Index

TITRE EN FRANÇAIS

Méthode de Test et Conception en Vue du Test pour les Réseaux sur Puce Asynchrones : Application au Réseau ANOC

RESUME

Les réseaux sur puce (*NoC : Network on Chip*) et les architectures GALS (Globalement Asynchrone – Localement Synchrone) sont deux nouveaux paradigmes de communication pour les systèmes sur puce (*SoC : System on Chip*). Ces paradigmes ont conduit à la création de réseaux sur puce asynchrones. Cependant, faute de méthodologies et d'outils de test adaptés, le test de production des réseaux sur puce asynchrones constitue un grand défi pour la mise sur le marché de ces systèmes.

L'objectif de cette thèse est de proposer une nouvelle méthode de test pour les réseaux sur puce asynchrones. Afin de faciliter le test de l'infrastructure du réseau, nous avons tout d'abord proposé une architecture DfT (*Design-for-Test)* dans laquelle chaque routeur du réseau est entouré d'un wrapper de test asynchrone qui améliore sa contrôlabilité et son observabilité. Cette architecture DfT a été modélisée, implémentée en logique asynchrone QDI (*Quasi-Delay Insensitive*), et validée avec un réseau sur puce asynchrone ANOC développée au CEA-LETI. La génération des vecteurs de test a été alors faite en analysant les fonctionnalités et l'implémentation structurelle du routeur et de ses interconnexions. Ensuite, nous avons également introduit une stratégie pour tester un réseau complet. La méthode de test complète développée dans cette thèse permet une couverture de faute de 99,86% pour le réseau ANOC en utilisant un modèle de faute de collage simple.

MOTS CLES

Réseau sur puce ; NoC ; Système sur Puce ; SoC ; Conception en Vue du Test ; CVT ; Méthodologie de test ; Testabilité ; Architecture de communication ; Globalement Asynchrone – Localement Synchrone ; GALS ; Logique asynchrone QDI.

TITRE EN ANGLAIS

Test Method and Design-for-Test of Asynchronous Networks-on-Chip: Application to ANOC Network

ABSTRACT

Networks-on-Chip (NoCs) are emerging as a new on-chip communication paradigm for large complex Systems-on-Chip, together with the Globally Asynchronous – Locally Synchronous (GALS) paradigm, which lead to asynchronous NoCs. Nevertheless, manufacturing test is a big challenge for asynchronous NoCs before they can be brought to market due to a lack of testing methodology and support.

The objective of this thesis is to propose a novel testing method for asynchronous NoCs. In this method, to ease the test of the network infrastructure, we have developed a Design-for-Test (DfT) architecture, in which each network router is surrounded by an asynchronous test wrapper in order to improve the controllability and the observability of the routers. This DfT architecture has been designed, implemented in Quasi-Delay Insensitive (QDI) asynchronous logic, and validated with ANOC, an asynchronous NoC architecture developed at the CEA-LETI. The corresponding test pattern generation is done by analyzing both functionalities and structural implementation of network routers and links. We have also introduced a complete testing strategy to test the whole network architecture. With the generated test patterns, the testing method presents high fault coverage (99.86%) for the ANOC architecture using a single stuck-at fault model.

KEY WORDS

Network-on-Chip; NoC; System-on-Chip; SoC; Design-for-Test; DfT; Testing methodology; Testability; on-chip communication; Globally Asynchronous – Locally Synchronous; GALS; QDI asynchronous logics.

SPECIALITE : MICRO NANO ELECTRONIQUE