Conception de réseaux de communication sur puce asynchrones : application aux architectures GALS

J. Quartana

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

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

THESE

pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité : Micro et Nanoélectronique

préparée au laboratoire **TIMA** dans le cadre de
l'Ecole Doctorale d'« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »

présentée et soutenue publiquement

par
Jérôme QUARTANA

Le 20 décembre 2004

Titre :

CONCEPTION DE RESEAUX DE COMMUNICATION SUR PUCE ASYNCHRONES : APPLICATION AUX ARCHITECTURES GALS

Directeur de Thèse : Marc RENAUDIN

JURY

M. Michel AUGUIN	, Président
M. Paolo IENNE	, Rapporteur
M. Lionel TORRES	, Rapporteur
M. Didier MOULIN	, Examinateur
M. Marc RENAUDIN	, Directeur de thèse
M. Laurent FESQUET	, Codirecteur de thèse

Résumé

Cette thèse porte sur l'étude d'architectures de communication sans horloge pour la conception de réseaux de communication asynchrones appliqués aux systèmes globalement asynchrones et localement synchrones. Elle s'intègre également dans le cadre du développement de l'outil de conception automatique de circuits asynchrones TAST (TIMA Asynchronous Synthesis Tool).

L'importance des besoins de communication au sein des systèmes intégrés modernes fait du réseau d'interconnexion un acteur majeur de la complexité et des performances de ces systèmes. Parmi les nombreuses méthodologies de synchronisation de systèmes existantes, nous montrons l'intérêt de choisir un réseau d'interconnexion sans horloge pour la communication des systèmes globalement asynchrones et localement synchrones.

Nous développons dans ce manuscrit une méthodologie de conception d'un réseau d'interconnexion qui utilise les propriétés d'excellente modularité des circuits sans horloge. Nous découpons la construction de nos réseaux sur silicium asynchrones en quatre modules majeurs : arbitrage, transport, routage et synchronisation. L'objectif de ce découpage simple est de permettre à terme la synthèse automatique d'arbitres et de réseaux de communication sans horloge dans l'outil de conception TAST. Les modules du réseau sont spécifiés en CHP, un langage de modélisation de haut niveau adapté à la synthèse de circuits asynchrones. A travers ces modélisations, nous mettons en relief l'importance des problèmes d'arbitrage et de synchronisation entre les blocs du système. Nous présentons un système de communication qui illustre cette méthodologie de construction de réseau par assemblage de modules et son degré d'automatisation actuel.

Mots-clefs

Circuits asynchrones, architectures Globalement Asynchrones et Localement Synchrones, réseaux de communication sur silicium, protocoles et interfaces de communication, arbitrage, synchronisation, synthèse automatique de circuits asynchrones, modélisation en langage de haut niveau.

Abstract

This thesis tackles a research on self-timed communication architectures for the design of asynchronous Networks-on-Chip (NoCs), dedicated to Globally-Asynchronous Locally-Synchronous (GALS) systems. This study also takes part within the framework of the TIMA Asynchronous Synthesis Tool (TAST) suite.

The needs for communication within modern Systems-on-Chip (SoCs) turns the interconnect network into a major contributor of complexity and performances of these systems. Among many existing methodologies adressing the problem of synchronization within SoCs, this work demonstrates the advantages to choose an asynchronous interconnect network for the communication of GALS systems.

This manuscript puts forward a design methodology for interconnect networks which uses the modularity property of asynchronous circuits. We cut out the construction of our asynchronous NoCs in four major modules: arbitration, transport, routing and synchronization. The aim of this classification is to help the automatic synthesis of arbiters and of asynchronous interconnect networks using TAST. The basic modules of these communication networks are specified in CHP (Communicating Hardware Processes) language. CHP is a high-level modelling language adapted to describe and to synthesize asynchronous circuits. Through these modelling, the proposed methodology throws into relief arbitration and synchronization problems between concurrent elements of the system. Also, a communication system case-study is presented to illustrate the asynchronous NoC design methodology and its current automation level.

Keywords

Asynchronous circuits, Globally-Asynchronous Locally-Synchronous (GALS) architectures, Networks-on-Chip (NoC), communication protocols and interfaces, arbitration, synchronization, automatic synthesis of asynchronous circuits, high-level language modelling.

Remerciements

Bon, j'en ai du monde à remercier, en plus je suis bavard, alors c'est parti !!

Les travaux de cette thèse ont été réalisés au sein du laboratoire TIMA, sur le site Viallet de l'Institut National Polytechnique de Grenoble. Je remercie son directeur Bernard Courtois pour son accueil au sein de son laboratoire. Mes remerciements également à monsieur Pierre Gentil, directeur de l'Ecole Doctorale EEATS, de m'avoir accueilli et conseillé pour démarrer ce chemin de croix que fut cette thèse.

Je tiens à exprimer ici toute ma gratitude, mon profond respect et mon admiration pour les trois personnes qui ont souffert à encadrer mes travaux. Ces personnes m'ont offert une confiance, un soutien et une compréhension qui sont pour moi exemplaires.

Il s'agit de Marc Renaudin, mon directeur de Thèse, qui a accepté en 2000 de me prendre comme stagiaire à la dernière minute, ignorant ce qui l'attendait. Et il a récidivé en acceptant de me prendre ensuite en thèse !

Didier Moulin fut mon encadrant au cours de ma période passée chez STMicroelectronics. J'ai pu apprécié avec quelle finesse politique il a su manœuvrer pour préserver son équipe au sein des remous de l'entreprise. C'est un manager au meilleur sens du terme. Je regrette de n'avoir pu répondre aux attentes professionnelles qu'il plaçait dans ces travaux de thèse.

Enfin Laurent Fesquet fut mon co-encadrant de thèse. En plus de m'avoir nourri, conseillé et assisté avec une immense disponibilité, il m'a offert une amitié discrète et sincère qui m'est précieuse. Je tiens à saluer ici sa femme et ses craquantes Iris & Clara.

Je vous remercie tous les trois du temps et de l'intérêt que vous m'avez accordé tout au long de ces années. Quel que soit mon avenir professionnel, j'espère être à la hauteur de vos qualités humaines.

Je remercie messieurs Lionel Torres, professeur au laboratoire LIRMM de Montpellier, et Paolo Ienne, professeur et directeur du département d'Architecture des Processeurs à l'EPFL de Lauzanne, qui ont accepté d'être les rapporteurs de mes travaux de thèse. Ma gratitude à monsieur Ienne pour les circonstances précipitées dans lesquelles il a dû rapporter ces travaux.

Merci à Michel Auguin, Professeur au Laboratoire I3S de Nice, de m'avoir fait l'honneur de présider mon jury de thèse.

Je remercie ces personnes pour l'intérêt et l'analyse avisée qu'ils ont porté sur mes travaux. Pour reprendre mon mentor Jean-Baptiste (qui a fait beaucoup mieux en l'occurrence) : « Je suis particulièrement redevable envers tous les membres du jury pour avoir accepté de se déplacer et de se réunir à une date si tardive dans l'année et si proche des fêtes de Noël ».

Je remercie Pierre Delerue, Jean-Baptiste Rigaud, Anh-Vu Din-Duc, Salim Renane, Kamel Slimani et Arnaud Baixas pour leur appréciable et agréable collaboration professionnelle et tout ce que j'ai appris auprès d'eux. Je remercie aussi Pascale pour sa contribution à ma plus grande découverte scientifique personnelle au cours de la thèse. Cette découverte récente (le 1er octobre 2004) a nécessité un matériel de pointe : un ordinateur de bureau Dell tip top classe avec tous ses périphériques et un téléphone portable Nokia très vieille génération avec son kit Piéton (oreillette et micro). Lorsque je converse avec ledit kit et que je bouge le cordon ombilical qui relie le téléphone au micro et à l'oreillette, le curseur de la souris sur l'écran de l'ordinateur se déplace aussi. Bon, toujours vers la droite jusqu'à présent, mais je poursuis l'expérience.

Enfin ces remerciements s'adressent à tout les secrétaires et le personnel des ressources humaines de FranceTelecomR&D et de STmicrolectronics tant à Crolles qu'à Grenoble, ainsi qu'au personnel du CIME, aux administrateurs réseau et aux secrétaires du laboratoire qui ont tous toujours été d'une grande disponibilité et d'une aide précieuse.

Au cours de ces années passées à Grenoble, j'ai rencontré de nombreuses personnes passionnantes qui m'ont enrichi, soutenu et diverti, me permettant ainsi d'arriver au bout de cette thèse, qui il faut l'avouer ressembla parfois aux douze travaux. D'emblée je les remercie tous d'avoir été présents dans les moments difficiles.
De mon passage chez FranceTelecomR&D et STmicroelectronics je conserve des amitiés profondes. Pour Fanny et Philippe, les heureux possesseurs de la ruine la plus chère du Grésivaudan, même pas classée monument historique. Pour Joumana et son mari Clovis (roi des) Francis, les libanais chers à mon cœur. Merci de nous avoir accueilli si chaleureusement en votre beau pays. Je vous souhaite tout le bonheur possible et tout le courage pour construire votre avenir. Pour Helder l'européen en soif d'aventure, toujours à la recherche de LA montre technique GPS-cardio-alti-boussole-thermo-téléphone-frigoabier-chauffmergez. En plus si elle donne l'heure, c'est géant.
Je n'oublie pas bien sûr ma chère Carolynn, avec son délicieux accent et sa géniale recette de salade aux algues qui m'a valu tant de succès. J'espère que nous continuerons à réaliser de merveilleux réveillons. Et puis Hélène, de compagnie toujours si douce et agréable. Bonne route avec ton Jessy !
J'en profite ici pour saluer et remercier tous les membres de la « ST team » pour leur joyeuse camaraderie : Pierre et sa patience infinie (il a marché, mon bus !), Philippe, Thierry, Joël, Silvia et tous les autres.
De mon long séjour au laboratoire TIMA je conserve également des amitiés sincères. Pour Jean-Baptiste « 1000W », mon mentor. May the Zen attitude be with you même si ce n'est pas gagné. Merci pour ton immense soutien aux heures sombres autant que pour ta connivence aux jours clairs. Pour Mariana « beau sourire » et Joao « champignon de surf attitude » (j'ai pas dit de surf), le seul brésilien à connaître en français plus de blagues que n'importe quel autochtone du coin (pour un brésilien, la France, c'est un gros bled). Mes hommages au Grands Anciens, Damien « jus d'ananas » (le seul homme à faire du roller dans les calanque) et Lovic le fantastique. Encore toutes mes excuses pour Dongeon&Dragon's. Mes respects à maître Anh Vu, coupeur de têtes de cobra en son lointain Vietnâm et accessoirement fournisseur de logiciel à déboguer.
Je salue ici mes compagnons de route, de galère et de promotion, Kamel « barres d'haltères », Amine le philosophe, Manu le tombeur-frimeur (ça dépend de la bonne fortune) qui fut d'un inestimable secours ces derniers mois grâce à ses innombrables « templates ». Une pensée particulière pour les moues adorables de Dhanistha, ma si charmante voisine de bureau pendant trois ans, qui fit de moi un presque-maître Fenshui. « Mimine » et Yanick (digne successeur de Damien en blagues légères), merci pour les ballades en moto.
Je n'oublie pas Fabien « peluche lover », 100kg (peu ou prou) de franche rigolade, d'amour de la vie et d'immense générosité, si bien assorti à sa belle Juliette. Pour la collec' de BDs, respect.
Une pensée d'affection sincère pour Lobna et son amitié, merci à toi et à ta famille de m'avoir accueilli à bras ouverts pour me permettre de retrouver la terre de mes ancêtres. C'est un grand geste que je n'oublie pas.
Je salue tous les autres membres du groupe CIS et plus largement du laboratoire TIMA : Sophie « boîte à bonbons », la meilleure organisatrice de soirées auxquelles je ne participe jamais, Estelle toujours réconfortante, Yann l'homme aux mille savoirs, Bertrand le roi de la piste, Freddy et sa généalogie remarquable, Yannick, David, Livier et Cédric (alors, mariés ou pas ?), Aurélien, Pierre, Greg « trompette et ciboulette » et tous les autres.
Je poursuis avec joie en pensant aux amis et camarades des années lyonnaises, et leurs « pièces rapportées » (« Sexe » François, la grande famille Schuppadolphe, « Chorale » Laurent). Pierre « le chauve » (désolé, c'est de ma mère) qui fut toujours prompt à me soutenir, de même que les Pinoncely. Pif, je suis très flatté d'avoir été le mari intérimaire de la femme de mon parrain. Bienvenue parmi nous à leur petit Noah, de même qu'à l'adorable Lou-Anne, entourée de ses merveilleux parents Marine et Arnaud. Nat&Moule, je crois que c'est votre tour !!
Merci à Estelle, à Audrey, à Betty et Boop, à Lucie, à Celia, à Ariane, à Marie-Hélène et Silvia pour m'avoir rendu, chacune à sa manière, un peu de confiance en moi.
Je remercie le professeur Charnallet du temps qu'elle m'a généreusement consacré et pour m'avoir tant appris sur moi-même.
J'exprime toute ma gratitude et mon éternelle amitié à Isabelle et Sébastien qui surent m'offrir un port d'attache au milieu de la tempête.
Je tien à NE PAS remercier mes parents et mes frères pour m'avoir toujours soutenu et encouragé dans mon obstination à choisir les voies les plus tortueuses, difficiles et mal payées et à avoir cru en moi, m'obligeant ainsi à aller jusqu'au bout !
Je termine pudiquement par le co-auteur de ce manuscrit, à qui je dois tant. Pascale, je t'embrasse de tout mon cœur. Merci pour tout ce que tu m'offres de meilleur chaque jour.

Bonne route à tous, et surtout… ne tournez pas la page !

A mes parents dont la fortune est d'amour
A Gilles et Sylvain, mes mousquetaires
A Pascale, la princesse pour qui mon cœur flanche

Liste des figures

Liste des tableaux

Introduction : la problématique des communications dans les systèmes sur silicium

Contexte de l'étude : la synchronisation par communication des systèmes sur silicium

Lorsqu'il s'agit de réalisations industrielles à objectif commercial, la majorité des méthodologies de conception numériques synchrones utilisent comme méthode de synchronisation un signal d'horloge périodique, servant de référence de temps, et global à l'ensemble du système sur silicium[1]. Ce paradigme de la conception synchrone présente l'avantage de fournir un contrôle d'exécution (régulation de l'opération du système) largement éprouvé. Mais le rôle central joué par cette horloge nécessite d'investir d'importants efforts dans la conception, l'optimisation et la vérification de ces réseaux de distribution d'horloge. La

Figure 0-1 présente la croissance rapide[2] des fréquences d'horloge des microprocesseurs Intel depuis le processeur 386 cadencé à 20 MHz jusqu'au dernier Pentium 4 atteignant des vitesses d'horloge de 3 GHz en technologie 0.13 µm.

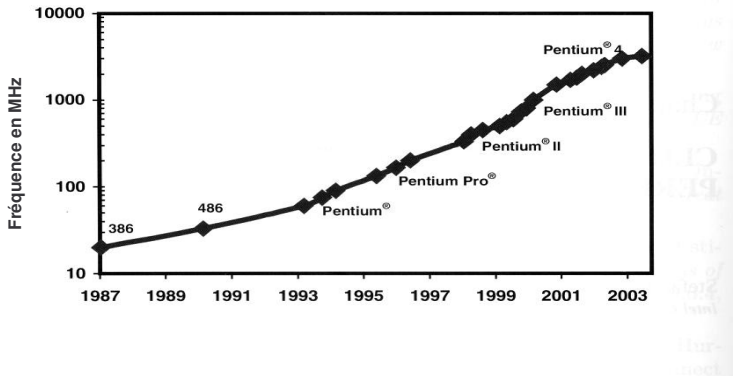


Figure 0-1 : courbe d'évolution temporelle de la fréquence des processeurs Intel

[1] Le paradigme de système sur silicium s'est imposé au milieu des années 1990. Il est étroitement lié aux concepts de conception pour la réutilisation (*Design Re-Use*) et de composant virtuel ou bloc de propriété intellectuelle (*IP* pour *Intellectual Property*). Le portail *Design & Reuse* (*D&R*), qui catalogue en véritable bibliothèque l'ensemble des composants virtuels disponibles et les outils de conception du monde de la microélectronique, a d'ailleurs vu le jour en 1997 [D&R 03].

[2] La haute performance d'un processeur est en général associée à la haute fréquence de son horloge, mais pas toujours : un processeur spécifique dédié à une application peut se révéler bien plus performant dans son domaine qu'un processeur généraliste fonctionnant pourtant à une fréquence bien plus élevée, et donc consommant davantage d'énergie.

Cependant les perspectives dressées dans les "Roadmap" [SIA 04] consacrés à la conception de systèmes intégrés utilisant des technologies submicroniques avancées, mentionnent la nécessité de résoudre le problème d'interfacer des fonctions intégrées animées par des horloges différentes. En effet, de plus en plus de systèmes sur puce cherchent à intégrer un grand nombre de fonctions de nature très différentes : logiques, analogiques, synchrones et asynchrones. La distribution d'une horloge unique, ou d'un ensemble d'horloges synchronisées entre elles, dans ce type de systèmes s'avère très difficile pour plusieurs raisons.

Du point de vue de la réalisation tout d'abord, la distribution d'une horloge à grande échelle dans un circuit pose des problèmes de temps de propagation, d'intégrité du signal, de bruit et de consommation. Du point de vue fonctionnel ensuite, la synchronisation de l'ensemble des fonctions d'un système complexe par une horloge unique complexifie de façon significative le développement des applications et constitue un frein important à une réduction des temps de développement. Le système de communication a alors pour rôle de découpler les composants du système, le rendant plus modulaire et extensible, donc plus flexible.

Cette difficulté, croissante avec la taille du circuit, de concevoir une horloge globale en la maîtrisant totalement, a remis le vent en poupe des techniques de conception des systèmes auto-séquencés (ou asynchrones, bref sans horloge) à la fin des années 1990. L'engouement fut suffisant pour que certains prédisent même l'affranchissement total de la tyrannie de l'horloge [TRI 01], tandis que les défenseurs du tout synchrone poursuivaient leur quête de la réduction du déphasage d'horloge [XAN 01].

Aujourd'hui aucun changement brutal de paradigme n'est survenu. Le tout synchrone n'est plus viable pour les systèmes sur silicium de grandes dimensions, sans toutefois que le tout asynchrone se soit imposé. La tendance est aux méthodologies hybrides, complexes à répertorier et à classer, qui apparaissent maintenant autant dans les prototypes de recherche que les réalisations industrielles, et dont le désir commun est de s'affranchir des contraintes de réalisation d'une horloge globale, afin de disposer de fréquences de fonctionnement totalement indépendantes pour les différents composants d'un système. Le consensus commun désigne l'ensemble de ces nouveaux circuits sous le label de systèmes GALS pour Globalement Asynchrones et Localement Synchrones.

De manière plus formelle, le concept de système globalement asynchrone et localement synchrone désigne un assemblage de composants intégrés sur une même puce, dont le séquencement interne de chaque composant est synchrone d'une horloge, mais ces horloges sont indépendantes. Les composants synchrones se synchronisent de manière locale au moyen de canaux et de protocoles de communication. Grâce à cette approche, l'effort de conception des horloges est fortement réduit, car local à chaque bloc synchrone, dont la performance est alors ajustable de manière indépendante des autres composants. De plus, la synchronisation de ces blocs est facilitée, grâce au potentiel de modularité des protocoles de communication. Enfin, la conception des composants de communication et des composants de calcul ou de mémoire peut être séparée.

Mais à nouveau concept, nouveaux problèmes. En apportant une solution nouvelle à la synchronisation d'un circuit, le paradigme des systèmes GALS amène également de nouveaux problèmes. La conception des composants de communication, assemblés en un réseau interconnectant l'ensemble des composants de calcul et de mémoire du système, devient le cœur

du système. En remplaçant le rôle central joué par l'horloge dans les systèmes synchrones, ce réseau d'interconnexion devient la pierre angulaire des performances du système et donc son goulot d'étranglement potentiel. De plus, la synchronisation par communication de domaines d'horloges indépendants doit être abordée avec une attention particulière pour garantir au système fiabilité et performance. Cette synchronisation aux interfaces de domaines de fonctionnement et la conception de réseaux de communication performants sont les deux défis à relever par les concepteurs de systèmes globalement asynchrones et localement synchrones.

Partenariat industriel

Cette importance du réseau de communication n'a pas échappé aux acteurs du monde de la conception en microélectronique. Ainsi la société STMicroelectronics, qui conçoit et fabrique des circuits et systèmes intégrés dans des domaines très variés pour les marchés professionnels et grand public, propose dans son portefeuille de composants le STBus, un standard d'interface de communication associé à un jeu de topologies de réseaux d'interconnexion synchrones, capable d'assembler un grand nombre de composants hétérogènes. Dans le contexte de réalisation de systèmes globalement asynchrones et localement synchrones, STMicroelectronics s'est intéressé en particulier à la technique de conception asynchrone pour la réalisation d'architectures de communication. De forts potentiels peuvent en effet être attendus de ce type de circuits : robustesse, calcul en temps moyen, faible bruit, consommation conditionnelle, forte modularité et niveau d'abstraction plus élevé.

Un partenariat a donc vu le jour avec le laboratoire TIMA (Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs), et en particulier le groupe CIS (*Concurrent Integrated Systems*), qui possède une expertise reconnue dans le domaine de la conception de circuits et systèmes asynchrones. Le groupe travaille sur les méthodologies et les outils de conception de circuits et systèmes asynchrones. Dans le cadre général de la conception de systèmes sur puce, il est très intéressé par la réalisation d'architectures globalement asynchrones et localement synchrones faisant appel à des réseaux de communication asynchrones.

Cette collaboration s'est concrétisée sous la forme de ce travail de thèse. L'objectif principal était de comparer les performances de réseaux de communication de même spécification, réalisés respectivement en technologies synchrones et asynchrones. Ces réseaux devaient être intégrés dans l'architecture du système Prepor de STMicroelectronics, une plateforme de prototypage pour les applications dédiées aux réseaux sans fil basse puissance (*WLAN* pour *Wireless Local Area Network*).

Malheureusement pour diverses raisons la réalisation de ce projet n'a pas pu aboutir. Cette collaboration fut tout de même un cadre applicatif très riche pour les travaux présentés ici.

Objectif : la conception de réseaux de communication en technologie asynchrone pour des architectures globalement asynchrones et localement synchrones

Avant de parvenir à cet objectif principal, plusieurs directions de travail ont été définies pour atteindre six objectifs identifiés au cours de la collaboration avec l'industriel STMicroelectronics.

1. Connaître les techniques de synchronisation globale d'un système sur silicium

Pour réaliser une architecture globalement asynchrone et localement synchrone, il est nécessaire de connaître les techniques de conception à la fois synchrones et asynchrones. Cette maîtrise double des techniques de conception est également indispensable pour la réalisation et la comparaison de performances des réseaux d'interconnexion de séquencement interne synchrone et asynchrone. Le premier objectif est donc la connaissance de l'ensemble des techniques de synchronisation d'un système sur silicium, depuis la synchronisation par une horloge globale jusqu'à l'utilisation de FIFOs asynchrones, en passant par les techniques à horloge interruptible.

2. Identifier les critères de conception d'un réseau sur silicium

Nous avons cherché à identifier tous les paramètres de conception qui caractérisent le réseau d'interconnexion d'un système sur silicium, qu'il soit synchrone ou asynchrone, de manière à pouvoir répondre au mieux aux exigences du système en termes de performances de communication, de fiabilité, de flexibilité et de coûts. Cet objectif de spécification est essentiel pour une architecture de communication devenue le cœur de la conception des systèmes sur silicium.

3. Identifier et étudier les problèmes de synchronisation et d'arbitrage

Après le réseau de communication, le deuxième enjeu majeur pour réussir la conception efficace d'une architecture GALS est la synchronisation de composants possédant des vitesses de fonctionnement différentes. Identifier et étudier les problèmes de synchronisation et d'arbitrage au niveau le plus fin des protocoles de niveau signal fut le troisième objectif de ces travaux.

4. Déterminer les atouts potentiels de la conception asynchrone

Il s'agit de déterminer les atouts potentiels que peut apporter la conception asynchrone pour répondre à cette exigence : réaliser la synchronisation par communication d'un système GALS de manière performante, flexible et robuste. Par exemple, les protocoles auto séquencés offrent une grande robustesse et donc une grande fiabilité de communication. De même, la localité des communications rend les architectures asynchrones très modulaires et facilement extensibles.

5. Dégager une méthodologie de conception

Un objectif majeur pour un industriel comme STMicroelectronics est de réduire les temps de cycles de conception des circuits. Pour cela, de nombreuses méthodologies de conception en vue de la réutilisation sont mises en place. Dans ce cadre, et à partir de la synthèse des études précédentes, notre objectif était de dégager une méthodologie de conception de réseaux de communication permettant l'assemblage le plus modulaire possible des composants de communication constitutifs du réseau.

6. Formaliser en vue de la synthèse automatique

Le laboratoire TIMA développe un environnement de conception dédié à la synthèse de circuits asynchrones. Dans la perspective d'aboutir à la synthèse automatique de réseaux sur silicium asynchrones et leur application aux systèmes GALS, l'objectif ici était en particulier d'identifier et de formaliser les points de choix non-déterministe au sein d'un réseau de communication, à savoir :

a) les arbitres en environnement globalement asynchrone, devant échantillonner des requêtes mutuellement asynchrones ;

b) les interfaces de synchronisation entre domaines de fonctionnement de vitesses différentes, dont au moins un domaine est synchrone.

Contributions

La première contribution de ces travaux est l'important travail bibliographique dont l'objectif était double : identifier les problèmes de conception spécifiques aux réseaux sur silicium et renforcer les connaissances dans ce domaine du groupe de recherche CIS ainsi que des collaborateurs de l'équipe de conception de circuits de STMicroelectronics. La thèse de Laurent Fesquet sur les commutateurs optiques dans les architectures multiprocesseurs [FESQ 97] fut au laboratoire une précieuse base de départ de ces travaux, ainsi que les compétences en architecture des systèmes de Joël Cambonie chez STMicroelectronics.

Nous avons ainsi identifié et spécifié les critères de conception d'un réseau sur silicium, en particulier en développant les critères qualitatifs de flexibilité et de variété de services, ainsi que l'étude de la fiabilité des protocoles de synchronisation (ou protocoles de niveau signal).

Nous avons mené une étude qualitative comparative de ces coûts de synchronisation au sein d'une architecture globalement asynchrone et localement synchrone. Tout d'abord aux interfaces entre les composants du système, pour un réseau de communication synchrone et pour un réseau de communication sans horloge, puis entre des arbitres synchrones et asynchrones en environnement asynchrone. Nous montrons que le réseau et l'arbitre asynchrone permettent des synchronisations plus fiables, plus performantes et moins coûteuses.

Ces deux points de synchronisation entre domaines de fonctionnement à des vitesses différentes, dont au moins un domaine est synchrone, et d'arbitrage en environnement asynchrone constituent d'ailleurs les deux sources du potentiel de non-déterminisme d'une architecture globalement asynchrone et localement synchrone, rendant sa validation difficile. Ce

sont également deux sources de risque de métastabilité et donc de risque d'erreur dans le fonctionnement du système.

Nous proposons une méthodologie de conception d'une architecture globalement asynchrone et localement synchrone en utilisant un réseau de communication sans horloge modulaire, flexible (en terme de services, d'extensibilité et d'interfaçage), robuste et performant. Cette méthode repose d'une part sur la modélisation en langage CHP (*Communicating Hardware Processes*) de haut niveau d'abstraction, de problèmes très fins de synchronisation, et d'autre part sur l'utilisation de composants autonomes construits pour répondre chacun à des critères de spécification bien déterminés du réseau de communication.

Au sein de ce réseau nous proposons deux composants pour améliorer la qualité des circuits générateurs d'un risque de métastabilité. D'une part, une classe d'arbitres asynchrones à échantillonnage parallèle parfaitement fiables qui réalisent des fonctions de priorité indépendantes de l'ordre d'arrivée des requêtes. Ces arbitres offrent une excellente modularité, sont aisément extensibles et améliorent la latence de réponse aux requêtes.

D'autre part, nous améliorons une architecture de FIFO circulaire asynchrone existante pour réaliser des interfaces de synchronisation mixtes synchrones/asynchrones fiables, à faible latence ou faible consommation.

Plan du manuscrit

Les travaux et contributions réalisés en vue d'atteindre les objectifs précédemment énoncés, se retrouvent dans ce manuscrit, agencés de la manière suivante.

Le chapitre 1 présente les différentes techniques de synchronisation par horloge et les problèmes auxquels elles sont confrontées. Il introduit ensuite le paradigme des architectures globalement asynchrone et localement synchrone, ses avantages (modularité, localité, effort de conception réduit) et l'attention particulière qu'il faut porter aux problèmes de synchronisation entre les composants et au réseau de communication. Enfin, les atouts potentiels – mais aussi les inconvénients – apportés par l'utilisation, dans une architecture globalement asynchrone et localement synchrone, d'un réseau de communication sans horloge sont développés.

Le chapitre 2 est conséquent. Il commence par passer en revue l’ensemble des paramètres qui caractérisent le réseau d’interconnexion d’un système sur silicium, qu'il soit synchrone ou asynchrone, avant d'étudier plus en détails trois caractéristiques fondamentales de ces réseaux : le degré de flexibilité d'un réseau, la fiabilité et la qualité de ses protocoles de communication, et les différentes familles de topologies existantes. La caractérisation de ces paramètres est illustrée à travers la présentation de réalisations de réseaux sur silicium industriels et universitaires représentatifs.

Le chapitre 3 introduit rapidement les techniques de conception asynchrone, puis le formalisme de modélisation de circuits asynchrones que nous utilisons en entrée de l'outil de conception automatique de circuits asynchrones TAST (*TIMA Asynchronous Synthesis Tool*). Ce formalisme de modélisation est basé sur le langage CHP (*Communicating Hardware Processes*), un langage de description de haut niveau, sous forme de processus communicants.

Le chapitre 4 est très important. Il définit les problèmes de non-déterminisme et de métastabilité. L'élément synchroniseur destiné à prévenir la métastabilité est introduit dans ce chapitre, ainsi que la modélisation du non-déterminisme par le langage CHP. A partir de l'étude du non-déterminisme et du synchroniseur, nous évaluons le coût d'une synchronisation par communication à travers le réseau d'interconnexion. Comparé à son homologue synchrone, un réseau de communication sans horloge résout les problèmes de synchronisation de manière plus fiable, plus performante et moins coûteuse. Enfin, nous proposons une méthodologie de conception de réseaux d'interconnexion au sein d'une architecture globalement asynchrone et localement synchrone. Cette méthodologie s'appuie sur les propriétés d'excellente modularité des circuits sans horloge. Nous proposons un découpage modulaire des composants du réseau en ressources critiques élémentaires : arbitrage, transport, routage et synchronisation. A cela il faut rajouter une ressource supplémentaire de mise en forme des communications, qui se place d'un point de vue méthodologique au sein des périphériques synchrones. Nous illustrons cette méthodologie à partir d'un exemple plus concret d'architecture GALS : le système WUCS (*Wireless Universal Communicating System*) dans lequel s'intègre un réseau de communication sans horloge de topologie d'interconnexion totale, que nous appelons FAN (Fully-interconnected Asynchronous NoC).

Ces quatre chapitres de la première partie, présentent l'ensemble des études menées, nécessaires pour aboutir à la présentation de notre méthodologie de conception de réseaux de communication sans horloge par assemblage de composants. Dans la deuxième partie, nous revenons sur une étude plus détaillée de chacun de ces composants, ou modules.

Ainsi le chapitre 5 présente la modélisation d'arbitres à priorité et leur synthèse à partir du langage CHP. Il détaille ensuite l'intégration d'une instance d'arbitre à échantillonnage parallèle dans les modules d'arbitrage et de routage, qui constituent le cœur du réseau de communication sans horloge, quelle que soit sa topologie distribuée.

Le chapitre 6 introduit l'ensemble des techniques de synchronisation existantes pour les architectures GALS. Il s'agit essentiellement des techniques à base d'interruption de l'horloge et des solutions à base d'un ensemble circuit synchroniseur plus FIFOs. Nous proposons une solution simple et performante de FIFO entièrement asynchrone, adaptée à des périphériques synchrones acceptant une synchronisation par protocole de communication. Combinée à un circuit synchroniseur robuste, elle fournit le module de synchronisation de nos réseaux de communication sans horloge.

Le chapitre 7 enfin se penche sur l'importance du problème du transport des données le long des lignes physiques d'interconnexion. Il présente les moyens d'arranger les lignes de communication pour atténuer le coût des délais d'interconnexion.

PARTIE I

Chapitre 1 : Les systèmes sur silicium globalement asynchrones

En introduction de ce mansucrit nous avons présenté le paradigme de la conception globalement asynchrone et localement synchrone comme une alternative favorable pour dépasser les limites de la conception de circuits entièrement synchrones. Les techniques de conception synchrones ont toutefois été attentivement étudiées au cours de nos travaux pour plusieurs raisons essentielles :

1. Tout d'abord, il nous fallait être certain du potentiel que peut offrir la méthodologie de conception des systèmes globalement asynchrones, une idée qui n'avait pas encore convaincu quand ce travail de recherche a démarré;
2. ensuite, si le système GALS est globalement asynchrone, les cœurs de composants qui le constituent restent synchrones et la connaissance précise de leurs techniques de synchronisation interne reste indispensable;
3. enfin, nous désirions promouvoir avec des arguments solides l'utilisation au sein de ces systèmes GALS d'un réseau intégré asynchrone au lieu d'un bus synchrone. Or la comparaison de performance nécessite la maîtrise de conception d'un bus synchrone;
4. plus généralement, une bonne connaissance des techniques de conception synchrones nous fut nécessaire pour une analyse et une défense critiques du potentiel de conception des circuits auto séquencés.

C'est pourquoi dans ce chapitre nous rappelons en premier lieu l'ensemble des principales techniques de synchronisation d'un système sur silicium, du tout synchrone (§1.1) vers le globalement asynchrone (§1.2). De là nous présentons et justifions le choix et l'intérêt pour une architecture de type GALS de disposer en son sein d'un réseau sur silicium asynchrone (§1.3), malgré un certain nombre de défis que doit encore relever la conception asynchrone pour convaincre de son intérêt réel (§1.4).

1.1. Les méthodologies de synchronisation par horloge : revue des systèmes globalement synchrones

Messerschmitt, dans un article destiné à l'origine aux communications numériques [MES 90], définit un certain nombre de termes relatifs à la synchronisation de signaux en transit entre deux blocs logiques aux horloges différentes. Ces termes vont nous être utiles pour distinguer et classer les méthodologies de synchronisation :

- Système mésochrone :
    - Les sous-systèmes ont des fréquences de fonctionnement identiques avec des déphasages fixes
    - Une unique horloge globale de référence
- Système multi-synchrone :
    - Les sous-systèmes ont des fréquences de fonctionnement identiques avec des déphasages arbitraires
    - Une unique horloge globale de référence
- Système plésiochrone :
    - La fréquence de fonctionnement nominale des sous-systèmes est identique, mais la phase est arbitraire
    - Les sous-systèmes ont des générateurs d'horloge distincts destinés à fonctionner à la même fréquence mais soumis aux aléas de variations de procédés de fabrication et d'environnement de fonctionnement (température, alimentation…). Par conséquent, une légère différence de fréquence de fonctionnement entre deux domaines est possible, qui peut conduire soit à un double échantillonnage soit au ratage d'un transfert de données.
- Système hétérochrone :
    - Les horloges des sous-systèmes sont distinctes en fréquence et en phase, mais les fréquences sont des multiples.
    - La source des horloges peut être commune ou non.

1.1.1. Les systèmes globalement synchrones à verrouillage de phase

La méthode traditionnelle pour générer une horloge est de synchroniser, au moyen d'une boucle à verrouillage de phase (PLL pour *phase-locked loop*), une horloge lente externe au circuit intégré avec une horloge rapide intégrée. La sortie de la PLL est distribuée à travers le circuit au moyen d'un arbre d'horloge en H équilibré [WES 93] ou d'un réseau équivalent, qui

vise à égaliser les temps de propagation du signal d'horloge vers chacun des périphériques. La Figure 1-1 présente les structures les plus communes de distribution d'arbres d'horloge.

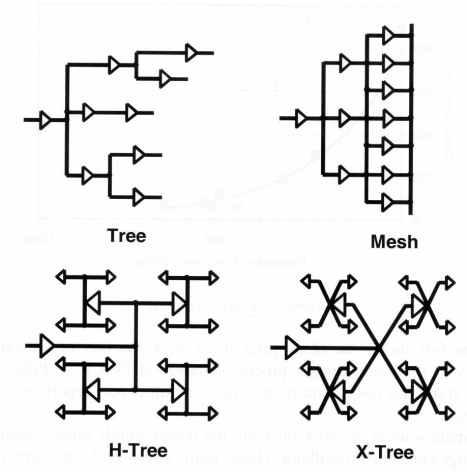


Figure 1-1 : structures communes de distribution d'arbres d'horloge

Cependant la PLL elle-même introduit une gigue de la phase (*jitter*), qu'il faut contenir mais qui n'est pas critique pour le système. Il s'agit de la variation temporelle de l'horloge par rapport à un front de référence. Plus pénalisantes par contre sont les différences de phase introduites, par les variations de process, dans les interconnexions et les répéteurs des branches de l'arbre d'horloge. Ce retard dans le temps de propagation, plus connu sous le terme de "*clock skew*", est à même de réduire fortement la portion de cycle d'horloge utile à l'exécution. La Figure 1-2 présente la mesure de ce déphasage en pourcentage de temps de cycle de l'horloge. La marge de déphasage considérée comme acceptable est de 5%, mais cette limite devient difficile à préserver pour des processeurs fonctionnant à des vitesses supérieures au gigahertz. Plus exactement, c'est l'effet de l'augmentation des fréquences de fonctionnement combiné à la réduction des dimensions des cellules qui accroît ce déphasage de l'horloge.

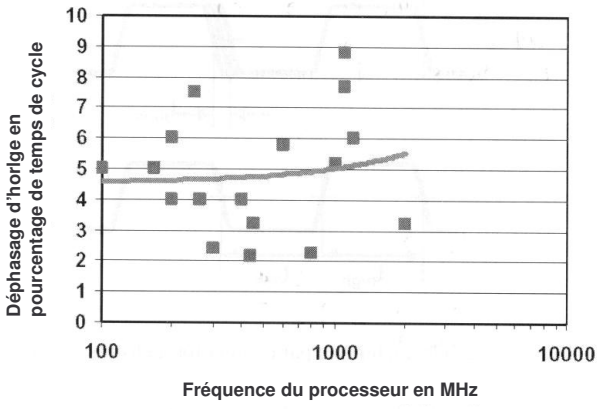


Figure 1-2 : déphasage d'horloge en pourcentage du temps de cycle

En effet, la densité d'intégration croissante d'éléments de plus en plus réduits ne s'accompagne pas d'une semblable évolution des lignes de transmission au sein des systèmes intégrés [SIA 04]. Le délai physique le long des lignes d'interconnexion devient de plus en plus significatif, à cause de l'augmentation de la résistance de ces lignes, en regard des vitesses de commutation des transistors et donc des cadences d'horloge. On observe alors la transition d'un

même signal à des instants différents en différents endroits du circuit, rendant le système mésochrone. Si ce signal est l'horloge, la nécessité de prévenir ce déphasage par des niveaux de bufferisation de plus en plus nombreux sur des circuits de plus en plus grands, limite la vitesse de fonctionnement du circuit synchrone. Car tout système gouverné par une horloge doit se conformer à un principe de base : il faut que le temps mis par le signal d'horloge pour parvenir au composant le plus éloigné soit inférieur à l'intervalle de temps qui sépare deux fronts d'horloge identiques.

Pour alléger ces contraintes temporelles, de nombreuses stratégies de contrôle de l'horloge ont été développées et sont présentées dans les paragraphes 1.1.2 et 1.1.3. Il est possible de retrouver simultanément plusieurs de ses stratégies dans la distribution de réseaux d'horloge de systèmes particulièrement complexes. Lorsque nous parlerons de "déphasage d'horloge" dans la suite du manuscrit, nous nous référerons à ce phénomène de "*clock skew*".

1.1.2. Techniques de distribution d'horloge à déphasage nul

1.1.2.1. Méthodologies mésochrones

Distribution de l'horloge par une grille

Cette technique de *clock gridding* consiste à transformer tout ou partie des feuilles (les extrémités) d'un arbre d'horloge équilibré en une grille sur laquelle le placement des blocs synchrones consommant l'horloge est arbitraire. La Figure 1-3 présente la distribution de l'horloge du microprocesseur Power4 [REST 02] dont une partie de l'arbre est une grille de 1024 points. La PLL est placée au centre du circuit pour minimiser le délai de distribution de l'horloge globale.

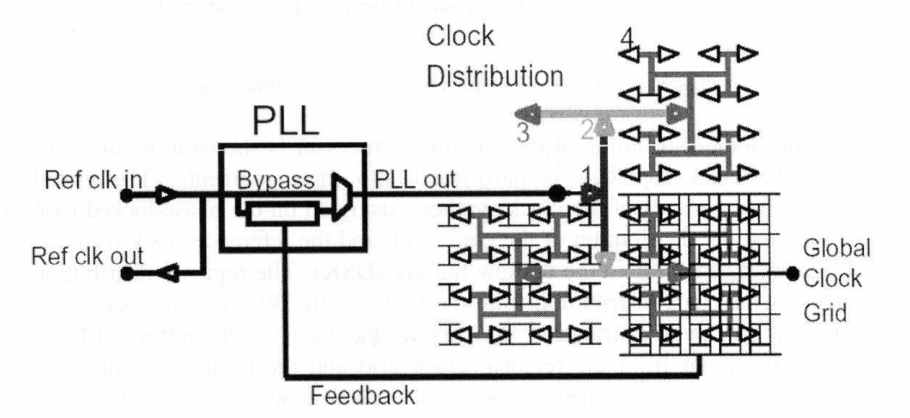


Figure 1-3 : distribution de l'horloge par une grille dans le processeur Power4

Cette stratégie vise à moyenner le déphasage de l'horloge. L'ennui est que les lignes de la grille augmentent de manière sensible la charge totale de l'arbre d'horloge. En outre elle interdit la mise en place de solutions de réduction de la consommation par la mise en veille de modules inactifs grâce aux techniques d'interruption locale de l'horloge, dites "*clock gating*" ou "*stoppable clock*".

Une amélioration est apportée par le processeur SPARC 800 MHz [HEALD 00] qui utilise un schéma à deux niveaux de grille. La sortie de la PLL est distribuée par l'arbre d'horloge à un premier niveau de grille. Cette grille alimente des buffers d'horloge qui déphasent l'horloge globale pour créer des domaines d'horloge distincts, grâce justement à une technique de contrôle d'horloge par logique combinatoire ("*clock gating*"). Chacun de ces nouveaux domaines redistribue son horloge aux sous-systèmes par un arbre équilibré terminé par une deuxième grille.

Boucles à verrouillage par délai (DLL pour ***Delay-Locked Loop***)

En cas de déphasage excessif de l'horloge, la technique de distribution de l'horloge par une grille peut conduire à des contentions entre les sous-systèmes au lieu de moyenner la phase. C'est le cas par exemple lorsqu'on désire intégrer un processeur synchrone dans un nouveau système sur silicium globalement synchrone en voulant retoucher au minimum le réseau de distribution de l'horloge interne au processeur.

La réponse fut apportée par les concepteurs du processeur Alpha 1,2 synchrone [XAN 01], constitué d'un précédent cœur de processeur, d'un ensemble de deux nouvelles mémoire de cache à deux niveaux et d'un sous-système composé de la mémoire principale et du bus de communication. Chacun de ces quatre sous-systèmes possède son propre arbre de distribution de l'horloge à faible déphasage, mais piloté chacun par la PLL du cœur de processeur pour assurer un système globalement synchrone. Pour compenser les inévitables retards entre la PLL et les racines respectives des arbres, des boucles à verrouillage de retard (DLL pour *Delay-Locked Loop*) sont insérées pour aligner les distributions de ces différentes horloges. La version 21264 de ce processeur est présentée sur la Figure 1-4.

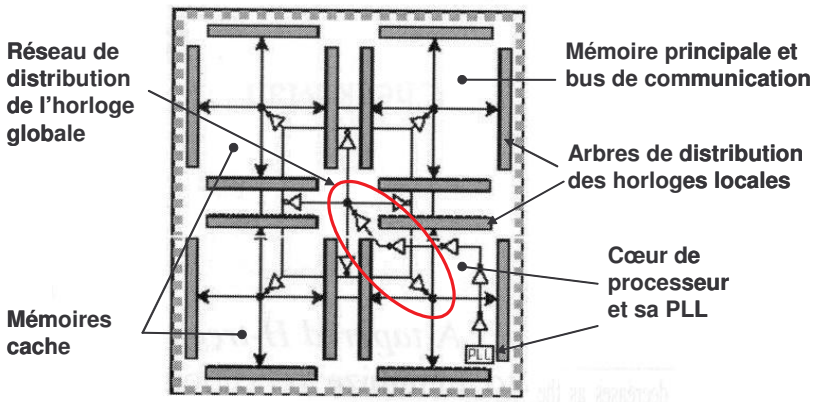


Figure 1-4 : réseau de distribution d'horloge par boucles à verrouillage de phase du processeur Alpha v21264

1.1.2.2. Méthodologie plésiochrone

Réseau distribué de PLLs

Pour répondre au besoin de flexibilité des systèmes sur silicium, [GUT 00] propose d'améliorer l'extensibilité des réseaux de distribution de l'horloge grâce à un réseau distribué de PLLs. Chaque sous-système, appelé carreau dans cette méthodologie (de l'anglais *"tile"*), dispose de sa propre PLL, ainsi que l'illustre la Figure 1-5. Des détecteurs de phase permettent à chacune de ces PLLs de se verrouiller sur le déphasage moyen des horloges des carreaux voisins.

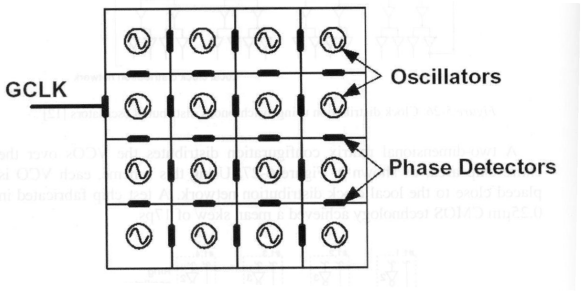


Figure 1-5 : réseau distribué de PLLs

Il faut cependant prendre garde au phénomène de verrouillage sur un modulo de phase, ou "*modelock*". En effet la nature cyclique de la phase peut entraîner le système à se stabiliser sur des offsets de phase non nuls entre les PLLs mais modulo 2π.

1.1.3. Techniques de distribution d'horloge à contrôle de déphasage non nul

1.1.3.1. Méthodologies mésochrones

Arbre d'horloge double

Les précédentes approches de distribution de l'horloge visent à s'affranchir complètement du déphasage. Au contraire le premier des processeurs Intel Itanium 64 bits [RUSU 00], dont le réseau de distribution de l'horloge est illustré par la Figure 1-6, inclut un module de test qui injecte volontairement un déphasage d'horloge pour détecter les chemins critiques du système. La PLL pilote deux arbres d'horloge :

- Le premier arbre, minutieusement équilibré et de charge minimale, sert d'horloge de référence (*reference clock* sur la Figure 1-6) ;
- Le second arbre distribue l'horloge système proprement dite (*main clock* sur la Figure 1-6), il supporte une charge maximale et ajuste sa phase sur l'horloge de référence du premier arbre.

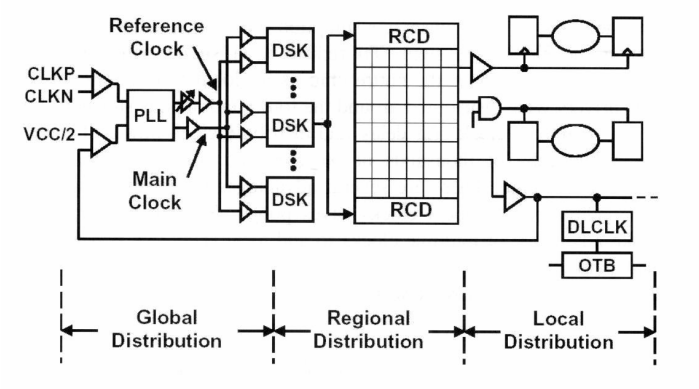


Figure 1-6 : réseau de distribution de l'horloge pour le processeur Intel Itanium 64 bits

Ces deux arbres alimentent chacun 32 *deskew buffer* (*DSK* sur la Figure 1-6) qui eux-mêmes pilotent chacun un sous-arbre équilibré qui se termine en grille (*RCD* sur la Figure 1-6). En fonctionnement normal, les *deskew buffer* sont chargés de détecter et de corriger les différences de phase entre l'horloge système et l'horloge de référence. En mode test, un port d'entrée-sortie spécifique permet au contraire d'injecter un déphasage dans les arbres de distribution d'horloge, en contournant les détecteurs de phase que sont les *deskew buffer*. Cela permet d'analyser les longueurs des chemins au sein du circuit intégré.

Réseau de détecteurs de phase mésochrone

La technique précédente de contrôle de phase non nulle se limite au mode test. Le processeur Intel Pentium 4 [KUR 01], dont le réseau de distribution de l'horloge est illustré par la Figure 1-7, reprend et étend cette technique en créant un réseau hiérarchique de 46 détecteurs de phase (*PD* pour *Phase Detector* sur la Figure 1-7) en remplacement de l'horloge de référence globale. Ces détecteurs pilotent non plus 32 mais 47 fuseaux d'horloge programmables (*DB* pour *Domain Buffer* sur la Figure 1-7).

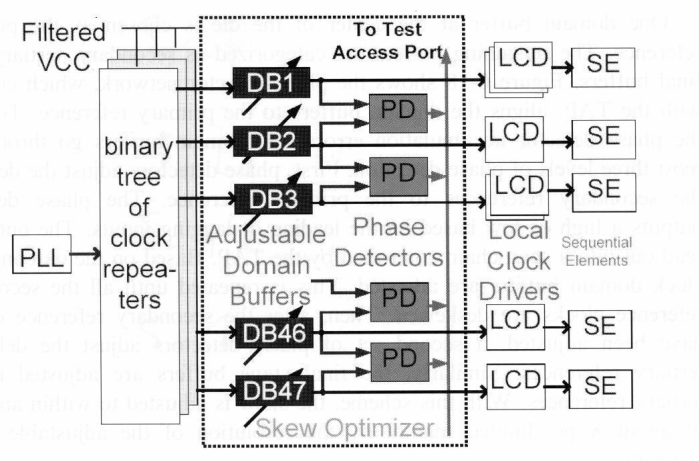


Figure 1-7 : arbre d'optimisation du déphasage d'horloge dans le processeur Intel Pentium 4

1.1.3.2. Méthodologies hétérochrones

Réseau de détecteurs de phase hétérochrone

Le précédent système mésochrone, où une PLL unique pilote tous les fuseaux, fut rapidement remplacé au sein des versions suivantes du Pentium 4 par une version hétérochrone composée de trois horloges système distinctes de fréquences multiples (principale, double, et fréquence moitié), ainsi que de trois horloges pilotant les ports d'entrée/sortie [HIN 01].

Méthode DCD (***"Delayed Clock Domino***")

Dans cette méthode le niveau de granularité de contrôle de déphasage de l'horloge passe du niveau larges sous-systèmes au niveau flip-flop. Les horloges sont préchargées dans un pipeline de portes domino et déclenchées en fonction du déphasage qui leur est attribué. Cette technique est implémentée entre autre dans les processeurs SPARC [HEALD 00], Pentium 4 [HIN 01] et IA-64 [RUSU 00].

Méthodes sur niveau

Ces méthodes, comme les méthodes par retenue de temps ("*Time Borrowing*") ou par relâchement ("*Slack Passing*") [CHAN 01] travaillent également sur niveau et non plus uniquement sur front d'horloge grâce à l'utilisation de latchs. Elles tirent avantage de la transparence des latchs pour découpler, jusqu'à un certain point, le temps d'exécution des blocs logiques et la date d'arrivée du prochain front d'horloge.

1.1.4. Conclusion

Nous venons de passer en revue un certain nombre de politiques extrêmement sophistiquées de réseaux de distribution de l'horloge. Mais dans l'ensemble, ces solutions nécessitent un nombre élevé de niveaux de métaux et coûtent cher en surface de silicium et en consommation électrique. En outre elles nécessitent de développer des modèles de délai et des simulations très poussés. Parvenus à une maîtrise de techniques d'hétérochronicité très avancées mais très complexes, les concepteurs désireux de disposer de fréquences de fonctionnement totalement indépendantes se sont alors tournés vers les architectures globalement asynchrones.

1.2. Les méthodologies de synchronisation par communication : introduction aux systèmes globalement asynchrones

1.2.1. Circuit synchrone et circuit asynchrone : définitions

Seitz définit un circuit synchrone comme un circuit dans lequel les séquences d'exécution sont temporisées par une horloge globale ou un ensemble d'horloges synchronisées entre elles [SEITZ 80]. Les transitions des signaux sont temporellement prédictibles.

Un circuit auto séquencé est un circuit dans lequel les séquences d'exécution satisfont une représentation insensible aux délais des éléments et des lignes de connexion. Les dates de transition des signaux ne sont pas prédictibles. Les circuits asynchrones définissent donc une classe très large de circuits, dont le contrôle ou le séquencement est assuré par toute autre méthode que le recours à un signal périodique globalement distribué.

Ainsi les méthodes présentées au paragraphe 1.1 précédent sont bien destinées à des systèmes globalement synchrones, même si ces derniers présentent un certain degré d'asynchronicité, voir même une très forte hétérochronicité (47 fuseaux d'horloge pour le Pentium 4, pilotés par 6 horloges distinctes [HIN 01]).

1.2.2. Architectures globalement asynchrones et localement synchrones

1.2.2.1. Définition du paradigme

Dans sa thèse Chapiro posait les fondations des systèmes GALS [CHAP 84], même si son approche adressait des systèmes de petites dimensions et n'est plus compatible avec les méthodologies de conception actuelle. Un assemblage de composants intégrés sur une même puce, synchrones selon la définition du paragraphe 1.2.1 précédent, mais dont les horloges sont indépendantes, est appelé architecture globalement asynchrone et localement synchrone (GALS). Les composants synchrones communiquent et se synchronisent au moyen de canaux et de protocoles de communication décrits en détail au paragraphe 3.1.

Précisons qu'une architecture globalement asynchrone dont les fréquences de fonctionnement synchrones locales seraient multiples l'une de l'autre n'est pas une architecture hétérochrone, car le principe de synchronisation est fondamentalement différent :

- Dans le cas d'une architecture globalement asynchrone, les sous-systèmes synchrones communiquent par des canaux de communication. Le format de codage des données dans ces canaux peut être du type (Cf. §3.1.3) :
    - "*bundle data*" où les signaux de contrôle nécessaires à la synchronisation sont transmis en parallèle avec le bus de données;
    - insensible aux délais (*DI* pour "*Delay Insensitive*") où le contrôle et les données sont mélangés (codage de type 1 parmi N).

D'un point de vue du formalisme informatique, précisons que ces canaux de communication sont dits *synchrones*, car ils synchronisent deux processus concurrents. Cependant la communauté des concepteurs en microélectronique parle de communication asynchrone pour la distinguer d'une communication synchrone, alors qu'il s'agit de distinguer une synchronisation par communication d'une synchronisation par horloge ;

- La synchronisation d'un système hétérochrone se fait par contrôle de phase sur les arbres de distribution des horloges des sous-systèmes, comme illustré au paragraphe 1.1 précédent. Les horloges sont dépendantes.

Notons que les sous-systèmes synchrones qui composent l'architecture GALS peuvent être localement hétérochrones, plésiochrones, mésochrones, ou purement synchrones.

1.2.2.2. Classification des signaux

Tous les signaux qui sont échantillonnés par une horloge sont conçus pour respecter des contraintes de temps de propagation pire cas. Ils se stabilisent à leur valeur logique finale bien avant le front d'horloge suivant (respect des règles de *setup time*). Ces signaux sont dits synchrones de l'horloge et sont déterministes (Cf. §4.1.1.2) car ils basculent à des instants prédictibles. C'est l'horloge qui détermine la validité des signaux synchrones.

Les signaux asynchrones sont des signaux dont la date de transition n'est pas prédictible. Leur validité doit être codée spécifiquement, soit par des signaux de contrôle explicites (requête et acquittement) selon un protocole particulier, soit en intégrant cette information de validité dans les signaux de données. Les signaux asynchrones se classent en deux catégories :

- Les signaux asynchrones dont le comportement est déterministe. La transition de ces signaux est certaine, bien sa date de réalisation soit inconnue. Ces signaux se synchronisent par rendez-vous grâce à un circuit spécifique dit porte de Müller, dont le détail est donné au §3.1.4. Cette porte attend que tous les signaux sur ses entrées soient au rendez-vous (occurrence d'une transition sur chacun d'eux) pour valider sa sortie.
- Les signaux asynchrones dont le comportement est non déterministe (Cf. §4.1.1.3). Aucune transition sur ces signaux n'est certaine. Un circuit spécifique appelé synchroniseur est alors nécessaire pour déterminer la valeur de ces signaux à l'instant d'échantillonnage par un signal de référence, qui peut être l'horloge ou un autre signal asynchrone. Ce type de circuit est présenté au §4.1.3 pour les synchroniseurs en environnement synchrone et au §5.1.2 pour les synchroniseurs en environnement asynchrone. Le problème de synchronisation fait l'objet du §4.2 et du chapitre 6. En outre, ces signaux asynchrones non déterministes ne peuvent pas se donner rendez-vous, mais doivent au contraire êtes arbitrés lorsque plusieurs d'entre eux peuvent potentiellement être vrais en même temps. Cette fonction peut être réalisée par l'élément d'exclusion mutuelle (Cf. §4.1.4.2) : si deux signaux asynchrones en entrée transitent en même temps, un seul est arbitrairement choisi.

La technique de conception asynchrone est abordée en détail au chapitre 3.

1.2.2.3. Avantages des architectures GALS sur les architectures globalement synchrones

On peut dire que l'approche GALS combine les avantages des deux techniques de conceptions synchrone et asynchrone [MUTT 99]. Un système GALS est en effet divisé en plusieurs blocs synchronisés par des horloges indépendantes et conçues avec les outils du flot standard de conception des circuits synchrones. Les générateurs de ces horloges sont locaux et aucune horloge externe n'est nécessaire (hormis une horloge de test distribuée globalement et destinée à la synchronisation avec le testeur matériel externe). Chacun de ces blocs synchrones communique avec le reste du système au moyen d'une interface asynchrone, ou pour mieux dire auto-séquencée. Cette interface spécifie un certain protocole de communication et un certain format du codage des données transmises à travers les canaux de communication. Ces protocoles et formats de codage sont présentés au chapitre 3, tandis que l'étude des interfaces fera essentiellement l'objet des chapitres 4 et 6. Les avantages essentiels à concevoir un système GALS avec une stratégie de synchronisation par communication, plutôt qu'un système globalement synchrone, sont alors les suivants :

- L'effort de conception des horloges est fortement réduit, car le contrôle du déphasage de l'horloge reste interne à chaque bloc synchrone ;
- La performance de chaque module synchrone est alors ajustable de manière indépendante aux autres modules ;
- L'effort de synchronisation des sous-systèmes est également fortement réduit, grâce au potentiel de modularité (voir §1.3.4) des protocoles de communication entre les blocs. La synchronisation par une horloge globale limite la réutilisation de sous-systèmes préexistants car ils présentent des interfaces souvent incompatibles ;
- En outre les protocoles auto séquencés offrent une grande robustesse et donc une grande fiabilité de communication (Cf. §1.3.4.1 et §2.3.1) ;
- Motivant les premiers travaux de cette thèse, Hemani et al. ont publié une étude comparative de la consommation entre des architectures synchrones et leurs équivalents GALS [HEM 99]. Au sein d'un système composé de N modules synchrones, ils montrent que la consommation d'énergie pour la synchronisation des GALS est réduite d'un facteur au moins $\sqrt{N}$ vis-à-vis des architectures globalement synchrones[3]. Le coût de synchronisation de l'architecture synchrone inclut toutes les horloges du système et leurs synchronisations, tandis que la synchronisation de l'architecture GALS prend en compte les horloges des sous-systèmes et la logique de communication par canaux.

Bien évidemment le paradigme des systèmes GALS, en apportant une solution nouvelle à la synchronisation d'un circuit, amène également de nouveaux problèmes, que présente le suivant paragraphe.

---

[3] Dans la configuration GALS la plus favorable présentée dans ces travaux, la consommation d'énergie du réseau d'horloge est réduite de 70% lorsqu'on s'affranchit de la synchronisation globale du système par l'horloge. Cela correspond à une réduction de 20% de la consommation globale totale du système GALS par rapport à son équivalent synchrone.

1.2.2.4. Contraintes des architectures GALS

La synchronisation

La contrainte de conception d'une architecture GALS vient de sa spécificité : la synchronisation par communication de domaines d'horloges indépendantes. Cette synchronisation entre les blocs est uniquement assurée par des signaux de contrôle locaux (Cf. §3.1 sur les protocoles de communication). Ces signaux de contrôle locaux, appelés les requêtes ou les acquittements (Cf. 3.1.1), se trouvent être asynchrones vis-à-vis des entrées des blocs qu'ils synchronisent. C'est-à-dire que leurs dates d'évènement ne sont ni prédictibles ni corrélées. En effet, ils sont échantillonnés par l'horloge locale des blocs synchrones et donc l'ordre relatif des transitions de ces signaux d'entrées et des transitions d'horloge sont imprédictibles.

Cet échantillonnage d'un signal asynchrone au moyen du signal d'horloge d'un bloc synchrone est à l'origine de deux problèmes connexes : la métastabilité et le non-déterminisme :

- Le phénomène de métastabilité caractérise un état dont la durée est indéterminée, non bornée [REN 04b]. Un circuit peut entrer dans un état métastable lorsque un signal est échantillonné à une valeur intermédiaire qui ne permet pas de déterminer son niveau logique ;
- Le non-déterminisme est la caractéristique, généralement indésirable, que possède un système à fournir plusieurs séquences de sortie différentes en réponse à une même séquence d'entrée. Ainsi, si un des blocs synchrone possède de multiples entrées asynchrones, les délais relatifs entre les différents signaux de contrôle de ces communications rendent la séquence d'entrée, et par conséquent la séquence de sortie, non déterminée.

Des circuits spécifiques dits synchroniseurs sont utilisés pour résoudre ces problèmes et permettre la synchronisation de domaines d'horloges différentes. Mais la fiabilité de ces circuits n'est pas parfaite et ils introduisent une latence importante dans les communications. Ces circuits synchroniseurs sont nécessaires dans les architectures GALS chaque fois qu'un bloc synchrone échantillonne au moyen de son signal d'horloge un signal asynchrone de l'horloge. Or au cœur des échanges se trouve le réseau de communication. Si ce réseau d'interconnexion est synchrone, il va nécessiter l'usage de synchroniseurs pour échantillonner en entrée chaque signal de contrôle de communication reçu d'un périphérique. Un réseau de communication sans horloge s'affranchit en revanche de l'usage de synchroniseurs car comme tout circuit auto-séquencée, ses séquences d'exécution sont insensibles aux délais.

La section 4.1 présente en détail le non-déterminisme, la métastabilité et le circuit synchroniseur. Elle illustre également leurs modélisations respectives en langage de haut niveau CHP (*Communicating Harware Processes*) dans sa version modifiée utilisée au sein du groupe CIS du laboratoire TIMA. La section 4.2 détaille le coût de cette synchronisation dans une architecture GALS et montre comment un réseau de communication localement sans horloge peut avantageusement réduire l'impact du coût de ces synchroniseurs aux interfaces entre le réseau d'interconnexion et les périphériques. Le chapitre 6 enfin passe en revue toutes les techniques d'interfaces de synchronisation performantes entre les périphériques synchrones et le réseau de communication sans horloge.

Le réseau d'interconnexion

Nous avons vu au paragraphe 1.1.1 que c'est le délai physique le long des lignes qui, devenant de plus en plus significatif en regard des vitesses de commutation des transistors, a conduit à la nécessité de résoudre tous ces problèmes rencontrés par l'horloge. Et finalement au désir de s'en affranchir en se tournant vers les architectures GALS à synchronisation par communication. Cependant cette importance croissante du temps de propagation au sein des lignes vis-à-vis des temps de calcul ne concerne pas uniquement l'horloge, mais l'ensemble des lignes de communication du système. Or le réseau d'interconnexion, qui remplace l'horloge pour synchroniser les architectures GALS par canaux de communication, constitue l'essentiel des liens de communication. Ce réseau reproduit donc un certain nombre de difficultés liées à la conception d'une horloge synchrone. De par sa nature, il occupe une surface de circuit importante, connectant des blocs éloignés sur de longues distances. Il nécessite donc des efforts de routage importants, en particulier pour son réseau de distribution de l'horloge. Ayant tendance à occuper une grande surface de circuit, son horloge devient une source majeure de pollution électromagnétique.

Le choix d'une architecture GALS est une première étape fondamentale pour s'affranchir en grande partie des problèmes liés à la distribution de l'horloge sur une grande surface d'un circuit. La seconde étape pour s'en affranchir complètement est de choisir un réseau de communication sans horloge.

1.2.3. Conclusion

Nous avons donc identifié ci-dessus deux difficultés principales à la construction d'un réseau sur silicium : d'une part les problèmes de synchronisation aux interfaces entre les composants et le réseau, tous synchrones, du système sur silicium et d'autre part le réseau d'interconnexion lui-même, qui se révèle être la pierre angulaire des performances du système, et par conséquent son goulet d'étranglement. L'objectif des travaux de ce manuscrit est de passer en revue les problèmes spécifiques aux réseaux intégrés, en focalisant particulièrement l'attention sur la synchronisation, puis de proposer le choix d'un réseau de communication sans horloge pour lever ou réduire certaines des difficultés, notamment la synchronisation. Pour commencer, la section suivante examine comment la technologie de conception asynchrone peut répondre aux problèmes d'implémentation de réseaux d'interconnexion dans une architecture globalement asynchrone et localement synchrone.

1.3. Les avantages de la conception sans horloge pour les réseaux sur silicium

Les systèmes asynchrones s'avèrent, de manière générale, plus robustes, plus économes en énergie et potentiellement plus rapides que leurs équivalents synchrones s'ils sont conçus proprement et rigoureusement [COR 02], [NOW 99]. Mais en contrepartie les circuits asynchrones souffrent d'un coût supplémentaire élevé en logique de contrôle qui accroît de manière significative leur surface. C'est pourquoi leur utilisation dans les architectures GALS reste limitée à l'implémentation des interfaces de synchronisation par communication. Ce paragraphe se propose cependant de présenter les caractéristiques spécifiques de la conception de circuits sans horloge qui peuvent se révéler des avantages de plus en plus significatifs pour les réseaux sur silicium au sein d'une architecture GALS.

Avant-propos : les architectures asynchrones…

Avant toute chose, il nous paraît bon de rappeler ici qu'un système peut être asynchrone vis-à-vis de son environnement tout en restant lui-même cadencé par une horloge. Pour faire la distinction entre de tels circuits de séquencement synchrone, mais globalement asynchrones, et des circuits véritablement asynchrones, on parle pour ces derniers de circuits auto séquencés ou encore de circuits sans horloge.

… ne sont pas antinomiques aux systèmes globalement synchrones

Les circuits sans horloge peuvent être rendus globalement synchrones s'ils communiquent avec leur environnement synchrone au moyen d'une interface de synchronisation par horloge. Par exemple l'intégration d'une unité de multiplication et division (MDU) asynchrone dans un microprocesseur MIPS 4Ksc synchrone [SLI 02]. Panyasak présente dans sa thèse [PAN 04] les interfaces compatibles cycle à cycle qui permettent d'encapsuler un cœur de composant asynchrone dans un système globalement synchrone.

1.3.1. Affranchissement du déphasage de l’horloge (clock-skew)

Le problème du déphasage d'horloge a été longuement développé au paragraphe 1.1. Il nous suffit ici de rappeler que les systèmes asynchrones sont sans horloge à distribuer. Le problème de déphasage ne se pose donc pas, ce qui est un atout considérable pour un réseau sur silicium réparti sur plusieurs niveaux de métal et sur l'ensemble de la surface d'un système sur silicium.

1.3.2. Les atouts pour la consommation

La consommation d'énergie est le point clé des systèmes embarqués où la ressource en énergie est finie. Les systèmes haute performance voient même leur "*packaging*" affecté, ces derniers devant à la fois fournir avec le moins de pertes possible l'énergie du système, tout en dissipant la chaleur produite par ces derniers et pouvant atteindre plusieurs dizaines de watts. L'importance du problème est révélée par l'apparition dans les manuels d'utilisation, de chapitres entiers sur le management thermique des processeurs [INTEL 99].

La conception asynchrone s'affranchit naturellement de deux problèmes de consommation spécifiques au synchrone [HAU 95] :

1. Toutes les parties d'un circuit synchrone s'activent au front d'horloge, même si elles ne réalisent aucune fonction utile ;
2. Les lignes d'horloge proprement dites ont des capacités de charge élevées et nécessitent des drivers dissipant une importante quantité d'énergie. L'absence d'horloge dans un réseau intégré asynchrone va permettre d'économiser ces coûts de manière significative, comme présenté au paragraphe 1.2.2.3 sur les travaux de Hemani et al [HEM 99].

Les solutions synchrones à ces problèmes, comme le *clock-gating* ou *stoppable-clock* [WU 00], sont complexes, tandis que les systèmes asynchrones ne nécessitent aucune mise en œuvre particulière pour les résoudre.

Pour répondre au premier problème, il faut savoir que les systèmes asynchrones ont la propriété de mise en veille/réveil automatique ("*zero power quescient state*"). En l'absence d'activité, ils cessent d'eux-mêmes de fonctionner et lorsque celle-ci redémarre, ils se remettent instantanément à fonctionner à pleine puissance. Au contraire des systèmes synchrones qui en l'absence d'activité doivent choisir :

- soit de continuer à activer l'horloge au prix d'une consommation d'énergie inutile ;
- soit d'arrêter l'horloge. Dans ce cas une coûteuse logique de décision doit déterminer la pertinence d'un tel choix, car redémarrer ensuite l'horloge sans perdre d'information nécessite de compter avec la latence de stabilisation du signal d'horloge.

Cette capacité de mise en veille/réveil paraît moins cruciale pour un réseau d'interconnexion destiné, s'il est correctement dimensionné, à fonctionner en permanence pour répondre aux besoins des périphériques du système. Cependant, l'évolution des réseaux de communication vers des topologies distribuées en maille ou en crossbar (voir §2.4.) tend à fournir des réseaux intégrés dont le taux d'activité n'est plus le plein régime. Il reste donc avantageux de pouvoir éteindre les parties inactives du réseau de manière passive, grâce à une propriété inhérente des systèmes asynchrones.

Une revue des techniques asynchrones dédiées à la conception de circuits et plus particulièrement de processeurs faible consommation est présentée au chapitre 22 de l'ouvrage *Low-Power Electronics Design* [SLI 04].

1.3.3. Les émissions électromagnétiques

La synchronisation globale par une horloge provoque l'activité de l'ensemble du circuit aux mêmes instants. Le spectre d'émission d'énergie se retrouve donc concentré aux harmoniques de la fréquence d'horloge. Les techniques de conception synchrone pour répartir le spectre d'émission, comme la variation de la période d'horloge ("*spread spectrum clock*"), existent mais ne peuvent que réduire les pics d'activité liés à l'horloge sans toutefois pouvoir les supprimer. En outre ces techniques n'autorisent qu'une très faible variation sur la période d'horloge si le contrôle du déphasage d'horloge veut être éviter [PAN 04] et diminuent les performances du système (on ne peut jouer que sur l'allongement de la période de l'horloge).

Les circuits asynchrones produisent une émission d'interférences "*broadband*" (à large bande) distribuée sur l'ensemble du spectre et d'intensité réduite comme démontré sur le processeur asynchrone ASPRO [*ASPRO]. L'avantage de cette discrétion est significatif pour les applications sécurisées [BOU 04b] qui doivent protéger la lecture et le décodage possibles du spectre du circuit et pour les systèmes radiofréquences qui cherchent à minimiser les interférences, autant leur propre pollution d'émission que celle de l'environnement [MON 04], car ces systèmes associent des parties numériques et analogiques, qui sont par définition très sensibles. Allier [ALL 03] démontre dans le cadre de ses travaux de thèse le faible bruit généré par les circuits asynchrones dans les systèmes mixtes de conversion analogique numérique.

Par exemple, le circuit pour carte sans contact MICA a été conçu dans [ABR 01], intégrant le système de réception, de traitement et d'alimentation. Le fait que le microcontrôleur utilisé soit asynchrone permet de le faire fonctionner pendant les phases de réception/émission ce qui n'était pas possible jusqu'alors avec des processeurs synchrones.

Dans ce sens, un réseau de communication asynchrone intégré dans une architecture GALS et connectant entre eux des blocs synchrones offre un certain nombre d'intérêts. Comme introduit précédemment au paragraphe 1.2.2.4 et développé au chapitre 2, le réseau sur silicium devient prépondérant dans les systèmes en terme de performance critique, de consommation et de surface. Un réseau sur silicium synchrone couvrant l'ensemble de la surface du circuit est donc une source majeure de pollution. Le même réseau sur silicium asynchrone voit son intensité de bruit et sa signature réduits car répartis plus largement sur le spectre.

1.3.4. La modularité

Un système asynchrone fonctionne sur le principe de modules se synchronisant par un ou plusieurs canaux de communication. Chaque module est donc indépendant de ses voisins à tout point de vue. A condition de respecter ce(s) protocole(s) de communication, la modularité des composants offre alors un certain nombre d'avantageuses propriétés présentées ci-dessous. Notons que nous parlons ici de système asynchrone de manière générale : le bénéfice de la modularité provient de l'utilisation des protocoles de communication pour se synchroniser, il profite donc autant aux architectures globalement asynchrones et localement synchrones qu'aux circuits asynchrones.

1.3.4.1. La localité

Un système synchrone ne peut être amélioré qu'en augmentant sa fréquence d'horloge, ce qui nécessite la plupart du temps de repenser la quasi-totalité des interfaces du système, soit en adaptant plusieurs domaines d'horloge en son sein, soit en réimplémentant les composants pour qu'ils fonctionnent à des fréquences plus élevées.

Au contraire l'indépendance des modules dans un système globalement asynchrone rend possible l'amélioration globale des performances en optimisant uniquement [SLI 04] :

- La vitesse des parties critiques les plus actives du circuit, de manière à augmenter la vitesse globale du système de manière significative,
- La consommation et la surface des parties au taux d'activité le plus bas, sans pour autant pénaliser la vitesse globale du système.

Et cela sans être obligé de reconstruire tout l'arbre d'horloge et tous les composants pilotés par cet arbre.

Cette propriété de localité se retrouve bien sûr à l'échelle du réseau de communication, vu comme une *IP* du système sur silicium. La méthodologie de conception des réseaux sur silicium présentée au chapitre 4 et développée en deuxième partie de ce manuscrit (chapitres 5 à 7) repose fortement sur cette propriété de modularité-localité.

Le chapitre 4 introduira quatre modules ou ressources critiques internes qui vont construire nos réseaux sur silicium asynchrones, plus une interface propre à chaque périphérique du système, considérée comme une cinquième ressource externe :

1. la ressource de Transport de l'Information (*ITR* pour "*Information Transport Ressource*") ;
2. la ressource de Routage de l'Information (*IRR* pour "*Information Routing Ressource*") ;
3. la ressource d'Arbitrage (*PRS_Arbiter* pour *Parallel-Request-Sampling Arbiter*) pour l'attribution des ressources de Transport, de Routage et de Périphérique ;
4. la ressource d'Interface de Synchronisation entre les ressources d'Arbitrage, de Routage et de Périphérique ;
5. la ressource d'Interface de Communication, intégrée au Périphérique.

1.3.4.2. L'extensibilité

L'extensibilité *(scalability)* est la gamme de valeurs que peuvent prendre les paramètres génériques d'un composant sans altérer ni ses fonctionnalités, ni ses performances globales. Pour un réseau d'interconnexion, il s'agit typiquement du nombre de périphériques concurrents que celui-ci peut supporter (Cf. §2.2.1).

La modularité induite par les protocoles de communication permet de fournir des réseaux d'interconnexion asynchrones très fortement extensibles. L'absence de signaux globaux favorise l'extensibilité.

1.3.4.3. Migration technologique

Les circuits asynchrones se prêtent facilement aux migrations technologiques et les rendent plus « souples ». En effet, le comportement fonctionnel d'un circuit asynchrone est indépendant de la réalisation des cellules qui le constituent pourvu que le protocole de communication soit respecté. Ainsi, au niveau le plus bas, il est possible de modifier l'implémentation ou la technologie des cellules de base sans modifier la fonction. La vitesse de fonctionnement obtenue sera simplement la vitesse maximale permise par la nouvelle technologie étant donnée la structure du circuit.

1.3.5. La performance moyenne meilleure que l'alignement sur le "pire cas" synchrone : propriété de calcul en temps minimum

Dans un système synchrone la période d'horloge minimale est contrainte pour prendre en compte un certain nombre de variations de paramètres :

- la tension d'alimentation ;
- la température ;
- la vitesse de commutation des transistors ;
- le temps d'exécution en fonction des données. Ainsi un bloc additionneur réalise plus vite son calcul si la retenue se propage sur une courte distance [HAS 95].

L'occurrence de la combinaison pire cas de variation de ces paramètres s'avère très rare en pratique. Il se trouve alors qu'un circuit asynchrone, non contraint de fonctionner à une fréquence fixe imposée, peut réaliser dans la plupart des cas de meilleures performances.

En effet un opérateur asynchrone peut évaluer une fonction en un temps variable, compris entre une borne inférieure et une borne supérieure. Ce temps correspond en fait au temps nécessaire à l'écoulement des données, des entrées vers les sorties. En fonction des données elles-mêmes, le chemin emprunté peut changer et le temps varie en conséquence. Etant donné que par définition, l'opérateur implémente une signalisation de communication, la donnée peut être utilisée à sa sortie par l'opérateur qui lui est connecté, et ce dès que possible.

En plus de cette variation qu'on peut qualifier de fonctionnelle (car uniquement liée à l'expression des dépendances présentes dans l'algorithme), il existe une autre source de variation du temps de calcul liée à l'implémentation physique [HAS 95] [FRAG 03].

Enfin, le fonctionnement *flot de données* des circuits asynchrones les rend très robustes vis-à-vis des variations en fonctions des paramètres qui influencent le fonctionnement des dispositifs élémentaires, tels que les variations des paramètres technologiques, de la température ou des tensions d'alimentation [ESS 02]. La fin de traitement étant détectée et signalée au niveau de chaque cellule, les variations de la vitesse induites par les modifications des propriétés physiques n'altèrent pas le comportement fonctionnel. Les circuits asynchrones fonctionnent toujours à la vitesse maximale permise par les dispositifs élémentaires et les conditions de fonctionnement.

Il est donc possible de caractériser un opérateur par une latence maximale et une latence minimale. Evaluer une latence moyenne est plus difficile car cela impose de modéliser

les propriétés statistiques des entrées et des dépendances fonctionnelles de l'algorithme implémenté. Cependant une chose est sûre, le temps de traversée sera le temps minimal requis pour réaliser la fonction demandée, étant donné le chemin emprunté par les données, la vitesse opérationnelle des dispositifs élémentaires, et les conditions de fonctionnement de la technologie. Alors que la correction fonctionnelle est garantie, le circuit traite les données à la vitesse maximale. La caractérisation de performances d'un système asynchrone se fait alors en mesurant sa performance moyenne à conditions d'environnement fixes (température, alimentation…).

De tels systèmes s'avèrent en outre très robustes à de grandes variations de procédés [REN 99] et peuvent dépasser la contrainte de coïncidence des conditions pire cas imposées à un système synchrone de spécification équivalente.

1.3.6. Un pipeline élastique

En asynchrone, la technique du pipeline s'applique également, mais le nombre de données présentes dans le pipeline n'est pas imposé [RIGA 02]. En effet, les registres de pipeline se comportent comme une pile de type FIFO, c'est-à-dire que les données progressent dans le pipeline aussi longtemps qu'elles ne rencontrent pas de ressource occupée, et ceci indépendamment des données qui suivent. En technologie synchrone, c'est l'occurrence d'un front d'horloge sur tous les registres de pipeline qui provoque le déplacement des données. L'horloge impose donc une synchronisation relative forte des données entre elles. Une fois entrées dans le pipeline, deux données sont toujours séparées par le même nombre d'étages. Ce ne sera évidemment pas le cas pour le pipeline ou les FIFOs des réseaux sur silicium sans horloge (Cf. chapitres 4 et 6), où une donnée qui entre tardivement dans le réseau pourra rattraper une file de données plus lente en amont.

1.3.7. Conclusion

Les composants asynchrones facilitent la conception d'architectures GALS. Ils s'intègrent parfaitement à un réseau de communication sans horloge, d'une part bien sûr grâce à la compatibilité des protocoles de communication. D'autre part parce que ce protocole de communication permet l'adaptation automatique des vitesses relatives des domaines de fonctionnement entre le composant et le réseau, sans avoir recours à des interfaces de synchronisation complexes, de fiabilité non parfaite et de latence élevée (Cf. chapitre 4 et chapitre 7) nécessaires pour les composants synchrones.

En outre l'intérêt rapidement croissant pour l'utilisation des architectures GALS a conduit les concepteurs de réseaux d'interconnexion et les architectes réalisant l'intégration des systèmes à utiliser les techniques de conception asynchrones de manière de plus en plus large [MUTT 99, VILL 02]. Et donc, comme nous allons le voir en détail en partie II du manuscrit, à s'intéresser à l'amélioration des performances de tels circuits [ZHU 02], [CHAT 03], ainsi que des outils et méthodologies associées [AHO 04].

1.4. Les limitations actuelles de la conception asynchrone

Malgré les avantages certains présentés au paragraphe précédent vis-à-vis des circuits synchrones, d'autres obstacles spécifiques à la conception asynchrone sont à surmonter pour proposer des réalisations robustes et performantes d'architectures de communication intégrées. Il ne paraît pas non plus raisonnable d'envisager dès à présent, hormis au sein des groupes de recherche universitaires, des réalisations entièrement asynchrones, pour les raisons suivantes.

1.4.1. Le manque de maturité et de familiarité

La technique de conception asynchrone n'a pas atteint la maturité des outils de conception synchrone. En effet l'accumulation de connaissances et d'expériences sur les circuits synchrones est gigantesque, y compris pour les langages et les outils de conception qui permettent à un circuit d'être modélisé, synthétisé, testé, vérifié et fabriqué avec un haut degré d'automatisation. Cependant la communauté asynchrone connaît de réelles avancées dans le domaine et la trentaine d'outils de conception de circuits asynchrones propose aujourd'hui des solutions performantes, spécifiques et évolutives [EDW 04].

Reconnaissons également que les ingénieurs actuels ne sont pas formés à ces paradigmes de conception : la méthode de conception asynchrone oblige à revoir de fond en comble les méthodes de conception industrielles actuelles, même s'il est possible de détourner les outils destinés aux systèmes synchrones pour concevoir rapidement mais sans réelles améliorations de performances des circuits sans horloge [KOND 02]. En outre, cette technique de conception reste confidentielle car elle est très peu enseignée dans les universités.

L'exploitation des techniques de conception asynchrones est donc à ce jour pertinente pour des niches techniques très précises comme la sécurité des circuits, les architectures faible bruit ou les réseaux intégrés de communication.

1.4.2. La complexité

Le paradigme de la conception synchrone énonce deux règles fondamentales :

1. chaque étage de calcul doit terminer son traitement dans un intervalle de temps inférieur à la période de l'horloge. Cette règle est simple à suivre car c'est le bloc de calcul le plus lent qui conditionne le choix de la fréquence d'horloge ;
2. le signal d'horloge doit parvenir au composant de traitement le plus éloigné dans un intervalle de temps inférieur à la période de l'horloge. Cette règle de plus en plus difficile à mettre en œuvre sur de larges systèmes à conduit à s'intéresser aux architectures globalement asynchrones.

La simplicité de la première règle assure encore de beaux jours aux circuits synchrones. Au contraire un bloc asynchrone requiert une logique de contrôle supplémentaire importante pour réaliser les synchronisations locales de transfert de données. Le non-déterminisme des arbitres asynchrones [RIG 02b] était jusqu'à peu un frein supplémentaire car il

représente un obstacle majeur dans la testabilité et la vérification des systèmes. Aujourd'hui cependant existent des outils performants de vérification formelle de circuits asynchrones [BOR 03b], [YAK 03] (Cf. §3.2.2).

En outre les architectures globalement asynchrones, même avec un réseau d'interconnexion synchrone, rencontrent à présent le même obstacle à franchir. L'étude de ce non-déterminisme et de son coût dans la construction de réseaux sur silicium est d'ailleurs l'objet en grande partie du chapitre 4 et a motivé les travaux des chapitres 5 et 6. Ce manuscrit se propose de montrer à partir du chapitre 4 et suivants, que le coût de synchronisation d'un système globalement asynchrone et localement synchrone est un argument supplémentaire en faveur d'un réseau de communication sans horloge.

1.5. Conclusion

Ce chapitre a présenté les différentes techniques de synchronisation par horloge et les problèmes auxquels elles sont confrontées. En particulier la distribution d'une horloge à grande échelle dans un circuit pose des problèmes de temps de propagation, d'intégrité du signal, de bruit et de consommation et complexifie de façon significative la conception du système.

Le paradigme des architectures globalement asynchrone et localement synchrone offre au contraire une synchronisation locale grâce à des protocoles de communication. Le système de communication, synchrone ou asynchrone, permet alors de découpler les composants du système, le rendant plus modulaire et extensible, donc plus flexible, et diminuant de fait l'effort de conception. Une attention particulière doit cependant être portée aux problèmes de synchronisation aux interfaces entre les composants et le réseau de communication qui fonctionnent dans des domaines de vitesse différents.

Le réseau d'interconnexion se révèle d'ailleurs être la pierre angulaire des performances du système, et par conséquent son goulet d'étranglement. L'utilisation d'un réseau de communication localement sans horloge présente un certain nombre d'avantages, en commençant par la suppression de son horloge locale. En effet, de par sa nature, le réseau occupe une surface de circuit importante, connectant des blocs éloignés sur de longues distances. Il nécessite donc des efforts de routage importants, en particulier pour son réseau de distribution de l'horloge s'il est synchrone. Ayant tendance à occuper une grande surface de circuit, son horloge devient une source majeure de pollution électromagnétique. Ce qui n'est pas le cas pour un réseau de communication sans horloge. Mais le réseau de communication sans horloge nous offre également un degré de modularité très élevé qui va faciliter la conception, l'adaptation et la réutilisation des composants de nos réseaux.

Le chapitre suivant s'attache à présenter l'ensemble des critères de conception qui caractérisent le réseau d'interconnexion d'un système sur silicium, qu'il soit synchrone ou asynchrone, avant d'illustrer la caractérisation de ces paramètres à travers la présentation de réalisations de réseaux sur silicium industriels et universitaires représentatifs.

Chapitre 2 : Les réseaux de communication intégrés sur silicium

Le système de communication[4] est devenu le cœur de la conception des systèmes sur silicium. La plupart des grands acteurs du monde de la conception en microélectronique offrent aujourd'hui dans leur portefeuille de composants un réseau de communication intégré. Le Tableau 2-1 présente une liste, non exhaustive mais déjà fort représentative, de la grande variété de réseaux de communication intégrés déjà présents sur le marché des semi-conducteurs (d'après [SIL 03]). Ce tableau montre aussi que les consortia d'alliance se sont dotés de spécifications de protocoles de communication devant servir de standard et de label de reconnaissance pour ses membres. Certaines petites sociétés et start-up font même de la conception de réseaux sur silicium leur spécificité [ART 04a], [FUL 04a].

Réseaux d'interconnexion synchrones	Compagnie propriétaire/intendante
Advanced Microcontroller Bus Architecture (AMBA)	ARM
CoreConnect	IBM
CoreFrame	Palmchip
FISPbus	Mentor Graphics
IDT Peripheral Bus (IPbus)	Integrated Device Technology
Peripheral Interconnect Bus (PI-Bus)	Open Microprocessor systems Initiative (OMI)
SiliconBackplane	Sonics, Inc.
Wishbone Interconnect (open standard)	Silicore Corporation

Tableau 2-1 : Une revue des réseaux d'interconnexion pour systèmes sur silicium

Ce chapitre commence par passer en revue l'ensemble des paramètres qui caractérisent le réseau d'interconnexion d'un système sur silicium, qu'il soit synchrone ou asynchrone, avant d'étudier plus en détails trois caractéristiques fondamentales de ces réseaux : le degré de flexibilité d'un réseau, la fiabilité et la qualité de ses protocoles de communication, et les

[4] *NoC* pour *Network on Chip*. Il s'agit du terme anglo-saxon adopté comme standard de désignation d'un réseau de communication intégré sur silicium.

différentes familles de topologies existantes. Enfin nous illustrerons la caractérisation de ces paramètres à travers la présentation de réalisations de réseaux sur silicium industriels et universitaires représentatifs.

2.1. Les critères de conception d'un réseau sur silicium

2.1.1. Les réseaux sur silicium : un paradigme nouveau

Les premiers travaux universitaires conséquents sur le thème des réseaux intégrés furent publiés dès la fin du précédent millénaire. Ils portaient tous sur la réalisation de réseaux de communication complets. Citons entre autres le routeur à commutation de paquets R-SPIN développé à l'université Paris VI [GUE 00a], le bus centralisé asynchrone MARBLE destiné au processeur ARM-like Amulet3i développé par l'université de Manchester [BAIN 00] ou enfin le réseau asynchrone à mailles du processeur ultra faible consommation Maia développé par l'université de Berkeley [BEN 00].

Ces travaux cherchaient à l'époque à déterminer une architecture performante de réseau sur silicium la plus générique possible. Ils parvinrent à peu près tous à la même époque à cette conclusion : l'utilisation traditionnelle d'un bus centralisé est obsolète et doit être remplacée par des topologies de réseaux distribués sur le modèle des réseaux *off-chip* utilisés en télécommunications et pour les calculateurs parallèles haute performance [FESQ 97]. En outre, la topologie efficace de ces réseaux est fortement dépendante de l'application et des contraintes de performances (Cf. §2.4).

Ces travaux et d'autres plus récents [LYO 03] mirent également en évidence l'émergence des problèmes nouveaux liés à la réalisation de larges systèmes complexes hétérogènes multi horloges et aux contraintes de performance élevées : très faible consommation ou très haut débit ou très grande hétérogénéité des modules intégrés au sein d'un même système, impliquant une très grande flexibilité. Ces nouveaux défis ne dispensant évidemment pas le système de communication de respecter les contraintes d'implémentation physique que l'on exige de lui.

2.1.2. Les exigences de conception d'un réseau sur silicium

Ces nouvelles contraintes très fortes qui pèsent sur les réseaux de communication intégrés déterminent un ensemble d'exigences de conception que les concepteurs de ces réseaux et les intégrateurs de système vont devoir spécifier, implémenter et évaluer pour concevoir des réseaux sur silicium performants. Ces exigences, ou critères, sont les suivants :

- La performance en termes de latence et débit que peut garantir le système de communication : les systèmes sur silicium actuels intègrent plusieurs périphériques capables chacun de réaliser des traitements à plusieurs gigabits par seconde. Tous ces périphériques opérant simultanément, le réseau d'interconnexion doit leur offrir

une bande passante suffisante pour établir entre eux des communications multi-Gbit/s.

- Le placement et le routage des réseaux sur silicium qui présentent le risque d'occuper une surface de silicium plus grande que les périphériques intégrés [BER 04]. Il s'agit d'un cas extrême, mais il reste vrai que l'accroissement de la densité d'intégration ne sera d'aucune utilité si celui, parallèle, de la surface dévolue aux interconnexions, conduit à plafonner les ressources disponibles pour le traitement.
- Les temps de communication qui deviennent plus coûteux que les temps de calcul [iHPerf 00].
- Les contraintes d'ordre système deviennent prépondérantes, en particulier la vérification [SCH 04] et l'interfaçage du systèmes de communication avec les couches logicielles basses de l'application [LYO 03].
- La testabilité. Les plus complexes des réseaux de communication peuvent embarquer de la logique de tests permettant d'espionner leur états internes [HEATH 04]. Pour les plus simples, le matériel de test des composants de calcul ou de mémorisation connectés aux terminaisons suffit amplement à vérifier la bonne transmission des informations.
- L'aptitude du système de communication à synchroniser des blocs cadencés par des horloges différentes (asynchronisme), comme nous venons de le voir au chapitre 1.
- La flexibilité dans la définition des protocoles de communication. C'est une nécessité pour permettre la connexion au bus de blocs fonctionnels de nature et d'origine très diverses. L'étude de ce critère très qualitatif est développée dans le §2.2
- La flexibilité dans la définition des services offerts par le système de communication. On peut citer : la gestion de priorités, l'accès parallèle aux ressources, la mémorisation de messages…. Cette variété de services est présentée dans les §2.2 et §2.3.3
- La fiabilité des mécanismes de communication est une propriété très importante, comme en discute le §2.3. Il faut prendre en compte la fiabilité des protocoles de synchronisations au niveau le plus bas (problème de métastabilité) et aussi d'un point de vue fonctionnel, au moyen de protocoles de services.
- L'ensemble de ces exigences ou critères détermine le choix d'une topologie du réseau sur silicium synchrone ou asynchrone. L'ensemble des topologies est passé en revue au §2.4.

Ainsi les performances du réseau de communication vont dépendre de la topologie de ses composants et de leur bande passante, du nombre de liens logiques se partageant les mêmes ressources et nécessitant donc une latence d'accès à une ressource partagée, mais aussi du protocole d'échange. La consommation du réseau de communication doit être soigneusement réduite car le réseau représente une part très importante de la charge de fonctionnement d'un système sur silicium. En outre, le réseau d'interconnexion est à la fois un émetteur puissant et un récepteur sensible d'ondes électromagnétiques, pouvant parasiter l'ensemble du système.

Enfin, la surface occupée sur le circuit par le réseau doit rester la plus faible possible par rapport à la surface attribuable sur ce même circuit à la logique de calcul.

Tous ces critères ne se retrouvent pas de manière critique dans toutes les applications. Cependant une infrastructure de communication se doit aujourd'hui d'être conçue performante bien sûr, mais également flexible afin d'être réutilisable pour le plus grand nombre possible d'applications. Il s'agit de fournir la fonction « réseau de communication » comme un bloc *IP* (*Intellectual Property*) configurable. Cet objectif de réutilisation améliore les temps de cycle de conception, dans la limite où il n'aboutit pas à un système irréalisable.

2.1.3. Conclusion

La plupart des critères de conception d'un réseau sur silicium présentés ci-dessus sont clairement définis et illustrés en détails dans [GUE 00a]. Cependant certains d'entre eux revêtent une importance particulière en regard des problèmes de synchronisation déjà présentés au chapitre 1. En effet, si la résolution des problèmes de synchronisation est la motivation première à l'origine de la méthodologie de conception d'un réseau sur silicium sans horloge décrite dans les chapitres 3 à 6, les critères que nous allons développer ici n'en ont pas moins fortement influencé les travaux présentés dans la suite de ce manuscrit.

Ainsi donc les paragraphes suivants 2.2 à 2.4 discutent respectivement :

- du degré de flexibilité du réseau d'interconnexion,
- de la fiabilité et de la qualité de service des protocoles de communication,
- et enfin du choix de la topologie du réseau.

Le paragraphes 2.5 présente des réalisations reconnues de réseaux de communication intégrés sur silicium, industriels et universitaires, dédiés à des systèmes globalement synchrone. Le paragraphe 2.6 fait de même pour des systèmes globalement asynchrones.

Dans la suite de ce manuscrit, lorsque nous parlerons des critères de performance et de coût, essentiellement ces critères seront respectivement la latence et le débit pour la performance et la surface et la consommation pour le coût.

2.2. La Flexibilité

Il s'agit à la fois du degré d'adaptation du réseau d'interconnexion à différents modules périphériques hétérogènes et sa facilité de réutilisation dans de futurs systèmes.

Ces deux exigences peuvent s'avérer difficiles à concilier avec les exigences de performances d'un réseau. Par exemple, si le protocole de communication du réseau d'interconnexion est conçu spécifiquement en regard des protocoles utilisés par les périphériques, les interfaces d'adaptation entre le réseau et les périphériques seront simples de conception donc performantes. En revanche ce réseau sera difficilement réutilisable dans de nouvelles architectures avec des protocoles de communication différents devant fournir de nouveaux services. Au contraire un réseau de communication plus générique sera moins

performant car il devra développer des interconnexions spécifiques nombreuses pour adapter les protocoles. Il sera donc capable de connecter de nombreux périphériques hétérogènes et sera extensible plus aisément vers de nouvelles architectures. Sa généricité lui permet donc de pouvoir fournir une plus grande variété de communications.

Ces exemples montrent qu'il est possible de caractériser le degré de flexibilité d'un réseau de communication selon trois indicateurs :

- L'extensibilité du réseau vers de nouvelles architectures ;
- La variété des communications de niveau services offertes par le système d'interconnexion ;
- La capacité d'interfaçage avec d'autres composants, périphériques ou réseaux.

Les trois paragraphes suivants introduisent respectivement les trois critères énoncés ci-dessus et renvoient aux chapitres qui les développent plus en détails dans la suite du manuscrit.

2.2.1. L'extensibilité

L'extensibilité *(scalability)* est l'aptitude du système à supporter un nombre donné de périphériques concurrents sans altérer ni ses fonctionnalités, ni ses performances globales.

Elle dépend d'un point de vue système de la régularité de l'architecture par le choix du réseau de communication. La régularité d'une architecture est l'identification dans sa structure de la reproduction d'un motif à base de composants périphériques autour du réseau d'interconnexion. Nous distinguons aux paragraphes 2.2.1.1 et 2.2.1.2 les architectures respectivement régulières et hétérogènes. Le degré de régularité et donc d'extensibilité sera précisé pour chaque topologie de réseau d'interconnexion étudiée au §2.4. La régularité de l'architecture ne contraint en rien l'hétérogénéité des composants du système.

D'ailleurs, pour permettre la connexion au bus de blocs fonctionnels de nature et d'origine très diverses, et donc améliorer l'extensibilité du système, il est nécessaire de définir des protocoles flexibles de communication de bas niveau dits protocoles de signal ou de synchronisation, détaillés au §2.3.2.

2.2.1.1. Les architectures régulières

Ces architectures ont une topologie régulière sous forme d'assemblage systématique des composants autour du réseau de communication. Cette solution permet une systématisation ou automatisation de l'assemblage des composants. Malheureusement, ces réseaux de communication génériques deviennent les nouveaux goulots d'étranglement face à l'accroissement des exigences en performances globales des applications. Des solutions à base de topologie matricielle sont alors envisagée, comme le système picoArray de PicoChip [BAI 03]. La Figure 2-1 illustre cette topologie de 430 processeurs RISC 16 bits faiblement hétérogènes organisés en matrices de grappes de quatre processeurs.

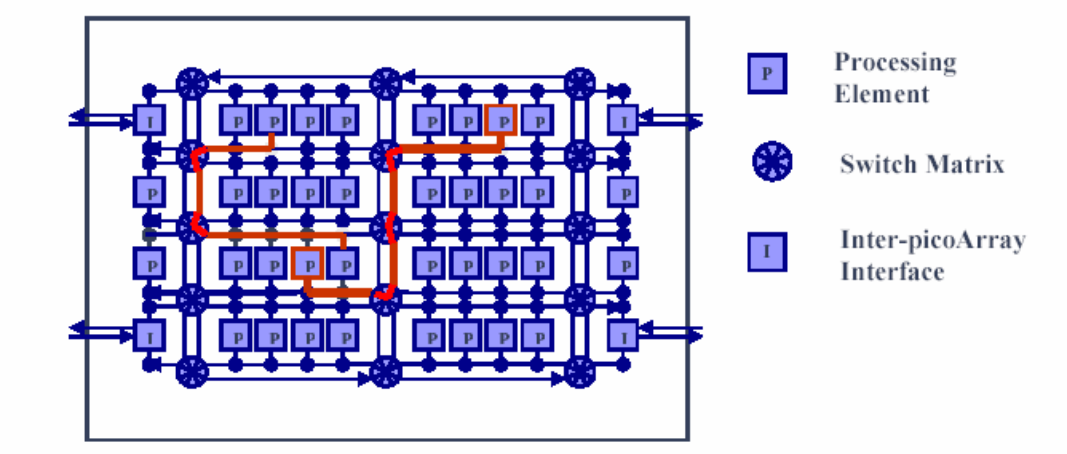


Figure 2-1 : matrice crossbar du système de processeurs parallèles picoArray

2.2.1.2. Les architectures hétérogènes

Ces architectures ne possèdent pas de motif reproductible pour constituer le système. Des ressources spécifiques sont mises en œuvre pour localement améliorer l'adéquation entre l'implémentation et les fonctionnalités exécutées. L'hétérogénéité permettant de spécialiser l'implémentation, il est plus aisé d'équilibrer les coûts avec le niveau de performance. De telles architectures sont donc préconisées pour les applications ayant un marché à fort volume. La Figure 2-2 présente par exemple l'architecture du système Prepor de STMicroelectronics, une plateforme de prototypage pour les applications dédiées aux réseaux sans fil basse puissance (*WLAN* pour *Wireless Local Area Network*).

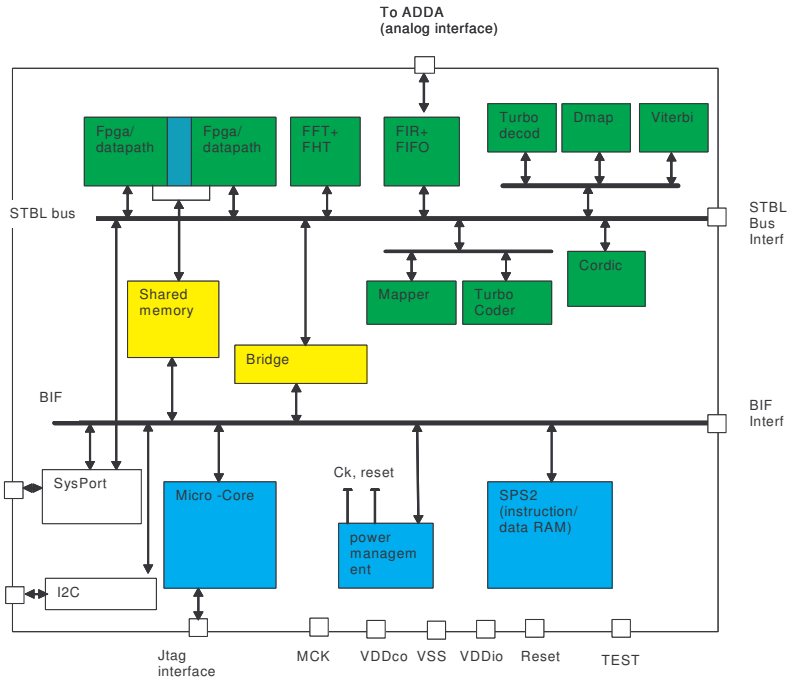


Figure 2-2 : architecture du système Prepor

Le réseau de communication ne présente pas une structure régulière capable d'être étendue. Le bus de communication BIF connecte le processeur et sa mémoire et l'unité de gestion d'énergie. Un autre bus de communication, le STBL Bus, connecte des accélérateurs matériels directement ou au moyen de sous-bus. Certains composants sont connectés aux deux bus principaux, tandis que les autres composants communiquent d'un bus de communication à l'autre grâce à la mémoire partagée et un pont d'adaptation (Cf. 2.2.3.2). Tous ces média de communication sont des implémentations spécifiques dédiées à l'architecture du système Prepor et qui permettent d'accommoder le besoin exact des performances de communication requises par l'application exigeant en particulier une très faible consommation. Son extensibilité s'avère en revanche très mauvaise.

2.2.2. La Variété et la qualité des services

La variété de services que doit offrir le réseau de communication correspond aux modèles ou modes de communication exigés par l'application. On peut citer les communications point à point ou multipoint, les transferts en rafale, avec ou non préemption, la gestion de priorités, la mémorisation de messages….

La flexibilité du réseau de communication sera d'autant plus grande que ce dernier sera capable de fournir un mode de communication adapté aux exigences spécifiques d'un composant initiateur de transactions sur le réseau. Cette variété de services répond à un besoin en variété de types de trafics des applications. Le détail des protocoles de service répondant à ces besoins est présenté au §2.3 suivant.

2.2.3. La compatibilité d'interfaçage

La facilité d'interfaçage avec d'autres réseaux ou périphériques éventuellement de natures différentes est le troisième critère de flexibilité d'un réseau de communication. Il s'agit plus exactement d'un degré d'adaptabilité. L'étude de cette compatibilité d'interfaçage est développée directement dans ce paragraphe.

La facilité de la connexion des composants périphériques au réseau va dépendre de leur hétérogénéité. La Figure 2-3 présente la structure de l'architecture régulière Prophid de Philips, composée d'un cœur de processeur logiciel RISC (*CPU*) et d'accélérateurs matériels pour le traitement du signal (*ADS*). Bien que ces blocs *ADS* puissent être réalisés par des composants matériels ou logiciels divers, leur relative homogénéité les rend facilement adaptables aux deux bus à travers une interface unifiée.

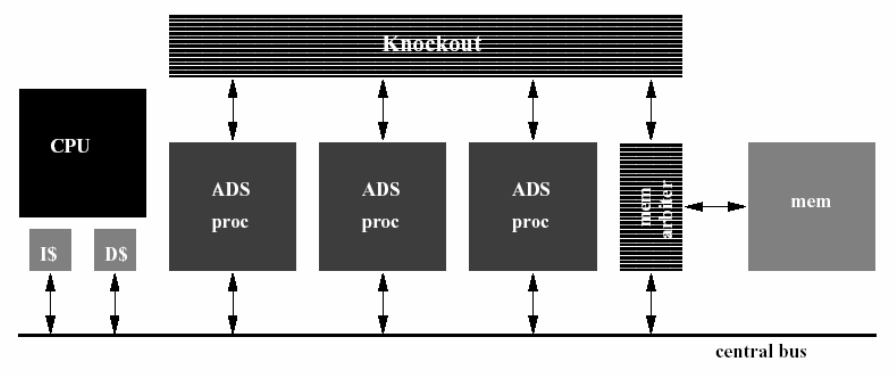


Figure 2-3 : architecture Prophid de Philips

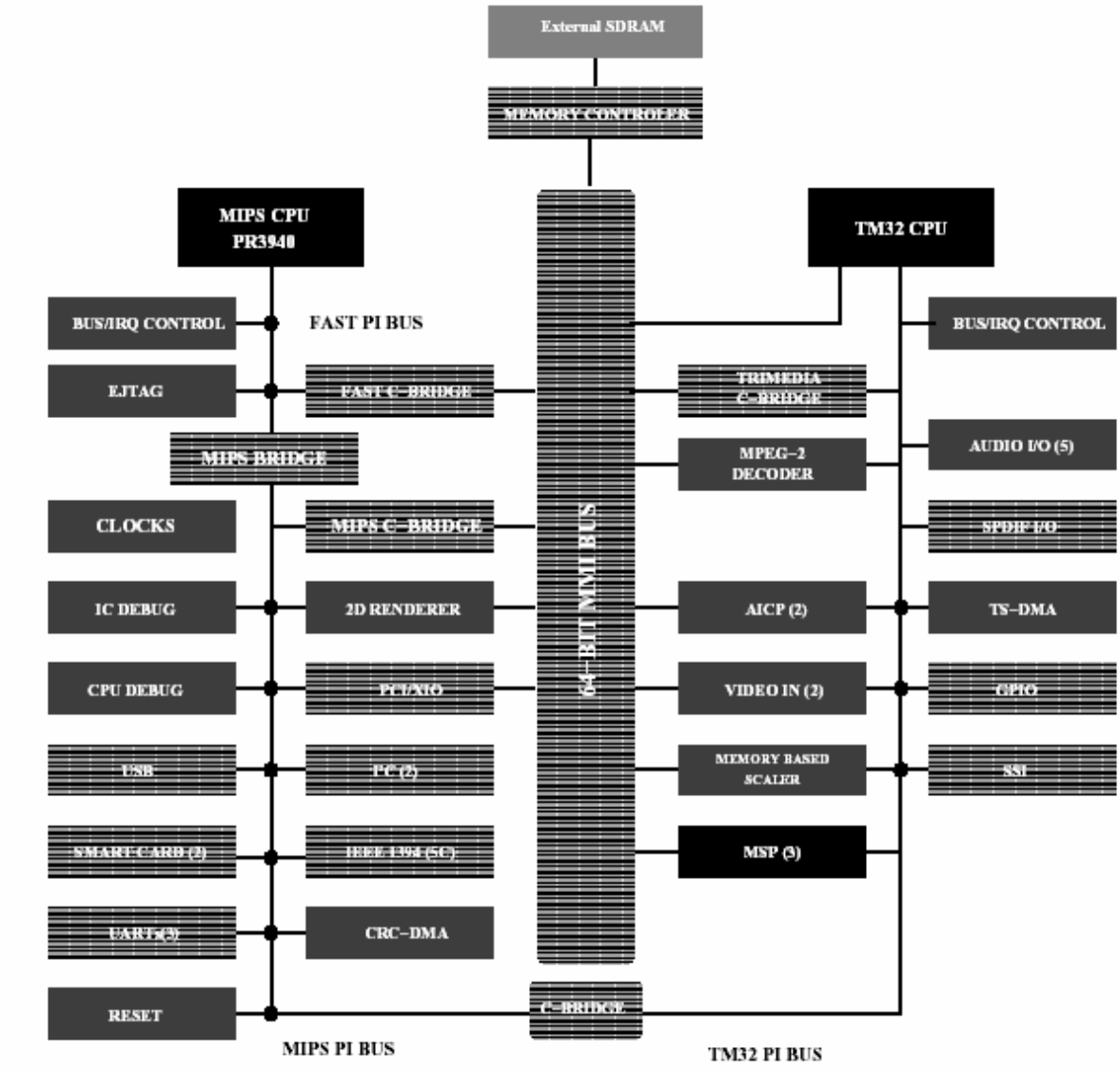


Figure 2-4 : architecture Viper de Philips

Le *Viper pnx8500* de Philips hérite pour ses caractéristiques principales de l'architecture Prophid. Son architecture, toujours régulière (Cf. §2.2.1.1), est illustrée par la Figure 2-4 et se compose de cinq processeurs logiciels, une demi-douzaine d'accélérateurs

matériels, des unités dédiées au test et des composants de gestion mémoire. Les processeurs logiciels sont hétérogènes : un cœur RISC *MIPS* 32 bits, un cœur VLIW *Trimedia* 32 bits et un cœur RISC 16 bits par unité *MSP* (2 ou 3 instances). Les composants matériels sont aussi hétérogènes : accélérateurs de traitement de l'image (*2D Graphics, AICP et Memory Based Scaler*), décodeur de flux vidéo (*Mpeg2 Decoder*), unités de test (*EJTAG, IC Debug, CPU Debug*) et composants de gestion de la mémoire (*CRC-DMA* et *TS-DMA*). Enfin la communication aussi est hétérogène et se présente sous une topologie complexe : trois bus partagés de type *PI-Bus* [OMI 95] sont utilisés pour les communications processeurs ↔ périphériques tandis que l'étranglement de l'accès mémoire est évité grâce au *MMI Bus.*

Ce système nécessite donc des ponts d'adaptation de connexions afin de prolonger les accès d'un *PIBus* vers un autre ou vers les connexions point à point du *MMI Bus*. En outre des interfaces relativement complexes d'adaptation de communication sont nécessaires pour adapter les instances de cœurs de processeurs et certains périphériques matériels préalablement conçus, aux différents bus.

Ces exemples illustrent une contrainte forte : une interconnexion de système hétérogène doit être compatible avec plusieurs modèles de communication. On pourrait imaginer des interconnexions très efficaces spécifiques à certains modèles de communication délimités, mais elles ne peuvent suffire. L'intérêt d'un système sur silicium étant la coopération de composants hétérogènes, qui peuvent potentiellement tous être amenés à communiquer, il faut donc fournir des ponts et des adaptateurs prédéfinis permettant d'offrir un interfaçage entre des systèmes hétérogènes aux protocoles de communications incompatibles. Ce problème d'interfaçage de protocoles est abordé en profondeur dans la thèse de Lyonnard [LYO 03] qui propose un outil d'assemblage de composants à l'aide de bibliothèques d'adaptateurs de communication. Nous nous contentons ici de définir le rôle de ces adaptateurs et ponts.

2.2.3.1. Les adaptateurs ou interfaces de communication

Ce sont des blocs matériels destinés à adapter le périphérique au réseau de communication, lorsque le protocole natif de cette unité matérielle est incompatible avec celui du réseau de communication. Les performances de ces interfaces sont fortement dictées par la nature de l'application et du réseau de communication :

- Pour les applications dominées par le contrôle, le non-déterminisme des communications (Cf. §4.1) rend difficile leur modélisation et contraint les outils d'exploration d'architecture à les négliger. La monopolisation des ressources de calcul pour traiter des tâches de communication nuit fortement aux performances de l'exécution des algorithmes applicatifs;
- Pour les applications à dominantes calcul, bien que le déterminisme des communications et les mécanismes de *bypass* permettent de masquer les latences, le débit des communications doit cependant être suffisamment élevé pour suivre les cadences imposées par les unités de calcul.
- Les ressources des réseaux de communication sont généralement partagées par un nombre massif de connexions logiques, l'utilisation trop longue de celles-ci peut empêcher une tâche de respecter les échéances qui lui étaient imposées.

Les fonctionnalités simples prises en charge par ces interfaces nécessitent très peu d'énergie. Les primitives de traduction mises en jeu réclament peu de silicium. Il est courant d'estimer leur occupation surfacique à moins d'un dixième de la surface totale. Suivant la nature des deux protocoles à adapter, la nécessité de découpler le noeud de calcul du réseau par *bufferisation* et/ou l'usage de multiples horloges, la conception de ces unités fluctue de la simple réutilisation de composants prédéfinis à la conception de blocs spécifiques. Les protocoles usités étant généralement issus de standards, cette étape est limitée à l'instanciation de composants et à leur dimensionnement. La faible complexité de ces unités ne supportant pas le surcoût lié à l'usage de matériel de test, on préfère utiliser la logique de test embarquée aux terminaisons des communications pour tester l'intégrité de ces dernières et donc les adaptateurs.

La fonctionnalité de ces blocs étant statiquement définie, leur conception étant abordée au cas par cas, leur flexibilité est très restreinte, voir inexistante. Pour contrevenir à ce coût systématique de conception d'interfaces d'adaptation à chaque (ré-)utilisation d'un bloc périphérique, les alliances et autres consortium ont tenté de mettre en place des *guidelines* de méthodologie de conception dans lesquelles les blocs doivent être conçus avec une interface de communication correspondant au standard émis par ces consortia. En pratique ces louables intentions rencontrent peu de succès. D'une part, il arrive fréquemment que le choix du protocole de communication soit décidé trop tardivement par rapport à l'avancement de conception des blocs. D'autre part, même les sociétés ayant spécifié leur standard de communication en utilise un grand nombre pour satisfaire les exigences des clients (internes ou externes). Enfin, ces standards ne répondent pas forcément aux besoins spécifiques de l'application. Les concepteurs les adaptent alors à leurs contraintes, mais si le label persiste, la compatibilité de protocole aura disparu le jour où ces composants voudront être réutilisés.

2.2.3.2. Les ponts

Les ponts sont une application spécifique des adaptateurs à la connexion de plusieurs bus supportant chacun plusieurs maîtres. Ils doivent donc s'enquérir de:

- L'attribution du bus. En termes d'organisation, l'initiateur de la communication ou maître prend le contrôle de son bus et accède au pont se comportant alors comme un esclave. Le pont formule alors une requête pour obtenir l'accès au bus connecté à l'esclave et agit ainsi en tant que maître sur ce bus.
- La transformation de protocole dans le cas de bus hétérogènes. Toutes données échangées par un maître sur son bus suivent un protocole d'échange qui peut ne pas être supporté par le bus connecté à l'esclave ciblé. Le pont doit alors adapter ces deux protocoles.

Ils sont généralement implémentés en logiques câblées. Leur utilisation est requise lorsque l'adaptation entre deux sous-réseaux de communication doit être opérée. Cet état de fait se présente lorsque le réseau est hiérarchisé afin de:

- offrir localement les performances requises;
- accentuer le parallélisme en utilisant des ressources de transports concurrentes;

- reporter les problèmes d'incompatibilité de protocoles des différents ensembles processeur/bus natif, sur un unique composant et permettre aussi loin que possible l'utilisation des protocoles natifs.

Pour découpler la synchronie des différents bus il est souvent fait appel à des mécanisme de mémoire tampon ou «buffer ». Les coûts et les performances relatifs de ces composants découlent donc des implémentations des machines d'états exécutant les divers protocoles et gérant ces mémoires. Ces composants doivent supporter les bandes passantes propres à chacun des bus adapté afin d'optimiser l'usage de ces ressources partagées de communication. La monopolisation des deux ressources de calculs dépréciant fortement les performances globales, une attention particulière est portée sur les performances de ces goulots d'étranglement. La complexité de ces composants restant relativement faible, leur consommation énergétique est réduite. Cette simplicité fonctionnelle résulte aussi en une occupation surfacique héritée pour l'essentiel des mémoires tampons embarquées. Ces dernières étant dimensionnées pour ne contenir qu'un petit nombre de paquets sont très peu volumineuses. Comme pour les adaptateurs, lorsque les protocoles à adapter ne sont pas des plus coutumiers, il est nécessaire de développer des modèles spécifiques aux paires de bus considérées. Qu'on réutilise des composants préconçus ou qu'on les conçoive spécialement, il est nécessaire de dimensionner les mémoires tampons pour efficacement découpler les deux bus et ainsi diminuer les latences. La complexité de ces composants nécessite rarement l'usage de ressources de test. Ces composants répondent à une spécification bien précise : adapter deux ressources de communication en supportant une bande passante donnée. La seule flexibilité envisageable concerne la mise à l'échelle de la gestion des canaux concurrents connectés (ex: nombre de lignes d'un crossbar).

2.2.4. Conclusion

La flexibilité recouvre à la fois le degré d'adaptation du réseau d'interconnexion à différents modules périphériques hétérogènes et sa facilité de réutilisation dans de futurs systèmes. Nous avons caractérisé le degré de flexibilité d'un réseau de communication selon trois indicateurs.

Le premier indicateur est la capacité d'interfaçage avec plusieurs modèles de communications supportés par d'autres composants, périphériques ou réseaux, à l'aide de ponts et interfaces de communication. Ensuite, l'extensibilité est l'aptitude du système à supporter un nombre donné de périphériques concurrents sans altérer ni ses fonctionnalités, ni ses performances globales. L'extensibilité du réseau dépend de la flexibilité des protocoles de niveau signal et de la régularité de son architecture. Une architecture régulière assure une bonne extensibilité en pénalisant les performances. Enfin, la variété des communications de niveau services offerte par le système d'interconnexion répond à un besoin en types de trafics différents des applications. Ces protocoles de niveau signal, ou synchronisation, et de services sont abordés dans la section suivante.

2.3. Les protocoles de communication : synchronisation & services

La spécification d'un réseau sur silicium doit donc répondre aux critères de performance et de coût énoncés aux deux paragraphes précédents. Il est donc nécessaire de réaliser une implémentation la plus performante possible. La couche physique bien sûr doit répondre aux problèmes tels que l'optimisation des délais et des répéteurs dans le réseau (Cf. chapitre 7), le bruit, la surface et la synchronisation (Cf. chapitres 4 et 6). Dans une architecture GALS, la synchronisation se faisant par communication (Cf. §1.2), le protocole asynchrone de communication devient alors fondamental dans la spécification du réseau.

En effet ce protocole de plus bas niveau va spécifier le nombre de signaux et le nombre de transitions de chacun de ces signaux pour réaliser une action atomique de communication. Il va donc être déterminant pour les performances et les coûts du réseau de communication. La robustesse de ce protocole de synchronisation (problème de métastabilité abordé au §4.1.2) s'avère également déterminante pour la fiabilité du système.

Cependant, ce protocole de synchronisation au niveau signal ne suffit pas à déterminer les performances et les coûts d'un réseau d'interconnexion. Avec l'évolution des topologies depuis le bus centralisé vers des architectures distribuées, inspirées des topologies des réseaux de calculateurs haute performance et des routeurs de télécommunications (Cf. 2.4), il est devenu nécessaire de définir des protocoles de communication de plus haut niveau d'abstraction permettant de mettre en forme des communications aux formats complexes, construites à partir des actions atomiques des protocoles de niveau signal. Nous regroupons ces protocoles sous le nom de protocoles fonctionnels ou de services (récemment définis par le terme *QNoC* pour *Quality of Service NoC* par [BOL 03]). Un réseau sur silicium proposant une grande variété de services, adaptés à l'application cela s'entend, va permettre de parvenir aux exigences de performance et de coût du système complet. La fiabilité des mécanismes de communication étant une propriété très importante des réseaux sur silicium, une catégorie de services va en outre offrir la possibilité d'améliorer la fiabilité des communications en détectant et en corrigeant des erreurs de transmission, ou en autorisant la réémission des données erronées ou perdues.

Comme nous insistons sur l'importance de la fiabilité des mécanismes de communication pour un réseau de communication, le paragraphe 2.3.1 aborde la question de la fiabilité des protocoles de synchronisations et aussi d'un point de vue fonctionnel, au moyen de protocoles de services. Ces protocoles de synchronisation et de service sont ensuite présentés respectivement aux paragraphes 2.3.2 et 2.3.3.

2.3.1. La Fiabilité

Par suite d'un dysfonctionnement même temporaire, l'interconnexion peut faillir à délivrer la donnée[5] prévue. L'échec de transmission peut se produire selon trois angles différents :

- La donnée est transmise de manière erronée. La faute peut venir d'une perturbation de l'environnement, d'une mauvaise régénération du signal sur une ligne longue ou d'erreurs d'encodage. Il est évident que les deux derniers points ne sont pas admissibles. Quant à la tolérance aux fautes, elle dépasse le contexte de ces travaux. Le lecteur curieux trouvera son bonheur dans [QLF 03].
- La donnée n'est pas transmise. Il y a eu une collision dans le réseau ou un problème de synchronisation avec un périphérique et la donnée est perdue. Il existe de nombreuses politiques pour gérer ces défaillances. Les premières imposent au réseau d'assurer le bon transport des données. Le réseau doit alors s'adapter aux exigences de protocoles des périphériques pour ne pas perdre de données. De telles architectures peuvent coûter chères en éléments de mémorisation, mais permettent une réutilisation aisée de composants. La deuxième politique privilégie une latence élevée en bon fonctionnement. Par contre en cas de perte de donnée, le périphérique destinataire doit être capable de détecter le dysfonctionnement et l'émetteur doit pouvoir réémettre le paquet (attention à la date de première émission en cas asynchrone).
- La donnée n'arrive pas dans les spécifications temporelles du système. C'est le problème de métastabilité développé au Chapitre 4 : le système n'est pas capable de prendre une décision sur la valeur de la donnée au moment de son échantillonnage et son temps de réponse devient non prédictible. La contrainte de fiabilité, outre la conservation des données, impose alors une latence de réponse au réseau d'interconnexion.

L'augmentation des fréquences de fonctionnement, du nombre de composants d'un système, travaillant en outre à des fréquences différentes, ainsi que les besoins croissants en communication rendent de plus en plus difficiles la conception d'un système fiable vis-à-vis de l'application. Cette fiabilité est exprimée par le temps moyen entre deux défaillances du système (MTBF), que l'on cherche à rendre le plus grand possible, en général d'un ordre de grandeur supérieur à la durée de vie du circuit [GIN 02]. Ceci fixe le niveau de confiance que l'on exige des périphériques et de l'interconnexion. Le problème du coût de synchronisation nécessaire pour rendre le réseau d'interconnexion fiable est abordé dans la section 4.2.

Lorsque la donnée est transmise de manière erronée ou n'est pas transmise, des protocoles de service peuvent être mis en place pour détecter ces erreurs de transmission, puis pour les corriger ou autoriser la réémission des données erronées ou perdues. A ces méthodes

[5] Nous entendons par « donnée », l'ensemble des signaux transmis au sein d'un paquet élémentaire de communication : l'erreur peut donc survenir sur les données proprement dites, mais aussi sur le champ d'adressage ou de contrôle du paquet transmis, affectant de manière variable le système

correctives on préfère cependant les méthodes préventives de robustesse des communications (prévention d'erreur) et de mémorisation dans le réseau (prévention de perte). Lorsque la donnée n'arrive pas dans les spécifications temporelles du système, c'est le protocole de synchronisation et les circuits de synchronisation qui doivent être revus.

2.3.2. Le protocole de niveau signal ou protocole de synchronisation

Le protocole de signal garanti le bon acheminement des informations entre les deux extrémités de la communication. Il détermine l'aptitude du système de communication à synchroniser des blocs cadencés par des horloges différentes (asynchronisme). Il fournit le degré de fiabilité dit matériel du système, et s'il est jugé insuffisant, doit être renforcé par des solutions de *QNoC* fournies par les protocoles de services. Ce protocole de plus bas niveau va spécifier le nombre de signaux et le nombre de transitions de chacun de ces signaux pour réaliser une action atomique de communication, selon un protocole de communication choisi pour fournir le meilleur compromis entre performances, coûts et fiabilité, en fonction des exigences du système. Ce protocole de niveau signal peut être synchrone sur une horloge ou asynchrone (Cf. §1.2.2.2 pour l'abus de langage).

2.3.2.1. Actions atomiques sur les signaux

Le protocole de niveau signal ou de synchronisation a pour fonction la réalisation des actions atomiques sur les signaux, destinées à permettre la synchronisation entre deux composants matériels. Ces actions atomiques que nous relevons au nombre de six permettent de construire n'importe quel protocole de communication synchrone ou asynchrone. C'est au concepteur de garantir la fiabilité de ce protocole de niveau signal, les performances qu'il offre au système en terme de cycle de communication (latence d'achèvement d'une communication depuis son initiation jusqu'à sa terminaison) et la facilité d'implémentation à partir de ce protocole de synchronisation de la couche supérieure de protocoles de services (Cf. §2.3.3).

Ces actions atomiques sur les signaux sont :

3. attente d'un événement sur un signal de contrôle en entrée (*spooling*)
4. assignation d'une valeur sur un signal de contrôle en sortie
5. lecture d'une valeur sur un signal de donnée en entrée
6. écriture d'une valeur sur un signal de donnée en entrée
7. test d'absence d'événement sur un signal de contrôle en entrée (*spooling*)
8. attente pendant un temps fixe (*watchdogs*).

A noter à propos du *spooling* : cette attente ou ce test est en général actif dans un circuit synchrone mais elle est intrinsèquement passive pour un circuit asynchrone. C'est-à-dire qu'un circuit asynchrone reste naturellement à l'écoute de cet événement sans l'échantillonner, alors qu'un circuit synchrone l'échantillonne à chaque cycle d'horloge.

2.3.2.2. Protocoles de niveau signal synchrones

Les protocoles de niveau signal synchrones s'associent à toutes les techniques de synchronisation par horloge présentées au §1. Le système étant entièrement synchronisé à la même fréquence, les protocoles synchrones tirent parti de cet avantage pour réaliser une synchronisation implicite des signaux de contrôle des communications. En effet, ces signaux peuvent se passer d'une phase de remise à zéro et rester valides autant de cycles d'horloge que nécessaire : c'est l'échantillonnage de leur valeur au front d'horloge qui rafraîchit leur validité. Pour les données, les protocoles synchrones visent la meilleure performance possible en plaçant, autant que faire se peut, les données nouvelles sur le bus correspondant à chaque cycle d'horloge. La latence de gestion du réseau et la mise en forme des données étant souvent de plusieurs cycles, ces protocoles travaillent en mode pipeliné.

Ainsi le protocole du bus partagé AHB (*Advanced Microcontroller Bus Architecture*) [ARM 99] développé par ARM est représentatif des protocoles synchrones des bus centralisés. Il supporte les transactions en rafales, non préemptives et différées (Cf. §2.3.3.1 et 2.3.3.2). Pour de meilleures performances, les données associées à une adresse sont présentes sur le bus un cycle après l'adresse, ce qui permet un fonctionnement en pipeline. Un certain nombre de signaux de contrôle associés permettent de gérer la validité des transactions. Voir [ARM 99] pour plus de détails.

2.3.2.3. Protocoles de niveau signal asynchrones

Les protocoles asynchrones permettent de synchroniser deux composants dans un environnement concurrent du type globalement asynchrone et localement synchrone présenté au §1. Le choix du séquencement interne par horloge (composant synchrone) ou par communication (composant asynchrone) des composants va imposer ou non des circuits de synchronisation spécifiques abordés dans la suite de ce manuscrit (Cf. §4.2 et §6).

Mais le fait d'avoir un composant synchrone ou asynchrone va également contraindre le choix du codage de l'information au sein du protocole de communication asynchrone. En effet, il existe en technique de conception asynchrone plusieurs codages possibles du format des données, selon le modèle temporel choisi. Les codages possibles (Cf. explications détaillées au §3.1.3) sont :

- le codage "données groupées" qui nécessite des hypothèses temporelles et est supporté par les circuits synchrones moyennant un coût de synchronisation supplémentaire ;
- les codages insensibles au délai à trois ou quatre états, plus robustes mais plus coûteux en surface que le codage "données groupées" ;

Les différents modèles temporels sont présentés Figure 2-5 et expliqués en détail au §3.1.6. Plus le fonctionnement respecte fondamentalement la notion d'asynchronisme et plus le circuit est robuste et donc fiable (Cf. §1.3.4 et §1.3.5), mais complexe.

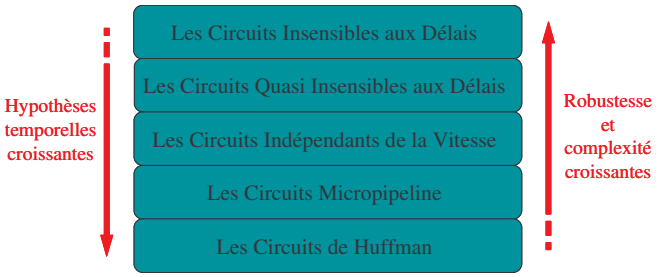


Figure 2-5 : Classification des circuits asynchrones.

Les composants synchrones nécessitant des hypothèses temporelles fortes ne vont donc pas supporter les codages des données insensibles au délai. Le chapitre 4 et notamment la section 4.3 présente une méthodologie de conception permettant d'adapter les modules synchrones d'une architecture GALS avec un réseau de communication sans horloge implémentant une classe de circuits asynchrones parfaitement fiables. Le paragraphe 4.3.3.3 en particulier illustre l'ensemble des circuits nécessaires pour parvenir à assembler de manière robuste et performante des composants périphériques synchrones d'horloges distinctes avec un cœur de communication localement sans horloge, tant du point de vue des protocoles de synchronisation que des protocoles de services.

La nature des protocoles asynchrones proprement dits est indépendante du modèle temporel choisi. Ces protocoles de communication peuvent être à quatre ou deux phases et sont expliqués en détail en section §3.1. La localité de ces protocoles les rend très flexibles (Cf. 1.3.4) et améliore la modularité et l'extensibilité du système.

L'utilisation judicieuse du protocole deux ou quatre phases et du modèle temporel en termes de performances et de coûts est abordée au chapitre 7.

2.3.3. Le protocole fonctionnel ou de services

Le protocole de communication fonctionnel ou de services permet de répondre aux exigences de performances et de fiabilité du système, au moyen d'une variété de services adaptés à l'application. La spécification fonctionnelle du réseau de communication décrit ce protocole de services et détermine l'action de communication proprement dite : quelles sont les fonctions ou services offerts par le réseau d'interconnexion ? La détermination de services adaptés à l'application va dépendre des paramètres de spécification de l'architecture du système, à savoir entre autres :

- le nombre de périphériques, avec éventuellement la décision du nombre de maîtres et d'esclaves ;
- le degré de parallélisme d'accès aux ressources (nombre de communications capables de s'établir en parallèle), la taille des bus de données et la fréquence de fonctionnement du réseau, paramètres exigés pour obtenir la bande passante nécessaire à l'exigence de communication du système ;
- la taille d'une communication, depuis un mot unique à une taille potentiellement infinie ;
- le type de transactions (*split*, *burst*, …) ;

- le degré et le type de mémorisation de messages (mémoires de bufferisation pour les interfaces de communication et piles de contexte pour les transactions interruptibles) ;
- la gestion des erreurs (Cf. §2.3.1) ;
- la possibilité de reconfiguration statique ou dynamique,… ;

Pour préciser le propos de l'introduction de cette section 2.3 sur les protocoles de services, l'évolution des topologies depuis le bus centralisé vers des architectures distribuées, a rendu nécessaire d'enrichir ces protocoles de communication de plus haut niveau d'abstraction, qui existaient déjà pour les bus centralisés mais offraient une variété réduite de services, contraints à des mécanismes d'arbitrage centralisé inefficaces à gérer des applications au trafic élevé (Cf. §2.4.3 la conclusion sur la comparaison des topologies). La variété de services ne doit cependant pas faire de la surenchère pour éviter de pénaliser d'une inutile latence le système.

Ces protocoles de service permettent d'élaborer des communications aux formats complexes, construites à partir des actions atomiques des protocoles de niveau signal indifféremment synchrones ou asynchrones. Il est important de noter que la mise en forme des communications utilisant ces protocoles de services doit être faite par le périphérique et non par le réseau lui-même, comme nous le verrons au paragraphe 4.3. Le dimensionnement des composants qui implémentent la spécification des fonctionnalités du réseau de communication doit être très soigné car ces services, destinés pour la plupart à améliorer latence et débit, influent directement sur les performances de l'ensemble de l'application. L'impact sur les coûts est en général moins sensible mais ne doit bien sûr pas être négligé non plus.

Les services offerts par le réseau d'interconnexion n'ont dans l'absolu pour limite que l'imagination des concepteurs. En pratique on distingue trois familles de services :

9. les services d'arbitrage pour résoudre les contentions d'accès à un nœud du réseau (Cf. §2.4.2) ou à une ressource périphérique ;
10. les services de transactions destinés à mettre en formes les données pour optimiser les performances de la communication ;
11. les services de fiabilité pour détecter les erreurs de transmission, puis pour les corriger ou autoriser la réémission des données erronées ou perdues.

Nous considérons cependant que les lignes de communication utilisant des protocoles asynchrones sont suffisamment fiables, ou robustes (Cf. §4.2), et que la capacité de mémorisation dans le réseau est une méthode préventive préférable à l'utilisation des services de fiabilité. Cette mémorisation permet d'adapter les fréquences des différents domaines de fonctionnement et d'améliorer le débit des communications en pipelinant les transferts (Cf. §4.3). Nous détaillons donc ici seulement les services spécifiques d'arbitrage et de types de transactions autorisées (modes de transfert des données sur les canaux du réseau) pour construire un protocole de communication de niveau services.

2.3.3.1. Les services d'arbitrage

Pour résoudre le problème de la compétition entre plusieurs composant pour l'accès à une unique ressource partagée, ceux-ci doivent d'abord soumettre une requête à l'arbitre qui en élira un unique. L'élu disposera de la ressource jusqu'à ce qu'il n'en ait plus besoin ou bien que l'arbitre le somme de la rendre. La transaction du demandeur mis en attente est mise en attente et dite différée. Dans le cas de l'accès de N composants vers P ressources, c'est également l'arbitre qui est en charge de sélectionner la ressource qui doit répondre à la transaction. Il existe deux grandes familles d'arbitre.

Les arbitres équitables

Un arbitre équitable est un circuit qui donnera accès à une ressource commune de façon équilibrée entre les différents canaux souhaitant accéder à cette ressource. Un arbitre est dit fortement équitable lorsque la requête d'une communication en attente est acquittée après un nombre fini et borné de requêtes sur les autres demandeurs. Un arbitre est dit faiblement équitable lorsqu'une requête est servie après un nombre fini mais non borné d'autres requêtes.

La Figure 2-6 présente la réalisation d'un arbitre équitable proposé par Martin. Il s'agit d'un arbitre asynchrone en protocole quatre phases avec prise en compte des requêtes grâce à des synchroniseurs (Cf. 4.1.3) notés SYNC. Cette réalisation est coûteuse en énergie car elle teste en permanence l'activité des canaux [RIG 02a].

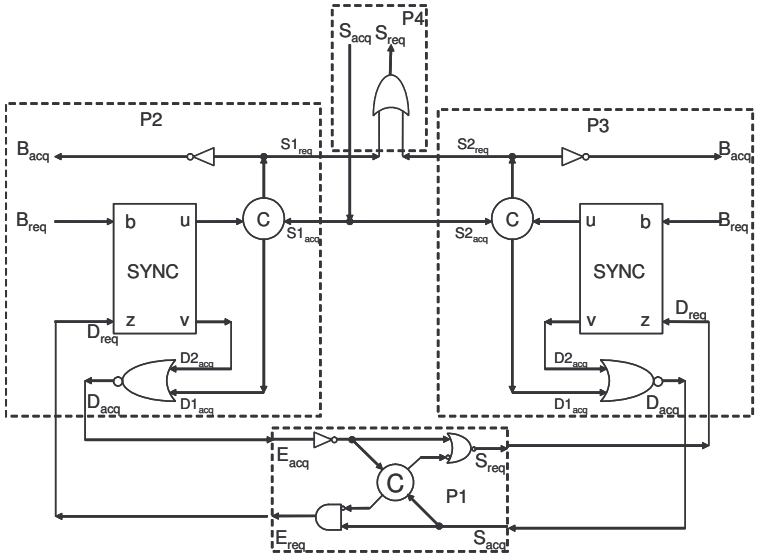


Figure 2-6 : Arbitre équitable quatre phases parallélisé [RIG 02a]

Les arbitres équitables conviennent mal à des architectures GALS où des composants hétérogènes peuvent avoir des vitesses de fonctionnement et des besoins en communications très variables. En outre, l'arbitrage dans les nœuds d'un réseau distribué peut intervenir dans les décisions de routage et nécessite donc d'implémenter des algorithmes plus complexes dits à priorité.

Les arbitres à priorité

Ces arbitres implémentent une politique de choix parmi les requêtes à servir. La classification de ces arbitres repose sur la façon dont le service d'une ou des ressources communes est partagé entre les composants demandeurs. Ces arbitres peuvent malgré tout tenir compte de l'historique du traitement des demandes pour équilibrer le service entre les demandeurs, de manière à être faiblement équitables.

En pratique, on distingue plusieurs politiques d'attribution telles que :

- l'élection d'un demandeur parmi tous ceux qui réclament l'accès en fonction de priorités qui leur sont statiquement assignées *(fixed-priority arbiter)* ;
- le choix d'un demandeur parmi tous ceux qui sont en attente en fonction de priorités dont les valeurs sont dynamiquement et circulairement assignées. On parle alors d'ordonnancement par priorité tournante ou tourniquet *(round-robin arbiter)* ;
- l'attribution systématique et cyclique par tranche de temps ou Time Division Multiple Access (TDMA) du réseau de communication à tous les demandeurs (comprenant ceux qui n'en n'ont pas l'usage) [FESQ 97] ;
- l'élection systématique et cyclique d'un demandeur par libération et circulation d'un jeton *(token-ring arbiter)*, sans tenir compte de la réelle nécessité ;
- l'élection par priorités dynamiques en fonction de la périodicité des communications *(Rate Monotonic Analysis (RMA))* ;
- l'attribution par priorités dynamiques selon l'ordre d'apparition des requêtes ou *First-In First-Granted (FIFG)* ;
- l'élection par priorités dynamiques en fonction de la proximité d'une communication de son échéance ou *Earliest-Deadline First (EDF)*.

L'étude détaillée des services et des ressources d'arbitrage d'un réseau d'interconnexion est abordée au chapitre 5.

2.3.3.2. Les services de transactions

Les services de transaction sont destinés à mettre en forme les transferts de données de manière à optimiser les performances de la communication. La taille atomique d'un transfert de données est appelée un mot de communication. Ce mot peut être assemblé à plusieurs autres pour former un paquet de communication. L'émission de ce paquet de taille N (avec N nombre de mots ≥ 1) depuis un module initiateur vers un autre module récepteur constitue l'opération de transaction ou de transfert. Lorsque le paquet est de taille supérieure à un mot, on parle de transaction en rafale (*burst transaction*). Ce type de transaction en rafale est couramment utilisé car il offre le meilleur compromis entre le gain en performance de débit et le coût de latence de mise en forme des données. La Figure 2-7 présente le format usuel d'un mot de communication dans une transaction par paquet. La définition du format et des propriétés du mot et du paquet constitue le protocole de services.

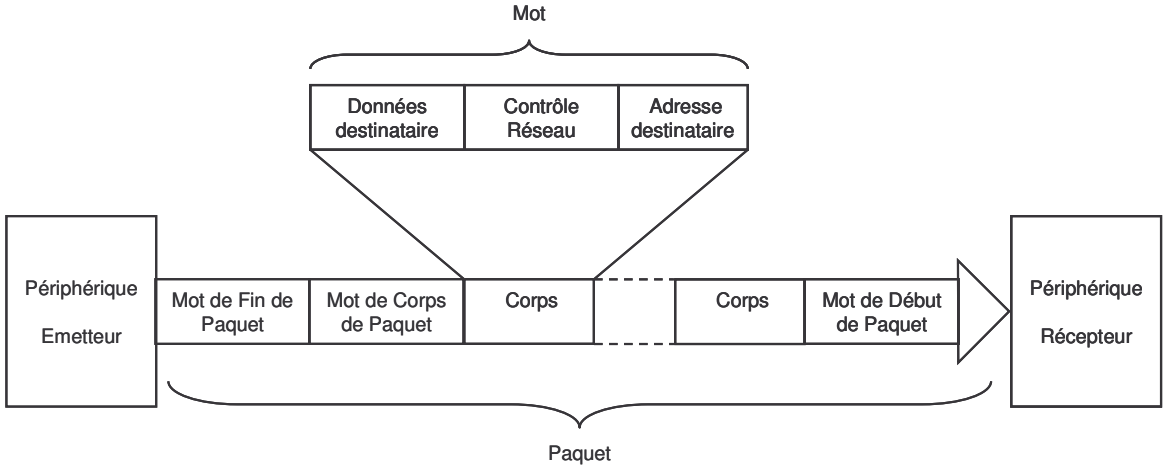


Figure 2-7 : format d'un mot de communication dans une transaction par paquet

Le format du paquet peut être de taille fixe unique ou déterminées ou encore de taille quelconque. Le premier champ du format du mot, appelé Adresse Destinataire constitue l'Adresse du périphérique récepteur de la transaction. Dans un mode rafale, ce champ n'est nécessaire que pour le mot de début de paquet, puisque tous les mots suivants seront routés vers cette même Adresse. Le deuxième champ Contrôle Réseau contient toutes les propriétés de services nécessaires à la bonne gestion des mots et du paquet par le réseau d'interconnexion. Ce champ va donc contenir les trois familles de services précédemment identifiées :

- Le degré de priorité associé à la transaction. Il s'agit d'un service de priorité.
- L'indication du type de transaction :
    - en rafale avec ou sans préemption. Avec préemption, le paquet peut être interrompu en cours de transfert par un paquet de communication plus prioritaire. Il est mémorisé par le réseau en attendant d'être restauré ;
    - Il existe d'autres types de transactions plus spécialisés en fonction des besoins de l'application. Par exemple le mode découpé (*split transaction*) où les transactions de commande/réponse entre deux modules supportent d'être séparées par d'autres transactions insérées par le réseau de communication.
- Eventuellement un codage de redondance d'information pour la détection et la correction d'erreurs, comme les codes CRC. Il s'agit des services de fiabilité ;

Le troisième champ Données Destinataire contient l'ensemble des informations destinées au périphérique récepteur de la transaction. Il s'agit des données proprement dites mais également de toutes les informations nécessaires à ce périphérique pour la gestion de ces données. Par exemple dans le cas d'une mémoire dans laquelle on désire écrire, ce champ contiendra la donnée, la commande d'écriture en mémoire et l'adresse associée.

D'ailleurs, du point de vue des périphériques, le protocole de communication de niveau service est transparent. Toutes les opérations de transfert de données sont vues par un périphérique soit comme des écritures (le maître envoie des données à l'esclave) soit comme des lectures (le maître demande des données à l'esclave qui les lui renvoie).

Ce protocole de communication de niveau service peut lui-même servir de socle à la construction de protocoles de communication de niveau d'abstraction encore plus élevé, permettant de mettre en forme des communications très complexes. C'est le cas par exemple de Bolotin et Ginosar qui définissent dans [BOL 03] quatre classes de services de haut niveau, chaque classe faisant appel à un jeu de protocoles de niveau service tel que nous venons de les définir :

- Les services de Signalisation (*Signaling*) couvrent les messages urgents et courts (paquets de faible taille) de plus haute priorité, comme les interruptions ou les signaux de contrôle ;
- Les services Temps Réel (*Real Time*) garantissent une bande passante efficace (notion de "*guaranteed throughput"*) aux applications temps réel comme les traitements audio ou vidéo. Ce service utilise des paquets de grande taille mais sans l'emploi de canaux virtuels trop coûteux en latence (Cf. §2.4.2.4) ;
- Les services de Lecture/Ecriture (*Read/Write*) modélisent un format de communication compatible avec un bus partagé (Cf. §2.4.2.1) ;
- Les services appelés Transfert de Blocs (*Block Transfer*) pour les opérations nécessitant les plus grandes communications (messages longs ou blocs de données volumineux), comme les opérations de DMA ou la gestion de cohérence des mémoires de cache.

La Figure 2-8 illustre une séquence de communication synchrone utilisant les protocoles de service de haut niveau du QNoC.

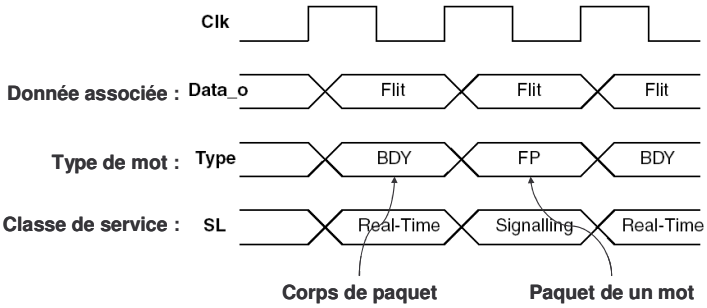


Figure 2-8 : exemple de séquence de protocoles de service de haut niveau du QNoC

Ces quatre classes de services permettent de couvrir l'ensemble des besoins en communication existants pour des systèmes sur silicium. Ces services sont implémentés en modifiant une architecture générique de réseau sur silicium. Cette architecture générique est un réseau à maille irrégulière appelé *QNoC* et dont la topologie est présentée sur la Figure 2-9. La Figure 2-10 détaille l'architecture du routeur d'un nœud du réseau, où l'on retrouve pour chaque lien de communication point à point quatre formats de communication correspondant aux quatre classes de services, qui sont routés au moyen d'un crossbar complet (Cf. §2.4.2.3).

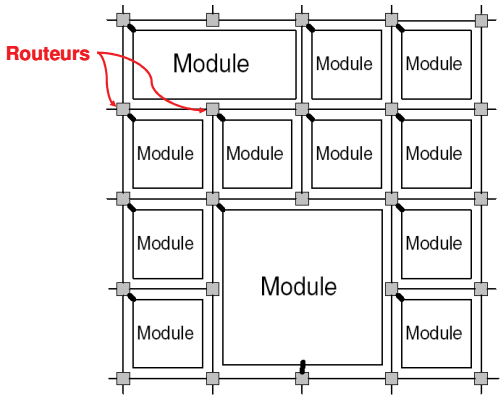


Figure 2-9 : Topologie en réseau à maille irrégulière du QNoC

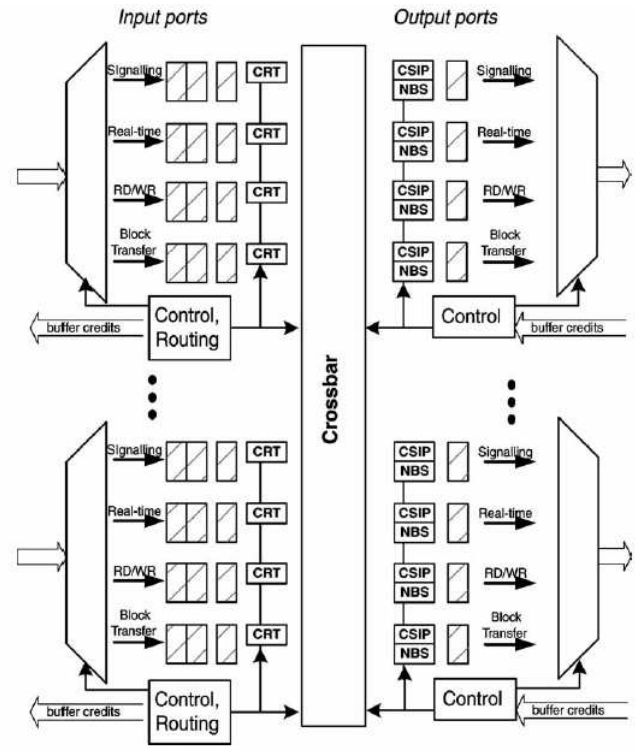


Figure 2-10 : architecture du routeur d'un nœud du réseau.

2.3.4. Conclusion

Nous venons de passer en revue les paramètres de fiabilité, ainsi que les critères de synchronisation et de qualité de service des protocoles de communication, nécessaires à la conception d'un réseau sur silicium.

L'ensemble de ces critères associés aux exigences des précédentes sections 2.1 et 2.2 participe de la détermination d'une topologie du réseau sur silicium synchrone ou asynchrone, qui sont passées en revue au §2.4.

2.4. Les topologies

L'augmentation constante du nombre de fonctions intégrées dans une seule puce a fortement augmenté les contraintes et les exigences sur les réseaux de communication. Les concepteurs de ces réseaux et les intégrateurs système se sont tournés vers les techniques éprouvées des réseaux à grande échelle utilisés dans le monde des télécommunications et des calculateurs parallèles haute performance [iHPerf 00] [FESQ 97].

Les réseaux de communication présentent donc aujourd'hui une grande variété d'implémentations physiques, qu'il serait difficile de toutes énumérer. Ce paragraphe présente donc les principes des familles de topologies les plus utilisées, sachant que de nombreuses solutions spécifiques sont développées à partir de ces familles pour adapter le réseau de communication à l'application cible. La Figure 2-11 présente ainsi le schéma de principe de quelques unes des topologies les plus couramment employées.

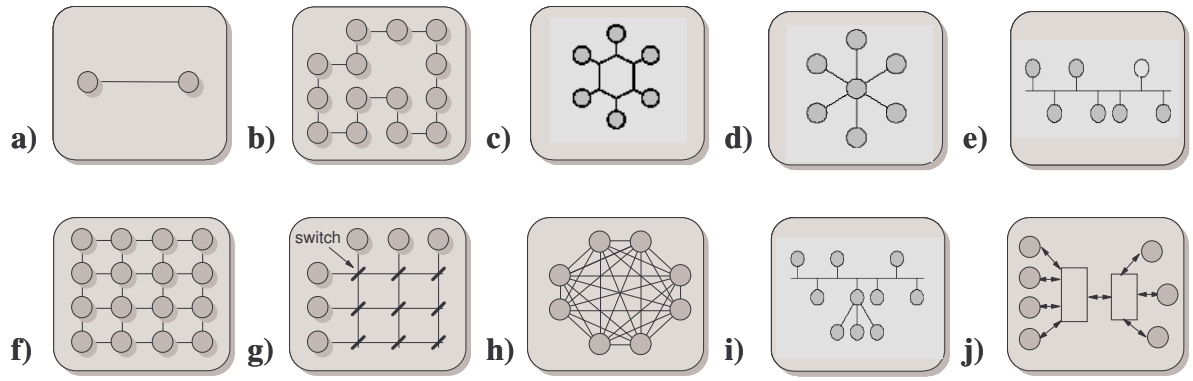


Figure 2-11 : quelques topologies d'interconnexions courantes [FESQ 97] : a) liaison point à point, b) anneau simple, c) anneau découplé, d) étoile, e) bus centralisé, f) réseau à maille complètement connecté, g) crossbar complet, h) interconnexions totales, i) topologie hybride bus + étoile, j) liaison des nœuds à des commutateurs.

Ces réseaux intégrés sont composés de ressources de routage et de transport de l'information, mais aussi d'unités de décision de l'attribution de ces ressources lorsque leur usage est partagé par l'implémentation de plusieurs liens logiques de communication concurrents. Ces ressources doivent à la fois permettre l'échange de données entre périphériques – on associe les termes chemin de données ou « *datapath* » à cette utilisation performante, mais très déterministe des composants de communication – et la synchronisation et la configuration/contrôle entre deux périphériques distants et concurrents – on parle alors d'utilisation non déterministe et fortement orientée contrôle de l'application. Le chapitre 3 présentera ces ressources critiques que sont le transport de l'information, le contrôle et la synchronisation. Ces ressources ou modules critiques seront détaillés respectivement aux chapitres 6, 5 et 4.

2.4.1. Chemins de communication point à point

Il s'agit de l'utilisation de ressources de transmission spécifiques et dédiées à l'implémentation d'un lien entre deux éléments.

La ressource de transmission étant dédiée à une unique communication, elle est toujours disponible pour cette dernière et peut ainsi lui offrir sa pleine bande passante. De plus, du fait de la simplicité du contrôle de telles connexions, les débits y sont élevés et les latences dépendent principalement de la longueur du chemin. Par contre la surface de ces connexions sera proportionnelle au nombre de liens.

L'effort d'intégration est très faible grâce à la complexité réduite des unités de contrôle. La testabilité est inexistante, et reportée aux tests des extrémités afin de ne pas apporter de surcoût prohibitif. Ces connexions sont non extensibles et spécifiques à un protocole.

Les architectures cascadées (« pipeline ») en sont très friandes, en particulier les applications orientées traitement de données et nécessitant des latences faibles et des débits importants qu'une ressource partagée ne peut offrir. Ces communications point à point sont avantageuses pour les communications longues distances, mais cela nécessite de connaître très tôt dans le flot de conception le placement des blocs périphériques.

2.4.2. Les topologies multipoints

Dans l'ensemble des topologies présentées ci-dessous, les besoins gourmands en performance ont conduit de manière universelle au développement de réseaux d'interconnexion constitués de bus de données et d'Adresses parallèles non multiplexés.

Le bus centralisé est en abandon progressif en tant que médium de communication principal par l'ensemble des acteurs qui se veulent à la pointe des solutions "système sur silicium". Le marché se déplace, de plus en plus rapidement d'ailleurs, vers des solutions globales de transport et de management du trafic au sein des systèmes sur silicium, s'inspirant fortement des réseaux de calculateurs parallèles [ART 04b].

Nous prenons toutefois la peine de l'expliquer car il fut longtemps le seul représentant de la famille des réseaux intégrés et il reste encore aujourd'hui le standard de facto dans de nombreuses applications. En outre, les systèmes sur silicium présentant une forte hétérogénéité de composants possèdent rarement un seul type de réseau embarqué. Ils privilégient d'avantage l'utilisation conjuguée [MPSoC 00] :

- D'un réseau central d'interconnexions point à point, du type maille ou crossbar (Cf. 2.4.2.3 et 2.4.2.4) pour les besoins de parallélisme élevés entre sous-ensembles de composants du système,
- avec un ou plusieurs bus partagés pour les communications entre les composants de chaque sous-ensemble du système, ces communications nécessitant beaucoup de contrôle et une économie de surface.

2.4.2.1. Le Bus centralisé ou partagé

Il s'agit d'implémenter plusieurs liens logiques de communication par un unique médium de communication à accès partagé : le bus. Ce médium permet la mise en place de liens uni ou multidirectionnels (*broadcast* ou *multicast*) entre un maître et un ou plusieurs esclaves.

Les bus Ethernet et USB sont deux exemples célèbres de bus partagés. Tous les éléments sont susceptibles de piloter ou d'échantillonner les lignes du bus de données, mais seuls un ou certains d'entre eux peuvent piloter le bus d'Adresse. Ces éléments sont appelés maîtres du bus, les autres esclaves. Les maîtres initient des transactions sur le bus avec des esclaves. Les maîtres peuvent aussi se comporter en esclaves.

Le bus est alloué dynamiquement à une communication, plus précisément à un maître, en fonction d'une police d'attribution gérée par l'arbitre du bus (Cf. §2.3.3). Les échanges peuvent être cadencés par :

- une horloge globale (système) : bus globalement synchrone ;
- une horloge spécifique au bus : bus de séquencement local synchrone. Le système devient Globalement Asynchrone et Localement Synchrone (GALS) ;
- l'état du maître et des esclaves : bus asynchrone. Le système est toujours de type GALS, sauf si l'ensemble complet des périphériques est également asynchrone. On obtient alors un système Globalement Asynchrone et Localement Asynchrone (GALA).

Les deux dernières catégories permettent de faire communiquer des éléments de l'architecture ayant des horloges indépendantes, alors que la première nécessite l'utilisation de la même horloge pour tous les composants connectés au bus (Cf. §1).

La structure de tels composants peut encore se décomposer en une partie contrôle et une partie chemin de donnée. Le chemin de donnée est constitué de ressources de transport généralement implémentées par des pistes métalliques. Plusieurs fonctionnalités incombent à la partie contrôle des composants de communication :

1. l'arbitrage pour l'attribution du chemin de donnée ;
2. la synchronisation des extrémités ;
3. le routage de l'information.

Ces bus sont généralement utilisés pour des systèmes ne nécessitant pas de larges bandes passantes. Ils sont donc employés dans les applications orientées contrôle.

Le bus *Advanced Multi-masters Bus Architecture (AMBA)* [ARM 99] présenté au paragraphe 2.5.1 et le bus *CoreConnect* [IBM 99] sont des exemples de bus partagés. Ils peuvent supporter la connexion de plusieurs composants initiateurs de communication (*Multi-Master* ou multi-maîtres), et doivent donc gérer en temps réel l'attribution des média pour une communication logique entre deux interlocuteurs. Les vitesses de fonctionnement de ces bus en font des unités de communication très performantes. Cependant, cette caractéristique doit être revue drastiquement à la baisse en fonction de l'activité et du nombre des communications se partageant cette unité. En effet, plus un bus est partagé plus la latence moyenne d'attribution est élevée. Lorsqu'ils ne sont pas disponibles pour une communication, on les dits «bloquants ». Ils gèlent ainsi l'exécution de parties de l'application. En tant que goulot d'étranglement des performances globales, leur propre performance devient donc un critère limitant sensible. Il apparaît en fait que le concept même de bus partagé devient rapidement incompatible avec un système fortement multi-maîtres comme par exemple les applications multiprocesseurs.

Du fait du nombre restreint de ressources, les bus partagés consomment peu d'énergie. Cependant ils peuvent pousser les composants qu'ils connectent à consommer plus. En effet, le partage de ressources implique inévitablement des cycles d'attente d'attribution durant lesquels de l'énergie est inutilement gaspillée.

L'encombrement de cette solution est le plus réduit. Le partage intensif d'un nombre restreint de ressources offre le meilleur rapport coût/nombre de communications. La conception de ces composants doit sa complexité à la recherche de meilleures performances. Leur réutilisation qui concerne l'architecte du système, se limite à configurer statiquement ces composants pour obtenir la bande passante et l'ordonnancement des communications adéquats. Le test de cette famille de composant est indirectement réalisé par le test effectif des composants connectés en périphérie. Ces composants de communication sont :

1. non extensibles, une augmentation du nombre de composants de calcul connectés réduit drastiquement les performances globales des communications.
2. compatibles uniquement avec une famille ou un jeu restreint de protocoles.

Le 2° point se résout par l'insertion d'adaptateurs de communication (Cf. §2.2.3.1) entre les périphériques et le bus. L'utilisation de plusieurs bus potentiellement de natures différentes permet de résoudre le 1° point en fournissant localement les performances requises. Mais cette solution n'est réalisable qu'en découplant les différentes instances de bus par des ponts (Cf. §2.2.3.2).

2.4.2.2. Les réseaux en étoile et en anneau

Une première alternative intéressante au bus partagé est les réseaux en étoile, de type Ethernet avec un serveur central, et en anneau [FESQ 97]. L'Octagon a récemment su tirer parti des avantages conjugués de ces deux architectures (Cf. § et [KAR 02]).

Ces systèmes contiennent des commutateurs de paquets, au centre de l'étoile ou sur chaque nœud de l'anneau, en charge de router le message passant par le nœud auquel ils sont associés vers un lien de sortie parmi n, ce lien étant soit un canal de communication vers un autre noeud du réseau soit la connexion au périphérique branché sur ce noeud. Les premières réalisations implémentèrent les commutateurs de paquets avec un bus local partagé, avant de se tourner vers des circuits à base de multiplexeurs pour s'affranchir des problèmes de haute impédance et de bus trois états. Nous ne nous étendrons pas plus sur ces types de réseaux, qui sont en fait des réseaux commutés particuliers, où le noeud de commutation est associé à un nœud de calcul ou de mémorisation (hormis pour le centre de l'étoile dont la charge de routage est élevée), sauf pour les topologies en anneau découplées des modules de l'architecture. Ces topologies présentent l'avantage de dissocier l'anneau de communication des nœuds de calcul (Cf. Figure 2-11e)). Elles peuvent ainsi offrir une grande bande passante dans le réseau et une bande passante réduite moins coûteuse avec le composant périphérique, tout en libérant ce dernier de son implication dans la communication. C'est le cas du bus synchrone micro-commuté *µNetwork* de Sonics présenté au §2.5.2 ou du réseau *STRING* (*Self-timed Ring for GALS*) de l'université ETH Zürick [VILL 03]. Le réseau *STRING* est dédié aux architectures GALS grâce à un anneau de communication asynchrone découplé qui se synchronise avec le

module périphérique synchrone par une technique d'interruption de l'horloge (*Stoppable Clock*), uniquement lorsqu'une communication avec ce dernier est nécessaire (Cf. §2.6.1.3).

2.4.2.3. Les réseaux crossbar

Il s'agit de la mise en parallèle de *n* lignes de transmission concurrentes dont l'attribution est régie par un multiplexage de type TDMA [FESQ 97].

Le réseau de périphériques s'articule autour d'une matrice (appelée *FPIC* ou *Switch Fabric*) d'interrupteurs programmables permettant la mise en oeuvre concurrente de plusieurs canaux de communication.

La configuration de cette matrice peut être :

- statique par programmation de la matrice lors de la fabrication, élevant ainsi la réutilisabilité de la plateforme ;
- faiblement dynamique par reconfiguration des interconnexions à chaque démarrage de l'application ;
- fortement dynamique lorsqu'au cours de l'exécution de l'application, une reprogrammation est décidée pour mieux répondre aux besoins courants en communication.

Lorsque ce réseau forme un graphe complet ou fortement connexe[6] il est dit *full-connect*. Les implémentations de crossbar sont généralement synchrones. On retrouve ces bus lorsqu'une grande bande passante et surtout une latence faible sont nécessaires. Il s'agit des applications à composantes fortement orientées calcul. Ils sont aussi utilisés comme bus de cohérence pour le mécanisme de cache système dans les architectures multiprocesseurs.

Sur de tels bus, le parallélisme des communications est élevé, ils sont très faiblement bloquants. La vitesse de chacune des lignes d'un crossbar est très élevée. L'ordonnancement TDMA et le nombre élevé de ressources rendent cette solution énergétiquement coûteuse. Le coût surfacique qui est proportionnel à n^2 rend difficile l'utilisation de ces bus pour les applications dites massivement parallèles. La conception de tels bus est assujettie de contraintes fortes concernant le placement sur la puce des lignes de transmission. Heureusement, la régularité de ces composants permet la mise au point de générateurs de modèles placés et routés sur silicium. La matrice n'opérant aucune décision sur le routage, le test est reporté aux interfaces des lignes de transmission. La disponibilité de modèles hautement paramétrés ou de générateurs dédiés permet d'étendre aisément le nombre de lignes de transmission. L'usage d'adaptateur de communication n'est pas nécessaire, car il est aisé de spécialiser une ligne de transmission pour le protocole de ses extrémités. Évidemment, si les deux extrémités d'une ligne ne supportent pas le même protocole, alors l'une d'elle devra être adaptée.

[6] toute paire de sommets (périphériques) est reliée par un arc (ligne de transmission)

2.4.2.4. Les réseaux commutés ou à maille

Ce type de réseau comporte plusieurs étages, dont le franchissement s'opère au travers d'un commutateur en charge de router le message vers un canal parmi n. Pour des raisons de performances et/ou de tolérances aux pannes, il est possible d'introduire des chemins redondants permettant ainsi d'augmenter la concurrence des communications et les solutions de routage. Leur utilisation au sein d'architectures mono puces est abordée par [GUE 00b]. En général, l'information s'y propage sous forme de paquets dont l'entête contient l'Adresse de destination.

L'implémentation de chaque commutateur peut être à base de multiplexeurs voire à base de petits bus crossbar (Cf. 5.3.4.2). La propagation de l'information peut suivre l'un des deux schémas suivants:

1. une connexion source-destination (aussi appelée *session*) permettant le transport continue et transparent de l'information au travers des étages;
2. une propagation de l'information d'étage en étage où elle est temporairement bufferisée dans une unité de routage. Ce mode de transmission est appelé *worm-hole*. Si deux paquets entrants doivent être routés vers la même sortie, on parle alors de collision, et un paquet doit être élu et transmis tandis que l'autre est bufferisé pour ne pas être perdu.

Une configuration particulière de ces réseaux, le commutateur « Batcher-Banyan » est utilisée pour l'implémentation des routeurs de protocole *Asynchronous Transfert Mode* (ATM) [FESQ 97] présentés par la Figure 2-12.

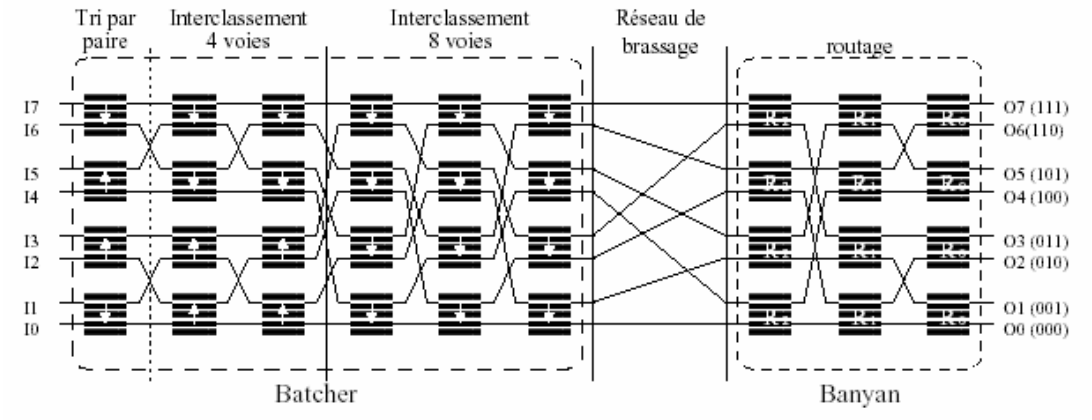


Figure 2-12 : réseau commuté ATM de Batcher et Banyan

La bande passante typique d'un bus commuté est proche de celle des bus crossbar. Par contre, les latences qui sont proportionnelles à la profondeur, ou nombre de commutateurs traversés, (surtout en transmission *wormhole*) peuvent être bien plus élevées. L'activité des routeurs est élevée afin d'atteindre une vitesse globale de transmission conséquente. Ces unités étant nombreuses, la consommation de l'ensemble du réseau est importante. Bien qu'étant moins performants que les bus crossbar, ces bus commutés leur sont préférés afin de réduire le coût de la mise en oeuvre des communications, car leur complexité en $n \log 2\ n$ (avec n nombre d'entrées) est bien moindre. Là encore, la réutilisation de composants placés, routés est

nécessaire. Chaque commutateur du réseau peut se voir associer de la logique de test autorisant ainsi le test en ligne et la modification dynamique de l'algorithme global de routage lorsqu'un chemin est défaillant. Ces composants peuvent être mis à l'échelle à condition de préserver leurs caractéristiques de régularité. Ainsi leur nombre de lignes peut évoluer suivant une règle précise (ex : puissance de deux pour les réseaux ATM et Fat-Tree).

2.4.3. Conclusion

Les réseaux de communication se différencient des bus partagés par :

- leur nombre de ressources de transport et de routage supérieur à l'unité ;
- la distribution du routage sur l'ensemble du réseau ;
- la possibilité d'intégrer des services, en particulier de gérer des modes de transaction interruptibles (Cf. 2.3.3) dans un environnement multi-maîtres avec des niveaux de priorité multiples.

L'orientation faiblement multi-maîtres du bus centralisé le rend favorable à une conception en technologie synchrone, que le système soit globalement synchrone ou GALS. L'horloge du bus de communication est alors synchronisée de manière mésochrone ou hétérochrone (Cf. 1.1) sur l'horloge du maître pilotant le système. En outre Stevens montre dans [STEV 03] que ces architectures centralisées sont les plus défavorables en terme de performances et de coûts à la technologie asynchrone.

Par contre les réseaux de communication présentent un certain nombre de caractéristiques les rendant favorables à une conception en technologie asynchrone au sein d'une architecture globalement asynchrone et localement synchrone :

- Ils sont favorables aux transmissions sur de grandes distances, à condition de s'affranchir des problèmes de propagation du signal, comme le permet la technologie asynchrone (Cf. 1.3.1 à 1.3.3).
- Ils conviennent parfaitement à des systèmes sur silicium complexes intégrant plusieurs maîtres et donc multi-horloges. Un réseau sur silicium asynchrone va permettre :
    - de libérer des problèmes de synchronisation globale d'horloge – il nécessite en revanche une synchronisation locale avec l'horloge dont le traitement est introduit au Chapitre 4 et détaillé au chapitre 5 –
    - une conception modulaire du système, comme présentée aux paragraphes 1.2.2.3 et 1.2.2.4. Cette conception modulaire va guider l'ensemble de notre méthodologie de conception développée à partir du chapitre 3.

Ces réseaux prennent place dans la réalisation d'applications « temps réel » et « traitement du signal » nécessitant une grande bande passante et une faible latence. Celles-ci mettant en oeuvre des calculs massivement parallèles, leurs nombreuses communications à haut débit ne peuvent être réalisées que par un réseau.

L'utilisation d'un nombre élevé de ressources de transport et de routage permet d'offrir de manière quasi-permanente la disponibilité des unités de transports. Les solutions à base de réseaux de communication offrent donc les meilleurs débits et les latences les plus basses. La consommation du réseau augmente proportionnellement avec le nombre de ressources utilisées. De plus, les unités de contrôle et de routage essaimées sur toute la surface du réseau consomment inutilement lorsqu'ils ne sont pas sollicités, à moins que ces unités ne soient construites en logique asynchrone, auquel cas elles ne consomment qu'en cas d'activité (Cf. 1.3.2). L'augmentation des ressources se traduit bien évidemment par un accroissement de l'occupation surfacique. Mais si la surface nécessitée par les unités de contrôle et de routage est proportionnelle à leur nombre, il en va différemment pour les ressources de transport. En effet ces dernières utilisent des pistes métalliques, dont on dénombre jusqu'à huit niveaux sur les puces modernes. En pratique, la logique câblée ne nécessite pas plus de quatre de ces niveaux, laissant ainsi les deux niveaux supérieurs disponibles pour l'implémentation des communications. Mais les réseaux de communication fortement parallèles peuvent nécessiter tant de pistes que ces deux niveaux ne suffisent à les fournir. Il faut alors définir sur la puce, des emplacements réservés au réseau de communication afin de bénéficier de la totalité des niveaux métalliques. Du fait de la complexité de leur structure et de leurs algorithmes de routage, les réseaux de communication sont de véritables défis à relever par les concepteurs spécialisés. Leurs caractères génériques et configurables sont les causes essentielles de cette complexité, mais permettent en contrepartie une réutilisation aisée. Le test du bon transport de l'information par unique observation de la périphérie du réseau ne suffit plus. Les nombres élevés d'unités de routage, de chemins possibles pour une même communication poussent à l'intégration de fonctionnalité de test au sein du réseau. Le nombre de connexions supportées par ces réseaux est rendu facilement extensible par leur régularité structurelle. Quand aux protocoles des composants de calcul connectés aux extrémités du réseau, leur incompatibilité peut ici encore, nécessiter l'usage d'adaptateurs de communication.

2.5. Architectures de communication pour des systèmes globalement synchrones

La grande diversité actuelle des réseaux de communication et de leurs implémentations rend difficile une modélisation générique. La solution adoptée par les acteurs tant industriels qu'universitaires consiste à restreindre les modèles à une famille de topologies et/ou de protocoles, voir même au sein d'une famille à un jeu restreint et figé de protocoles et parfois de réseau de communication.

L'ensemble des réseaux de communication et des interfaces propriétaires présentés ici contraint le système à fonctionner sur une horloge globale unique. L'ordre de présentation respecte une évolution des architectures des réseaux de communication à peu près chronologique. La section 2.5.1 présente des topologies propriétaires à base de bus centralisé peu flexible. La section 2.5.2 présente une étape intermédiaire de standardisation des interfaces uniquement, cherchant à laisser libre le choix de la topologie. La section 2.5.3 revient à des topologies figées mais distribuées, toujours plus facilement évolutives vers des applications GALS.

La section suivante 2.6 présentera les réalisations de réseaux de communication pour des systèmes globalement asynchrones. Les avantages et inconvénients comparés des systèmes respectivement à horloge unique, multi-horloges ou sans horloge sont développés au §1.

2.5.1. Les architectures de communication utilisant des bus partagés multi maîtres

2.5.1.1. Standards de bus industriels

Certains fournisseurs de composants réutilisables offrent des modèles de bus systèmes compatibles avec un grand nombre de leurs composants. Ces bus s'élèvent de facto aux rangs de standard, spécifiques à une chaîne de développement et une famille de composants. Ils sont alors couramment utilisés pour interconnecter un jeu de composants aux protocoles de communication compatibles. Le PI-Bus d'OMI [OMI 95] et AMBA [ARM 99] de ARM Inc. sont deux standards représentatifs des bus partagés multi-maîtres que nous présentons ici. Le bus centralisé CoreConnect d'IBM [IBM 99] est également un standard de bus qui fournit une famille de trois bus synchrones. L'architecture de ces bus est semblable à celle de la famille de bus AMBA et leur concept d'utilisation étant également très proche, cette famille n'est pas présentée ici mais une bonne présentation synthétique en est faite dans [BAG 02].

2.5.1.2. Peripheral Interconnect Bus (PI-Bus) – OMI

Le PI-Bus [OMI 95] a été développé dans le cadre du projet européen SMILE de standardisation des bus (on ne parlait pas encore de réseau) intégrés par le collectif Open Microprocessor Systems Initiative [OMI 98]. Le PI-Bus fut la première spécification du protocole d'un bus partagé multi-maîtres intégré et a été de ce fait très employé dans l'industrie. Ce bus reprend les grands principes des bus pour circuits imprimés (VME, PCI, ISA...). Les bus plus récents n'ont fait qu'améliorer la variété des services offerts (Cf. §2.2.2).

Ce bus et ce protocole restent intéressants et d'ailleurs largement utilisés pour spécifier proprement un réseau d'interconnexion aux faibles performances, notamment les bus dédiés au contrôle, évitant ainsi la surenchère de fonctionnalités.

2.5.1.3. Advanced Microcontroller Bus Architecture (AMBA) – ARM

Le bus AMBA [ARM 99] regroupe une famille de bus partagés destinée à satisfaire le plus grand nombre de besoins :

- *Advanced Peripheral Bus (APB)* : ce bus fournit une interface de complexité minimale et n'autorise qu'un seul maître, la plupart du temps le pont de connexion avec un autre bus, multi-maîtres nécessairement, et la plupart du temps de la même famille. Ce bus destiné à minimiser la consommation n'autorise qu'une faible bande passante et ne fournit aucun pipeline des transactions.

- *Advanced High-performance Bus (AHB)* : c'est un bus multi-maîtres synchrone à large bande passante/haut débit. Il supporte les transactions en rafales, non préemptives et différées, cela avec un nombre élevé de maîtres (jusqu'à 16 par bus). Pour de meilleures performances, les données associées à une Adresse sont présentes sur le bus un cycle après l'Adresse, ce qui permet un fonctionnement en pipeline. Un chemin de données multiplexé – d'une largeur extensible à 128 bits – est préféré à des lignes de bus trois états. Cela permet une fréquence de fonctionnement plus élevée et simplifie/améliore l'automatisation de la synthèse du système. L'arbitre et le décodeur d'Adresse sont centralisés et reliés point à point avec l'ensemble des périphériques, au contraire du PI-Bus ou du AMBA ASB où le maître devait passer par le bus pour demander et obtenir une communication.

La Figure 2-13 illustre une architecture type à base de bus hiérarchiques AMBA.

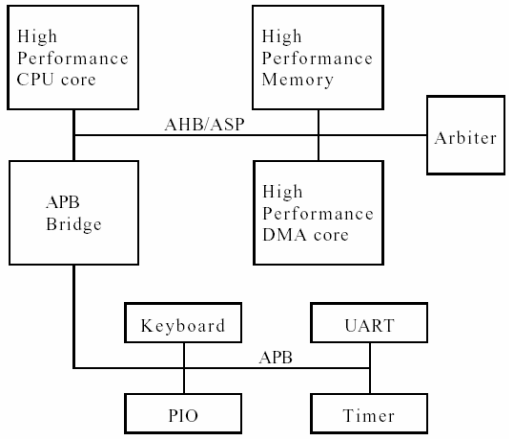


Figure 2-13 : une architecture type à base de bus hiérarchiques AMBA

La politique de l'arbitre est laissée à la discrétion de l'implémentation. Elle peut être équitable ou non, avec ou sans famine, selon les besoins de l'application et des éléments du système. Le bus AMBA spécifie un protocole d'accès spécifique efficace, via le bus, pour le test des modules raccordés (mais il ne s'agit pas d'un test du bus en lui-même).

Le bus AHB a été développé pour apporter des solutions à la précédente version du bus AMBA haute performance, le ASB (*Advanced System Bus*). Le protocole de ce dernier, très proche de celui du PI-Bus, rencontrait quelques limitations. Depuis 1999, le AHB a lui aussi fortement évolué et dérivé. Pour contourner le problème de son manque d'extensibilité (Cf. §2.2.1), qu'aucune intelligence de l'arbitrage ni des caches ne peuvent masquer indéfiniment, la largeur du bus et la fréquence d'horloge on progressivement augmentées.

Toutefois, un bus trop large ne sera pas plus efficace pour les transactions dont la taille est inférieure à la taille du mot. Si l'interface doit concaténer les transactions pour être efficace, elle perd ses avantages de simplicité et de faible latence.

Même lorsque le débit est suffisant, le bus ne convient plus aux grands systèmes avec de nombreux éléments à raccorder. Le goulot est alors l'arbitre central, qui doit être dimensionné pour élire un maître parmi des dizaines. S'il ne peut plus accomplir cette tâche

efficacement (ce qui est inévitable), la latence de service des requêtes s'accroît et une proportion notable des cycles disponibles est perdue dans les délais d'arbitrage [GUE 00a].

- *Multi-layer AHB* [ARM 02a] : pour contourner l'absence d'extensibilité et finalement assurer la continuité du standard AMBA (pour ne pas dire AHB), ARM offre depuis juin 2002 un réseau d'interconnexion basé sur le protocole AHB pour remplacer le bus centralisé. Il s'agit d'une matrice d'interconnexion qui distribue un bloc de décodage par maître et un bloc d'arbitrage par esclave. Cette extension a cependant rencontré peu de succès, car elle reste mono-horloge et nécessite donc la gestion globale – déjà délicate pour un bus centralisé – sur un réseau distribué, comme nous l'avons vu au chapitre 1. La Figure 2-14 illustre un exemple de matrice d'interconnexion Multi-layer AHB de 3 maîtres et 4 esclaves.

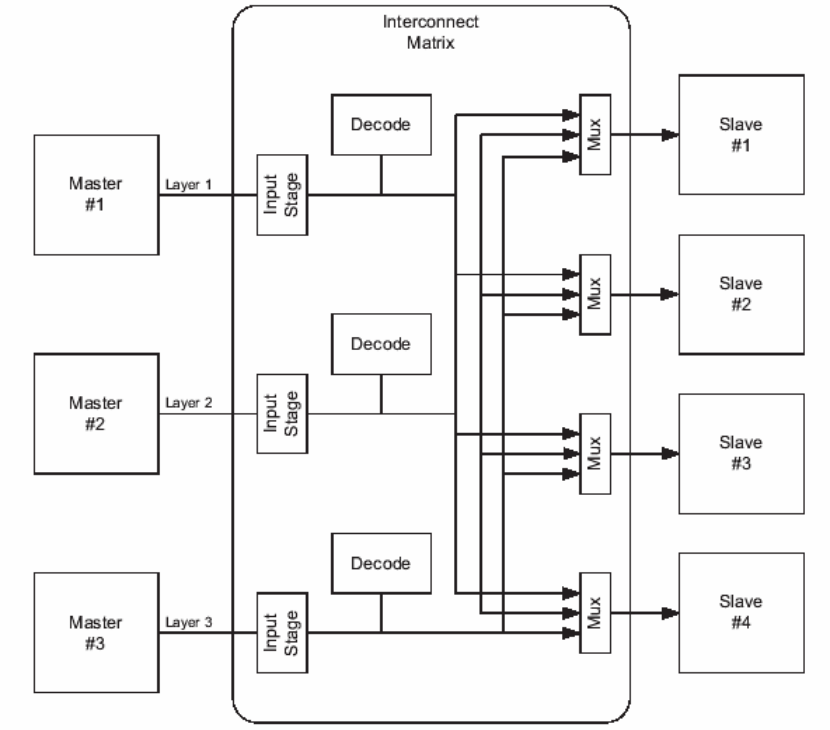


Figure 2-14 : exemple de matrice d'interconnexion Multi-layer AHB 3 maîtres / 4 esclaves

- *AHB-Lite* [ARM 02b] : il s'est avéré à l'usage que bon nombre de concepteurs avaient besoin des performances liées au protocole de bus AHB mais avec un usage amélioré des services du protocole APB destiné à un maître unique. Le bus AHB implémentait des fonctionnalités inutiles dans ce cas-là. Le protocole AHB-Lite fournit le jeu de services réduit pour un maître unique tout en conservant les potentialités de transaction sophistiquées : en rafale, réémissibles,…

2.5.2. Les standards d'interfaces

2.5.2.1. Les initiatives de collectifs

Afin de faciliter l'intégration de divers composants au sein d'une même architecture, plusieurs standards de protocoles et interfaces ont été érigés pour que lors de la conception de ces composants, ces protocoles soient adoptés. Ces standards, afin de devenir tels, ont été élaborés par des initiatives de collectifs et des consortia afin de proposer des solutions « ouvertes » dont le coût d'élaboration ne soit pas supporté par une seule compagnie. On trouve ainsi :

- la spécification Wishbone de Silicore Corporation [SIL 04], adoptée par le collectif OpenCores [OC 04] pour sa simplicité ;
- la spécification OCP (*Open Core Protocol*) développée par Sonics et soutenue par l'OCP-IP (*OCP International Partnership*) [OCP-IP 04] ;
- le consortium Virtual Socket Interface Alliance (VSIA) [VSI 04a] définit un ensemble plus large de standards [VSI 04b] destinés à la réalisation normée complète d'un système sur silicium. Dans le cadre de la conception d'un réseau sur silicium, le concepteur sera concerné par le standard de documentation et de spécification SLIF, un standard d'abstraction des interfaces de bus *On-Chip Virtual Component Interface (VCI),* et une représentation standardisée des types de données, le *System-Level Data Types Standard*.

De manière générale, ces spécifications de protocoles définissent une interface point à point qui fournit un ensemble standard de signaux de données de contrôle et de test permettant aux cœurs du système de communiquer entre eux. Ces standards offrent une manipulation et une utilisation simplifiées des composants grâce à leur compatibilité directe. Contrairement aux bus systèmes, ils n'imposent pas de topologie du réseau. Cependant, ils n'en offrent pas non plus. Et ils ne répondent pas toujours pleinement aux besoins spécifiques des applications. Leur respect peut conduire à l'implémentation d'un jeu de fonctionnalités qui ne sont pas toutes utilisées. Leur utilisation reste faiblement répandue, car les utilisateurs potentiels leur reprochent les points précédemment cités ou alors la stratégie ou la politique des entreprises utilisatrices ne s'accommode pas des décisions des organismes de standardisation souvent « orientées » par d'autres entreprises. Des solutions propriétaires leur sont donc préférées.

2.5.2.2. Le STBus – STMicroelectronics

Ainsi STMicroelectronics propose une solution intermédiaire entre l'interconnexion sous forme d'*IP* et le standard d'interface : le STBus [BRINI 03]. Le STBus définit un standard de trois types différents d'interfaces compatibles avec les recommandations VSI Alliance. L'avantage de définir ces standards d'interface est de conserver la liberté de choix de la topologie d'interconnexion. Cependant le STBus propose également un choix restreint mais suffisant de topologies en bus centralisé ou en cas complet. La topologie en cas combine ici les avantages d'une architecture distribuée (large bande passante, allègement des contentions) avec un standard d'interface conventionnel et relativement flexible. L'interconnexion autorise le

pipeline par FIFOs aux interfaces des maîtres et esclaves, les services de transactions *split* et *burst*. Chaque nœud peut implémenter son propre algorithme d'arbitrage. La Figure 2-15 présente le modèle fonctionnel de l'interaction des services de communication du STBus indépendamment de la topologie choisie.

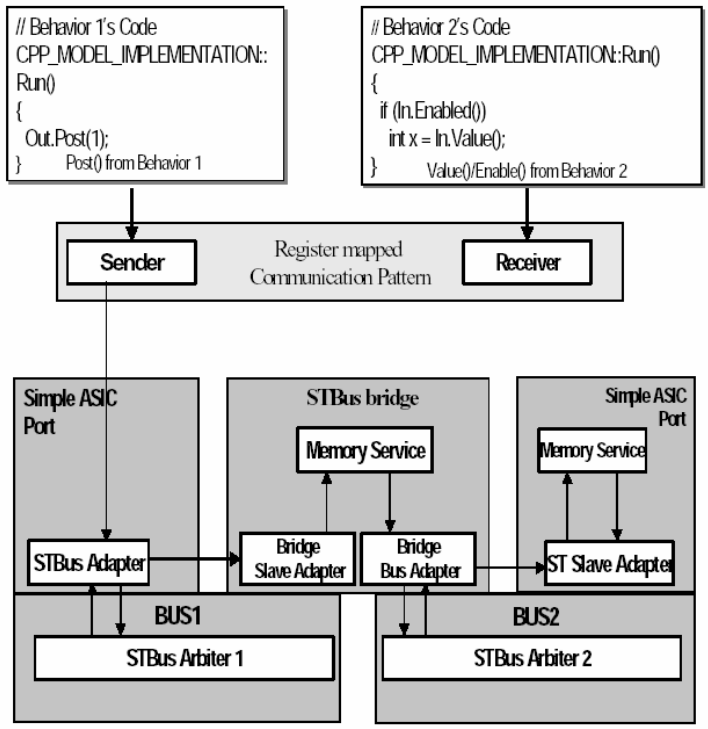


Figure 2-15 : modèle fonctionnel des services de communication du STBus

Un des avantages du STBus est que sa conception est appuyée par des outils de traces et d'analyse de performance et des outils d'exploration des paramètres du réseau d'interconnexion (topologie, arbitrage, FIFOs,…), en particulier l'outil *Virtual Component Co-design* (*VCC*) de Cadence qui permet l'exploration d'architectures et le partitionnement logiciel/matériel [BAG 02]. En outre la bibliothèque de modules constituants le STBus fournit des ponts vers les autres modèles de réseaux d'interconnexion, en particulier vers les systèmes ARM et Tensilica.

2.5.3. Les architectures de communication utilisant des réseaux commutés

Cette section présente des topologies de réseaux distribués, dont en particulier l'architecture Octagon qui est facilement évolutive vers des applications GALS.

2.5.3.1. Le bus micro-commuté Silicon Backplane µNetwork – Sonics

Le bus µNetwork de Sonics est un précurseur de la récente technologie des bus micro-commutés. Les Figure 2-16 et Figure 2-17 en illustrent respectivement la structure et une utilisation typique au sein d'un système sur silicium.

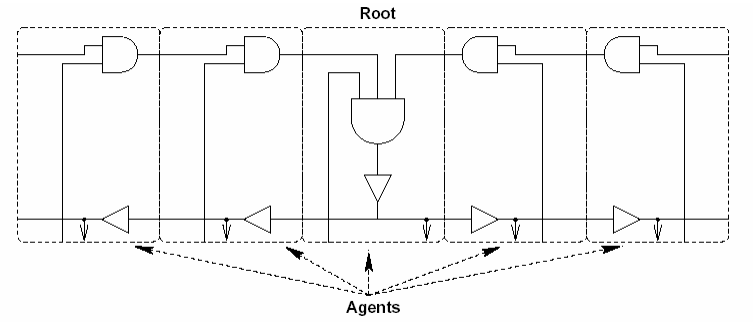


Figure 2-16 : couche physique du Sonics µNetwork

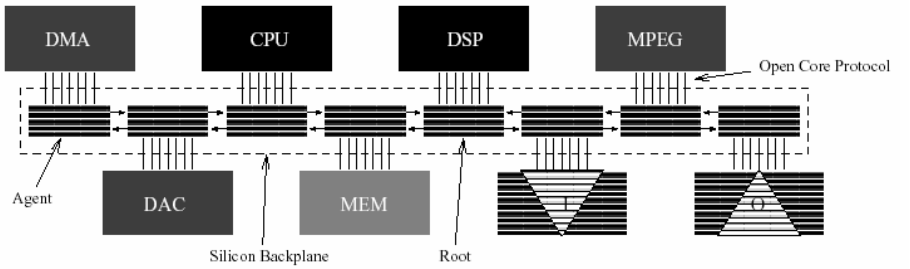


Figure 2-17 : architecture typique à base de Sonics µNetwork

Les éléments de commutation sont enfouis au sein même du bus, comme des macrocellules (rectangles pointillés sur la Figure 2-16). C'est l'assemblage de ces macrocellules qui constitue le médium physique de communication comme illustré sur la Figure 2-17. Cette famille de bus est de plus en plus utilisée (avec une complexité variable) dans les systèmes sur silicium. Ce sont d'ailleurs les systèmes sur silicium qui sont à l'origine de la création de cette famille de composants de communication. Le même principe de bus de communication, appelé U-bus a été élaboré pour des systèmes optiques dans [FESQ 97].

Tout comme leurs aînés les bus partagés, ces bus ne possèdent qu'une unique route. Cependant, ils n'en sont pas pour autant bloquant. En effet l'attribution de l'accès au médium est régi par une règle équitable à base de police tourniquet ou «round robin » et attribution par tranches de temps ou «Time Division Multiple Access (TDMA) ». Ceci permet d'obtenir des latences acceptables et de garantir une bande passante. Ces caractéristiques sont dans la littérature regroupées sous la dénomination de « Qualité de Service » [BOL 03]. La simplicité des macrocellules (quelques portes logiques chacune) permet de limiter la consommation qui se situe entre celle des bus partagés et celle des bus commutés. La simplicité des macrocellules a aussi pour conséquence de ne nécessiter qu'une surface réduite. Cependant, ces dernières empêchent l'implémentation du bus dans les couches métalliques supérieures et des zones du circuit devront lui être spécifiquement dédiées. Ces bus étant extrêmement réguliers, leur conception est aisée. Le faible coût d'implémentation de ces bus étant un argument principal de leur utilisation, aucun matériel de test n'est adjoint aux macrocellules. Le test du bus est reporté dans le test des composants en périphérie. L'extension des modèles de tels bus pour supporter un plus grand nombre de connexions peut se faire aisément grâce à leur régularité. De plus, la dispersion des éléments de routage (macrocellules) au sein même du bus fait que celui-ci dispose de répétiteurs de l'information augmentant ainsi la distance critique séparant deux terminaisons. Cependant, on prendra garde à l'évolution de la fréquence d'horloge séquençant

le bus qui, pour supporter les bandes passantes BWi requises par les communications, doit suivre la loi : f>= BWi/W, avec W la largeur des données véhiculées par le bus.

Ce bus supporte une interface compatible au protocole OpenCoreProtocol [OCP 01].

2.5.3.2. L'architecture SPIN à commutation de paquets – Université Paris VI

Un réseau de commutation de paquets déplace les données d'un noeud à l'autre dans des petits morceaux formatés appelés paquets. Puisque les décisions de routage sont réparties sur les routeurs, le réseau peut rester très réactif même pour des paquets de tailles importantes. Malgré les nombreuses étapes de routage, une faible latence peut être maintenue si les routeurs expédient l'en-tête des paquets aussi tôt que possible, sans attendre la queue. C'est le principe de routage par trou-de-vers (*wormhole routing*). Les travaux de [GUE 00b] présentent un réseau à commutation de paquets, intégré, programmable et extensible, appelé SPIN (*Scalable, Programmable, Integrated Network*).

La topologie de réseau influençant la complexité des décisions distribuées de routage, celle employée ici est le Fat-Tree, illustré par la Figure 2-18, car Leiserson prouve formellement [LEI 85] que cette topologie est la plus efficace pour la réalisation de systèmes VLSI.

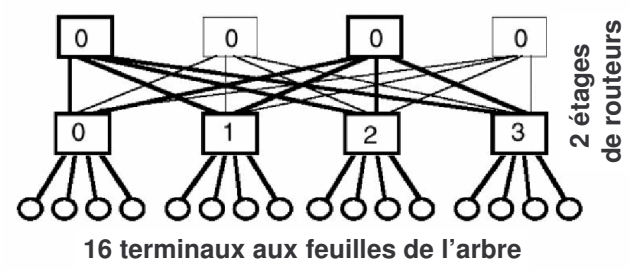


Figure 2-18 : réseau de type Fat-Tree

Le lien élémentaire est un lien parallèle constitué de deux chemins de donnés unidirectionnels de 32 bits. La taille d'un paquet peut être quelconque, voire infinie. Le protocole d'accès au réseau implémente nativement le modèle de communication à passage de message, mais des messages peuvent être employés pour établir des protocoles émulant d'autres modèles (flot de données, espace d'Addressage). Les inconvénients des réseaux de commutation de paquets sont leur complexité de conception et une latence intrinsèque arbitraire.

2.5.3.3. L'architecture Octagon – STMicroelectronics

La motivation de réalisation de cette architecture vient d'un besoin de réseaux intégrés à haute performance capables d'aider à fournir la capacité de traitement exigée par les versions les plus récentes des routeurs OC-768 [KAR 01]. Les architectures traditionnelles d'interconnexion telles que le bus partagé et le crossbar auront des difficultés à répondre à ces besoins de débits tout en maintenant des coûts raisonnables [KAR 01].

L'unité de base de l'Octagon se compose de huit nœuds et de douze liens bidirectionnels connectés selon la manière représentée sur la Figure 2-19.

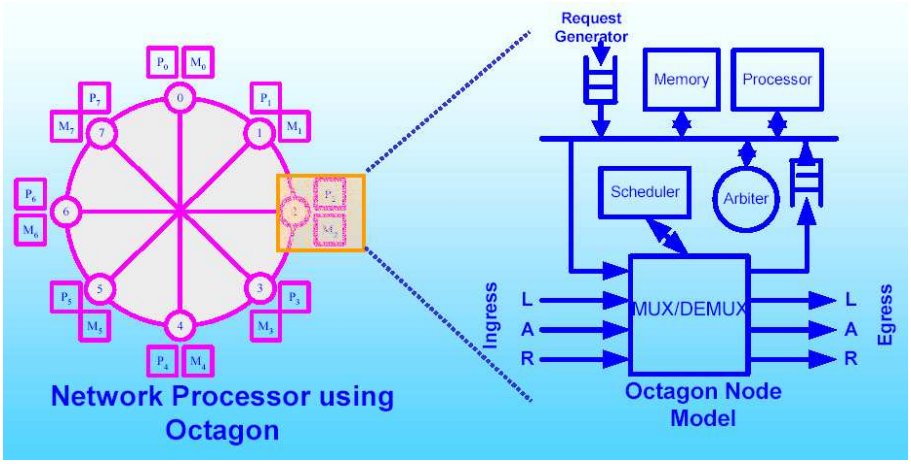


Figure 2-19 : modèle architectural de l'Octagon

L'architecture Octagon a les propriétés suivantes :

1. la communication entre n'importe quelle paire de nœuds peut être exécutée en passant au maximum par deux routeurs ;
2. le débit total est meilleur que pour un bus centralisé ou un crossbar interconnectant le même système ;
3. l'algorithme de routage de plus court chemin est très simple ;
4. L'implémentation physique coûte moins chère en câblage que le crossbar ;
5. La complexité augmente linéairement avec le nombre de nœuds ;
6. Cette architecture Octagon peut facilement évoluer pour interconnecter des systèmes globalement asynchrones et localement synchrones.

2.6. Architectures de communication pour des systèmes globalement asynchrones

Cette section présente les réalisations de réseaux de communication pour des systèmes globalement asynchrones. A l'époque de cette étude les réalisations industrielles étaient encore un peu timides, d'où la part belle aux réalisations universitaires. Le paradigme GALS ayant à présent convaincu la communauté des concepteurs, la plupart des acteurs du monde de la conception de systèmes intégrés propose une solution GALS, que le réseau de communication soit synchrone ou asynchrone

2.6.1. Réalisations universitaires

L'ordre de présentation de ce paragraphe respecte, comme pour la section 2.5, une évolution chronologique des architectures de communication asynchrones. Le paragraphe 2.6.1 présente un bus encore considéré comme une fonction de processeur. Le paragraphe 2.6.2 présente l'évolution d'une IP de bus partagé vers une fabrique de topologies d'interconnexion distribuées à partir de composants élémentaires pour des architectures essentiellement asynchrones (globalement et localement). Le paragraphe 2.6.3 présente une méthodologie de conception comparable dans l'esprit, mais pour des applications résolument GALS.

2.6.1.1. ASPRO Bus – Laboratoire TIMA

ASPRO [REN 99] [VIV 01] est un microprocesseur asynchrone de 16 bits à 3 étages de pipeline développé par l'ENST de Bretagne en collaboration avec France Telecom R&D Grenoble et STMicroelectronics en 1998. L'équipe a depuis rejoint le laboratoire TIMA en 1999. C'est un microprocesseur RISC régulier qui contient 16 registres de 16 bits.

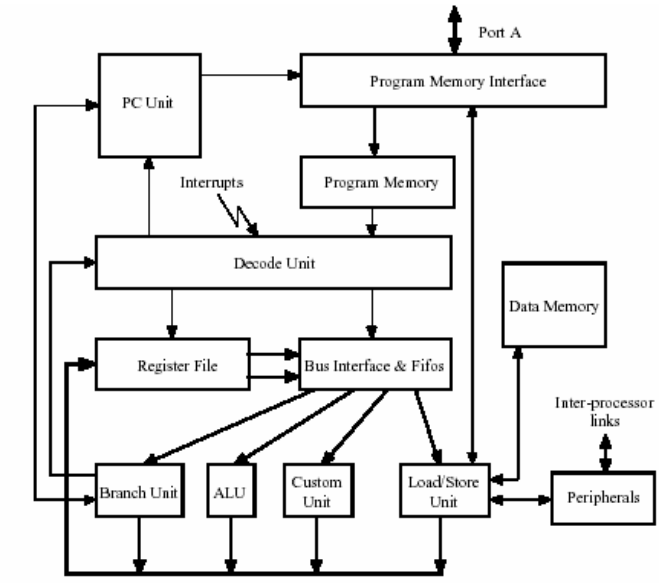


Figure 2-20 : architecture du processeur ASPRO

L'architecture de ASPRO est proposée Figure 2-20. Il s'agit d'un processeur scalaire qui décode les instructions dans l'ordre et les exécute dans le désordre. La motivation était de montrer que les techniques asynchrones peuvent améliorer efficacement les systèmes VLSI et peuvent permettre d'optimiser la vitesse et la consommation d'énergie. Cette réalisation met également en évidence les propriétés d'excellente modularité de l'asynchrone. Le cœur du processeur ASPRO a été implémenté en une architecture multi-rail quasi insensible aux délais (QDI) en utilisant un protocole de communication 4 phases (Cf. §3.1). Un convertisseur 4 phases double rails vers 4 phases micropipeline (Cf. §3.1) est nécessaire pour accéder aux mémoires internes et externes (Cf. §2.3.2.3 et §3.1.3.2 pour la compatibilité des protocoles). Le port parallèle A qui permet d'accéder à la mémoire centrale est également implémenté en micropipeline 4 phases pour les même raisons (compatibilité avec les mémoires standard, en

outre le double rail n'est pas utilisé au niveau de l'interface afin de minimiser le nombre de broches).

Le processeur ASPRO dispose de quatre liens d'interconnexion séries point à point *Nx, Ex, Wx* et *Sx* du type *transputer* implémentés en double rails associé à un protocole 2 phases (Cf. §3.1), ce type d'implémentation ayant été adopté afin d'augmenter la vitesse et la robustesse des communications (Cf. §7.2.2). Cependant à l'époque le bus était encore considéré comme une fonction de processeur et non pas comme un composant singulier réutilisable.

2.6.1.2. CHAIN et MARBLE – Université de Manchester

MARBLE (*Manchester Asynchronous Bus for Low Energy*) est le seul bus partagé asynchrone conçu comme un composant réutilisable (ou *IP*). Il a été développé dans le cadre de la thèse de Bainbridge [BAIN 00] comme l'ossature du processeur asynchrone AMULET3i [FURB 00]. Ce processeur implémente une version asynchrone de l'architecture des processeurs de la famille ARM, et le bus MARBLE offre des services compatibles avec les bus de la famille AMBA.

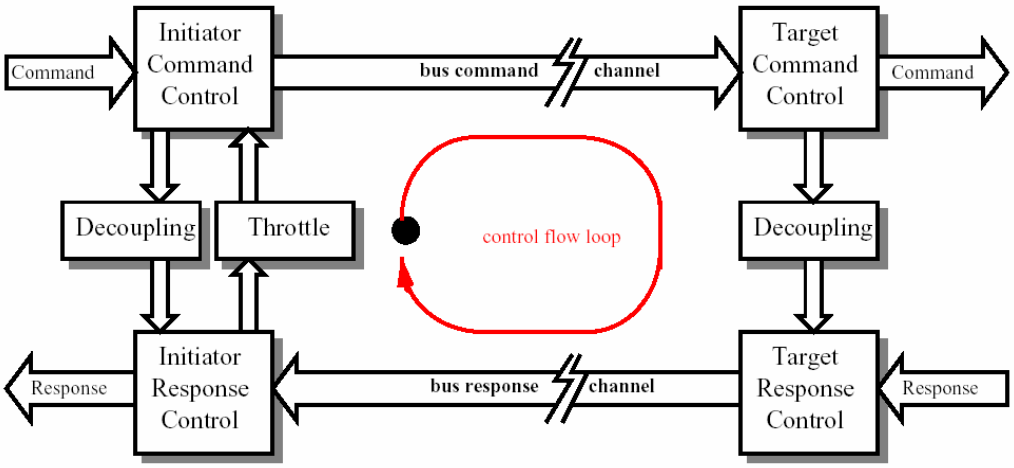


Figure 2-21 : le bus partagé asynchrone MARBLE

C'est un bus double canaux comme le montre la Figure 2-21 (un canal de commande du Maître vers l'Esclave et un canal de retour) en codage "données groupées" quatre phases. L'arbitrage et le décodage d'adresse sont centralisés et ce bus supporte des transactions découpées (*split transactions*) (Cf. §2.3.3.2).

Les limitations des bus centralisés ont conduit rapidement les concepteurs de MARBLE à se tourner vers des architectures de réseaux de communication distribuées. Ainsi la fabrique d'interconnexion CHAIN (*CHip Area INterconnect*) [BAIN 02] utilise l'assemblage de composants élémentaires pour construire des topologies distribuées asynchrones régulières (Cf. §2.2.1.1) selon les besoins du système. La Figure 2-22 présente les composants élémentaires de CHAIN.

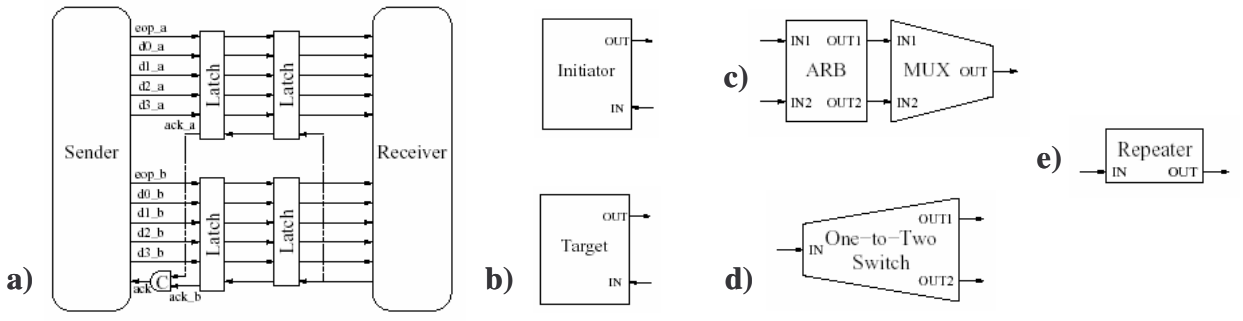


Figure 2-22 : les composants élémentaires de CHAIN

a) Le composant *Link* représente le format des canaux de communication. Chaque digit de canal est codé de manière insensible au délai sur six fils (Cf. §2.3.2.3 et §3.1.3.2) : un fil signalant la fin de paquet (*eop* pour *end of packet*), quatre fils de données et un fil d'acquittement collectif pour l'ensemble des digits du canal. Le format des paquets est construit sur le principe du paquet illustré par la Figure 2-7 du §2.3.3.2 ;

b) Les composants *Initiator* et *Target* adaptent les protocoles éventuellement incompatibles, en cas de périphériques synchrones par exemple, respectivement pour un composant maître et un composant esclave ;

c) Le composant *Two-to-one switch* est un arbitre de deux vers un à priorité fixe ;

d) Le composant *One-to-two switch* est un routeur qui consomme l'en-tête du paquet pour diriger ce dernier vers la bonne cible de la communication. Chaque routeur du réseau est ainsi assuré de lire le même champ du paquet et ce composant peut donc être générique ;

e) Le répéteur périphérique permet de régénérer électriquement le signal lorsque la ligne d'interconnexion est trop longue (Cf. §7.2.1).

A partir de ces blocs des topologies de réseau crossbar complet, d'anneau et de connexions à base de multiplexeurs/démultiplexeurs ont été simulées et comparées [LOV 02]. CHAIN a en outre été implémenté dans un circuit carte à puce de l'université de Manchester.

2.6.1.3. Shir Khan –Université ETH de Zürich

Le système sur silicium *Shir Khan* [VILL 03] est une architecture globalement asynchrone et localement synchrone directement dédiée à la validation de différentes topologies de réseaux de communication asynchrones. Le système est constitué de 25 processeurs quatre bits synchrones d'horloges distinctes, avec chacun quatre ports d'entrée et quatre ports de sortie. La Figure 2-23 présente la répartition sur silicium des quatre topologies, réparties comme suit :

- deux bus centraux partagés MOGLI (*Modular GALS Interconnect*) connectés à huit modules;
- une version MOGLI avec une interface compatible du bus AMBA AHB connecté à six modules ;
- SWING (*Switching Interconnect for GALS*), un réseau crossbar (matrice deux par deux) connecté à six modules;

- STRING (*Self-timed Ring for GALS*), un anneau double découplé (deux anneaux unidirectionnels) connecté à cinq modules.

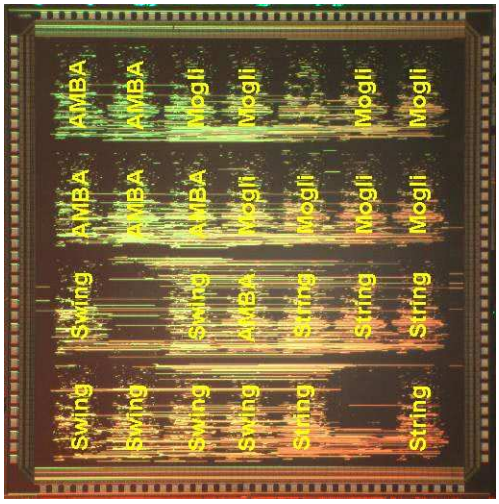


Figure 2-23 : le circuit Shir Khan d'évaluation de réseaux de communication asynchrones

L'ensemble de ces topologies utilise des adaptateurs asynchrones (*Self-Timed Wrapper* de la Figure 2-24) qui "entourent" les modules synchrones (*Locally-Synchronous Island*). Ces adaptateurs contrôlent le protocole de niveau signal (Cf. §2.3.2.3) et sont en codage "données groupées" quatre phases pour être compatibles avec les périphériques synchrones. Ils sont composés d'un certain nombre de ports de communication, de structures de test et surtout d'un générateur d'horloge interruptible. Chaque module (périphérique ou réseau) se synchronise en effet avec un autre module par technique d'interruption et d'extension de l'horloge (*Stoppable or Pausable and Stretchable Clock*). Cette technique a l'avantage de prévenir le risque de métastabilité (Cf. §4.1.2) plutôt que de chercher à le résoudre lorsqu'il s'est produit. Lorsqu'un module initie une transaction, son horloge est interrompue (la phase montante suivante du signal d'horloge n'a pas lu) par le *Wrapper* immédiatement après avoir émis les données et la requête, cela jusqu'à ce que le *Wrapper* reçoive l'acquittement valide de la part du module récepteur.

De l'autre côté, dès que la requête est parvenue au récepteur synchrone, le *Wrapper* de ce dernier suspend son horloge (le signal d'horloge reste dans la phase où il se trouve) jusqu'à complétion du protocole de communication entre les *Wrappers*. Ainsi les protocoles sont initiés par les périphériques synchrones mais sont achevés par les *Wrappers*. Cela assure que l'ensemble des signaux lus par les périphériques synchrones soient stables lorsqu'il sont échantillonnés, prévenant ainsi de toute métastabilité.

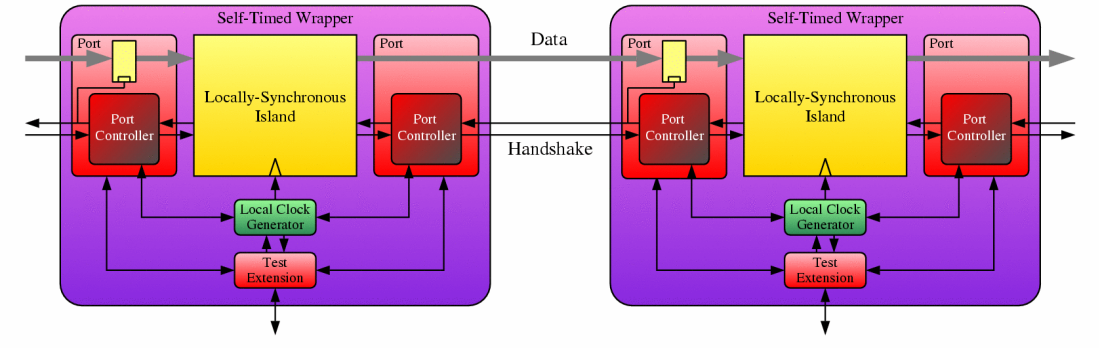


Figure 2-24 : principe des encapsuleurs asynchrones (*Self-Timed Wrapper*) autour des modules localement synchrones (*Locally-Synchronous Island*) pour contrôler l'horloge

La Figure 2-25 montre les performances de chaque topologie. Comme attendu, le bus partagé présente les performances de débit et de latence les plus faibles pour la plus petite surface, au contraire de la matrice d'interconnexion du crossbar. L'anneau découplé présente des performances et un coût de surface intermédiaires entre les deux précédentes topologies.

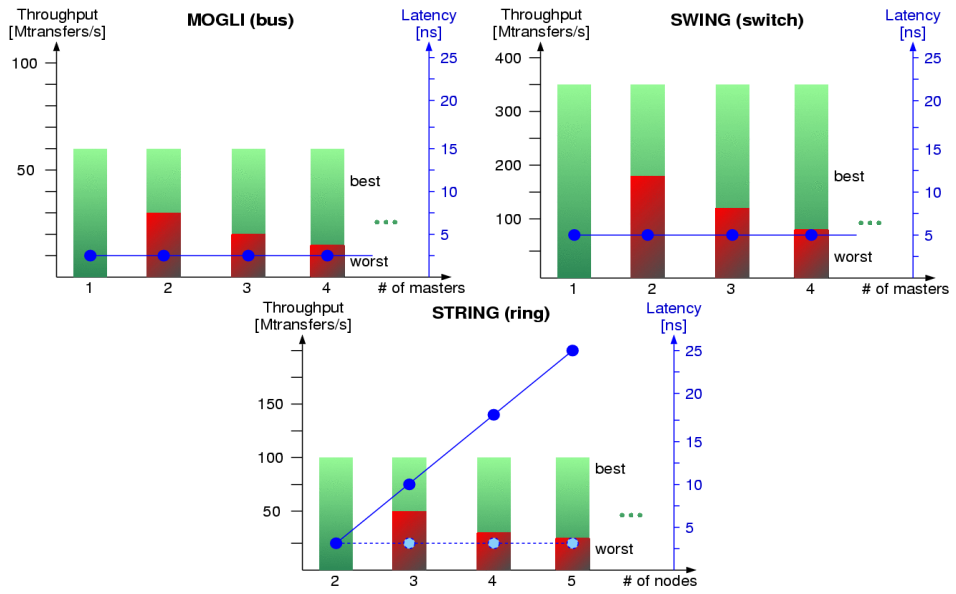


Figure 2-25 : résultats comparés des trois topologies de Shir Khan : bus centralisé, crossbar, anneau découplé

Shir Khan permet de comparer les performances des topologies de réseaux de communication sans horloge implémentées, et prouve le potentiel de modularité apporté par une conception en technologie asynchrone. La possibilité de programmation des processeurs valide également la flexibilité en terme de services de ces topologies. En revanche les modules périphériques sont conçus spécifiquement pour être compatibles avec les adaptateurs asynchrones et ne permettent pas de conclure sur la facilité d'interfaçage de ces réseaux, en particulier pour le contrôle de l'horloge interruptible.

2.6.2. Réalisations industrielles

AMBA AXI [ARM 04] – ARM

ARM annonce délivrer sous peu une spécification d'interconnexion point à point compatible avec les protocoles AMBA (Cf. §2.5.1.3) et de topologie flexible, pour la conception de systèmes globalement asynchrones et localement synchrones. L'architecture de cette interconnexion AMBA AXI sera très certainement de séquencement synchrone.

Fulcrum Microsystems [FUL 04a]

Cette compagnie travaille sur des systèmes sur silicium entièrement asynchrones pour fournir des solutions dans le domaine des architectures de réseaux informatiques. Son produit phare, le *PivotPoint* [FUL 04b], est un commutateur crossbar non bloquant de très haute performances (latence et débit) construit à base de processeurs MIPS asynchrones capable de fournir six interfaces de connexion (ou trois pour sa version réduite) pour développer des réseaux de norme SPI-4.2.

2.7. Les outils de conception automatique de réseaux

Bien que la structure fonctionnelle des réseaux de communication soit généralement très régulière et donc facile à modéliser pour autoriser l'automatisation de leur conception, les fortes contraintes physiques qui y sont relatives (couplage ou *crosstalk*, temps de propagation, non-déterminisme, intégrité du signal…) font que seuls des générateurs spécifiques et fortement conscients (dépendants) de la technologie sont aptes à remplir cette tâche. En outre, la diversité des réseaux de communication et de leurs implémentations rend difficile le support d'un modèle générique par les outils d'intégration. Enfin, les réseaux de communication doivent fournir un haut niveau de modélisation pour pouvoir être intégrés dans les outils d'exploration d'architecture (par exemple l'outil VCC (*Virtual Component Codesign*) de Cadence [BRINI 03]). L'exploration d'architecture consiste en l'élaboration de l'architecture la plus propice à répondre aux exigences de performances et de contraintes de coûts (surface, consommation…). Elle nécessite un grand nombre d'essais [BAG 02] fournissant chacun une évaluation fidèle d'architectures diverses.

Une première solution consiste à restreindre les modèles de réseaux supportés à une famille de topologies et/ou de protocoles et de proposer un outil d'intégration de composants autour de réseaux de communication propriétaire. Leur principale spécificité réside dans la disponibilité d'un jeu figé (non extensible) de protocoles et de réseau de communication profondément accouplé à un modèle d'assemblage. Par exemple l'outil Coral pour le bus *CoreConnect* d'IBM [IBM 99] ou la solution *µNetwork* de Sonics (Cf. §2.5.3.1). Cette solution limite les choix architecturaux et impose l'utilisation d'*IPs* propriétaires.

Une autre solution consiste à utiliser des outils génériques d'intégration de composants autorisant la synthèse du réseau de communication, et permettant la génération d'architectures

détaillées d'implémentation à partir de modèles de référence. Le principal défaut de ces outils est leur orientation dédiée multiprocesseurs encore trop fortement homogènes et l'absence de raffinement de niveaux d'abstraction jusqu'au plus bas niveaux d'implémentation. Cette critique reste à nuancer par l'étude détaillée de ces outils de synthèse de réseau de communication, tels que *CoWare N2C* ou le flot de conception *Roses* développé par le groupe SLS du laboratoire TIMA, qui est présentée par Lyonnard dans sa thèse [LYO 03].

Au-delà des outils, nous pensons en fait que c'est une méthodologie globale de conception centrée sur l'interconnexion et les réseaux sur silicium qui doit être envisagée pour pouvoir répondre aux exigences et contraintes des systèmes actuels. L'interconnexion est aujourd'hui un défi majeur des systèmes sur silicium modernes et vraisemblablement la clé de la réussite de ceux de demain.

Ces techniques de conception centrées sur le réseau d'interconnexion (*Interconnect-centric Design Methodologies*) [SIA 04], rompent avec les flots traditionnels de conception en fournissant au plus tôt dans la phase de conception des connaissances et des contraintes spécifiques à l'interconnexion et facilitent ainsi la conception globale du système sur silicium. La Figure 2-26 fournit l'illustration d'un flot de conception moderne en mettant en relief les outils qui doivent maintenant tenir compte des paramètres d'interconnexion, rompant les distinctions traditionnelles entre les différents niveaux de conception.

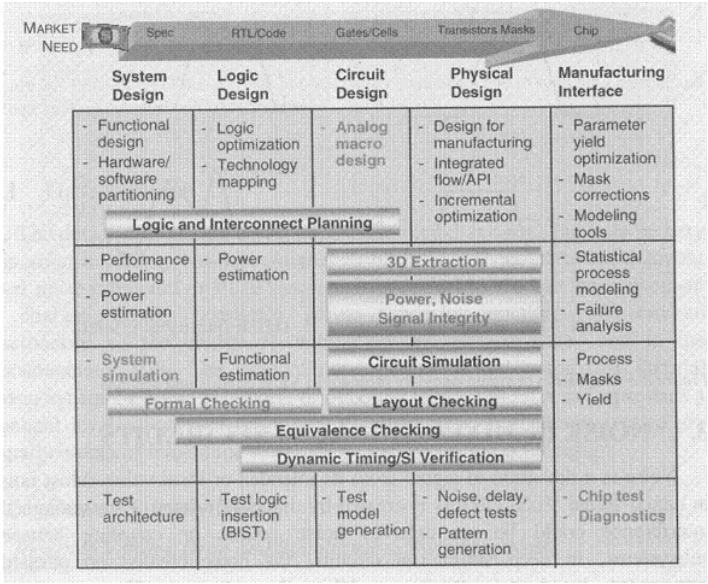


Figure 2-26 : flot de conception moderne montrant la nécessité de prendre en compte des paramètres physiques de plus en plus en amont [SIA 04]

L'enjeu fondamental de cette méthodologie centrée sur l'interconnexion est d'améliorer la connaissance au plus tôt de l'estimation des paramètres de l'interconnexion. Par exemple à l'aide d'outils d'exploration de topologies [AHO 04], d'analyse efficace des schémas de communication utilisés [HEM 98] ou encore d'estimation du coût de l'interconnexion physique sous contraintes de bruit et de performance [ZHE 99].

2.8. Conclusion

Dans ce chapitre important nous avons présenté l'ensemble des paramètres qui caractérisent le réseau d'interconnexion d'un système sur silicium, qu'il soit synchrone ou asynchrone, en insistant en particulier sur trois caractéristiques fondamentales de ces réseaux : le degré de flexibilité, la fiabilité et la qualité de services des protocoles de communication, et les différentes familles de topologies existantes.

La flexibilité définit à la fois le degré d'adaptation du réseau d'interconnexion à différents modules périphériques hétérogènes et sa facilité de réutilisation dans de futurs systèmes. Elle est caractérisée selon trois indicateurs :

- l'extensibilité du réseau vers de nouvelles architectures dépend de la régularité de sa topologie, de la flexibilité et de la robustesse de ses protocoles de niveau signal ;
- la variété et la qualité des communications de niveau services ;
- la capacité d'interfaçage avec d'autres composants, périphériques ou réseaux, grâce à l'utilisation de ponts et d'adaptateurs. La facilité de la connexion des composants périphériques au réseau va dépendre de leur hétérogénéité.

Le protocole de communication de niveau signal garantit le bon acheminement des informations entre les deux extrémités de la communication. Il détermine l'aptitude du système de communication à synchroniser des blocs cadencés par des horloges différentes (asynchronisme). Il fournit le degré de fiabilité dit matériel du système, et s'il est jugé insuffisant, doit être renforcé par des solutions fournies par les protocole de services.

Ces protocoles de service permettent d'élaborer des communications aux formats complexes, construites à partir des actions atomiques des protocoles de niveau signal indifféremment synchrones ou asynchrones. La mise en forme des communications utilisant ces protocoles de services doit être faite par le périphérique et non par le réseau lui-même. Nous avons présenté en détail deux familles de services :

1. les services d'arbitrage pour résoudre les contentions d'accès à un nœud du réseau ou à une ressource périphérique ;
2. les services de transactions destinés à mettre en formes les données pour optimiser les performances de la communication.

Les réseaux de communication présentent aujourd'hui une grande variété d'implémentations physiques, dont les principales familles de topologies et leurs propriétés ont été présentées : le bus centralisé inadapté au paradigme des systèmes globalement asynchrones et localement synchrones, les architectures distribuées en anneau, crossbar ou maille. Ces topologies ont été illustrées par des exemples de réalisations industrielles et universitaires de réseaux intégrés synchrones et asynchrones représentatifs. Tous les critères présentés dans ce chapitre et qui caractérisent le réseau d'interconnexion d'un système sur silicium, nous serviront pour orienter les choix de conception des réseaux de communication sans horloge présentés au chapitre 4. Mais avant cela le chapitre 3 introduit les techniques de conception asynchrone et le formalisme de modélisation de circuits asynchrones que nous utilisons en entrée de l'outil de conception automatique de circuits asynchrones TAST développé par le groupe CIS du laboratoire TIMA.

Chapitre 3 : Conception de circuits sans horloge : propriétés, techniques et outils.

L'intérêt de la conception de circuits sans horloge pour les réseaux de communication dans les architectures GALS a été présenté au chapitre 1, en particulier au §1.4. Les protocoles de niveau signal asynchrones ont été introduits au §2..3.2.3, en mettant en avant leurs atouts pour la robustesse, la modularité et l'extensibilité des systèmes. Ce chapitre 3 introduit les techniques de conception asynchrone, puis le formalisme de modélisation de circuits asynchrones, que nous utilisons en entrée de l'outil de conception automatique de circuits asynchrones TAST (*TIMA Asynchronous Synthesis Tool*). Ce formalisme de modélisation est basé sur le langage CHP (*Communicating Hardware Processes*), un langage de description de haut niveau, sous forme de processus communicants.

3.1. La conception de circuits sans horloge

Ce paragraphe présente les différentes notions concernant la conception logique de circuits asynchrones. Il s'agit de circuits sans horloge globale (Cf. §1.2.) et dont la synchronisation entre les blocs est uniquement assurée par des signaux de contrôle locaux. Nous présentons ici les principes de base qui seront nécessaires à la compréhension des autres paragraphes de ce chapitre et des chapitres suivants. Ces principes de base sont : les protocoles de communication, le codage des données, les différentes classes de circuits sans horloge.

3.1.1. Généralités

Les circuits asynchrones définissent une classe très large de circuits, dont le contrôle ou le à point est assuré par toute autre méthode que le recours à un signal périodique globalement distribué. Le point clef de ces circuits est que le transfert d'information est géré localement par une signalisation adéquate. Les opérateurs connectés se synchronisent en échangeant des informations indépendamment des autres opérateurs auxquels ils ne sont pas connectés. 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 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. Le contrôle local doit par conséquent prendre en charge une signalisation bidirectionnelle. Toute action de communication doit être

acquittée par le récepteur afin que l'émetteur puisse émettre à nouveau. Les communications sont dites à *poignées de mains* (ou *handshake*) ou de type *requêtes-acquittements* (cf. Figure 3-1).

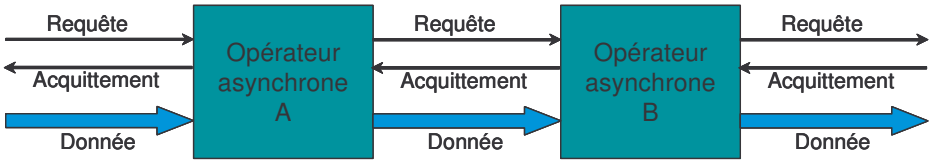


Figure 3-1 : Communications entre des opérateurs asynchrones.

3.1.2. Les protocoles de communications

Pour implémenter une telle gestion des échanges, deux principaux protocoles de communication sont utilisés : le protocole deux-phases (ou NRZ pour Non Retour à Zéro, ou encore *Half-Handshake*), et le protocole quatre-phases (ou RZ pour Retour à Zéro ou *Full-Handshake*).

3.1.2.1. Le protocole deux phases

Le fonctionnement du protocole deux-phases est décrit Figure 3-2, les données invalides sont en hachuré. Un cycle se déroule de la façon suivante :

Phase 1 : phase active du récepteur : il détecte la présence de nouvelles données par le signal de requête, effectue le traitement et génère le signal d'acquittement.

Phase 2 : phase active de l'émetteur : il détecte le signal d'acquittement et émet de nouvelles données si elles sont disponibles.

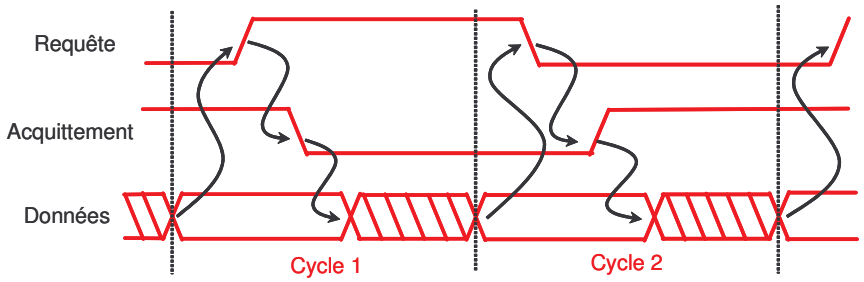


Figure 3-2 : Protocole de communications deux-phases.

3.1.2.2. Le protocole quatre phases

Le fonctionnement du protocole quatre-phases est décrit Figure 3-3. Le fonctionnement est très similaire au protocole deux-phases. La différence est que chaque signal de contrôle a besoin d'une phase de réinitialisation. La communication est alors sensible à des niveaux logiques et non à des transitions. Un cycle se déroule de la façon suivante :

Phase 1 : première phase active du récepteur : il détecte la présence de nouvelles données par le signal de requête, effectue le traitement et génère le signal d'acquittement.

Phase 2 : première phase active de l'émetteur : il détecte le signal d'acquittement et émet des données invalides.

Phase 3 : deuxième phase active du récepteur : il détecte la présence des données en état invalide et place le signal d'acquittement dans son état initial (retour à un).

Phase 2 : deuxième phase active de l'émetteur : il détecte le retour à un de l'acquittement. Il est alors prêt à émettre de nouvelles données.

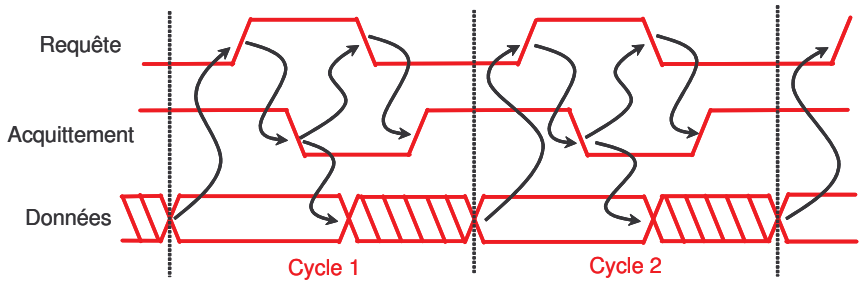


Figure 3-3 : Protocole de communications quatre-phases.

Il existe plusieurs variantes du protocole quatre-phases. Le point commun est que la donnée valide est toujours valide à partir du front montant de la requête. Ce qui les différentie est l'évènement qui déclenche le passage des données à l'état invalide. Il y a plusieurs possibilités : le front montant de l'acquittement (*Broad*), le front descendant de la requête (*Broadish*) ou enfin le front descendant de l'acquittement (*Early*).

Le protocole quatre-phases requiert deux fois plus de transitions que le protocole deux-phases. Il est à priori plus lent et consomme plus d'énergie. Toutefois le protocole deux-phases nécessite une logique de contrôle beaucoup plus importante que le protocole quatre-phases car il doit détecter des transitions et non des niveaux logiques. La consommation et la vitesse de ces deux types de codage sont donc très dépendantes de leur utilisation judicieuse. Ce point est repris au chapitre 7.

3.1.3. Codage des données

Contrairement à la logique synchrone où données et validités sont complètement décorrélées, la logique asynchrone doit non seulement coder la donnée, mais aussi sa validité. Le but étant d'assurer la synchronisation présentée dans le paragraphe précédent. Le codage est bien sûr lié au type de protocole choisi.

3.1.3.1. Le codage données groupées ou "bundle data"

C'est la façon la plus simple d'implémenter les bits de contrôle : parallèlement aux bits de données, un fil de requête est explicitement ajouté permettant ainsi de dissocier le contrôle des données proprement dites. Il est possible de faire de même pour le signal d'acquittement. Lorsque toutes les données sont disponibles, le bit de requête déclenche la mémorisation ou le

traitement des données associées. De même, une fois toutes les données reçues, le signal unique d'acquittement est généré (cf. Figure 3-4). Ce type de codage est principalement utilisé dans les montages *micropipeline*. Les protocoles de communication deux-phases ou quatre-phases pourront tous deux être utilisés avec ce type de codage. Un bit est codé sur un seul fil, le qualificatif *mono rail* (ou *single rail*) est souvent employé. Le fil spécifiant la validité des données (requête) est typiquement implémenté avec le délai adéquat, correspondant au temps de calcul du pire cas.

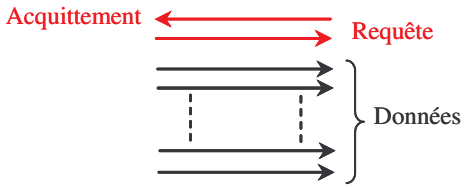


Figure 3-4 : Codage « données groupées ».

3.1.3.2. Les codages insensibles au délai

Cette approche plus complexe consiste à intégrer l'information de validité dans les données. Cette approche possède l'avantage que les données valides sont détectées sans aucune hypothèse temporelle. Ceci donne lieu à des circuits plus difficiles à concevoir mais plus robustes et portables.

Le codage quatre états

Dans ce type de codage, chaque bit de donnée est représenté par deux fils. Parmi les quatre états possibles, deux sont réservés à la représentation d'un *'1'*, les deux autres à la représentation d'un *'0'*. L'émission d'une nouvelle donnée se traduit par un changement sur un des deux fils. Deux conventions existent :

Pour la première, un fil est dédié à une valeur (*'1'* ou *'0'*) et tout évènement sur un des fils signifie qu'une valeur est valide.

Pour la deuxième, les valeurs *'1'* ou *'0'* d'un bit sont codées avec deux combinaisons : l'une avec une parité paire, l'autre de parité impaire. Chaque fois qu'une donnée est émise, sa parité est changée. Ceci permet donc de détecter le changement d'une donnée sans passer par un état invalide (cf. Figure 3-5).

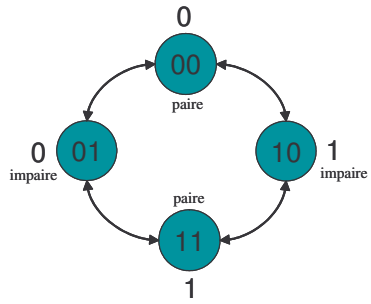


Figure 3-5 : Codage quatre états.

Le codage trois états

Comme pour un codage quatre états, ce type de codage utilise deux fils par bit pour le codage de chaque bit. Parmi les quatre possibilités fournies par ce codage, une seule représente le *'1'*, une autre le *'0'*, une troisième des données invalides et la quatrième n'est pas utilisée (cf. Figure 3-6). Ainsi une transition des données en entrée se fait automatiquement par le passage par l'état invalide.

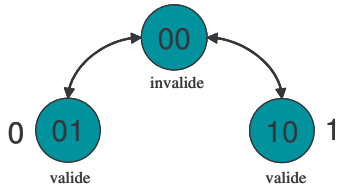


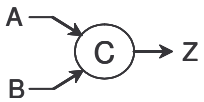
Figure 3-6 : Codage trois états.

Il est clair que le codage trois états est bien adapté au protocole quatre-phases, alors que le codage quatre états est adapté au protocole deux-phases. Le codage trois états est actuellement le plus utilisé pour des raisons d'implémentation et de sécurité. En effet, pour le codage quatre états, il faut implémenter de la logique pour détecter la parité. Le codage trois états est souvent appelé *double rail ou DR*. L'état invalide est déterminé par la remise à zéro de tous les fils. Chaque double rail représente un digit qui permet de coder 2 valeurs soit 1 bit équivalent.

Cette notion peut être généralisée en codant chaque digit en base *N*, où *N* fils sont utilisés (codage 1 parmi N ou Multi-Rail). En particulier le codage Multi-Rail à quatre voies ou MR4 présente un intérêt certain par rapport au codage DR (qui n'est autre qu'un codage MR2). Le codage MR4 représente un digit de quatre fils qui permet de coder les valeurs 0 à 3, soit 2 bits équivalents, avec à chaque fois une seule transition sur un des quatre fils. Or en codage DR il faut deux digits de deux fils pour coder le même nombre de valeurs, soit deux transitions à chaque fois sur deux des quatre fils. Pour le même nombre de lignes physiques, le codage MR4 utilise donc deux fois moins de transitions (à quantité d'information égale) que le codage DR, pour un coût en logique combinatoire équivalent.

3.1.4. La porte de Muller

L'implémentation des protocoles vus précédemment est impossible avec des portes logiques standard. La porte de base utilisée dans les circuits asynchrones est la porte de Muller ou *C-element*. Elle réalise le rendez-vous entre plusieurs signaux. Sa sortie est égale à ses entrées lorsque toutes les entrées sont égales. Dans tous les autres cas, la sortie est mémorisée. Une porte de Muller à deux entrées ainsi que sa table de vérité est donnée Figure 3-7.



A	B	Z
0	0	0
0	1	Z^{-1}
1	0	Z^{-1}
1	1	1

Figure 3-7 : porte de Muller à deux entrées et sa table de vérité

Deux implémentations électriques sont données Figure 3-8, à base de portes complexes et en transistors.

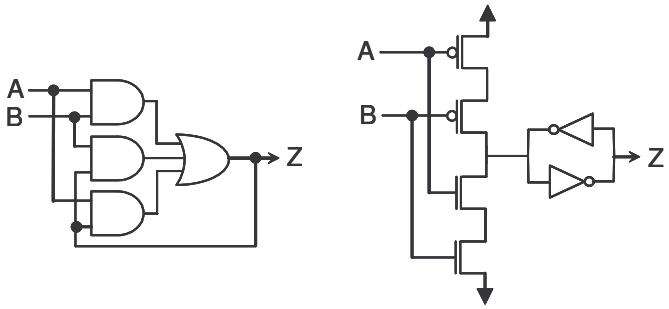


Figure 3-8 : implémentions électriques d'une porte de Muller deux entrées

Il existe des variantes des portes de Muller dites portes de Muller généralisées. Dans ce cas les signaux qui font basculer la sortie de porte à *'1'* ne sont pas toujours les mêmes que ceux qui la font basculer à *'0'*. La porte est alors qualifiée de dissymétrique [DIN 03], [RIG 02a].

3.1.5. Les modèles de délais

Les modèles qui régissent le comportement des délais interviennent dans la caractérisation des circuits asynchrones.

Les délais sont communément caractérisés par un comportement dit *pur* ou *inertiel*. Un *délai pur* décale dans le temps toute forme d'onde d'un signal de la valeur de ce délai. En VHDL, ce type de délai est appelé *transport delay*. Cependant cela n'est pas très réaliste car cela implique que les portes et les pistes du circuit possèdent une bande passante infinie. Un modèle de délai plus réaliste est le *délai inertiel*. En plus de décaler la forme d'onde du signal, celui-ci supprime les impulsions courtes (i.e. certains aléas). Le modèle de délai inertiel utilisé en VHDL se caractérise par deux paramètres : le temps de décalage (*delay time*) et le temps de rejet (*reject time*). Ainsi les impulsions plus courtes que le temps de rejet sont alors filtrées. Ce modèle de délai est celui utilisé par défaut en langage VHDL.

Les délais peuvent aussi être caractérisés vis-à-vis du temps :

- le *délai fixe* possède une valeur constante.
- le *délai borné* est compris entre une limite inférieure et une limite supérieure qui sont connues.
- le délai peut être *non borné* : sa valeur est comprise dans un intervalle dont les limites sont inconnues. Cela dit, sa valeur est finie, mais non connue. Par exemple, ce modèle de délai est utilisé pour les portes dans les circuits indépendants de la vitesse.

3.1.6. Les différentes classes de circuits asynchrones

Les circuits asynchrones sont communément classés suivant le modèle de circuit et d'environnement utilisé. Cette terminologie est présentée Figure 3-9. Plus le fonctionnement respecte fondamentalement la notion d'asynchronisme et plus le circuit est robuste et complexe. Les circuits vont donc des circuits synchrones avec les hypothèses temporelles les plus fortes aux circuits purement asynchrones.

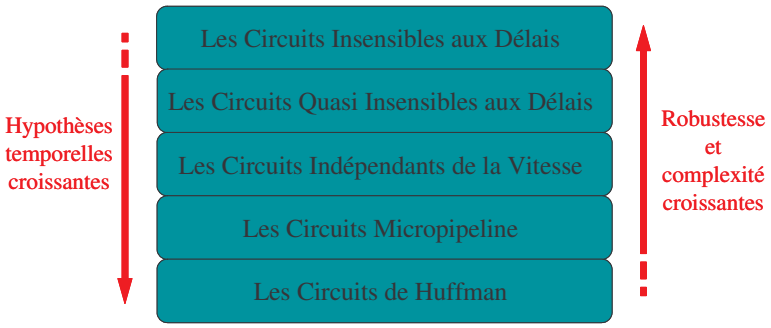


Figure 3-9 : Classification des circuits asynchrones.

3.1.6.1. Les circuits Insensibles aux Délais

Cette classe de circuits utilise un mode de fonctionnement purement asynchrone. Aucune hypothèse temporelle n'est introduite, c'est-à-dire qu'ils sont fonctionnellement corrects indépendamment des délais introduits par les fils ou les éléments logiques, quelle que soit leur complexité. D'un point de vue théorique, cela signifie qu'ils sont basés sur un modèle de délai pour les fils et les éléments qui est non borné.

Les circuits DI (*Delay Insensitive*) sont donc supposés répondre toujours correctement à une sollicitation pour effectuer un calcul. Ceci impose donc au récepteur d'un signal de toujours informer l'expéditeur que l'information a été reçue. Les circuits récepteurs doivent donc être capables de détecter la réception d'une entrée et/ou la fin de son traitement. Les circuits émetteurs doivent attendre un compte rendu avant d'émettre une nouvelle donnée. Les contraintes de réalisation pratique qu'impose l'utilisation de ce modèle sont très fortes. De plus, la plupart des circuits conçus aujourd'hui utilisent des portes logiques à une seule sortie. L'adoption du modèle de délai non borné ne permet pas leur utilisation. En effet, les portes logiques standards ont leurs sorties qui peuvent changer si une de leurs entrées change. Comme il a été souligné, toutes les composantes de ce type de circuits doivent s'assurer du changement des entrées avant de produire une sortie. Si la sortie change alors qu'une seule des entrées change, seul le changement de l'entrée active est acquitté, sans pouvoir tester l'activité des autres entrées. La seule porte à une sortie qui respecte cette règle est la porte de Muller. Mais les fonctions réalisables avec seulement des portes de Muller sont très limitées.

La seule solution est d'avoir recours à un modèle de circuits de type « portes complexes » pour les composants élémentaires. Dans ce cas, la construction de ces circuits se fait à partir des composants standards plus complexes que de simples portes logiques et qui peuvent posséder plusieurs entrées et plusieurs sorties.

3.1.6.2. Les circuits Quasi Insensibles aux Délais

Cette classe de circuits (QDI – *Quasi Delay Insensitive*) adopte le même modèle de délai non borné pour les connexions mais la notion de fourche isochrone (*isochronic fork*) est ajoutée. Une « fourche » est un fil qui connecte un expéditeur unique à deux récepteurs. Elle est qualifiée d'isochrone lorsque les délais entre l'expéditeur et les récepteurs sont identiques. Cette hypothèse a des conséquences importantes sur le modèle et les réalisations possibles. Elle résout notamment le problème de l'utilisation de portes logiques à une seule sortie. En effet, si les fourches sont isochrones, il est permis de ne tester qu'une des branches de la fourche en supposant que le signal s'est propagé de la même façon dans l'autre branche. En conséquence, l'acquittement d'une seule des deux branches de la fourche isochrone est autorisé.

A. J. Martin [MAR 93] a montré que l'hypothèse temporelle de la 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. Ainsi, les circuits QDI se caractérisent par l'adoption d'un modèle de délais pour les connexions qui est de type non borné, avec en plus l'hypothèse de « fourche isochrone » et un modèle de type « porte simple » pour les composants élémentaires du circuit. Les circuits QDI sont donc séquencé avec des cellules standard telles qu'elles sont utilisées pour la conception de circuits synchrones. 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.

3.1.6.3. Les circuits indépendants de la vitesse

Les circuits indépendants de la vitesse (SI –*Speed Independant*) font l'hypothèse que les délais dans les fils sont négligeables, tout en conservant la modèle « non borné » pour le délai dans les portes. Dans un modèle SI toutes les fourches sont donc isochrones. S. Hauck [HAU 95] montre comment une fourche isochrone peut être représentée par un circuit indépendant de la vitesse Figure 3-10 (les Δ sont des temps de propagation).

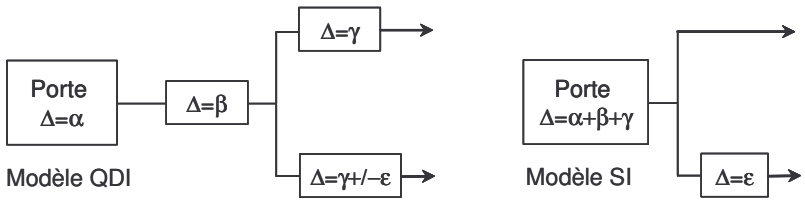


Figure 3-10 : équivalence entre les modèles QDI et SI.

3.1.6.4. Les circuits micropipeline

La technique micropipeline a été introduite par I. Sutherland [SUTH 89]. Les circuits de cette classe sont composés de parties contrôle insensibles aux délais qui commandent des chemins de données conçus en utilisant le modèle de délai borné. La structure de base de cette classe de circuits est le contrôle d'une queue de type FIFO. Elle se compose d'éléments identiques connectés tête bêche comme indiqué Figure 3-11.

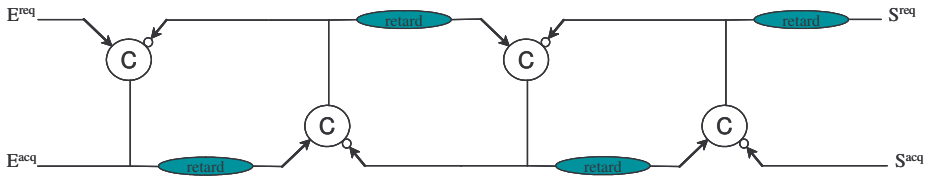


Figure 3-11 : structure de base des circuits micropipeline

Le circuit réagit à des transitions de signaux et non pas à des états (protocole deux-phases). Il peut également être qualifié de logique à événements, chaque transition étant associée à un évènement. Ainsi, si tous les signaux sont supposés à zéro initialement, une transition positive sur Ereq provoque une transition positive sur Eacq qui se propage également dans l'étage suivant. Le deuxième étage produit une transition positive qui d'une part se propage à l'étage suivant, mais qui d'autre part revient au premier étage l'autorisant à traiter une transition négative cette fois. Les transitions de signaux se propagent donc dans la structure tant qu'elles ne rencontrent pas de cellule « occupée ». C'est un fonctionnement de type FIFO. Cette structure peut être enrichie d'opérateurs de mémorisation ou pipeline, et d'opérateurs de traitement combinatoires comme indiqué Figure 3-12.

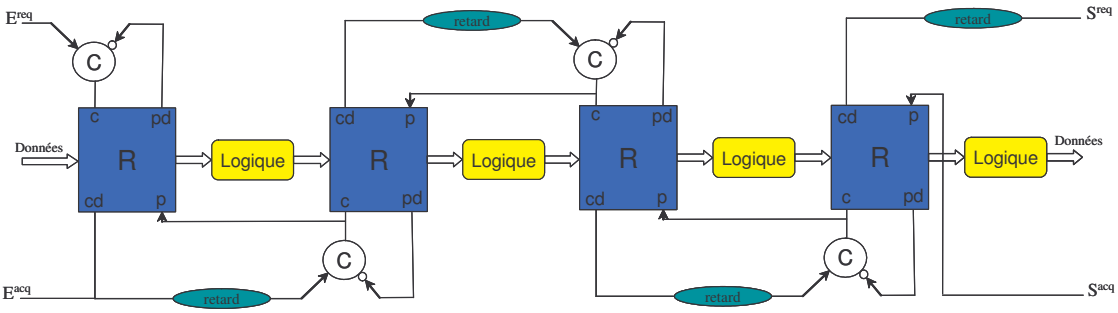


Figure 3-12 : structure micropipeline avec traitement et registre

Dans cette structure, les opérateurs notés R sont des registres qui capturent la donnée entrante sur l'occurrence du signal C. Ils produisent le signal Cd lorsque la donnée est mémorisée. En pratique Cd est une version retardée du signal C. Durant cette phase, la donnée précédemment mémorisée dans le registre est maintenue en sortie. Lorsque P est actif, le registre laisse passer la donnée d'entrée à la sortie. Le signal Pd indique que le registre est bien transparent. La structure de la Figure 3-12, dépouillée de la logique réalise une FIFO (initialement, tous les registres laissent passer les données). Avec la logique, la structure est celle d'un circuit asynchrone micropipeline.

Le protocole utilisé est de type données groupées. Lors d'une occurrence de Ereq, les données d'entrée sont stockées dans le premier étage. Ereq est propagée vers l'étage suivant qui stocke le résultat transmis par la logique du premier étage. La propagation de Ereq est retardée de façon à s'assurer que la logique combinatoire a bien convergé avant la capture du résultat dans le deuxième étage. La capture du deuxième étage étant effectuée, le premier registre est rendu passant, ce qui permet le traitement de la donnée présente dans le premier étage et autorise la prise en compte d'un nouvel évènement sur Ereq du premier étage.

La motivation première pour le développement de cette classe de circuits est de permettre un pipeline élastique. En effet, le nombre de données présentes dans le circuit peut

être variable, les données progressant dans le circuit aussi loin que possible en fonction du nombre d'étages disponibles ou vides. Cependant, ces circuits révèlent des inconvénients.

Tout d'abord, il faut remarquer que les problèmes d'aléas ont été écartés en ajoutant des délais sur les signaux de contrôle. Cela permet en fait de se ramener à un fonctionnement en temps discret dans lequel la mémorisation de données est autorisée seulement lorsqu'elles sont stables (à la sortie des parties combinatoires). Les délais étant de durée fixe, ce type de circuits effectue les traitements en un temps pire cas. Il ne peut donc pas être tiré parti de la variation dynamique de la chaîne critique des opérateurs de traitement.

Il est possible cependant de s'affranchir de cette contrainte en utilisant des registres et des opérateurs combinatoires capables de générer leur propre signal de fin de calcul sans avoir recours à des délais fixes. Des structures du type de celle donnée Figure 3-13 dans laquelle les délais fixes ont été remplacés par des délais variables implémentés dans les délais et la logique combinatoire sont obtenues. Ici le temps de calcul peut varier en fonction des données. Il ne s'agit plus de circuits utilisant le modèle de délai borné et le circuit peut être rendu indépendant de la vitesse puisque la seule hypothèse temporelle consiste à assurer que l'occurrence du signal de contrôle Ereq succède les données.

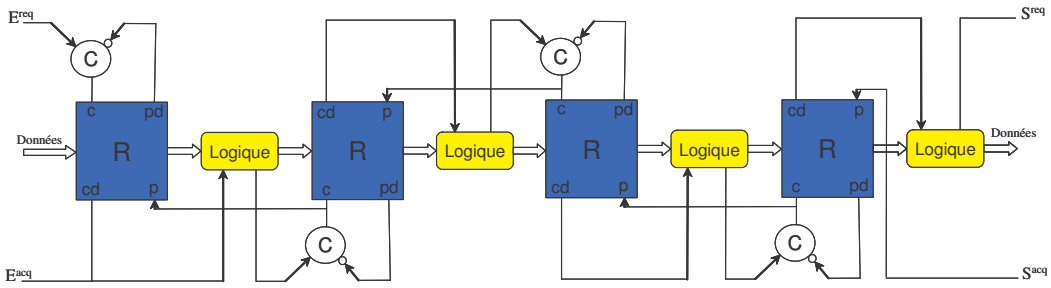


Figure 3-13 : circuit cas indépendant de la vitesse

D'autres optimisations peuvent être apportées à la structure de base donnée Figure 3-12. Il est par exemple possible d'utiliser des bascules de mémorisation sur double front dans le cas d'un protocole deux-phases [YUN 96a]. Enfin, des protocoles quatre-phases optimisés permettent un taux de remplissage de la structure important, afin d'augmenter les performances.

3.1.6.5. Les circuits de Huffman

Les circuits de cette classe utilisent un modèle de délai identique aux circuits synchrones. Ils supposent que les délais dans tous les éléments du circuit et les connexions sont bornés ou même de valeur connue. Les hypothèses temporelles sont donc du même ordre que pour la conception de circuits synchrones. Leur conception repose sur l'analyse des délais dans tous les chemins et les boucles de façon à dimensionner les signaux de contrôle locaux qui s'apparentent davantage à des horloges locales. Ces circuits sont d'autant plus difficiles à concevoir et à caractériser qu'une faute de conception ou de délai les rend totalement non fonctionnels.

3.2. Modélisation par un langage de haut niveau adapté à la synthèse de circuits asynchrones

Il n'existe pas aujourd'hui de langage de spécification universellement reconnu comme étant le plus apte à permettre la modélisation des circuits asynchrones. Cependant l'ensemble des langages, et des outils associés, développés pour la conception de circuits asynchrones se base sur le concept de processus concurrents communicants par canaux. A l'origine de ce concept se trouve le langage CSP (*Communicating Sequential Processes*) développé par Hoare [HOAR 78]. Les langages les plus populaires au sein de la communauté des concepteurs de circuits asynchrones sont probablement le CHP de l'Université de Caltech [MAR 93], Tangram de Philips [BERK-91], le langage Balsa de l'Université de Manchester [BARD 02].

Le formalisme de modélisation de circuits asynchrones que nous utilisons est une version modifiée du langage CHP qui permet en particulier la modélisation du non-déterminisme (Cf.Chapitre 4). TAST est l'acronyme de « *TIMA Asynchronous Synthesis Tools* ». C'est un environnement de conception dédié à la synthèse de circuits asynchrones.

Le paragraphe 3.2.1 présente le langage CHP modifié utilisé au sein du groupe CIS et le paragraphe 3.2.2 présente l'outil TAST.

3.2.1. Le langage CHP

Notre version du CHP possède une syntaxe et une sémantique pour modéliser l'indéterminisme dans un programme, qui sont présentées en détails en Annexe A. Cela permet au concepteur de spécifier explicitement les problèmes d'indéterminisme pour ensuite appliquer un outil de synthèse qui dérive correctement le circuit correspondant.

La spécification de ce langage est très proche du VHDL : un composant matériel est défini par son interface de communication (port d'entrées/sorties), une partie déclarative et un jeu de processus concurrents. La description de chaque processus est découpée en trois parties :

1. l'interface de communication, sous forme d'une liste de ports directionnels connectés à des canaux locaux ou directement aux ports d'entrées/sorties du composant ;
2. une section déclarative pour les variables locales et les canaux internes ;
3. une section d'instructions.

Une instruction peut être une affectation de variable, une lecture ou une écriture de canal de communication (Cf. 3.1.2). Elle peut aussi être une composition séquentielle ou parallèle d'instructions ou encore une structure de choix composée de commandes gardées.

Le CHP est basé sur les commandes gardées. Supposons que G soit une expression booléenne et S une liste de commandes, la commande gardées est alors : G => S. Elle signifie que S s'exécute quand l'expression G est vraie. La garde G est calculée par une expression arithmétique et logique de variables ou de "probes". La fonction probe ("#" en syntaxe CHP) teste l'activité d'un canal et peut aussi tester la valeur présente dans le canal.

Le CHP propose deux structures de contrôle : le choix et la répétition. Pour chacune d'entre elles le non déterminisme peut être spécifié. Au final les structures CHP possibles sont : le choix déterministe (1) et la répétition déterministe (2), le choix non déterministe (3) et la répétition indéterministe (4). Leur syntaxe est la suivante :

(1) @[G_1 => S_1 ; break ... G_n => S_n; break]

(2) @[G_1 => S_1 ; loop ... G_n => S_n ; loop]

(3) @@ [G_1 => S_1 ; break ... G_n => S_n ; break]

(4) @@[G_1 => S_1 ; loop ... G_n => S_n ; loop]

Dans un choix déterministe (1), au plus une garde est vraie. L'exécution est suspendue jusqu'à ce qu'une garde soit vraie. La commande S correspondant à la garde qui est vraie est exécutée et le choix se termine.

Dans une répétition déterministe (2) (noté par « *loop* »), au plus une garde est vraie. La répétition continue à exécuter les commandes associées à la garde qui est vraie. Quand aucune garde n'est vraie, la répétition se termine.

Dans le cas où l'exclusion mutuelle (Cf. 4.1.4.2) ou la stabilité des gardes ne peuvent être garanties [MAR 93], le choix (3) et la répétition indéterministes (4) sont utilisés. Cela se traduit au niveau circuit par un problème d'arbitrage. Dans ce cas plusieurs gardes peuvent être vraies en même temps, mais une seule, arbitrairement choisie est sélectionnée et la commande correspondante est exécutée (Cf. 4.1.4.2).

3.2.2. L'outil de synthèse TAST

TAST (*TIMA Asynchronous Synthesis Tool*) est un ensemble de logiciels dédiés à la conception de circuits asynchrones. Il est composé d'un compilateur, d'un simulateur et d'un synthétiseur offrant la possibilité de cibler plusieurs formats de sorties (description comportementale en VHDL, formats spécifiques de simulation, description structurelle au niveau porte logique en VHDL) à partir d'un langage de description haut niveau : le CHP (Cf. Annexe A). Le flot de synthèse est organisé autour des réseaux de Pétri et des graphes de données. La synthèse de circuits asynchrones est basée sur la spécification DTL (*Data Transfert Level*) [DIN 03]. La spécification DTL fournit un ensemble de règles qui garantissent que le réseau de Pétri et les graphes de données sont synthétisables vers des circuits asynchrones.

TAST cible plusieurs types de circuits à partir de la description sous la forme de processus communicants :

- une description comportementale en un format spécifique simulable par un simulateur C intégré dans TAST ;
- une description comportementale en langage VHDL simulable ;
- les circuits quasiment insensibles aux délais sous la forme d'une description VHDL au niveau porte ;
- les circuits micro pipeline sous la forme d'une description VHDL au niveau porte.

Le flot de TAST est présenté figure suivante :

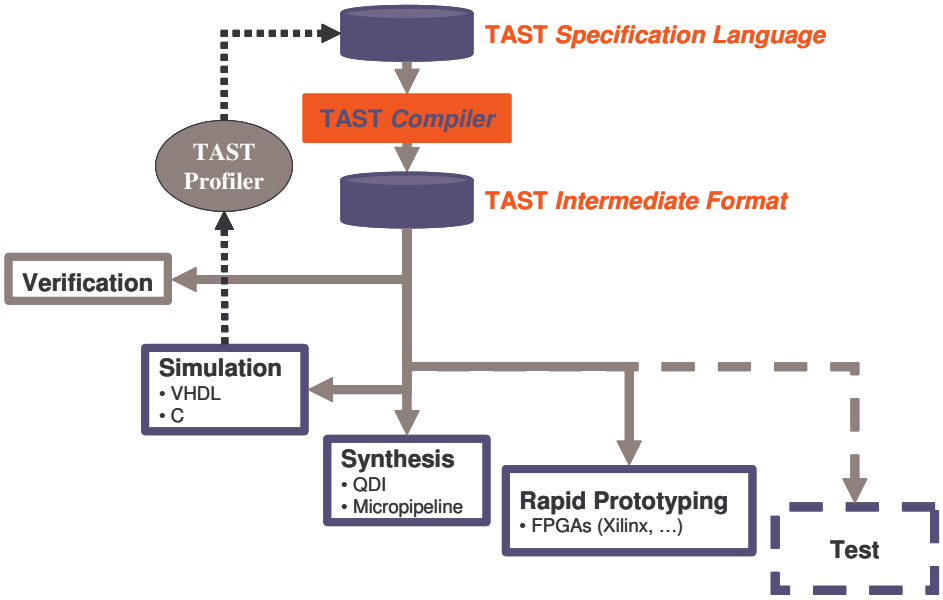


Figure 3-14 : flot de synthèse de l'outil TAST.

TAST accepte en entrée des systèmes décrits en CHP. Le CHP, de l'anglais *Communicating Hardware Processes*, est un langage de description haut niveau de circuits basé sur les processus communicants. Le CHP de TAST supporte les différents points suivants : les protocoles de communication, le codage des données, la spécification de l'indéterminisme, la hiérarchie, la gestion de projet.

3.2.2.1. Transformation en réseaux de Pétri

Le programme CHP est compilé et transformé par l'outil TAST en un ensemble de réseaux de Pétri (structures de contrôle et de choix, séquentialité, parallélisme) et de graphes de données (instructions associées aux places des réseaux de Pétri). La Figure 3-15 présente le modèle sémantique en réseaux de Pétri d'un canal utilisant un protocole quatre phases (Cf. 3.1.2.2).

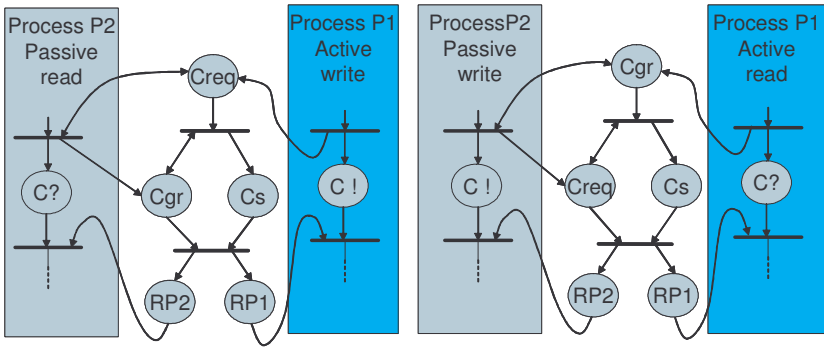


Figure 3-15 : modélisation en réseaux de Pétri du protocole quatre phase a) écriture active / lecture passive b) écriture passive / lecture active

Les réseaux de Pétri permettent la vérification formelle des modèles CHP. Ces réseaux autorisent une méthode de "*model checking*" [DUM 03] permettant l'analyse des propriétés de vivacité, de l'absence de blocages et de l'exclusion mutuelle des gardes. D'ailleurs une part importante du procédé de vérification est d'analyser si les structures de choix des processus

CHP satisfont l'exclusion mutuelle des gardes, ou non. L'impact de cette analyse sur le circuit implémenté est essentiel : si les choix sont mutuellement exclusifs, le circuit ne nécessite pas d'implémenter des éléments de résolution spécifiques. En revanche les structures de choix où plusieurs gardes peuvent être vraies ensemble doivent être protégés par des éléments d'exclusion mutuelle ou des synchroniseurs (Cf. §4.1.3 et §4.1.4).

Le travail sur la vérification formelle des circuits asynchrones se fait en collaboration avec deux partenaires universitaires : l'université de Newcastle upon Tyne et le groupde VDS du laboratoire TIMA. La Figure 3-16 présente le flot automatisé de conception de circuits asynchrones avec l'utilisation des deux outils de validation automatique des spécifications de circuits asynchrones écrites en CHP développés par ces deux groupes de recherche.

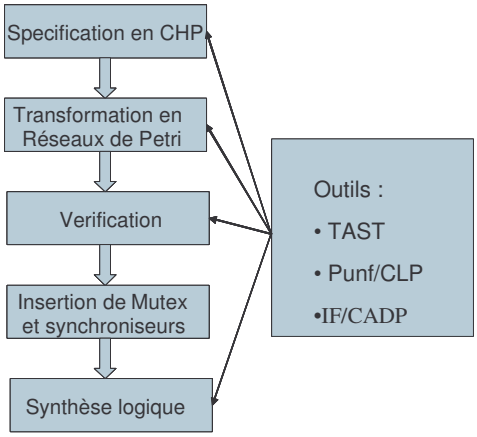


Figure 3-16 : flot de conception de circuits asynchrones

L'outil TAST supporte la construction des spécifications en CHP et leur conversion en réseaux de Pétri. L'insertion d'éléments d'exclusion mutuelle ou des synchroniseurs est encore manuelle. L'outil Punf/CLP développé par l'université de Newcastle upon Tyne permet ensuite de vérifier ces réseaux par la méthode dite *unfolding prefix method*. L'étude de cet outil n'est pas abordée ici mais le lecteur se référera à [YAK 03].

Le groupe VDS du laboratoire TIMA a travaillé sur deux approches pour la validation des circuits asynchrones. La première consiste à adapter la vérification symbolique de modèles, initialement dédiée aux circuits synchrones, pour la vérification des circuits asynchrones. Les spécifications de circuits sont alors traduites dans un modèle en VHDL pseudo synchrone et ensuite vérifiées par des outils industriels de vérification symbolique de modèles. Mais cette méthode impose de détailler les protocoles de communication en VHDL trop tôt dans le flot de conception, alors que l'expansion des communications en CHP est réalisée au moment de la synthèse en portes logiques, permettant de conserver un haut niveau d'abstraction.

Dans la deuxième approche, la sémantique de CHP, initialement donnée en termes de réseaux de Pétri, est reformulée en termes de Systèmes de Transitions Etiquetées Etendus (STEE) [BOR 03a]. Les spécifications de circuits sont alors validées par des méthodes énumératives de vérification de modèles. Pour augmenter les performances de l¹approche énumérative et faire face au problème d'explosion d'états, le groupe VDS a développé et implémenté un certain nombre de techniques automatiques de réduction et d¹abstraction sous forme d'un outil appelé IF/CADP [BOR 03b].

3.2.2.2. Le flot de vérification fonctionnelle

Afin de pouvoir vérifier par simulation la correction fonctionnelle du circuit obtenu, la spécification, une fois vérifiée formellement, peut être transformée en :

- un modèle de simulation utilisé dans un outil de traces propre à TAST, qui permet de visualiser le parallélisme des processus et le taux d'activité de chaque branche dans les structures de choix ;
- un modèle VHDL comportemental utilisé dans les outils standard de simulation.

3.2.2.3. Le flot pour la synthèse

Une fois la validation de la description CHP effectuée, il faut vérifier qu'elle est compatible avec les règles de spécification DTL. Si cela n'est pas le cas, le concepteur transforme le code CHP [DIN 03]. La synthèse permet le choix entre deux classes de circuits asynchrones : la classe de circuits micro pipeline et quasiment insensibles aux délais (QDI).

Le circuit, une fois synthétisé, se présente sous la forme d'une description VHDL au niveau porte (*netlist*) composées de portes de Muller et de cellules standard. La *netlist* obtenue doit être à son tour validée. A ce niveau du flot de conception, la connaissance de la technologie cible n'est pas obligatoire. Pour simuler le circuit synthétisé, une bibliothèque de cellules décrites en VHDL comportemental est disponible.

Enfin, le lien avec les étapes back-end du flot est réalisé par une nouvelle bibliothèque de cellules décrites cette fois de façon structurelle (à base de cellules standard ou spécifiques) [RIG 02a]. La connaissance de la technologie cible est maintenant nécessaire pour utiliser les outils standard associés qui permettent entre autre de faire de la simulation électrique et du placement routage.

TAST permet de cibler également les techniques de prototypage rapide et de mapper les circuits synthétisés sur les FPGA standard (circuits synchrones) [T. Q. HO 02]. Pour parvenir à cela, une dernière version de bibliothèque de cellules est disponible. Elles sont construites sur la base de LUT (« Look-Up Table »).

Pour mener à bien la conception et la réalisation d'un circuit asynchrone avec TAST, quatre bibliothèques génériques sont utilisées :

- une bibliothèque fonctionnelle pour la simulation indépendante de la technologie cible ;
- une bibliothèque structurelle pour la réalisation d'ASIC construite à partir de cellules standard ;
- une bibliothèque structurelle pour la réalisation de FPGA construite à partir de LUTs ;
- une bilbiothèque structurelle pour la réalisation d'ASIC construite à partir de cellules spécifiques de portes de Müller.

3.3. Conclusion

Nous venons de présenter succinctement les propriétés et les techniques de conception des circuits asynchrones mises en œuvre au sein de l'outil TAST grâce à une modélisation en langage de haut niveau, le CHP. Or l'un des objectifs de ces travaux de thèse est de formaliser la conception des réseaux de communication sans horloge, en particulier la modélisation du non-déterminisme (Cf.Chapitre 4), de manière à permettre leur génération automatique par l'outil TAST. Le chapitre 4 suivant présente donc une méthodologie de conception de réseaux de communication sans horloge utilisant ces techniques et propriétés des circuits asynchrones. Cette méthodologie vise trois objectifs :

1. répondre aux problèmes de conception des systèmes sur silicium présentés au chapitre 1,
2. répondre aux exigences spécifiques des réseaux d'interconnexion développées au chapitre 2,
3. permettre à terme la synthèse automatique de structures déterministes, des arbitres et des réseaux de communication par l'outil TAST.

Chapitre 4 : Méthodologie de conception semi-automatique d'un réseau de communication asynchrone décrit en langage de haut niveau

Ce chapitre commence par définir les problèmes de non-déterminisme et de métastabilité qui ont déjà été soulevés plusieurs fois au cours des précédents chapitres. L'élément synchroniseur destiné à prévenir la métastabilité est introduit dans ce chapitre, ainsi que la modélisation du non-déterminisme par le langage CHP (*Communicating Hardware Processes*). A partir de l'étude du non-déterminisme et du synchroniseur, nous évaluons le coût d'une synchronisation par communication à travers le réseau d'interconnexion. Enfin, nous proposons une méthodologie de conception de réseaux d'interconnexion qui s'appuie sur les propriétés d'excellente modularité des circuits sans horloge et l'illustrons à travers la spécification d'une architecture GALS.

4.1. Le cœur du problème de synchronisation au sein des réseaux : le non-déterminisme

4.1.1. Le non-déterminisme

4.1.1.1. Définition

La spécification d'un système définit une séquence d'états et de sorties qui doivent être produits en réponse à une séquence d'entrée. L'implémentation doit être conforme à cette spécification. Cependant, si cette dernière contient des séquences partiellement ordonnées ou des décodages partiels de valeur (données dont la valeur de certains bits est sans importance), l'implémentation peut fournir plusieurs séquences de réponses correctes possibles. Une implémentation qui choisira toujours la même séquence de réponses sera dite déterministe. Une implémentation non déterministe fournira une réponse arbitraire parmi les séquences correctes possibles.

Le non-déterminisme est donc la caractéristique, généralement indésirable, que possède un système à fournir plusieurs séquences de sortie différentes en réponse à une même

séquence d'entrée. Un circuit semi-conducteur peut être sensible à plusieurs facteurs physiques qui rendent sa sortie non déterminée :

- le bruit électrique ;
- les délais relatifs entre signaux, eux-mêmes dus à des variations de procédés de fabrication ;
- la sensibilité de l'horloge au déphasage et à la gigue.

Tous ces phénomènes s'accentuent avec l'avancée dans des technologies CMOS profondément submicroniques : dégradation du rapport signal sur bruit, variations accrues des procédés, augmentation de la fréquence de fonctionnement des composants.

Ce non-déterminisme se présente sous deux formes distinctes bien spécifiques :

- le problème, que nous venons d'introduire, de génération de séquences de sortie multiples pour une unique séquence d'entrée. Cette caractéristique est illustrée au paragraphe 4.1.1.4.
- le phénomène de métastabilité qui caractérise un état dont la durée n'est pas bornée. Le problème particulier de la métastabilité est abordé en détail au paragraphe 4.1.2.

4.1.1.2. Les circuits synchrones déterministes

Par construction un circuit synchrone est toujours déterministe. L'état suivant et les sorties du système sont uniquement déterminés par l'état courant et les entrées. Tous les signaux qui sont échantillonnés par l'horloge sont conçus pour respecter des contraintes de temps de propagation pire cas. Ils se stabilisent à leur valeur logique finale bien avant le front d'horloge suivant (respect des règles de *setup time*).

Il n'existe donc qu'un seul état futur correct du système, indépendamment du bruit ou des variations de procédés. L'observation éventuelle d'une séquence de sortie différente de la séquence prédite est considérée comme l'observation d'une défaillance du système (Cf. §4.1.1.3. et Figure 4-1). La seule adaptation possible d'un circuit synchrone face à une erreur de cette nature est d'adapter sa fréquence de fonctionnement: il faut concéder que ce circuit n'est pas capable de fonctionner à une telle fréquence et réduire sa vitesse d'exécution. Cependant si cette erreur apparaît de manière plus systématique sur des circuits d'une même série de production, il faut alors revoir la qualité des procédés de fabrication ou modifier la conception pour obtenir des circuits fonctionnant à la fréquence du cahier des charges.

4.1.1.3. Les circuits GALS non déterministes

Les circuits globalement asynchrones et localement synchrones sont non déterministes, bien que les blocs synchrones soient déterministes. Les signaux de contrôle, que ce soit les requêtes ou les acquittements (Cf. 3.1.1), qui permettent la communication entre les blocs synchrones sont des signaux asynchrones vis-à-vis des entrées des blocs synchrones. Leurs dates d'évènement ne sont ni prédictibles ni corrélées. Ils sont échantillonnés par l'horloge des

blocs synchrones et donc l'ordre relatif des transitions de ces signaux d'entrées et des transitions d'horloge sont imprédictibles.

Si un de ces blocs synchrones possède de multiples entrées asynchrones, les délais relatifs entre les signaux de contrôle de la communication rendent la séquence d'entrée, et par conséquent la séquence de sortie, non déterminées. Les paragraphes suivants illustrent le phénomène de génération de séquences multiples sur les exemples suivants :

- le comportement d'une architecture de pipeline GALS (Cf. 4.1.1.4) ;
- l'échantillonnage d'un signal asynchrone par une bascule D synchrone (Cf. 4.1.1.5).

En outre le deuxième exemple introduit l'autre source de non-déterminisme dont les GALS ne sont pas exempts, la métastabilité (Cf. 4.1.2).

4.1.1.4. Illustration du non-déterminisme au niveau comportemental

Contexte : un pipeline d'instructions GALS

Prenons l'exemple d'un pipeline d'instructions de profondeur quatre, inspiré d'un cœur de processeur implémenté en Verilog [HEATH 03], dont nous allons imaginer la réalisation et la validation complètes. Les quatre étages du pipeline sont :

1. chargement de l'instruction,
2. lecture et décodage,
3. exécution,
4. écriture du résultat.

Nous désirons que l'architecture soit du type GALS. Chaque étage est donc localement synchrone et globalement asynchrone. Chaque étage va exécuter un traitement sur les données en un temps fixe mais à une fréquence différente des autres étages. La communication par poignée de main va prendre un temps variable selon la date d'échantillonnage des signaux de synchronisation (Cf. paragraphe suivant). Ce pipeline est donc élastique [REN 00] : il autorise une variation des durées (exécution + synchronisation) de chaque étage du pipeline.

Pour valider ce pipeline, nous observons la sortie de chaque étage. Cette observation se fait à la fréquence de fonctionnement de l'étage 4 qui écrit le résultat du traitement des données dans la mémoire. Il est donc possible que relativement à la fréquence d'observation que nous venons de choisir, l'exécution des étages précédents prenne un nombre de cycles d'horloge variable.

Génération de la séquence d'entrée

On désire réaliser grâce à ce pipeline la séquence d'instructions suivante :

Séquence d'instructions exécutées dans le pipeline:

```
I1: ADD  R3, R1, R2
I2: MUL  R5, R3, R4
I3: SUB  R4, R2, R1
I4: MOV  R6, R3
I5: ADD  R4, R3, R2
```

Figure 4-1 : Exemple de séquence d'instructions exécutée dans un pipeline

L'ordonnancement est partiel. Ainsi les tâches d'écriture des résultats respectifs des instructions I2, I3 et I4 sont sans relation d'ordre et peuvent donc se terminer dans n'importe quel ordre. La Figure 4-2 présente l'ordonnancement des tâches correspondant à cette séquence d'instructions.

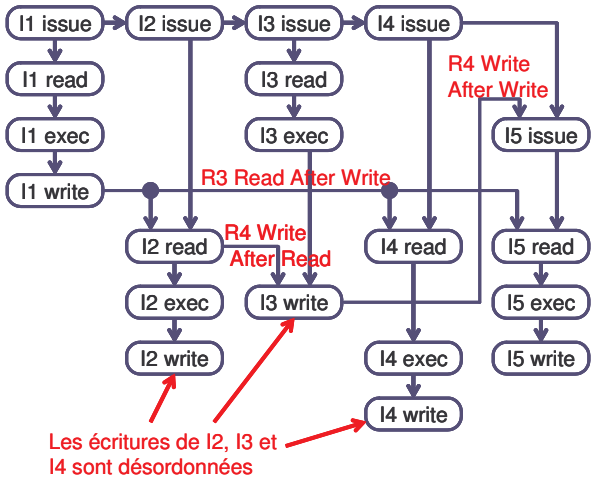


Figure 4-2 : ordonnancement des tâches dans le pipeline

Il existe par contre des relations d'ordre entre les autres tâches exécutant la séquence d'instructions. Le temps d'exécution de la tâche *I3 Write* par exemple va retarder d'autant le lancement de la tâche *I5 Issue*. La simulation des cycles d'exécution désordonnée de ce pipeline, une fois son architecture implémentée, fournit les résultats de la Figure 4-3. La tâche *I5 Issue* s'exécute au cycle 8 une fois seulement la tâche *I3 Write* exécutée au cycle précédent.

Clk	I1	I2	I3	I4	I5
1	Is				
2	Rd	Is			
3	Ex		Is		
4			Rd	Is	
5	Wr		Ex		
6		Rd			
7			Wr	Rd	
8				Ex	Is
9					Rd
10		Ex		Wr	Ex
11		Wr			
12					Wr

Figure 4-3 : cycles d'exécution désordonnée d'un pipeline GALS

Génération de séquences de sortie multiples

Le circuit une fois fabriqué, la séquence d'instructions de la Figure 4-1 est testée plusieurs fois (Figure 4-4) et l'on observe des séquences de résultats différentes : c'est l'illustration du problème de génération de séquences de sortie multiples au sein d'un GALS pour une unique séquence d'entrée.

Sur le premier test, les tâches d'exécution et d'écriture de résultat de l'instruction I3 sont retardées d'un cycle, provoquant également le décalage complet de l'instruction dépendante I5. Les instructions ont été exécutées selon la séquence suivante : I1, I3, I4, I2, I5. Sur le deuxième test, c'est l'exécution et donc l'écriture du résultat de l'instruction I2 qui sont retardées. Il n'y a pas de dépendance avec l'écriture du résultat de l'instruction I5. On obtient alors une séquence de sortie différente du premier test, où l'exécution de I2 se produit après celle de I5.

Clk	I1	I2	I3	I4	I5
1	Is				
2	Rd	Is			
3	Ex		Is		
4			Rd	Is	
5	Wr				
6		Rd	Ex		
7				Rd	
8			Wr	Ex	
9					Is
10		Ex		Wr	Rd
11		Wr			Ex
12					
13					Wr

Clk	I1	I2	I3	I4	I5
1	Is				
2	Rd	Is			
3	Ex		Is		
4			Rd	Is	
5	Wr		Ex		
6		Rd			
7			Wr	Rd	
8				Ex	Is
9					Rd
10				Wr	Ex
11		Ex			
12		Wr			Wr
13					

Figure 4-4 : test de la séquence d'instruction : obtention de séquences de résultats différentes

4.1.1.5. Illustration du non-déterminisme au niveau signal

Prenons l'exemple le plus simple : une bascule D qui échantillonne le signal reçu sur son entrée D à chaque transition positive du signal d'horloge. Cette bascule est placée à l'entrée d'un périphérique synchrone au sein d'une architecture GALS. Elle est donc destinée à échantillonner des signaux externes asynchrones de manière à se synchroniser avec son environnement.

La Figure 4-5 présente donc cette bascule D sous le nom de synchroniseur et illustre ses réponses possibles pour une unique stimulation d'entrée. La fonction du synchroniseur est présentée en détail au paragraphe 4.1.3.

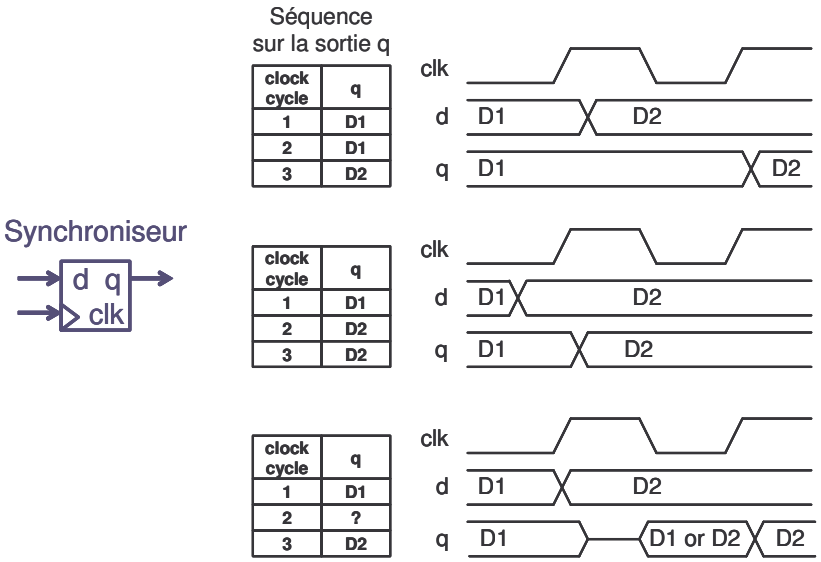


Figure 4-5 : échantillonnage d'un signal par le front d'horloge d'une bascule [HEATH 04]

Dans le chronogramme associé à la première séquence, le signal D asynchrone arrive juste après le premier front d'horloge. La sortie Q ne bascule donc qu'après le deuxième front d'horloge. Dans la deuxième séquence, le signal D asynchrone arrive juste avant le premier front d'horloge. La sortie Q bascule donc immédiatement après le premier front d'horloge. Le circuit localement synchrone lit donc en sortie de la bascule D deux séquences différentes[7].

La troisième séquence un peu particulière nous permet d'introduire la notion de métastabilité : le signal D commute au plus près du front d'horloge, la bascule D rentre dans un état métastable et ne fournit une réponse arbitraire qu'après un délai tout aussi arbitraire. Voyons cela plus en détail au paragraphe suivant.

4.1.2. La métastabilité

4.1.2.1. Définition

Le phénomène de métastabilité caractérise un état dont la durée est indéterminée, non bornée [REN 04b]. Un circuit peut entrer dans un état métastable lorsque un signal est échantillonné à une valeur intermédiaire qui ne permet pas de déterminer son niveau logique.

Reprenons l'exemple de la bascule D, appelée synchroniseur au paragraphe précédent. La Figure 4-6 rappelle la spécification temporelle de cette bascule :

- t_{SU} est le délai avant le front d'horloge à partir duquel le signal d'entrée doit être stable (*setup time*) ;

[7] Remarquons que dans le cas d'un système globalement synchrone, l'arrivée en retard du signal D dans la première séquence constituerait une violation de délai maximal.

- t_H est le temps de maintien de ce signal (*hold time*) après le front d'horloge qui doit être garanti avant que la valeur du signal d'entrée puisse varier à nouveau ;
- t_P est le temps de propagation dans la cellule, ou temps de réponse, avant d'obtenir l'établissement d'un signal de sortie stable (*settling time*).

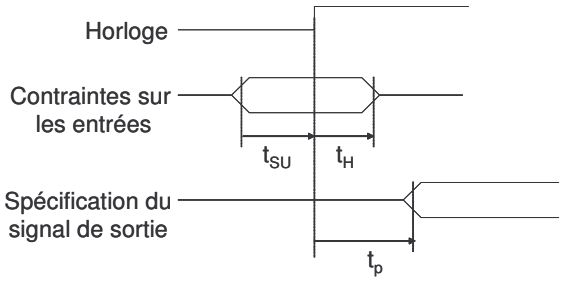


Figure 4-6 : spécifications temporelles d'une bascule D

Le phénomène de métastabilité va provenir de l'utilisation de cette bascule D comme synchroniseur (Cf. 3.3.2.2.) pour échantillonner sur front d'horloge un signal asynchrone qui ne respecte pas les contraintes temporelles, en particulier le temps d'établissement $t_{SU.}$

Dans les architectures GALS, la potentialité de métastabilité apparaît chaque fois qu'un bloc synchrone échantillonne au moyen de son signal d'horloge un signal asynchrone. Si la transition de ce signal asynchrone se produit trop près de la transition de l'horloge, on aboutit aux erreurs suivantes :

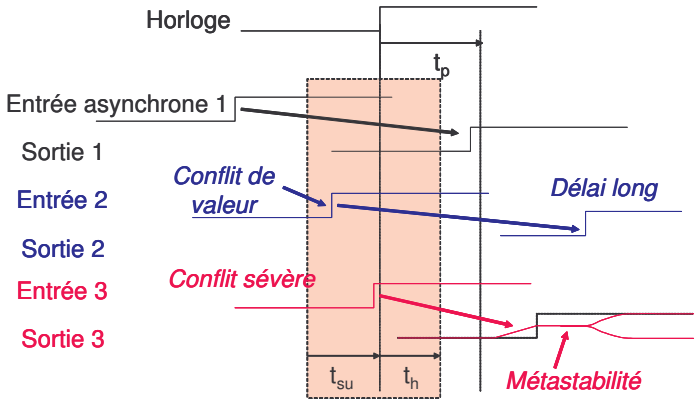


Figure 4-7 : erreurs de synchronisation dues au non respect des contraintes temporelles [GIN 02]

Le conflit de valeur sur l'entrée 2 entraîne un retard à l'établissement de la sortie 2, ce qui peut entraîner la violation des contraintes temporelles sur une éventuelle bascule suivante. Cependant la valeur fournie en sortie sera exacte en regard de l'entrée. Le conflit sévère sur la valeur 3 entraîne la métastabilité du circuit : le temps de stabilisation et la valeur de sortie seront arbitraires. L'élément synchroniseur est donc à l'origine de la métastabilité ! C'est pourtant ce circuit qui permet d'implémenter une structure de choix non-déterministe comme nous allons le voir au §4.1.4.2. De nombreux efforts se portent donc sur les architectures à base de synchroniseur pour améliorer ce circuit, afin de s'affranchir justement de cette métastabilité et améliorer la fiabilité des communications au niveau des protocoles de signal (Cf. 2.3.1et 2.3.2). Ce sera le propos du chapitre 6.

4.1.2.2. Probabilité d'occurrence de l'état métastable

La Figure 4-8 présente la courbe du temps de réponse de la bascule en fonction de la date d'arrivée du signal asynchrone en entrée de la bascule.

- Dans la zone 1, les contraintes temporelles sont respectées et le temps de réponse est fixe.
- Dans la fenêtre de délai long, le temps de réponse croît exponentiellement avec le retard de la donnée mais la sortie fournit une réponse avant la fenêtre de danger du prochain front d'horloge.
- Dans la fenêtre de métastabilité le temps de réponse dépasse un cycle d'horloge et est non borné.
- Dans la zone 4, la donnée arrive trop tard après le front d'horloge et n'est pas échantillonnée, d'où le temps de réponse nul : la sortie ne bascule pas.

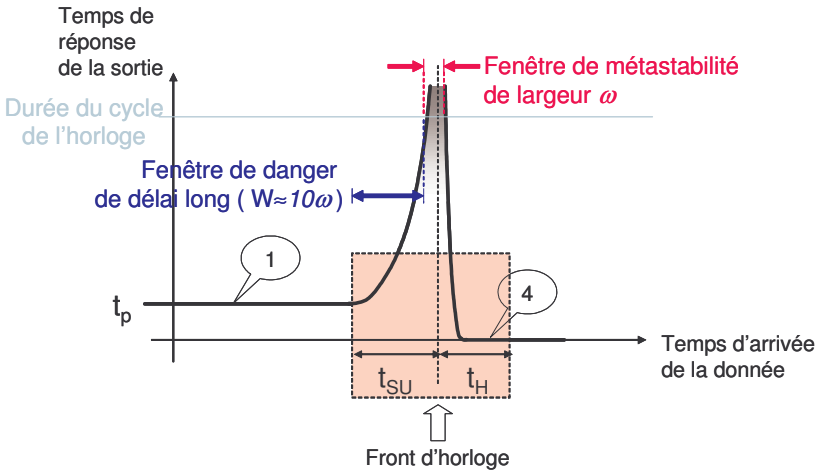


Figure 4-8 : temps de réponse d'une bascule D en fonction du délai d'arrivée de la donnée [GIN 02]

Dike et Burton présentent dans [DIK 99] la probabilité d'obtenir un délai long ou d'entrer dans un état métastable et montrent que la taille des fenêtres de danger augmentent linéairement avec la fréquence de fonctionnement du circuit . Soit F_c la fréquence d'horloge de la bascule et F_D la fréquence d'arrivée d'une donnée asynchrone. Les fréquences d'occurrence de délai long et de métastabilité sont alors :

- fréquence de délai long = $W \times F_c \times F_D$
- fréquence de métastabilité= $\omega \times F_c \times F_D$

Exemple [GIN 02]

Prenons une technologie CMOS 0,13µm et un système fonctionnant à F_c = 200 MHz, avec des données arrivant toutes les mille périodes d'horloge, soit à une fréquence de F_D = 0,2 Mhz. La fenêtre de délai long, pour une porte standard dans ce type de technologie, est estimée par [GIN 02] à W = 40 ps et la fenêtre de risque de métastabilité à ω = 4 ps. Le risque que la donnée se présente dans la fenêtre de délai long est alors estimé à 1,6 kHz, soit une donnée

toutes les 0,6 ms. Le risque que la donnée se présente dans la fenêtre de métastabilité est quant à lui estimé à 160 Hz, soit une donnée toutes les 6 ms en moyenne. On voit clairement qu'une solution doit être apportée pour réduire ce risque et fiabiliser le système : c'est la fonction du circuit synchroniseur présenté dans le paragraphe suivant.

4.1.3. Le synchroniseur

4.1.3.1. Fonctionnalité

La communication au sein des architectures GALS nécessite des interfaces asynchrones, caractérisées par un mécanisme de synchronisation. L'un de ces mécanismes permettant d'échantillonner un signal asynchrone est le synchroniseur, dont nous avons vu un exemple basique au paragraphe 4.1.1.5.

La fonction du synchroniseur est de résoudre l'un de ces deux problèmes équivalents :

1. étant donné une transition sur le signal asynchrone à échantillonner et une transition sur le signal échantillonneur[8], déterminer qui s'est produit en premier ; ou
2. étant donné une valeur de tension sur le signal asynchrone à échantillonner, déterminer à un instant donné si cette valeur est au-dessus ou en dessous des valeurs de seuil respectivement d'état logique haut ou d'état logique bas du signal.

La complexité d'un tel circuit va dépendre à la fois du degré de fiabilité exigé et du niveau de performances attendu. Le paragraphe suivant présente uniquement le synchroniseur traditionnellement utilisé. C'est le circuit le plus simple assurant une excellente fiabilité mais au prix d'une latence élevée. Le chapitre 6 se penchera plus en détail sur des réalisations plus performantes de synchroniseurs et plus généralement sur l'adaptation de domaines d'horloges.

4.1.3.2. Le synchroniseur double bascules

L'exemple du paragraphe §4.1.2.2 montre que l'utilisation d'une bascule unique ne peut pas convenir comme synchroniseur. Semiat & Ginosar montrent dans [SEM 03] que le circuit le plus simple permettant de réaliser un synchroniseur est de placer entre le signal de contrôle asynchrone et le bloc synchrone deux bascules en série synchronisées sur l'horloge du bloc. On parle de synchroniseur double flip-flop (FF) ou de *two flop synchronizer*. Le temps de stabilisation (*settling time*) autorisé pour le signal de sortie devient alors une entière période d'horloge.

La Figure 4-9 présente pour un codage données groupées quelques protocoles quatre phases de communication et leurs synchroniseur double flip-flop. Pour un codage multi-rail, il

[8] Nous avons considéré jusque-là l'horloge d'une bascule comme signal échantillonneur au sein d'un module localement synchrone. Ce signal échantillonneur peut être en fait n'importe quel signal de contrôle asynchrone (signal d'échappement) dans un environnement complètement asynchrone.

suffit d'extraire l'information de contrôle contenue dans les données (Cf. §3.1.3.2) en détectant une activité sur le canal (Cf. §3.2.1). La Figure 4-10 détaille la synchronisation pour le protocole *Push* de la Figure 4-9.

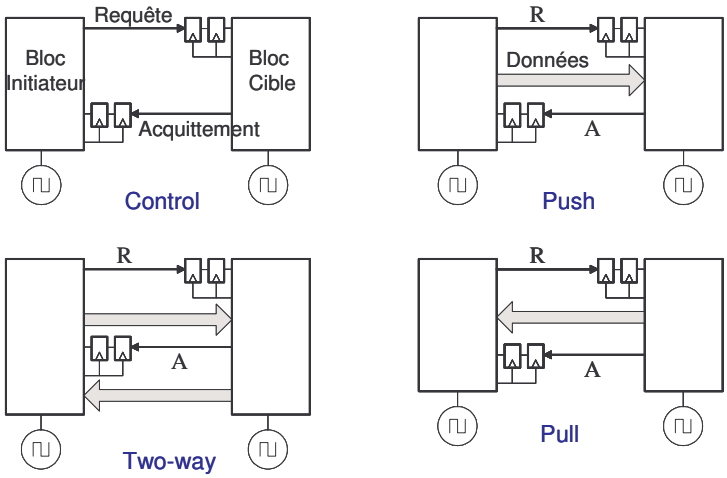


Figure 4-9 : protocoles quatre phases et synchroniseurs double flip-flop

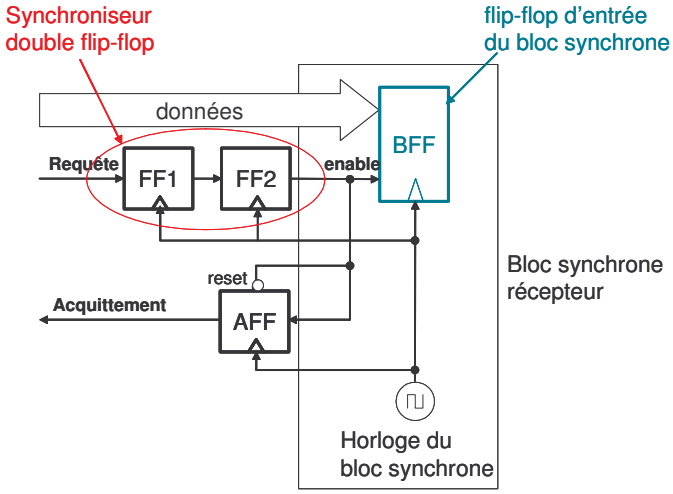


Figure 4-10 : synchroniseur double flip-flop

4.1.3.3. Temps moyen entre deux erreurs

La caractérisation d'un synchroniseur se fait sur deux paramètres essentiels :

- un paramètre de performance qui est son temps de réponse ou latence. Le synchroniseur double flip-flop induit une latence très importante de deux cycles d'horloge. Sa consommation d'énergie est également élevée.
- Un paramètre de fiabilité : le MTBF (*Mean Time Between Failures*) qui mesure la probabilité d'erreur de ce type de circuit. Le MTBF est donné par la formule suivante :

$$MTBF = \frac{e^{T/\tau}}{\omega . F_C . F_D}$$

Figure 4-11 : temps moyen entre deux erreurs d'un élément synchroniseur double flip-flop

On retrouve au dénominateur la formule du risque d'occurrence de métastabilité pour la bascule simple. Le facteur τ correspond au temps de réponse de cette même bascule (le *settling time* t_p vu précédemment). T représente la fenêtre de temps de stabilisation autorisé du signal entre les deux bascules du synchroniseur. Cela peut être une période complète si les deux bascules sont synchrones ou moins si on choisit par exemple de les rendre séquencées pour gagner en latence tout en respectant la fiabilité désirée.

Typiquement le MTBF est calculé pour être dix fois supérieur à la durée de vie estimée du circuit.

Exemple

En reprenant les données de l'exemple du paragraphe §4.1.2.2, avec un temps de réponse de la bascule estimé à τ = 30 ps et une fenêtre de temps de stabilisation T autorisée d'une entière période, on obtient un temps moyen entre deux erreurs de quelques 10^{63} secondes, ce qui garantit largement la durée de vie du système.

En revanche si maintenant on choisit une fréquence de fonctionnement Fc de 1 GHz au lieu du précédent 200 MHz, et qu'en outre le système impose pour des raisons de performance un déphasage d'une demie période d'horloge entre les deux bascules du synchroniseur, alors le MTBF tombe à $4*10^3$ secondes, soit à peine plus d'une heure. Il est donc clair que la question de la synchronisation ne peut être évitée par les concepteurs de systèmes sur silicium exigeant de hautes performances, et que la solution choisie pour assurer la synchronisation et donc la fiabilité du système doit être soigneusement étudiée et spécifiée pour offrir le meilleur compromis entre performance et fiabilité, comme nous le verrons au chapitre 6.

4.1.4. Modélisation et implémentation du non-déterminisme dans l'outil TAST

Ce paragraphe présente trois exemples de modélisation du non déterminisme en langage CHP : un générateur de nombre aléatoire pour se familiariser avec les conventions d'écriture en CHP, puis les deux structures de base indispensables pour résoudre les problèmes de synchronisation expliqués jusqu'à présent : l'élément d'exclusion mutuelle et le synchroniseur.

4.1.4.1. Un générateur de nombre aléatoire

L'outil TAST possède son propre simulateur qui utilise les représentations en réseaux de Petri produites par le compilateur. Ce premier exemple de générateur de nombre aléatoire permet de tester la capacité du simulateur à réaliser un tirage aléatoire entre deux évènements concurrents.

La Figure 4-12 fournit une description du générateur de nombre aléatoire. La syntaxe du langage CHP est détaillée en Annexe A.

Le composant "indet1" ne possède pas de port défini car il est fermé. La partie déclarative déclare deux constantes "zéro" et "un" et un canal local "data" formé d'un seul digit codé en double rail ("mr[2]" pour un multi-rail de deux bits soit un double rail et "[1]" pour le nombre de digits). Le générateur de nombre aléatoire est ensuite modélisé à l'aide de deux processus concurrents nommés "Generator" et "Client".

Client est un processus très simple qui boucle sur une seule action de communication sur le canal "data" : "?" symbolise l'action de lecture de la donnée émise par Generator ("!" pour une action d'écriture). Cela permet de compléter le protocole de communication sur ce canal afin que Generator puisse émettre à nouveau (Cf. 3.1.1).

Le processus Generator réalise une affection concurrente non déterministe de la variable x. Les deux valeurs "zéro" et "un" peuvent être potentiellement vraies en même temps, comme l'indique l'opérateur de parallélisme "," (Cf. Annexe A). L'opérateur ";" spécifie au contraire la séquentialité entre deux commandes.

Il est certain que la synthèse matérielle de ce circuit n'est possible que de manière déterministe, la variable "x" devenant une valeur figée. Par contre la simulation sur un temps très long du modèle comportemental de ce composant "indet1", généré par l'outil TAST, montre que le simulateur choisit la valeur "zéro" dans 50% des cas.

```
component indet1

constant zero : mr[2][1][1]:=0;
constant un   : mr[2][1][1]:=1;

channel data  : di mr[2][1];

process Generator
port ( data  : out di active mr[2][1] )
variable x   : mr[2][1][1];
begin
          [ x:=zero , x:=un ;
          data!x;
          loop
          ];
end;

process Client
port (data  : in  di passive mr[2][1])

begin
          [data?;loop ];
end;
```

Figure 4-12 : modèle CHP d'un système générateur de nombre aléatoire

Le langage CHP étant un langage de description sous forme de processus, la Figure 4-13 présente le même programme que celui de la Figure 4-12 mais sous une forme plus lisible et conviviale de processus CHP communicants. Ce format sera conservé pour les Figure 4-15 et Figure 4-17 modélisant respectivement l'exclusion mutuelle et le synchroniseur.

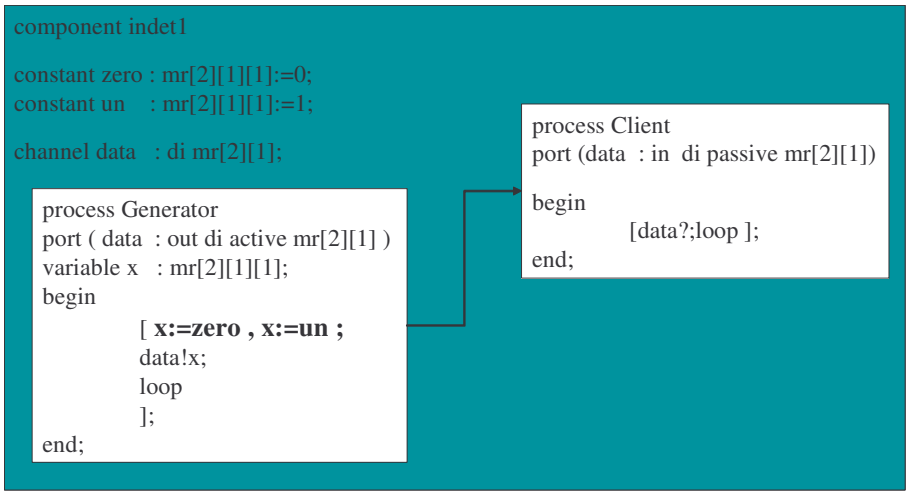


Figure 4-13 : système générateur de nombre aléatoire sous forme de processus CHP communicants

4.1.4.2. L'élément d'exclusion mutuelle

L'élément d'exclusion mutuelle permet de résoudre un conflit en environnement concurrent où deux demandeurs peuvent demander en même temps l'accès à la ressource partagée. La fonctionnalité de ce bloc est de choisir parmi deux requêtes celle qu'il faut considérer lorsqu'elles sont toutes les deux actives.

Cette situation est en réalité la plus simple situation d'arbitrage possible, illustrée par la Figure 4-14 : deux clients C1 et C2 sont en compétition pour accéder la ressource commune CR. Les canaux de communication et les signaux constitutifs de ces canaux sont représentés sur cette figure. Au niveau signal, C1req et C2req sont asynchrones et sans causalité entre eux. Ils doivent donc être synchronisés de manière à ce qu'un seul d'entre eux soit choisi pour accéder à la requête. Ainsi la propriété d'exclusion mutuelle est satisfaite et la communication sur le canal sélectionné peut être effectuée (validation du signal d'acquittement C1req ou C2req de manière exclusive).

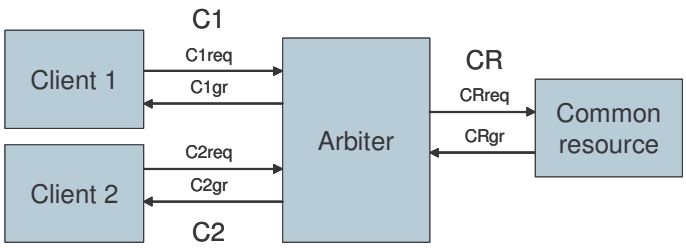


Figure 4-14 : environnement concurrent nécessitant un bloc MUTEX pour partager l'accès à une ressource commune

La modélisation en processus CHP communicants de cet arbitre à base d'élément d'exclusion mutuelle est présentée Figure 4-15. Sa dérivation sous forme de réseau de Petri est présentée en Annexe B.

Les processus Client_1 et Client_2 bouclent sur une action d'écriture sur leurs canaux respectifs C1 et C2. Le processus Common-Resource boucle sur une seule action de

communication sur le canal CR : il écrit la donnée émise par Arbiter dans la variable x. Cela permet de libérer le canal CR.

Le processus "Arbiter" implémente la compétition entre les deux clients C1 et C2 au niveau canal de communication par l'opérateur de choix non déterministe @@, présenté au §3.2.1. Cette notation spécifie que les gardes C1# et C2# peuvent être vraies simultanément, mais qu'une seule de ces deux gardes est sélectionnée et les instructions correspondantes exécutées.

L'opérateur de *probe* dénoté "#" permet de tester l'activité sur un canal. Ainsi, la sélection non déterministe @@ du processus Arbiter est en attente de requêtes sur les canaux C1 et C2. Lorsqu'au moins une de ces requêtes est active, au moins une des deux gardes devient vraie. Alors une unique garde est sélectionnée et les commandes correspondantes exécutées. Les canaux CR et C1/C2 sont accédés concurremment (opérateur ",") : la ressource commune est accédée par la commande CR!zero ou CR!un et concurremment la requête choisie est libérée par la commande C1? ou C2?. Ainsi le programme du processus Arbiter bouclera lorsque les deux communications auront été terminées.

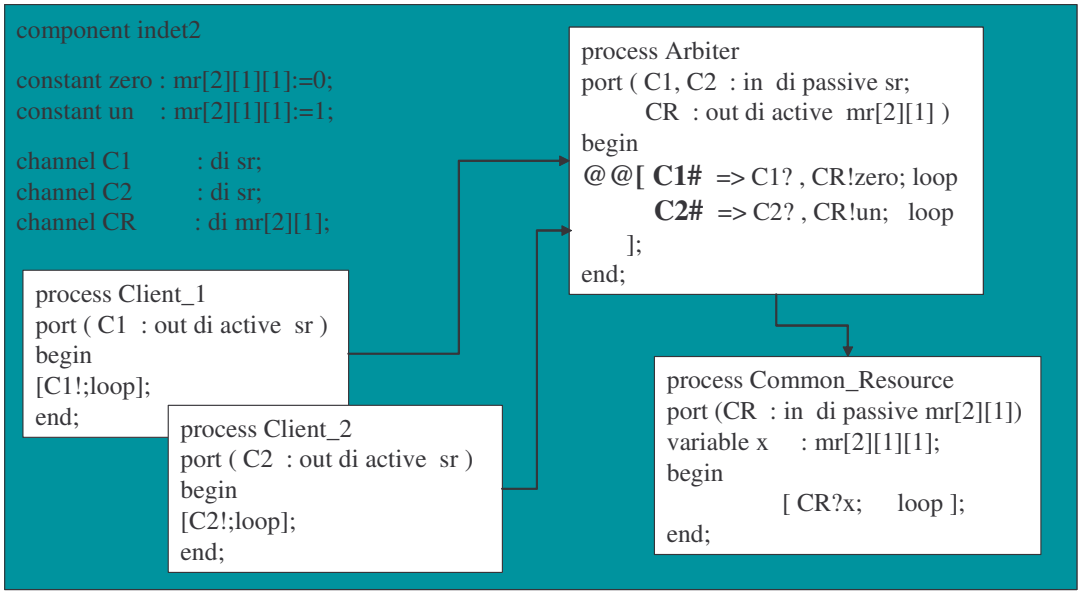


Figure 4-15 : modélisation en CHP de l'arbitre de la Figure 4-14

L'utilisation conjointe des opérateurs de choix non déterministe et de probe spécifie parfaitement le comportement de l'élément d'exclusion mutuelle, qui doit arbitrer entre deux requêtes asynchrones concurrentes. A noter que l'attente d'un évènement sur le canal d'entrée grâce à l'opérateur probe est intrinsèquement passive pour un circuit asynchrone (le circuit reste naturellement en veille tant que cet évènement n'a pas eu lieu).

4.1.4.3. Le synchroniseur

Le synchroniseur intervient dans les structures de choix qui implémentent des gardes instables. Le synchroniseur a pour fonction de déterminer la valeur logique d'un signal asynchrone vis-à-vis du signal de commande d'échantillonnage (Cf. 4.1.3). Dans un canal de communication asynchrone, cela revient à arbitrer si une requête est active ou non.

Considérons le système de la Figure 4-16, interruptible par le client "Interrupt Request". La tâche normale de ce système est la tâche A (processus "Task A") exécutée en boucle, et commandée par le processus "Arbiter" au moyen du canal A. Lorsque le client "Interrupt Request" émet une requête sur le canal INT, le processus "Arbiter" répond en insérant la tâche B (processus "Task B") dans le flot d'exécution.

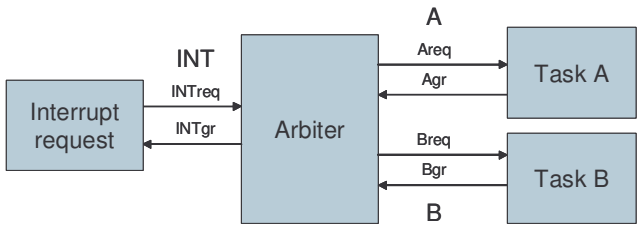


Figure 4-16 : échantillonnage asynchrone d'un signal d'interruption par un synchroniseur

La modélisation en processus CHP communicants de cet arbitre à base de synchroniseur est présentée Figure 4-17 (seule la tâche A est représentée). Sa dérivation sous forme de réseau de Petri est présentée en Annexe B.

```
component indet3

constant zero : mr[2][1][1]:=0;
constant un   : mr[2][1][1]:=1;

channel Req        : di sr;
channel CR         : di mr[2][1];

process Arbiter
port (INT : in  di passive sr;
      A, B  : out di active  mr[2][1] )
begin
@@[INT#     => INT?; A!zero; loop
  ¬INT#     => B!un;   loop
  ];
end;

process Interrupt Request
port ( INT  : out di active  sr )
begin
[INT!;loop];
end;

process Task_A
port (A  : in  di passive mr[2][1])
variable x     : mr[2][1][1];
begin
          [ A?x;     loop ];
end;
```

Figure 4-17 : modélisation en CHP du système de la Figure 4-16

La structure de choix non-déterministe @@ est également utilisée ici conjointement aux opérateurs de probe pour spécifier le comportement du synchroniseur. La différence ici est que le même canal INT est testé de manière non-déterministe :

- vrai avec la garde "INT#"
- faux avec la garde "¬INT#"

Ces gardes sont évidemment mutuellement exclusives mais elles ne sont pas stables. En effet, la valeur des signaux qu'elles scrutent peut changer après que l'une des commandes gardées ait été sélectionnée. Un mécanisme est donc nécessaire pour que la sélection reste stable, c'est-à-dire déterministe, même si un nouvel évènement survient sur le canal "INT". Le risque, présenté au §4.1.2.1, de créer un état métastable en utilisant un synchroniseur doit

cependant être pris en compte : le signal INTreq peut en effet transiter dans la fenêtre de danger (Cf. Figure 4-8) au moment où l'échantillonnage est réalisé.

De nombreux efforts se portent donc sur les architectures à base de synchroniseur pour améliorer ce circuit, afin de s'affranchir justement de cette métastabilité et améliorer la fiabilité des communications au niveau des protocoles de signal (Cf. 2.3.1et 2.3.2). Ce sera le propos du chapitre 6. La section 4.2 présente le coût de cette synchronisation dans une architecture GALS.

4.1.4.4. Degré d'automatisation de l'outil

A l'heure actuelle l'outil TAST ne reconnaît pas l'opérateur @@ de non déterminisme. La synthèse se fait donc en remplaçant cet opérateur par un opérateur classique @ de choix déterministe dans l'écriture du code CHP. Le concepteur doit ensuite modifier à la main le circuit généré, comme le présente le flot de conception de la Figure 3-16 du paragraphe 3.2.2.

4.1.5. Conclusion

Les deux derniers exemples de modélisation de comportements non déterministes présentés dans ce paragraphe peuvent être considérés comme des arbitres élémentaires. L'arbitrage à base de synchroniseur en particulier doit décider de la présence ou de l'absence d'un signal avant de résoudre une éventuelle contention entre plusieurs signaux de requêtes issus de demandeurs, tandis que celui à base d'exclusion mutuelle arbitre directement entre deux requêtes concurrentes. L'utilisation des éléments d'exclusion mutuelle et de synchroniseur pour la réalisation d'arbitres plus complexes est abordée au chapitre 5.

4.2. Le coût de la synchronisation

Cette section s'appuie sur l'étude précédente du non-déterminisme et du synchroniseur pour évaluer le coût d'une communication à travers le réseau d'interconnexion, selon le degré de synchronisme entre les composants du système sur silicium. Le Tableau 4-1 rappelle les techniques de synchronisation selon le degré de synchronisme.

Classe du circuit	$\Delta\phi$	Δf	Synchronisation
Synchrone	0	0	Non nécessaire
Mésochrone	ϕ_c	0	Compensation de phase
Plésiochrone	variable	$f_d<\varepsilon$	Compensation de phase adaptive
Hétérochrone		$f_d>\varepsilon$	Predictive
Asynchrone			Synchroniseurs à base de double flip-flop ou autre

Tableau 4-1 : type de synchronisation selon la classe de circuit

Les techniques de synchronisation par horloge ont été étudiées au §1.1 avec leurs avantages et leurs limites, qui nous ont amenées à choisir les architectures globalement

asynchrones et localement synchrones (Cf. 1.2). Nous allons donc nous intéresser à présent aux coûts de synchronisation des systèmes globalement asynchrones, selon le choix de leur réseau de communication avec ou sans horloge locale. La comparaison de ces coûts de synchronisation respectivement avec un réseau synchrone ou asynchrone est le sujet du paragraphe 4.2.1. Le paragraphe 4.2.2 se penche ensuite plus particulièrement sur la comparaison du coût de synchronisation des arbitres de séquencement respectivement localement synchrones et localement asynchrones, dans un environnement concurrent globalement asynchrone.

4.2.1. Synchronisation par communication

Rappel (Cf. §1.2.2.1) : par facilité de langage, on parle de communication asynchrone alors qu'on adresse le problème de synchronisation par communication.

4.2.1.1. Réseau synchrone

La Figure 4-18 présente une partie d'un système à l'architecture globalement asynchrone et localement synchrone. Le périphérique "Emetteur synchrone" échange des informations vers les autres périphériques au moyen d'un réseau sur silicium synchrone. Ce périphérique est muni d'une Interface de Communication (Cf. §2.2.3.1), non représentée sur la Figure 4.18, qui permet entre autres d'adapter le périphérique au réseau. Il adapte le protocole natif du périphérique en protocole asynchrone quatre phases données groupées (Cf. §3.1.2.2 et §3.1.3.1). Les signaux de contrôle de la communication (requête et acquittement) sont donc séparés des données. Pour des raisons de clarté, seuls ces signaux de contrôle sont représentés sur la Figure 4-18 ainsi que sur l'ensemble des autres figures suivantes de cette section 4.2.

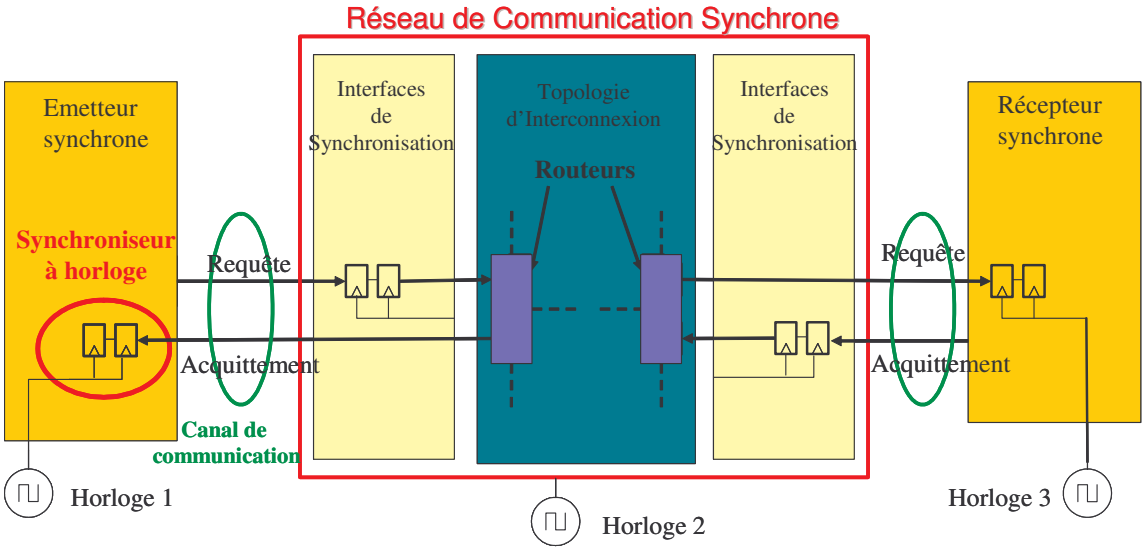


Figure 4-18 : synchronisation des communications d'un système GALS avec un réseau synchrone.

Le réseau d'interconnexion synchrone possède une horloge propre considérée asynchrone de celle des périphériques. Chacun des composants reçoit donc de l'autre composant un signal asynchrone vis-à-vis de son horloge. Conformément à l'étude menée au §4.1, les

communications entre les deux composants Emetteur Synchrone et Réseau doivent donc être synchronisées au moyen de circuits à base de synchroniseurs à horloge, car il s'agit ici d'échantillonner un signal de requête ou d'acquittement asynchrone vis-à-vis de l'horloge de chaque composant (Cf. §4.1.3 et §4.1.4.3). Il en va de même entre le Réseau et le Récepteur Synchrone. Les données n'ont pas besoin d'être synchronisées car le protocole *bundle data* garanti que les données sont prêtes lorsque la requête est émise.

La synchronisation de communication entre les périphériques synchrones à travers un réseau localement synchrone nécessite donc quatre synchroniseurs à horloge par transaction (ou deux synchroniseurs par canal de communication).

4.2.1.2. Réseau sans horloge

La Figure 4-19 présente la même architecture globalement asynchrone et localement synchrone que la figure précédente. Les périphériques synchrones restent identiques tandis que cette fois le réseau d'interconnexion est choisi asynchrone. Les périphériques synchrones nécessitent des synchroniseurs à horloge pour pouvoir échantillonner les signaux de requête ou d'acquittement émis par le réseau de communication et asynchrones vis-à-vis de leur horloge. En revanche le réseau de communication asynchrone s'affranchit de ces synchroniseurs. Il nécessite cependant l'utilisation de synchroniseurs sans horloge insensibles au délai pour permettre l'échantillonnage et l'arbitrage correct de requêtes concurrentes. Ce problème de coût de synchronisation d'un arbitre est développé au paragraphe suivant §4.2.2.

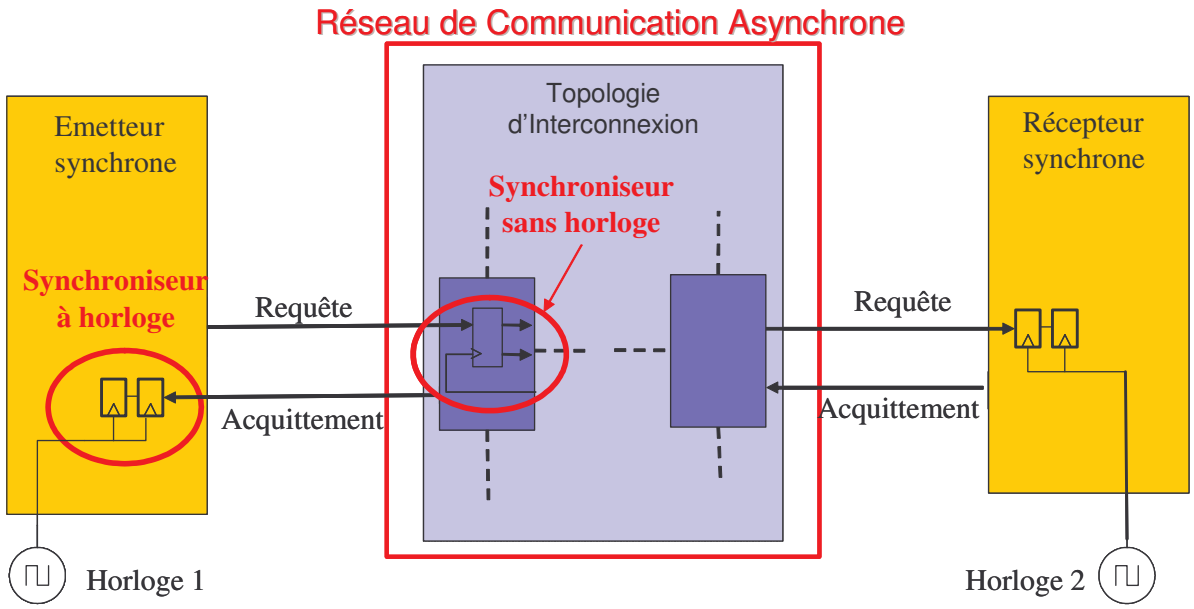


Figure 4-19 : synchronisation des communications d'un système GALS avec un réseau sans horloge.

La synchronisation de communication entre les périphériques synchrones à travers un réseau localement asynchrone nécessite donc deux synchroniseurs à horloge par transaction (ou un synchroniseur par canal de communication). Ainsi le nombre total de synchroniseurs à horloge aux interfaces de communication entre les composants et le réseau de communication

va doubler si ce dernier est synchrone, par rapport à un réseau asynchrone, et ce quelque soit le nombre de canaux de communication entre chaque périphérique et le réseau.

4.2.1.3. Comparaison des coûts de synchronisation

Il est important de préciser que cette comparaison porte uniquement sur le problème d'interfacer des composants intégrés animés par des horloges différentes. Les avantages potentiels du choix d'un réseau de communication sans horloge et ses inconvénients vis-à-vis d'un réseau sur silicium synchrone de même spécification ont été présentés dans les sections 1.3 et 1.4. Ce coût de synchronisation est un paramètre fondamental de conception des réseaux sur silicium qui doit être considéré à part entière. La synchronisation a été évoquée dans la section 2.3 à propos des protocoles de communication, et en particulier comme critère de fiabilité. La section précédente 4.1.1 a discuté en outre du phénomène de non-déterminisme causé par la nécessiter de synchroniser. Ce paragraphe va donc étudier l'influence de cette synchronisation sur le non-déterminisme et la fiabilité, mais aussi sur les paramètres de consommation et de vitesse.

Non déterminisme

Le potentiel de non-déterminisme d'une architecture GALS est proportionnel au nombre de synchroniseurs implémentés, ces derniers générant des séquences de sorties multiples (Cf. §4.1.1.5). Cependant il faut tenir compte du fait qu'à nombre de synchroniseurs donné, le nombre de composants va intervenir aussi. En effet, plus le nombre de composants est élevé, plus le nombre de domaine d'horloges l'est aussi, même s'il est possible que certains composants soient synchrones d'une même horloge ou d'un même ensemble connecté d'horloges. Ainsi le risque de non-déterminisme est plus élevé pour 20 composants ayant chacun 3 canaux de communication que pour 6 composants ayant chacun 10 canaux de communication. En outre ce potentiel de non-déterminisme va être exacerbé avec l'augmentation des fréquences de fonctionnement des composants, mais aussi l'augmentation du nombre de fréquences de fonctionnement (Cf. §4.1.1.1) au sein du système.

On peut donc dire que le potentiel de non-déterminisme d'une architecture GALS est fonction :

- du nombre de synchroniseurs aux interfaces de communication entre les périphériques et le réseau d'interconnexion ;
- du nombre de composants synchrones ;
- du nombre de fréquences de fonctionnement différentes au sein du système, et de leur vitesse.

Une architecture globalement asynchrone dont le réseau localement synchrone nécessite d'implémenter le double de synchroniseurs aux interfaces de communication que son homologue asynchrone, présente donc un potentiel d'indéterminisme bien plus élevé. De par le nombre de synchroniseurs bien sûr, mais aussi parce que le réseau synchrone rajoute un composant synchrone et potentiellement une fréquence de fonctionnement supplémentaires au sein du système. Ce potentiel de non-déterminisme rend les systèmes sur silicium extrêmement

difficiles à tester et à vérifier [HEATH 03] du fait de la génération de séquences de sorties multiples (Cf. §4.1.1) et l'utilisation de synchroniseurs dégrade la fiabilité des systèmes en augmentant le risque de métastabilité (réduction du temps moyen entre deux erreurs ou MTBF) (Cf. §4.1.2 et §4.1.3).

Fiabilité

La fiabilité d'un système dépend en partie du temps moyen entre deux erreurs ou MTBF que l'on cherche à rendre le plus grand possible, en général d'un ordre de grandeur plusieurs dizaines de fois supérieur à la durée de vie du circuit [GIN 02] (Cf. §4.1.3.3 et §2.3.1). Une conséquence quantitative du potentiel de non-déterminisme élevé d'un système est de réduire considérablement ce facteur MTBF et donc la fiabilité. La Figure 4-20 illustre la probabilité d'erreur d'un système composé de dix composants (J=10) localement synchrones communiquant avec leur environnement au moyen de dix synchroniseurs (I=10) chacun, considérant que tous les synchroniseurs ont le même MTBF d'une valeur de $P(synchronizer\ fails) = 10^{-10}$.

$$
\begin{aligned}
P(component\ fails) &= 1 - P(component\ ok) \\
&= 1 - \prod_{i=1}^{I} P_i(synchronizer\ ok) \\
&= 1 - \prod_{i=1}^{I} (1 - P_i(synchronizer\ fails)) \\
P(system\ fails) &= 1 - P(system\ ok) \\
&= 1 - \prod_{j=1}^{J} P_j(component\ ok) \\
&= 1 - \prod_{j=1}^{J} (1 - P_j(component\ fails)) \\
&= 1 - \prod_{j=1}^{J} \prod_{i=1}^{I} (1 - P_{ij}(synchronizer\ fails)) \\
&= 1 - (1 - 10^{-10})^{100} = 10^{-8}
\end{aligned}
$$

Figure 4-20 : calcul rapide du risque d'erreur pour un système GALS.

L'impact du potentiel de non-déterminisme sur le taux d'erreur du système pénalise donc les réseaux de communication synchrones, en particulier parce que ce réseau, s'il est correctement dimensionné, supporte une charge de travail importante et permanente au sein du système. Son activité élevée visant principalement à router mais aussi à synchroniser des communications parallèles de fréquences asynchrones, son impact sur la fiabilité est déterminant. Or un réseau asynchrone se dispense naturellement de synchroniseurs de communication : il est localement parfaitement fiable et améliore donc considérablement la fiabilité globale du système.

Le risque d'erreur augmente essentiellement avec le nombre de synchroniseurs et la fréquence de fonctionnement du composant. Une solution pour fiabiliser le réseau synchrone

consiste à améliorer les architectures de ces synchroniseurs pour renforcer la fiabilité de ces blocs, mais au prix d'une latence très pénalisante pour les performances du système. Sachant en outre que Pechoucek montre dans [PEC 76] que la probabilité d'erreur de n'importe quel synchroniseur est non nulle.

Vitesse

Nous avons vu au §4.1.3.3 que la caractérisation d'un synchroniseur se fait sur deux paramètres essentiels. Le premier est son paramètre de fiabilité, le MTBF. Le second est un paramètre de performance qui est son temps de réponse ou latence. Le synchroniseur rajoute donc une latence importante dans les communications, proportionnelle à sa fiabilité. Un réseau de communication sans horloge permet alors :

- de supprimer au sein du réseau les synchroniseurs prenant part à une communication et donc la latence associée. Sur l'exemple du synchroniseur présenté au §4.1.3.2 avec une latence de deux cycles d'horloge, le coût de traversée du réseau synchrone serait de quatre cycles d'horloge supplémentaires pour compléter le protocole de communication.
- de réduire par deux le nombre total de synchroniseurs prenant part à une communication. La latence de synchronisation globale ainsi gagnée sur une communication n'est pas forcément de moitié car elle dépend localement des vitesses de fonctionnement de chaque domaine d'horloge. Par contre en nombre de cycles, le gain en latence est de quatre cycles du domaine d'horloge du réseau synchrone sur l'exemple précédent.

L'adaptation des domaines d'horloge asynchrones nécessite donc une synchronisation pour garantir le bon fonctionnement du système. Mais si les vitesses de fonctionnement relatives de deux composants synchrones communicants sont très différentes, la communication sera souvent bloquée par l'un des deux qui sera trop long à répondre aux sollicitations. L'élément synchroniseur est donc la plupart du temps associé à un bloc de cellules FIFOs (*First-In First-Out*) destiné à améliorer le débit de la communication en tamponnant les vitesses relatives des deux composants communicants. L'utilisation d'Interfaces de Synchronisation à base de synchroniseurs et de FIFOs est présentée dans notre méthodologie de conception de réseaux de communication sans horloge de la section 4.3 suivante. L'optimisation de tels circuits fait l'objet du chapitre 6.

Consommation et Surface

L'absence de synchroniseurs dans un réseau de communication sans horloge évite bien sûr d'avoir à échantillonner systématiquement les canaux de requête et d'acquittement à la fréquence de l'horloge, mais tout simplement on gagne un grand nombre de portes. Une différence existe aussi sur l'architecture des interfaces de synchronisation (synchroniseur plus FIFO) des périphériques synchrones selon que le réseau de communication soit synchrone ou asynchrone. Ainsi, Chelcea et Nowick montrent dans [CHEL 04] qu'une interface de synchronisation mixte entre un périphérique synchrone et un réseau de communication sans horloge (on parle de "FIFO synchrone-asynchrone") est implémentée avec moins de logique

qu'une interface de synchronisation entre deux composants localement synchrones d'horloges différentes ("FIFO synchrone-synchrone").

4.2.1.4. Conclusion

Cette étude présente les avantages qu'apporte un réseau de communication sans horloge pour résoudre les problèmes de synchronisation d'une architecture GALS :

- le potentiel de non-déterminisme est fortement réduit et la fiabilité accrue car on s'affranchit du coût de synchronisation d'un domaine d'horloge complet, celui du réseau, pour le remplacer par un réseau de communication sans horloge entièrement fiable ;
- la performance est améliorée car on s'affranchit de la latence de la moitié des synchroniseurs, ce qui n'empêche pas de conserver uniquement un bloc de FIFO pour gagner en débit et débloquer les communications (l'Interface de Synchronisation devient alors une Interface de Performance, voir §4.3.1.2) ;
- on économise la surface et l'activité de la moitié des synchroniseurs.

Le paragraphe suivant traite le problème particulier de l'arbitrage asynchrone auquel aucun réseau distribué, qu'il soit synchrone ou asynchrone, n'échappe dans une architecture globalement asynchrone et localement synchrone.

4.2.2. Coût de synchronisation d'un arbitre

Le paragraphe 1.4.2 présentait le non-déterminisme comme une limitation forte des circuits à arbitres asynchrones du fait du non-déterminisme que cela induit. Nous venons de voir que ce non-déterminisme n'est plus une contrainte des seuls arbitres asynchrones mais de tous les circuits GALS, avec un potentiel plus élevé pour les architectures à réseaux synchrones que les architectures avec un réseau de communication sans horloge. Nous allons à présent comparer le coût de synchronisation des arbitres synchrones et asynchrones au sein d'un réseau de communication, dans un environnement concurrent globalement asynchrone et localement synchrone.

Les arbitres asynchrones doivent gérer le non-déterminisme dans toutes les architectures de communication asynchrones, quelle que soit leur topologie, car ils reçoivent des requêtes insensibles au délai et donc de date d'occurrence inconnue.

Les arbitres synchrones en revanche, dans des architectures centralisées comme un bus partagé ou un réseau en étoile, ne reçoivent que des requêtes synchrones et ne génèrent donc ni indéterminisme ni métastabilité, la fonction de synchronisation ayant été réalisée aux interfaces avec les périphériques de vitesses de fonctionnement différentes. En revanche dans des architectures distribuées comme les réseaux à maille, les réseaux crossbar ou l'Octagon, tous les arbitres synchrones localisés dans des nœuds en contact avec un périphérique d'horloge différente de celle du réseau d'interconnexion, vont être obligatoirement soumis à des requêtes asynchrones et donc devoir faire face au risque de métastabilité.

4.2.2.1. Arbitre synchrone

Pour être performant, l'arbitre synchrone est supposé ne pas consommer inutilement, c'est-à-dire ne pas tester ses canaux de requête à chaque cycle d'horloge mais attendre une présence d'activité pour fonctionner, ce qui suppose déjà un certain coût de logique de contrôle (Cf. 1.3.2). La Figure 4-21 illustre le principe d'un arbitre synchrone dans un environnement concurrent. Cet arbitre reçoit deux signaux de requête asynchrones (Cf. 4.1.1.3) de la part des demandeurs d'accès à une unique ressource partagée qu'il protège et un signal d'acquittement, également asynchrone, de cette même ressource. L'arbitre étant piloté par une horloge, tous ces signaux asynchrones nécessitent l'utilisation de synchroniseurs pour échantillonner des signaux asynchrones vis-à-vis de l'horloge. Les synchroniseurs de détection des requêtes sont appelés détecteurs d'activité. Cet arbitre de deux requêtes vers une ressource est appelé arbitre 2 vers 1 ou arbitre 21.

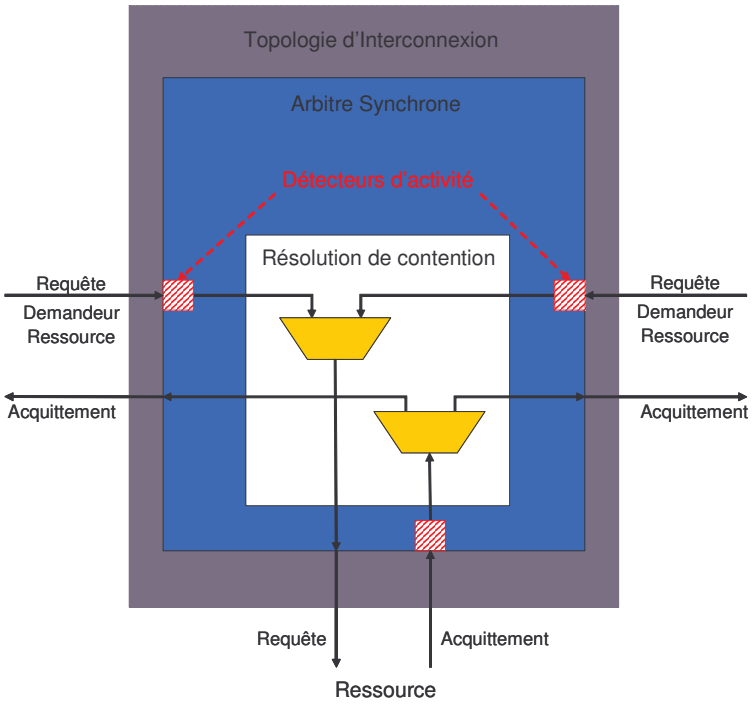


Figure 4-21 : échantillonnage de signaux asynchrones de l'horloge par un arbitre synchrone

La généralisation du nombre de synchroniseurs pour un arbitre synchrone gérant l'accès de N demandeurs vers P ressources (arbitre NP) est donc de N+P synchroniseurs (N requêtes et P acquittements asynchrones), avec N>1 et P>0 entiers.

4.2.2.2. Arbitre asynchrone

La Figure 4-22 illustre le principe d'un arbitre 21 asynchrone dans un environnement concurrent où tous les signaux sont asynchrones (Cf. 4.1.1.3). Cet arbitre reçoit deux signaux de requête asynchrones de la part des demandeurs d'accès à la ressource partagée qu'il protège et un signal d'acquittement, également asynchrone, de cette même ressource. L'arbitre asynchrone nécessite, même dans une architecture GALA, des blocs de synchronisation. En effet les canaux de requête d'accès à la ressource protégée par l'arbitre sont mutuellement non exclusifs et

peuvent potentiellement être vrais en même temps. Lorsque l'arbitre démarre son activité suite à son activation automatique par l'arrivée d'un signal de requête, il doit échantillonner l'ensemble des signaux de requête pour vérifier leur validité. La généralisation du nombre de synchroniseurs pour un arbitre NP asynchrone gérant l'accès de N demandeurs vers P ressources est donc de N synchroniseurs pour détecter l'activité des requêtes.

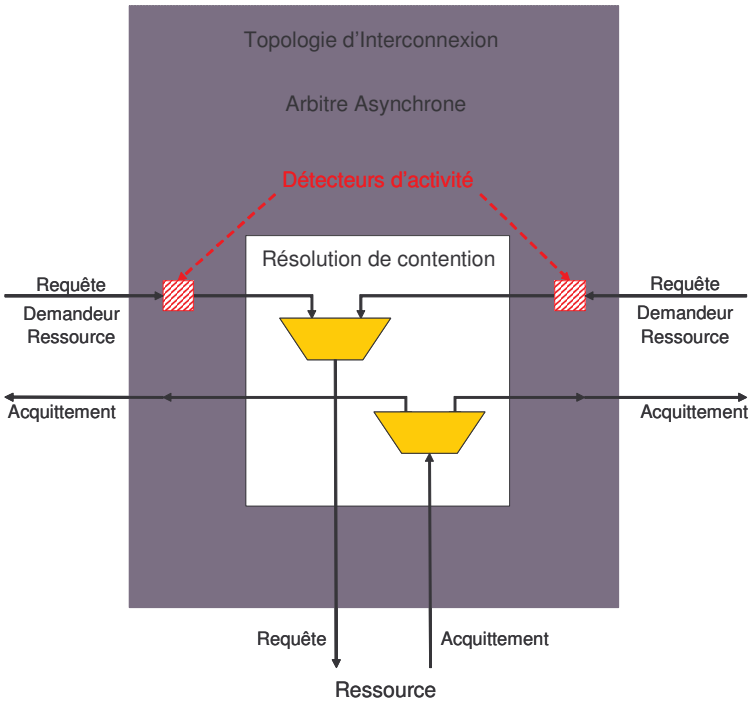


Figure 4-22 : échantillonnage de signaux asynchrones concurrents par un arbitre asynchrone

4.2.2.3. Comparaison des coûts de synchronisation

Non déterminisme

Le potentiel de non-déterminisme (séquences de sorties multiples) est proportionnel au nombre de synchroniseurs implémentés, ces derniers générant des séquences de sorties multiples (Cf. §4.1.1.5). Dans le cas de la comparaison des arbitres NP synchrones et asynchrones, la différence du nombre de synchroniseurs correspond au nombre de ressources partagées protégées par l'arbitre. Le cas le plus favorable à l'arbitre synchrone est donc un arbitre N1, lorsque la ressource partagée est unique (P=1). Dans ce cas l'arbitre synchrone possède un seul synchroniseur supplémentaire par rapport à l'arbitre asynchrone.

Fiabilité

La fiabilité d'un arbitre à échantillonnage asynchrone va dépendre de la fiabilité de ses synchroniseurs. Dans le cas des arbitres synchrones, cette fiabilité est mesurée par le temps moyen entre deux erreurs (MTBF), qu'il va falloir dimensionner en compromis avec la performance de latence exigible du bloc (Cf.4.2.1.3). Un arbitre asynchrone présente en revanche l'avantage vis-à-vis d'un arbitre synchrone de pouvoir traiter les problèmes de

métastabilité de façon parfaitement fiable [REN 00]. En effet, le phénomène de métastabilité est un état dont la durée n'est pas bornée et donc de fait non déterministe (Cf. 4.1.2). Or, dans les systèmes insensibles aux délais, la correction fonctionnelle de circuits contenant un élément de mutuelle exclusion ou un synchroniseur, est indépendante des temps de traitement (Cf. 3.1.5), et donc de la durée de l'état métastable. Un arbitre asynchrone est donc totalement fiable.

C'est pourquoi une réalisation relativement simple des structures de synchroniseurs peut être utilisée au sein des arbitres asynchrones comme le montre le chapitre 5 (Cf. §5.1), qui propose des instances d'arbitres asynchrones. Cependant, dans le monde synchrone, l'implémentation de ces blocs doit prendre en compte la contrainte supplémentaire que potentiellement l'état métastable durera plus longtemps que la période d'horloge, rendant le synchroniseur non entièrement fiable. On retrouve ici aussi le problème de réaliser des synchroniseurs offrant le meilleur compromis entre performance et fiabilité (Cf. 4.2.1.3). Les arbitres synchrones ne sont pas étudiés plus en détails dans ce manuscrit, mais on se référera au chapitre 6 pour l'étude de blocs de synchronisation synchrone/asynchrone.

Vitesse, consommation et surface

En vitesse, l'arbitre asynchrone gagne le temps de latence des synchroniseurs qui échantillonnent les signaux d'acquittement, soit deux cycles de l'horloge de l'arbitre synchrone.

Pour obtenir un composant consommant le moins possible d'énergie, il faut d'une part l'activité minimale (consommation dynamique) et d'autre part le moins de composants possible (consommation statique). La consommation dynamique est à l'avantage de l'arbitre asynchrone grâce d'une part à l'absence d'échantillonnage systématique à une fréquence d'horloge et d'autre part à un mode veille automatique. La consommation statique correspond au courant de fuite des composants. Quelle que soit l'activité du circuit, le nombre de composants doit donc être minimal car le courant de fuite est proportionnel au nombre de composants. Un circuit asynchrone occupe une surface plus importante que son équivalent fonctionnel synchrone, même si ici l'arbitre asynchrone est moins pénalisant car il se dispense de P synchroniseurs. A condition que les exigences de fiabilité soient faciles à atteindre (par exemple des fréquences de fonctionnement basses ou un faible nombre de requêtes et donc de fréquences différentes), l'arbitre synchrone s'avère donc avantageux en consommation statique et également en surface. Surtout pour les arbitres N1 qui sont les architectures les plus favorables aux arbitres synchrones, où le compromis entre le gain d'un seul synchroniseur et le coût en surface d'un arbitre asynchrone peut se discuter.

4.2.2.4. Conclusion

Le choix de l'arbitre asynchrone sur son homologue synchrone présente ces avantages :

- le potentiel de non-déterminisme est réduit de P synchroniseurs mais existe toujours car la présence de synchroniseurs reste nécessaire ;
- la fiabilité est totale ;
- en latence, on gagne sur les synchroniseurs d'acquittement soit deux cycles de l'horloge de l'arbitre synchrone ;

- une consommation dynamique plus réduite ;
- une consommation statique pénalisée par le nombre plus important de composants, en particulier pour les arbitres N1. Avec la réduction d'échelle des technologies, cette part de consommation statique devient de plus en plus importante.

4.2.3. Conclusion

A partir de l'étude du non-déterminisme et du synchroniseur de la section précédente, nous venons de mener une étude qualitative comparative pour évaluer les coûts de synchronisation à travers le réseau d'interconnexion, par communication au sein d'une architecture globalement asynchrone et localement synchrone. Tout d'abord aux interfaces entre les composants du système, pour un réseau de communication synchrone et pour un réseau de communication sans horloge, puis entre des arbitres synchrones et asynchrones en environnement asynchrone. Nous montrons que le réseau et l'arbitre asynchrone résolvent les problèmes de synchronisation de manière plus fiable, plus performante et moins coûteuse.

Nous allons voir à partir de la section suivante puis dans les prochains chapitres comment ces problèmes de non-déterminisme et de synchronisation sont abordés pour construire des réseaux de communication sans horloge.

4.3. Méthodologie de conception d'un réseau sur silicium sans horloge

4.3.1. Modèle abstrait du réseau

4.3.1.1. Présentation des modules du réseau

La méthodologie de conception de réseaux d'interconnexion asynchrones que nous présentons dans ce chapitre utilise les propriétés d'excellente modularité des circuits sans horloge. Nous proposons un découpage modulaire des composants du réseau en ressources critiques élémentaires : arbitrage, transport, routage et synchronisation. A cela il faut rajouter une ressource supplémentaire de mise en forme des communications, qui se place d'un point de vue méthodologique au sein des périphériques synchrones. Nous allons donc introduire (Cf. Figure 4-23 ci-dessous) :

1. des ressources d'Arbitrage (*PRSA* pour *Parallel-Request-Sampling Arbiter)* et d'Interface de Synchronisation (*SPI* pour "*Synchronisation & Performance Interface*") issues de l'étude des problèmes de synchronisation dans les architectures globalement asynchrones et localement synchrones ;
2. des ressources de Routage de l'Information (*IRR* pour "*Information Routing Ressource*") issues de l'étude des topologies et des besoins en services d'un réseau de communication ;

3. des ressources de Transport de l'Information (*ITR* pour "*Information Transport Ressource*") issues de l'étude des topologies et des performances d'un réseau de communication ;
4. des ressources d'Interface ou Adaptateur de Communication (*CA* pour "*Communication Adaptor*") issus de l'étude des protocoles de service et de niveau signal du réseau de communication, qui ne sont pas des ressources du réseau mais une interface intégrée au périphérique. Cette interface est chargée d'adapter le format des informations échangées entre le périphérique et le réseau.

L'objectif de ce découpage simple, mais modulaire et polyvalent, est de permettre à terme la synthèse entièrement automatique d'arbitres et de réseaux de communication sans horloge dans l'outil de conception TAST.

4.3.1.2. Modèle abstrait du réseau

La Figure 4-23 présente l'architecture GALS d'un système sur silicium, implémentant les ressources précitées au sein d'un réseau de communication sans horloge. Cette figure illustre l'utilisation de ces différentes ressources à travers un modèle abstrait de plus haut niveau. Seules les Interfaces de Communication et de Synchronisation sont connectées explicitement à ce niveau de représentation.

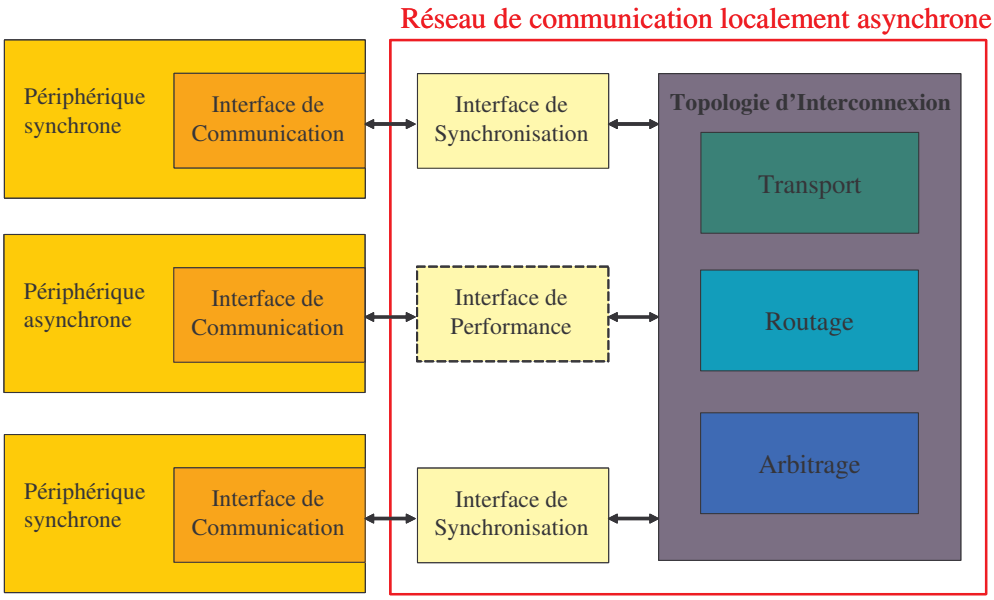


Figure 4-23 : modèle abstrait d'une architecture GALS

L'Interface de Communication

L'Interface de Communication permet au périphérique d'adapter son protocole de communication au niveau transactionnel de manière à être compatible avec le format des données transmises vers et depuis le réseau mais aussi pour pouvoir exploiter les services offerts par ce réseau (Cf. §2.3.3 et §4.3.3.2). Du point de vue des périphériques, le protocole de communication de niveau service est transparent. Toutes les opérations de transfert de données

sont vues par un périphérique soit comme des écritures (le maître envoie des données à l'esclave) soit comme des lectures (le maître demande des données à l'esclave qui les lui renvoie).

L'Interface de Synchronisation

L'Interface de Synchronisation est nécessaire uniquement pour les périphériques synchrones (Cf. 4.2.1). Sa structure permet :

- de prévenir les phénomènes de métastabilité lorsque le périphérique localement synchrone échantillonne au moyen de son signal d'horloge un signal qui lui est asynchrone (Cf. 4.1.2) ;
- d'adapter les vitesses de transmission/réception entre ces périphériques et le réseau. Cette fonction de tampon par pipeline, réalisée au moyen de FIFOs asynchrones élastiques (Cf. Chapitre 6), peut servir d'accélérateur de performance en améliorant le débit également pour des périphériques asynchrones , d'où le terme Interface de Performance.

La topologie d'Interconnexion

La spécification plus détaillée des modules de Transport, de Routage et d'Arbitrage est liée à la détermination de la topologie globale du réseau. Les topologies sont présentées en détail en section 2.4. Le choix d'une topologie va dépendre de l'application ciblée et des performances exigées par le système. Par exemple, le choix d'une topologie en crossbar va signifier un arbitrage distribué de faible latence tandis qu'une topologie en mailles va signifier un degré de services très élevé. Ces deux connexions se révèlent très flexibles et hautement parallèles au contraire de l'utilisation d'une connexion point à point qui va en revanche offrir une simplicité du contrôle pour un débit et une latence optimales, ou un bus centralisé offrant l'encombrement le plus réduit et une faible consommation (sous-entendu à condition que le réseau présente une topologie adaptée aux besoins de l'application).

Nous avons vu au §2.4.3 que le bus centralisé est plutôt favorable à une conception en technologie synchrone, que le système soit globalement synchrone ou GALS, tandis que les réseaux de communication présentent un certain nombre de caractéristiques les rendant favorables à une conception en technologie asynchrone au sein d'une architecture globalement asynchrone et localement synchrone :

- Ils sont favorables aux transmissions sur de grandes distances, à condition de s'affranchir des problèmes de propagation du signal, comme le permet la technologie asynchrone. Ce point est abordé en détail au chapitre 7.
- Ils conviennent parfaitement à des systèmes sur silicium complexes intégrant plusieurs maîtres et donc multi-horloges. Un réseau sur silicium asynchrone va permettre de libérer des problèmes de synchronisation globale d'horloge et va offrir une conception modulaire du système.

Le terme "Topologie d'Interconnexion" se référera donc par la suite toujours à des réseaux de communication distribués, qu'il s'agisse d'un crossbar, d'une étoile, d'un anneau, d'un

réseau à mailles ou de toute autre architecture distribuée hybride. Les modules de ces réseaux de communication, seront donc également distribués. Les modules d'Arbitrage et de Routage seront distribués à chaque nœud du réseau tandis que l'adaptation des protocoles de niveau signal du module de Transport ITR, en vue d'améliorer les performances de communication, se fera point à point sur chaque ensemble de canaux entre deux nœuds du réseau.

4.3.1.3. Conclusion

Nous venons d'introduire le principe de découpage modulaire de nos réseaux de communication sans horloge et d'expliciter la terminologie utilisée. Avant de raffiner la modélisation de ces réseaux vers des modèles d'implémentation de plus en plus détaillés, nous insistons dans le paragraphe suivant sur la manière dont nous avons identifié ces modules au cours des différentes études menées dans les chapitres précédents.

4.3.2. Synthèse des chapitres et paragraphes précédents : à l'origine des modules du réseau d'interconnexion

Nous développons ici comment les études réalisées au cours des chapitres précédents sont à l'origine des modules introduits au paragraphe précédent et assemblés en réseaux d'interconnexion asynchrones. La synthèse de ces études met en relief l'importance des problèmes d'arbitrage et de synchronisation entre les composants du système.

4.3.2.1. Synchronisation et arbitrage : le pivot de la construction d'un réseau sur silicium

L'objectif principal de ces travaux de thèse était d'acquérir les connaissances et les compétences sur la conception de réseaux de communication intégrés, et d'étudier les avantages potentiels de la conception de tels réseaux en technologie asynchrone. En passant en revue l'état de l'art du domaine, nous avons découvert qu'un point critique majeur de la conception de ces réseaux était la synchronisation. Ce qui rejoignait le problème connexe que nous venions d'aborder [RIG 02b] sur la modélisation du non-déterminisme en langage CHP en vue de la synthèse de circuits asynchrones, en particulier d'arbitres.

Nous avons donc au cours de nos études mis l'accent sur l'analyse et la résolution des problèmes de synchronisation rencontrés aux différents niveaux de conception d'un réseau sur silicium :

Stratégie de synchronisation globalement asynchrone avec un réseau de communication localement sans horloge

Choix de conception

Le chapitre 1 et la section 4.2 ont présenté les étapes de nos motivations à choisir une synchronisation d'un système sur silicium :

- premièrement par une architecture du système globalement asynchrone et localement synchrone plutôt que par une synchronisation par horloge globale (Cf. 1.2.2.3) ;
- deuxièmement en utilisant, au sein de cette architecture globalement asynchrone et localement synchrone, un réseau de communication asynchrone plutôt qu'un réseau synchrone (Cf. 1.3 et 4.2).

Ces choix de conception nous permettent de nous affranchir de la gestion critique de l'arbre d'horloge, le problème étant reporté et localisé dans les périphériques du système (Cf. 1.3.1). Ils nous offrent également un degré de modularité très élevé qui va faciliter la conception, l'adaptation et la réutilisation des composants de nos réseaux (Cf. 1.3.4). La construction de nos réseaux de communication sans horloge s'appuie fortement sur cet atout de modularité. En outre, la discrétion électromagnétique des circuits asynchrones (Cf. 1.3.3) convient avantageusement aux circuits mixtes ainsi qu'aux circuits sécurisés étudiés au sein du groupe de recherche C.I.S. Il s'agit aussi d'exploiter au mieux les atouts potentiels de la conception asynchrone pour la faible consommation d'énergie (Cf. 1.3.2). Enfin, les réseaux de communication sans horloge offrent une solution de réseaux d'interconnexion plus fiables et aux coûts de synchronisations moindres que leurs homologues synchrones (Cf. 4.2.1)

Modélisation en couches fonctionnelles

La Figure 4-24 illustre par un modèle dit en couches fonctionnelles le choix d'une architecture GALS avec un réseau de communication sans horloge.

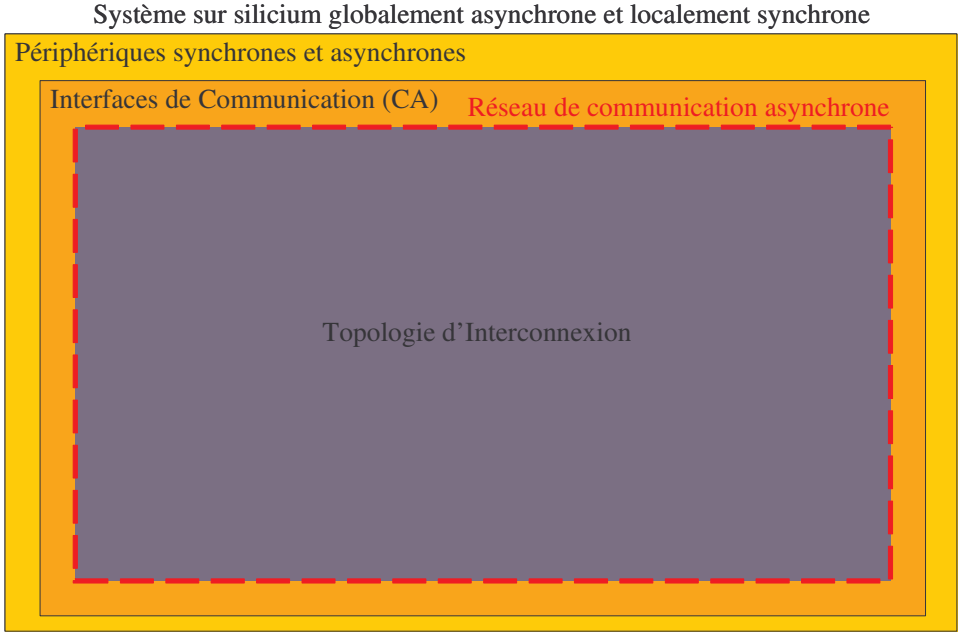


Figure 4-24 : modèle en couches fonctionnelles d'une architecture GALS avec un réseau de communication sans horloge

Pour des raisons de clarté lorsque le nombre de fonctions à représenter sera élevé, les différents composants correspondant à une fonction, par exemple ici les Périphériques, ne sont pas dissociés en blocs individuels pour chaque Périphérique mais unifiés en une couche fonctionnelle. La Figure 4-24 présente donc seulement trois couches : la couche externe des

Périphériques qui sont reliés entre eux au moyen de la couche interne du réseau de communication sans horloge. Cette couche interne est représentée par la topologie du réseau de communication, bien que celle-ci n'est pas encore été déterminée. A l'interface entre les périphériques synchrones et le réseau, le protocole de communication de chaque périphérique va contribuer à spécifier l'Interface ou Adaptateur de Communication (*CA* pour "*Communication Adaptor*") du périphérique qui permet d'adapter le protocole de communication et de gérer le format des transactions avec le réseau. Ce module *CA* fait partie intégrante des modules de construction de la communication mais se trouve externe au réseau de communication sans horloge proprement dit. Les modules *CA* sont synchrones avec l'horloge du périphérique auquel ils sont rattachés.

Environnement concurrent non déterministe

Le choix d'une architecture de système globalement asynchrone place les composants dudit système, périphériques et réseau d'interconnexion, dans un environnement concurrent. Cet environnement concurrent impose aux composants du système de se synchroniser pour pouvoir coopérer [TANEN 00]. Cette synchronisation recouvre deux problèmes connexes de non-déterminisme présentés au Chapitre 4 :

Les Interfaces de synchronisation pour les architectures GALS[9]

Le premier type de synchronisation doit garantir la non occurrence d'états métastables lorsqu'un composant localement synchrone échantillonne au moyen de son signal d'horloge un signal qui lui est asynchrone (Cf. 4.1.2). Il est donc nécessaire de synchroniser ces composants, asynchrones entre eux, au moyen d'interfaces à base de synchroniseurs, sur les principes présentés aux paragraphes 4.1.3 et 4.1.4.3. La section 4.2 présente et compare le coût de ces interfaces pour les réseaux de communication avec et sans horloge locale.

Ces interfaces de synchronisation vont constituer le premier des quatre modules ou ressources critiques de nos réseaux. La Figure 4-25 présente l'intégration de ces Interfaces de Synchronisation (*SPI* pour "*Synchronisation & Performance Interface*") dans le modèle en couches fonctionnelles. Les modules *SPI* interfacent les domaines synchrones des périphériques avec le réseau de communication sans horloge. Ils sont donc à la frontière des mondes synchrone et asynchrone, d'où l'inclusion dans la Figure 4-25 du réseau de communication sans horloge (*ANoC*) dans la couche *SPI*. Ces modules SPI sont construits à partir de synchroniseurs associés à des circuits tampons (*FIFO* pour *First-In First-Out*) mixtes synchrones/asynchrones (Cf. 4.2.1.3). Les FIFOs ont pour rôle de tamponner ou autrement dit d'adapter les vitesses d'exécution et donc de communication de domaines de fonctionnement différents mutuellement asynchrones.

Les FIFOs sont étudiées en détail au chapitre 6. Dans une architecture GALS, ces interfaces *SPI* sont la clef de voûte de la communication entre le réseau d'interconnexion et les périphériques synchrones hétérogènes, que le réseau soit synchrone ou asynchrone. Elle devient

[9] Ce problème est spécifique aux architectures GALS mais ne se pose pas pour les systèmes GALA, tous les composants étant localement asynchrones.

en revanche optionnelle lorsque le périphérique et le réseau sont tous deux asynchrones, l'adaptation des vitesses relatives des domaines de fonctionnement se faisant naturellement à travers le protocole de communication de niveau signal. On utilise alors une FIFO asynchrone/asynchrone dans le but d'améliorer les performances de débit de la communication.

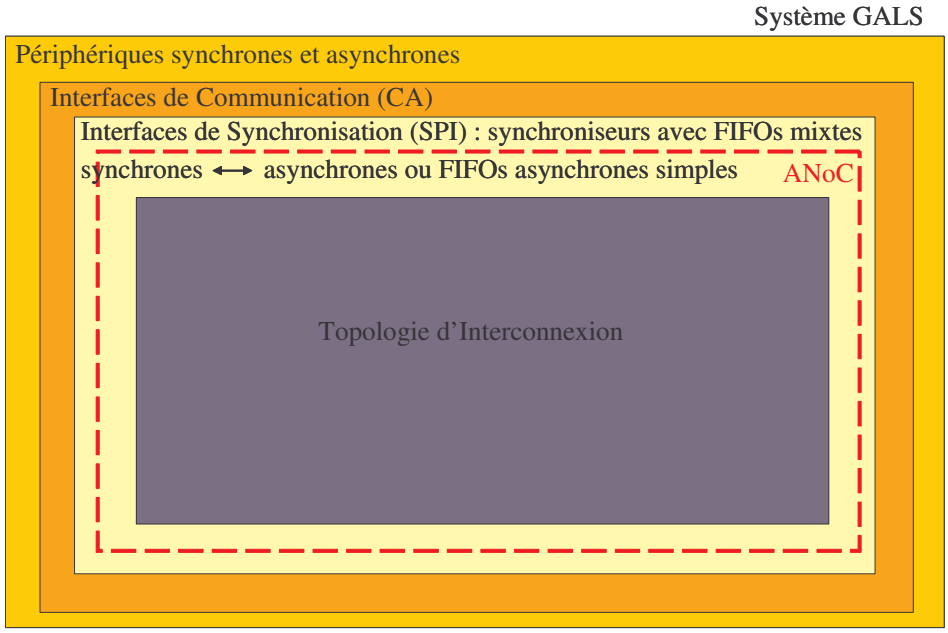


Figure 4-25 : modèle en couches fonctionnelles : les Interfaces de Synchronisation

La Ressource d'Arbitrage non déterministe

Le second type de synchronisation doit garantir que plusieurs composants n'accèdent pas simultanément à une ressource partagée (Cf. 4.1.4).

Au sein du réseau de communication se pose en effet le problème, connexe de la synchronisation, qu'est l'arbitrage de plusieurs modules[10] en contention pour l'accès à une ressource commune partagée (autrement dit ils initient tous une transaction vers un même autre module destinataire d'une transaction). Ces requêtes sont émises par des modules asynchrones concurrents : les arbitres implémentés sont donc non déterministes et doivent implémenter des synchroniseurs pour échantillonner les requêtes.

Ces arbitres non déterministes constituent le second des quatre modules ou ressources critiques de nos réseaux de communication sans horloge. Ils sont construits à partir d'un bloc d'échantillonnage en parallèle des requêtes (*PRSA* pour *Parallel-Request-Sampling Arbiter*) et d'un bloc capable d'implémenter l'ensemble des algorithmes de priorité présentés au paragraphe 2.3.3.1. Ils sont étudiés en détail au chapitre 5. La Figure 4-26 présente l'intégration de ces modules *PRSA* dans le modèle en couches fonctionnelles du réseau de communication. Les modules *PRSA* sont distribués à chaque nœud interne du réseau, d'où la représentation de la couche d'arbitrage intégrée au cœur de la Topologie d'Interconnexion.

---

10 On entend par module soit un nœud du réseau d'interconnexion soit le périphérique au moyen de son interface de communication et d'adaptation de protocole avec le réseau (Cf. §4.3)

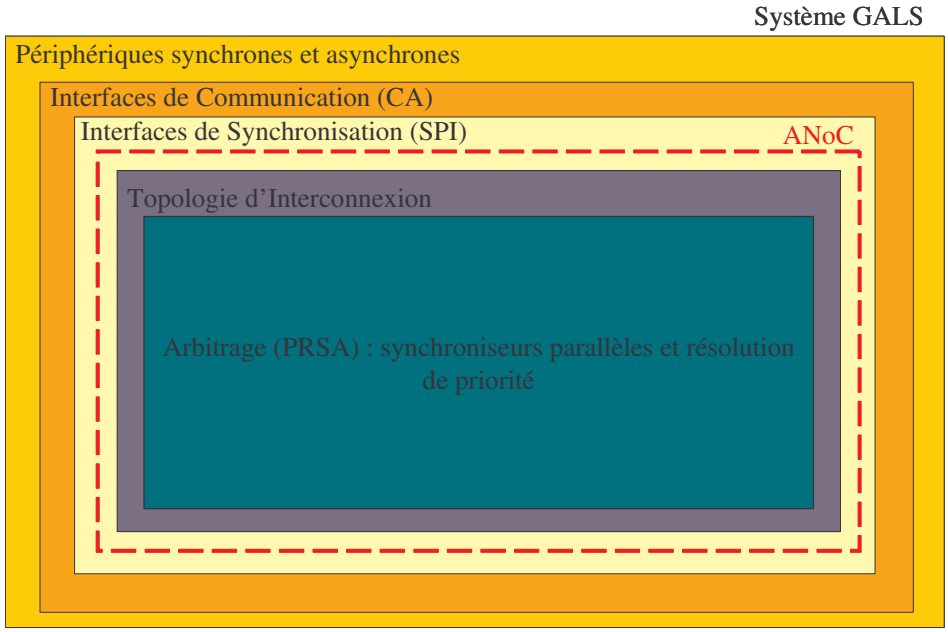


Figure 4-26 : modèle en couches fonctionnelles : l'Arbitrage de contention

4.3.2.2. Construction d'un réseau sur silicium performant

Le chapitre 2 a détaillé les caractéristiques essentielles de la construction d'un réseau performant. Il a insisté en particulier sur l'importance de la topologie du réseau de communication (Cf. §2.4.3). Ce paragraphe expliquait que les techniques de conception asynchrones étaient favorables aux architectures distribuées, telles que le crossbar ou le réseau à mailles. Le choix de la topologie du réseau d'interconnexion va donc contraindre évidemment la spécification des modules de Transport, d'Arbitrage et de Routage de l'Information. Le chapitre 2 a insisté également sur l'importance des protocoles de synchronisation au niveau de granularité fin dit niveau de signal (Cf. §2.3.2) et des protocoles de services (arbitrage et transaction) au niveau de granularité supérieur dit niveau fonctionnel (Cf. §2.3.3).

Le protocole de plus bas niveau matériel, ou protocole de niveau signal, va jouer sur les performances du réseau et la facilité d'interfaçage des composants périphériques. Au sein du réseau, la spécification de ces protocoles (protocole quatre ou deux phases et choix du format de codage des données) va fournir, pour chaque canal point à point entre deux modules de l'architecture, la Ressource de Transport de l'Information (*ITR* pour "*Information Transport Ressource*"). La Figure 4-27 présente cette ressource *ITR* dans le modèle en couches. Cette ressource ITR est répartie en deux couches car une adaptation des protocoles de niveau signal est possible entre chaque ensemble de canaux d'une liaison point à point, c'est-à-dire :

- entre les Interfaces de Synchronisation de la couche SPI et les nœuds de contrôle du réseau, représentés pour le moment dans ce modèle incomplet par la couche d'Arbitrage PRSA ;
- entre chacun des nœuds de cette couche PRSA à travers la Topologie d'Interconnexion.

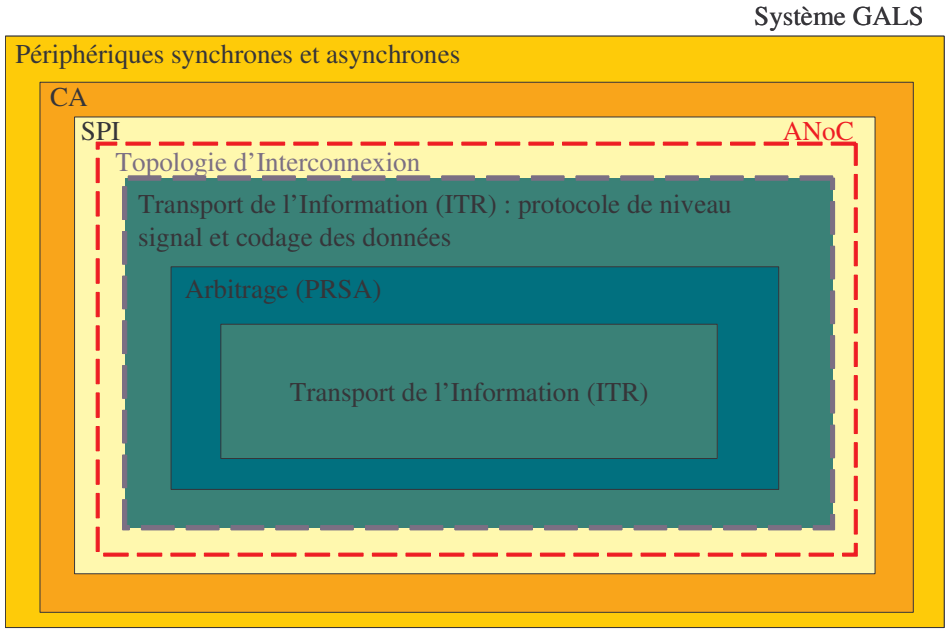


Figure 4-27 : modèle en couches fonctionnelles : le Transport de l'Information

Les protocoles de services (arbitrage et transaction) au niveau de granularité supérieur vont influer sur les performances du réseau et sur son extensibilité, à travers la variété et la pertinence des services offerts. Ces protocoles d'arbitrage et de transaction vont participer respectivement à la spécification des ressources d'Arbitrage et de Routage de l'Information (*IRR* pour "*Information Routing Ressource*"). La Figure 4-28 présente donc l'enrichissement des nœuds du réseau par les modules *PRSA* et *IRR* assurant respectivement les services d'arbitrage et de transaction. La spécification de l'ensemble de ces services va permettre le routage de l'information à travers la topologie d'interconnexion du réseau de communication.

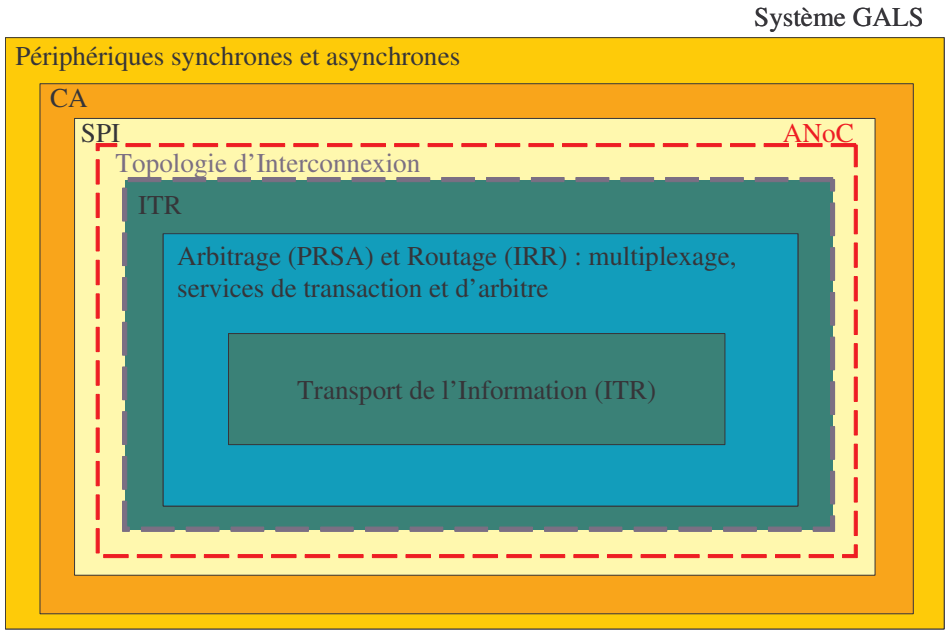


Figure 4-28 : modèle en couches fonctionnelles : les services de Routage et d'Arbitrage des nœuds du réseau

4.3.2.3. Conclusion

Nous venons de voir comment l'ensemble des études présentées au cours des chapitres 1 à 3 et dans les précédentes sections §4.1et §4.2 ont déterminées la méthode modulaire de réalisation de réseaux de communication sans horloge. La méthodologie de construction de ces modules a été illustrée par une représentation en couches fonctionnelles d'un système GALS complet. Nous allons à présent détailler l'architecture du réseau sur silicium.

4.3.3. Modèle architectural du réseau de communication sans horloge

Le modèle en couches fonctionnelles présenté ci-dessus reste très générique. Nous allons illustrer notre méthodologie à partir d'un exemple plus concret d'architecture GALS : le système WUCS (*Wireless Universal Control System*) dans lequel s'intègre un réseau de communication sans horloge. La première étape sera la détermination de la topologie globale du réseau, indispensable pour ensuite pouvoir spécifier entièrement les différents modules du réseau.

4.3.3.1. Présentation du système WUCS

Le système WUCS est une architecture globalement asynchrone et localement synchrone destinée dans sa première version à des applications de sécurité. Ce système permettra par exemple de valider des algorithmes de cryptographie ou d'implémenter des protocoles de communication sécurisés. L'architecture, illustrée par la Figure 4-29, est construite autour d'un réseau de communication sans horloge. Ce réseau interconnecte un microprocesseur MIPS synchrone et un microprocesseur MIPS asynchrone, une unité de cryptographie DES asynchrone [BOU 04a], deux bancs de mémoires RAM partagées, une liaison RS232 et une liaison directe de communication avec l'extérieur.

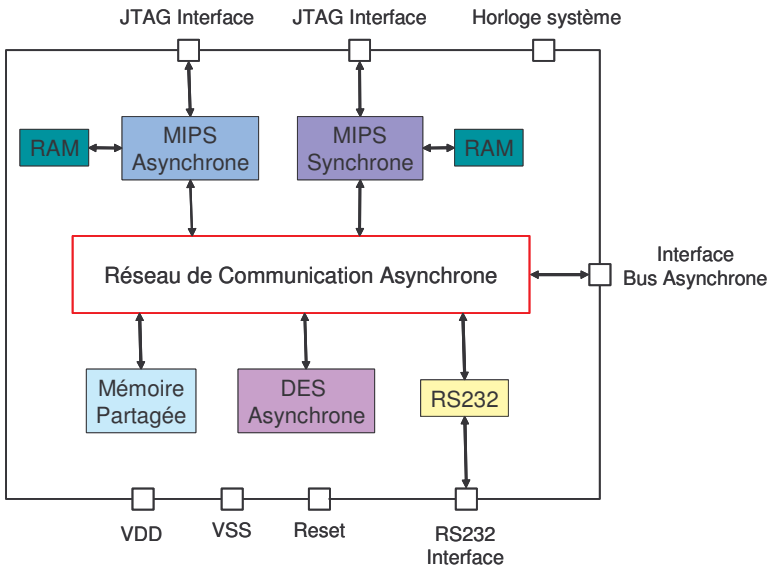


Figure 4-29 : architecture GALS du système sur silicium WUCS

Le système WUCS va permettre de valider les composants DES et MIPS asynchrones dans une architecture globalement asynchrone et localement synchrone. Il va permettre également de valider le concept et les choix de notre méthode de construction du réseau de communication sans horloge. Au cours de la conception du réseau nous évaluons la facilité d'interfaçage des composants, en particulier les composants synchrones avec les modules *SPI* et *CA*. Le circuit permettra ensuite de mesurer ses performances en terme de fiabilité et de vitesse de communication et ses coûts en terme de surface, de consommation et de bruit électromagnétique.

4.3.3.2. Spécification du réseau de communication sans horloge

Contraintes de conception

Nous avons vu au §2.4.3 que les techniques de conception asynchrones étaient favorables aux architectures distribuées plutôt que centralisées. Le choix d'un réseau de communication plutôt qu'un autre va ensuite dépendre des spécifications du système WUCS. Les contraintes de conception sont les suivantes :

- La flexibilité (Cf. 2.2) doit être la meilleure possible :
    - en terme d'extensibilité, le système est destiné à intégrer et valider les circuits asynchrones développés par le groupe CIS, en particulier un convertisseur analogique/numérique asynchrone [ALL 03] et des accélérateurs matériels de calcul arithmétique [FRAG 03];
    - en terme de compatibilité d'interfaçage, le système doit pouvoir intégrer les périphériques synchrones les plus divers. Le réseau de communication sans horloge, quelque soit sa topologie, présente déjà l'avantage de supporter de grandes différences de fréquences de fonctionnement entre les composants périphériques, grâce à une adaptation naturelle de vitesse de fonctionnement (Cf. 1.3.5) et une plus grande fiabilité (Cf. 4.2.1.3) ;
    - la variété des services devra répondre aux exigences du système en terme de priorités de communication, de débit garanti, de latence autorisée…
- Dans la première version du système WUCS, la latence est choisie la plus faible possible. En effet, même si les informations destinées à être traitées seront hétérogènes et donc les paquets de communication seront de tailles variables, en particulier pour les échanges avec les bancs de mémoire, l'application de cryptographie utilise des communications de petites tailles (paquets de 1 à 3 mots de 32 bits), qui ne devront donc pas être pénalisées par une latence importante relativement au temps de transfert des données.
- La fiabilité des protocoles de niveau signal asynchrones (Cf. §2.3.2.3) doit être démontrée par ce circuit.

- Le coût en consommation d'énergie par commutation de la logique du réseau doit être le plus faible possible, avec un nombre de portes le plus réduit possible pour diminuer l'impact des courants de fuite, importants dans les circuits asynchrones, plus gourmands en portes que les circuits synchrones de manière générale (Cf. §4.2.2.3).
- Le coût en consommation d'énergie de la communication proprement dite (commutations électriques sur les lignes physiques) doit également être minimisé.
- Le coût de la logique en surface ne sera pas très élevé, du fait des exigences de latence et de consommation.
- Le coût en surface de fils de communication doit être optimisé, mais le circuit relativement simple et destiné à être fabriqué en technologie 0,13 µm et 65 nm de STMicroelectronics va donc disposer de nombreuses couches de métal peu utilisées.

L'ensemble de ces critères de conception nous a amenés à faire les choix architecturaux suivants.

Le choix de la topologie

Il existe de nombreuses architectures de réseaux distribués intégrés très performantes, en fonction des critères de conception du système. Parmi les architectures étudiées et présentées dans ce manuscrit, nous relevons :

- Le crossbar (Cf. §2.4.2.3) présente une grande bande passante et surtout une latence faible, mais pour des systèmes avec de nombreux composants et pour des applications à composantes fortement orientées calcul. Pour un système simple comme WUCS, la matrice d'interconnexion se révèle relativement trop coûteuse.
- L'Octagon (Cf. §2.5.3.3) offre un débit total meilleur que pour le crossbar et son implémentation physique coûte moins chère. Son algorithme de routage et d'arbitrage équilibré convient à des architectures régulières de multiprocesseurs homogènes. Au contraire, le système WUCS présente une architecture et des composants hétérogènes.
- Le réseau à maille (Cf. §2.4.2.4) présente une bande passante proche de celle des bus crossbar pour une complexité moindre. Par contre, les latences peuvent être bien plus élevées. Si les unités sont nombreuses, la consommation de l’ensemble du réseau est importante.
- L'anneau découplé (Cf. §2.4.2.2) simple ou double, sur le principe de STRING (Cf. §2.6.1.3) présente des caractéristiques similaires à celles de l'Octagon, en avantages comme en défauts.
- Le QNoC (Cf. §2.3.3.2) est une topologie à maille irrégulière qui fournit une qualité de service très élevée pour un coût très performant mais encore trop élevé pour un système tel que WUCS.

Pour des systèmes de complexité supérieure à celle de WUCS et d'architectures plus homogènes, les réseaux Octagon et QNoC se révèlent des réalisations très performantes.

Cependant le système WUCS est encore dans la phase d'exploration d'architecture. Son objectif reste de valider, sur une application de cryptographie, le fonctionnement coopératif des modules synchrones et asynchrones qui le composent, avant d'étendre les versions futures à des applications plus complexes nécessitant des architectures de communication très performantes. Nous décidons donc de choisir comme topologie d'interconnexion une interconnexion totale qui permet la plus faible latence possible dans le réseau et la meilleure extensibilité, pour un contrôle totalement distribué. Le coût de la logique de contrôle du réseau est également minimisé, au détriment du coût en fils de communication, qui reste cependant raisonnable pour un tel système. Pour des versions futures plus complexes, il faudra cependant choisir des architectures distribuées inspirées de l'Octagon ou du QNoC qui présentent une évolution plus favorable du compromis coûts/performances en fonction de la complexité. Une discussion plus approfondie sur le choix de cette topologie et plus généralement de cette méthodologie de conception est abordée au §5.3.4.

Par commodité nous nommons cette topologie FAN pour *Fully-connected Asynchronous NoC*. La Figure 4-30 présente le schéma de principe d'une interconnexion totale.

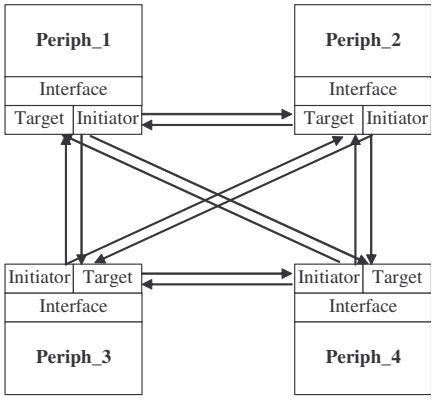


Figure 4-30 : schéma de principe d'une interconnexion totale

Le choix des protocoles de services

Le système WUCS supporte un nombre réduit de périphériques hétérogènes. La classe de services est donc spécifiée pour être la plus simple possible, de manière à fournir la meilleure latence mais en restant facilement extensible. FAN offre donc :

- Un service d'arbitrage : deux niveaux statiques de priorité. Les MIPS sont de priorité la plus élevée, les autres composants de priorité faible. L'arbitre est conçu pour pouvoir aisément réimplanter un nouvel algorithme d'arbitrage localement (Cf. chapitre 5 et [RIG 02b]). Ainsi, sur l'exemple du STbus, chaque nœud peut implémenter son propre algorithme d'arbitrage de manière statique ou dynamique ;
- Deux services de transaction :
    - des transactions en mode rafale non préemptif, de taille quelconque ;
    - l'architecture de contrôle totalement distribué de FAN supprime du point de vue du réseau la distinction entre un maître et un esclave. Outre le gain en modularité et en extensibilité que cela procure,

l'absence d'instance centrale permet de fournir un service particulier que nous appelons Réponse Indirecte : lors d'une transaction, un maître (la distinction existe toujours pour les périphériques) peut signifier à l'esclave une adresse d'éventuelle réponse différente de la sienne. Ce service est expliqué en détail au §5.3.2.2.

Les formats du mot et du paquet reprennent le principe de la Figure 2-7 et sont détaillés au §5.3.2.2.

Le choix du protocole de synchronisation

Lorsque le périphérique est asynchrone comme le MIPS asynchrone ou le DES, le protocole de niveau signal est parfaitement compatible entre le réseau FAN et les périphériques. Il s'agit d'un protocole quatre phases avec un codage des données insensible au délai trois états en multi-rail 4 voies (MR4) plus économe et plus rapide qu'un codage DR (Cf. §3.1 et §3.1.3.2) et bien sûr plus robuste qu'un codage "données groupées" (Cf. §2.3.2.3). En revanche, le protocole de niveau signal demande un certain nombre d'adaptations lorsque le périphérique est localement synchrone avant de pouvoir utiliser le codage MR4, qui sont présentées ci-dessous.

4.3.3.3. L'architecture FAN détaillée pour un périphérique synchrone

La Figure 4-31 présente l'architecture détaillée complète des étages de communication du réseau FAN pour un module périphérique localement synchrone.

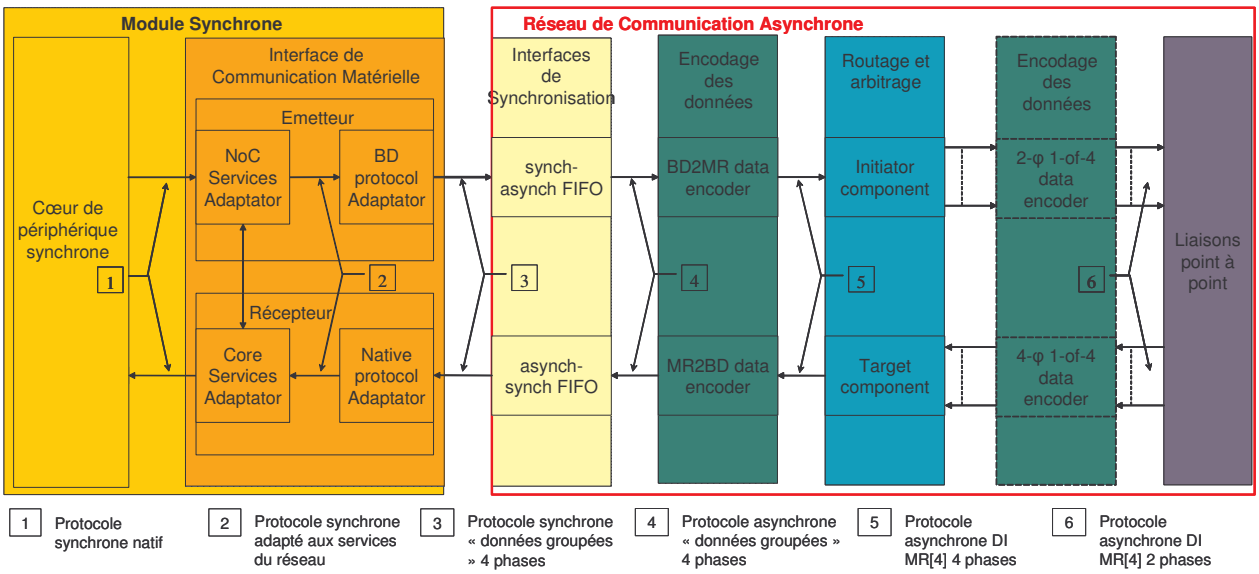


Figure 4-31 : architecture détaillée des étages de communication du réseau FAN pour un module périphérique synchrone

Chaque étage de communication réalise deux opérations duales selon le sens de la communication, du périphérique vers le réseau de communication ou réciproquement. Par commodité de lecture nous ne parlerons dans la suite que du sens périphérique vers réseau de

communication, sachant que pour chaque opération d'adaptation de protocole existe l'opération réciproque.

Interface de Communication Matérielle

A l'interface entre les périphériques synchrones et le réseau, le module CA (pour "*Communication Adaptor*" dans la Figure 4-28) permet d'adapter le protocole de communication et de gérer le format des transactions avec le réseau. Ce module *CA* est de même synchronisation locale que le périphérique auquel il est rattaché. Le protocole de niveau signal reste donc synchrone au niveau du module CA, malgré une mise en forme des données en codage "données groupées" qui est supporté par les circuits synchrones et permet donc l'adaptation progressive vers un protocole de niveau signal asynchrone.

Ainsi le module CA est découpé en deux blocs :

1. le bloc Adaptateur de Services (*NoC/Core Services Adaptor*) adapte le protocole de services synchrone natif du périphérique (label 1 de la Figure 4-31) au protocole de services offert par le réseau FAN (label 2). Il met en forme les données au format de mot et de paquet accepté par le réseau. Au niveau signal le protocole reste inchangé ;
2. le bloc Adaptateur de Protocole (*BD/Native Protocol Adaptor*) adapte le protocole de niveau signal synchrone natif du périphérique en protocole de niveau signal synchrone quatre phases "données groupées" ou BD (*Bundle Data*) (label 3 de la Figure 4-31).

Interface de Synchronisation

Les modules *SPI* (Cf. Figure 4-28) interfacent les domaines synchrones des périphériques avec le réseau de communication sans horloge au moyen de synchroniseurs et de FIFOs mixtes synchrones/asynchrones (*asynch-synch FIFO* et *synch-asynch FIFO*). Ils ont pour rôle d'adapter les vitesses d'exécution et donc de communication de domaines de fonctionnement différents mutuellement asynchrones. Le codage des données reste donc "données groupées", mais devient asynchrone du côté du réseau (label 4). Ces Interfaces de Synchronisation sont détaillées au chapitre 6.

Codage des Données insensibles au délai

Cet étage d'adaptation de la communication correspond à la ressource ITR du modèle en couches de la Figure 4-28 qui adapte les protocoles entre les modules représentés par les couches SPI et PRSA/IRR de la Figure 4-28. Cet étage adapte le protocole "données groupées" asynchrone en protocole multi-rail 4 voies ou MR4 (codage des données insensible au délai avec trois états), toujours en quatre phases ((label 5 de la Figure 4-31) au moyen du composant *BD2MR data encoder*.

Routage et Arbitrage de l'Information

Les modules d'Arbitrage *PRSA* et de Routage *IRR* assurent respectivement les services d'arbitrage et de transaction. La spécification de l'ensemble de ces services va permettre de réaliser à cet étage le routage de l'information à travers la topologie d'interconnexion du réseau de communication, grâce aux composants *Initiator component* et *Target component* qui sont détaillés au chapitre 5.

Codage des Données en protocole deux phases

Selon la longueur des lignes physiques de communication entre un module Emetteur et un module Récepteur, le protocole pourra être, localement sur une liaison point à point, transformé en protocole deux phases pour réduire le nombre de transitions sur les lignes grâce au composant *2-φ 1-of-4 data encoder*. Le choix d'un codage deux phases s'avère de plus en plus favorable car avec la réduction d'échelle, le coût en énergie de la communication devient de plus en plus élevé par rapport au coût respectif de la logique de contrôle. Ce codage étant pertinent sur des lignes longues, il est réalisé au plus tard pour éviter d'implémenter les étages de communication précédents (Routage et Arbitrage) en logique deux phases très complexe. Cette étude est cependant dépendante de la technologie et des étapes de placement et de routage du circuit complet. Elle est abordée au chapitre 7.

Liaisons point à point

Il s'agit des lignes physiques qui connectent point à point un composant Initiator avec un composant Target, après une éventuelle adaptation en protocole deux phases. Des répéteurs à base de FIFOs asynchrones sont disposés le long de ces lignes pour améliorer les performances. L'étude détaillée de cette couche physique du réseau est abordée au chapitre 7.

4.3.3.4. L'architecture FAN détaillée pour un périphérique asynchrone

La Figure 4-32 présente l'architecture détaillée complète des étages de communication du réseau FAN pour un module périphérique asynchrone. Ces étages d'adaptation de communication sont simplifiés lorsque le périphérique est asynchrone. L'étage de Codage des Données insensibles au délai n'est plus nécessaire et les étages d'Interface de Communication Matérielle et d'Interface de Synchronisation, devenue une Interface de Performance, sont simplifiés.

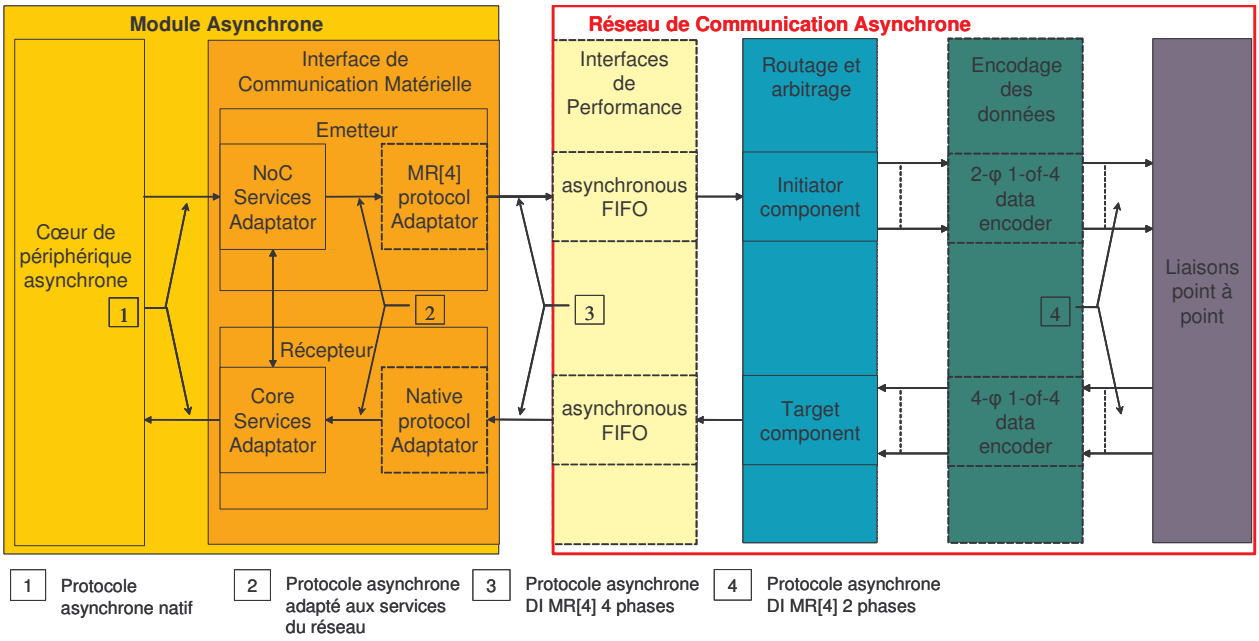


Figure 4-32 : architecture détaillée des étages de communication du réseau FAN pour un module périphérique asynchrone

Interface de Communication Matérielle

Le bloc Adaptateur de Services (*NoC/Core Services Adaptor*) reste nécessaire pour adapter le protocole de services asynchrone natif du périphérique (label 1 de la Figure 4-32) au protocole de services offert par le réseau FAN (label 2). Il met en forme les données au format de mot et de paquet accepté par le réseau. Au niveau signal le protocole reste inchangé ;

Les périphériques asynchrones du système WUCS que sont le DES et le MIPS sont codés en MR4 et ne nécessitent donc pas de bloc Adaptateur de Protocole (*BD/Native Protocol Adaptor*) dans leur Interface de Communication. Ce bloc peut cependant être nécessaire si le protocole de niveau signal natif du périphérique asynchrone n'est pas en codage MR4, mais par exemple en MR2 ou en BD.

Interface de Performance

Lorsque le périphérique et le réseau sont asynchrones, l'adaptation des vitesses relatives des domaines de fonctionnement se faisant naturellement à travers le protocole de communication de niveau signal, les Interfaces de Synchronisation ne sont plus nécessaires. On peut cependant utiliser une FIFO simple asynchrone sans synchroniseurs dans le but d'améliorer les performances de débit de la communication.

4.3.4. Degré d'automatisation de la synthèse

La conception du système WUCS présenté ci-dessus utilise les outils du flot standard de conception pour les périphériques synchrones et l'outil de conception automatique de circuits asynchrones TAST (Cf. §3.2.2) pour les périphériques asynchrones et le réseau FAN.

La conception et la réalisation des circuits développés en technologie asynchrone à l'aide de l'outil TAST, et constituants les modules du réseau de communication FAN, est découpée en trois étapes :

1. une modélisation fonctionnelle en CHP qui permet une validation par simulation fonctionnelle indépendante de la technologie cible, grâce à une bibliothèque fonctionnelle de composants asynchrones. Le non-déterminisme peut être modélisé de manière fonctionnelle, comme nous l'avons vu au §4.1.4.
2. A partir de cette modélisation, on synthétise la description structurelle du circuit grâce à une bibliothèque capable de cibler une implémentation sur FPGA. Le prototypage et la validation du système WUCS complet sont donc possibles sur FPGA (en l'occurrence le Stratix d'Altera).
3. Toujours à partir de la modélisation en CHP, il est alors possible de synthétiser la description structurelle du circuit à partir d'une bibliothèque structurelle pour la réalisation d'ASICs, construite à partir de cellules standard. Les opérateurs de choix non-déterministe @@ du langage CHP (Cf. §3.2.1) ne sont pas reconnu par le module de synthèse structurelle de l'outil TAST. Le circuit généré est donc entièrement déterministe, et les éléments synchroniseurs doivent être insérés manuellement dans le fichier de description structurelle de l'ASIC lorsque cela est nécessaire (Cf. §4.2 et chapitres 5 et 6).

Dans le cas de l'architecture du réseau de communication sans horloge FAN, les modules qui présentent un potentiel de non-déterminisme sont les modules d'Interface de Synchronisation (SPI) et d'Arbitrage (PRSA). Ces modules, qui devront donc subir une intervention manuelle d'insertion d'éléments synchroniseurs, sont étudiés en détail aux chapitres 6 et 5 respectivement.

L'étage d'adaptation du protocole en codage des données insensibles au délai ne nécessite pas d'être modélisé en CHP. Un outil de conversion automatique de protocole appelé *Interface Generator* et externe au flot de conception TAST permet de générer automatiquement les circuits d'adaptation du codage "données groupées" vers le codage 1 parmi N (*BD2MR data encoder*) et réciproquement (*MR2BD data encoder*).

4.4. Conclusion

Le chapitre 4 définit les problèmes de non-déterminisme et de métastabilité. L'élément synchroniseur destiné à prévenir la métastabilité est introduit dans ce chapitre, ainsi que la modélisation du non-déterminisme par le langage CHP.

A partir de l'étude du non-déterminisme et du synchroniseur, nous évaluons le coût d'une synchronisation par communication à travers le réseau d'interconnexion. Comparé à son homologue synchrone, un réseau de communication sans horloge résout les problèmes de synchronisation de manière plus fiable, plus performante et moins coûteuse.

Enfin, nous proposons une méthodologie de conception d'une architecture globalement asynchrone et localement synchrone en utilisant un réseau de communication sans horloge

modulaire, flexible (en terme de services, d'extensibilité et d'interfaçage), robuste et performant. Cette méthodologie repose d'une part sur la modélisation en langage CHP (*Communicating Hardware Processes*) de haut niveau d'abstraction, de problèmes très fins de synchronisation, et d'autre part sur l'utilisation de composants autonomes construits pour répondre chacun à des critères de spécification bien déterminés du réseau de communication. Nous proposons un découpage modulaire des composants du réseau en ressources critiques élémentaires : arbitrage, transport, routage et synchronisation. A cela il faut rajouter une ressource supplémentaire de mise en forme des communications, qui se place d'un point de vue méthodologique au sein des périphériques synchrones. Nous illustrons cette méthodologie à partir d'un exemple plus concret d'architecture GALS : le système WUCS (*Wireless Universal Communicating System*) dans lequel s'intègre un réseau de communication sans horloge de topologie d'interconnexion totale, que nous appelons FAN (Fully-interconnected Asynchronous NoC).

PARTIE II

Ces quatre chapitres de la première partie, présentent l'ensemble des études menées, nécessaires pour aboutir à la présentation de notre méthodologie de conception de réseaux de communication sans horloge par assemblage de composants. Dans la deuxième partie, nous revenons sur une étude plus détaillée de chacun de ces composants, ou modules.

Ainsi le chapitre 5 présente la modélisation d'arbitres à priorité et leur synthèse à partir du langage CHP. Ces arbitres font appel à des synchroniseurs construits en logique asynchrone capables de répondre en un temps fini mais non borné. Il détaille ensuite l'intégration d'une instance d'arbitre à échantillonnage parallèle dans les modules d'arbitrage et de routage, qui constituent le cœur du réseau de communication sans horloge, quelle que soit sa topologie distribuée.

Le chapitre 6 introduit l'ensemble des techniques de synchronisation existantes pour les architectures GALS. Il s'agit essentiellement des techniques à base d'interruption de l'horloge et des solutions à base d'un ensemble circuit synchroniseur plus FIFOs. Nous proposons une solution simple et performante de FIFO entièrement asynchrone, adaptée à des périphériques synchrones acceptant une synchronisation par protocole de communication. Combinée à un circuit synchroniseur robuste, elle fournit le module de synchronisation de nos réseaux de communication sans horloge. Ces synchroniseurs se trouvant aux interfaces des mondes synchrones et asynchrones, ils devront répondre en un nombre fini et fixe de cycles d'horloge.

Le chapitre 7 enfin se penche sur l'importance du problème du transport des données le long des lignes physiques d'interconnexion. Il présente les moyens d'arranger les lignes de communication pour atténuer le coût des délais d'interconnexion.

Chapitre 5 : Le cœur du contrôle au sein d'un réseau de communication : l'arbitrage et le routage

Les études présentées dans les sections 5.1 et 5.2 ont été réalisées en collaboration étroite avec J.B. Rigaud et ont fait l'objet de publications et de présentations communes [RIG 02b], [RIG 02c], [RIG 01]. On retrouvera donc dans sa thèse [RIG 02a] ces études sous une forme très semblable. La section 5.3 est la continuité de ces travaux qui font l'objet d'un dépôt de brevet en cours.

La section 5.1 présente la construction d'arbitres à base des éléments d'exclusion mutuelle et de synchroniseur présentés au §4.1.4. Ces circuits en technologie asynchrone sont parfaitement fiables car insensibles au délai. La modélisation d'arbitres complexes, en particulier à échantillonnage parallèle, est abordée dans la section 5.2. Enfin la section 5.3 illustre l'utilisation de ces arbitres dans les nœuds de contrôle de routage de l'information (Cf. §4.3.3.3) d'un réseau sur silicium sans horloge, selon la méthode de construction modulaire introduite au chapitre 4.3.Ces modules d'arbitrage et de routage, constituent le cœur du réseau de communication sans horloge, quelle que soit sa topologie distribuée.

5.1. Généralités

La spécification de l'indéterminisme grâce au langage CHP a été présentée au §3.2.1. Les circuits permettant de résoudre cet indéterminisme sont le bloc de mutuelle exclusion et le synchroniseur, qui sont les deux structures de bases indispensables pour résoudre les problèmes de synchronisation, comme nous l'avons vu au §4.1.4. Le but de cette section est de montrer que ces blocs élémentaires sont la base de la construction d'arbitres de n'importe quelle complexité.

5.1.1. Les arbitres à base d'éléments d'exclusion mutuelle

5.1.1.1. Le circuit d'exclusion mutuelle

L'exclusion mutuelle permet de faire le choix entre deux requêtes asynchrones concurrentes potentiellement vraies au même instant, comme détaillé au §4.1.4.2. La Figure 5-1 rappelle la modélisation en CHP de l'exclusion mutuelle, extraite du programme de la Figure

4-15 et exprimant la plus simple sélection possible entre deux gardes non exclusives, sachant que le choix entre les deux gardes n'est pas équitable.

```
@@[ C1# => C1?; …; loop
    C2# => C2?; …;  loop
  ];
```

Figure 5-1 : modélisation de l'exclusion mutuelle

La synthèse de cette fonction d'exclusion mutuelle est détaillée par Rigaud dans [RIG 02a] et fournit le circuit résultant suivant :

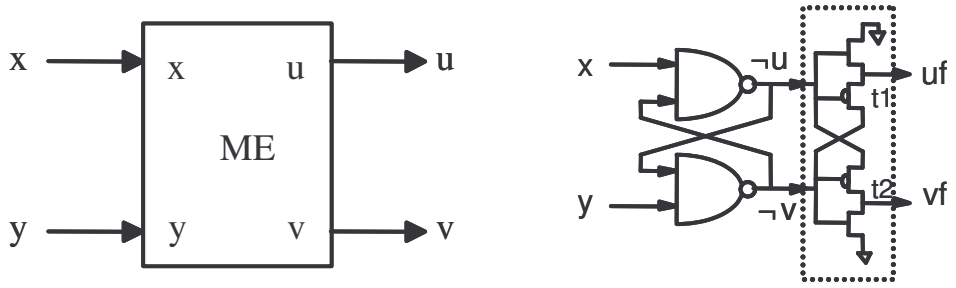


Figure 5-2 : circuit élémentaire de l'exclusion mutuelle

La réalisation de cet élément en technologie CMOS utilise les tensions de seuil à son avantage : le canal du transistor T1 ne conduit que si $(u \wedge \neg v)$ est vrai et le canal du transistor T2 ne conduit que si $(v \wedge \neg u)$. Lorsque le bloc de mutuelle exclusion est dans un état métastable, le deuxième étage de filtre se comporte comme une mémoire et conserve sur ses sorties (uf et vf) des valeurs stables jusqu'à ce que le premier étage se stabilise.

5.1.1.2. Exemples de circuits d'arbitres à base d'exclusion mutuelle

A partir de cette cellule de base d'exclusion mutuelle, il est possible de construire des arbitres plus complexes, dont la fonction reste de garantir qu'un unique client parmi deux demandeurs accède, tout en respectant la correction des protocoles de rendez-vous, à la ressource commune.

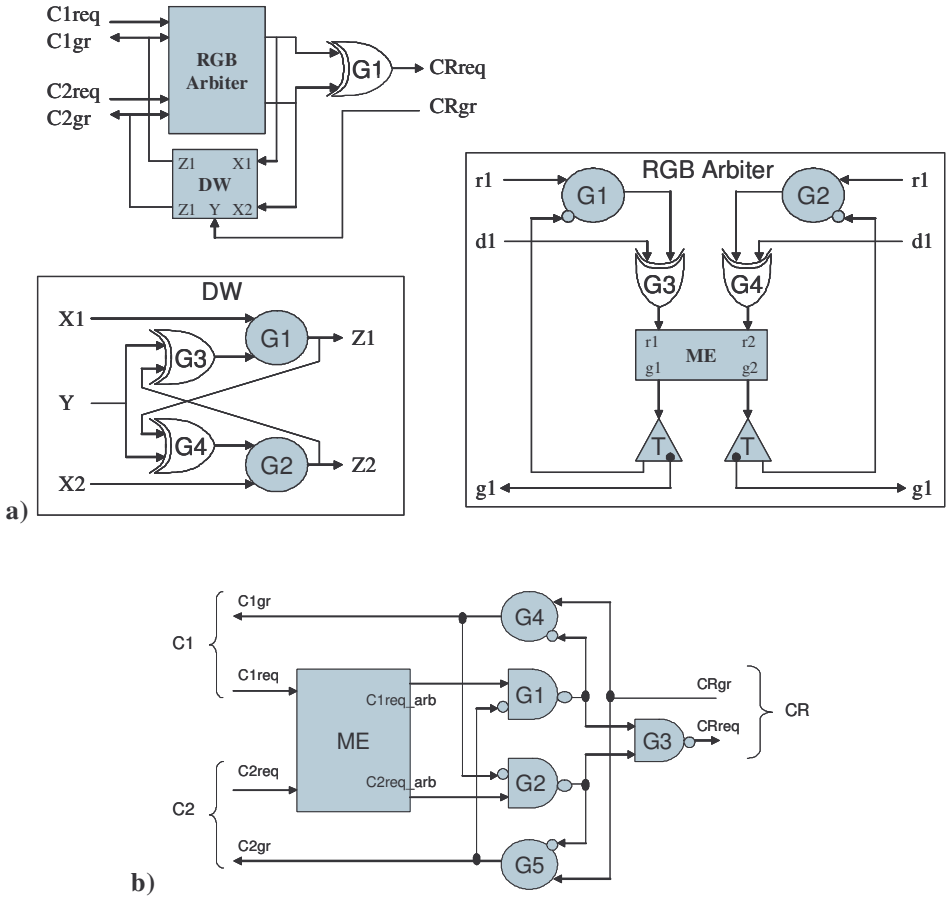


Figure 5-3 : schémas d'arbitres à base de Mutex : a) protocole deux phases, b) protocole quatre phases [REN 04b]

5.1.2. Les arbitres à base de synchroniseur

5.1.2.1. Le circuit synchroniseur

Le synchroniseur permet de déterminer la valeur logique d'un signal asynchrone vis-à-vis du signal de commande d'échantillonnage (Cf. §4.1.3). La Figure 5-4 rappelle la modélisation en CHP du synchroniseur, extraite du programme de la Figure 4-17.

```
@@[Int#    => Int?…; loop
  ¬Int#    => …;   loop
  ];
```

Figure 5-4 : modélisation du synchroniseur

La réalisation la plus élémentaire du synchroniseur peut être considérée comme la juxtaposition d'un bloc de mutuelle exclusion avec un étage de logique booléenne.

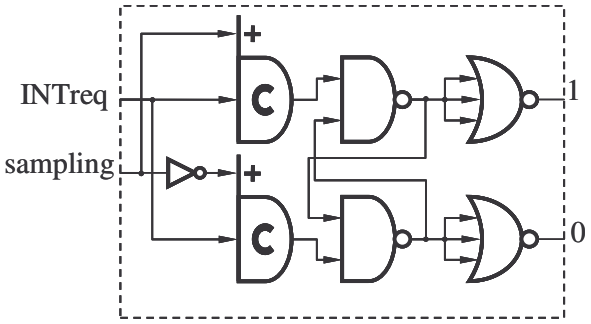


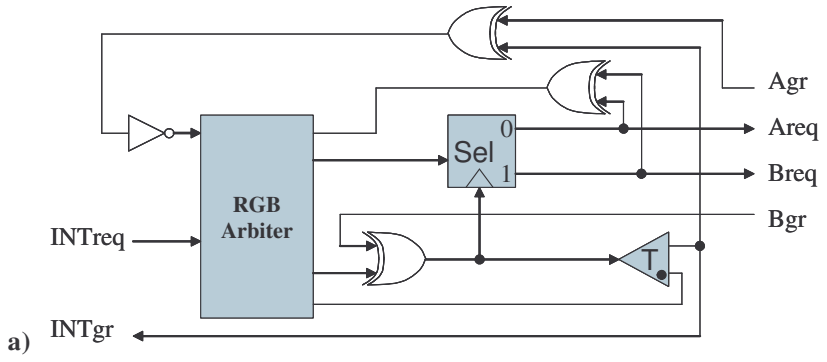
Figure 5-5 : circuit élémentaire du synchroniseur

On constate que ce synchroniseur construit en logique asynchrone présente une structure complètement différente du synchroniseur double bascules introduit au §4.1.3.2. La raison essentielle est que le synchroniseur double bascule est un synchroniseur destiné à un environnement synchrone sur horloge. Ce circuit est donc spécifié et dimensionné pour fournir un signal stable en un temps de réponse fini, par exemple deux cycles d'horloge dans le cas du synchroniseur DFF (double bascule) (Cf. §4.1.3.2), pour garantir une fiabilité suffisante au système (Cf. §4.1.2.2 sur le MTBF). Au contraire du synchroniseur construit en logique asynchrone qui peut se permettre de répondre en un temps certes fini mais non borné et qui est donc parfaitement fiable, la porte OU à trois entrées permetteant de filtrer d'éventuels états transitoires non désirés (Cf. §4.2.2.3 sur les coûts de synchronisation et [REN 00]).

Tous les circuits présentés dans ce chapitre 5 sont construits à partir d'éléments synchroniseurs en logique asynchrone insensible au délai. Le chapitre 6 traitera des circuits qui nécessitent des éléments synchroniseur devant répondre en un nombre fini et fixe de cycles d'horloge.

5.1.2.2. Exemples de circuits d'arbitres à base de synchroniseurs

A partir de cette cellule de base d'élément synchroniseur, il est possible de construire des arbitres plus complexes, dont la fonction reste de déterminer une garde potentiellement instable. Par exemple, le synchroniseur est nécessaire lorsque l'activité d'un canal intervient dans les différents choix d'un processus par l'intermédiaire de la commande probe en CHP (#) (Cf. §3.2.1).



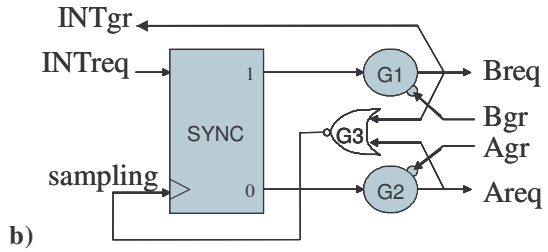


Figure 5-6 : schémas d'arbitres à base de synchroniseurs : a) protocole deux phases, b) protocole quatre phases [REN 04b]

5.1.3. Conclusion

Dans les systèmes insensibles aux délais, le fonctionnement correct de circuits contenant un élément de mutuelle exclusion ou un synchroniseur est indépendant de la durée de l'état métastable. C'est pourquoi une réalisation relativement simple de ces structures peut être utilisée. Cependant, dans le monde synchrone, l'implémentation de ces blocs doit prendre en compte la contrainte supplémentaire que potentiellement l'état métastable durera plus longtemps que la période d'horloge.

Il est possible à partir de l'élément synchroniseur présenté ci-dessus de réaliser des circuits d'arbitrage de n'importe quelle complexité, spécifiés à partir d'une des politiques d'arbitrage présentées au §2.3.3.1. La section suivante se propose de modéliser différents algorithmes de priorité au moyen du langage CHP et d'étudier les architectures obtenues après synthèse.

5.2. Spécification d'algorithmes d'arbitrage en CHP

Cette section illustre comment la modélisation de problèmes très fins de synchronisation en langage CHP de haut niveau d'abstraction, présentée dans la précédente section, permet de synthétiser des arbitres asynchrones de n'importe quelle complexité.

Nous présentons deux structures d'arbitres à priorité fixe: un arbitre chaîné et un arbitre à échantillonnage parallèle. Pour nos exemple l'ordre de priorité est le suivant : R3>R2>R1>R0. Ensuite nous verrons un arbitre à priorité dynamique et à échantillonnage parallèle. D'autres structures d'arbitres modélisés et synthétisés en CHP sont présentées en détail dans [REN 04b].

5.2.1. Arbitre chaîné "Daisy-Chain"

Spécification en CHP

La Figure 5-7 présente le code CHP qui modélise un arbitre chaîné quatre voix à priorité fixe.

```
Process Daisy_Chain_Arb
    Port ( R0, R1, R2, R3 : in di passive SR,        RS : out di active SR)
Begin
@@ [ R3# => RS! , R3? ; loop
       ¬R3# => @@ [R2 # => RS! , R2?; break
                     ¬ R2 # => @@ [R1 # => RS! , R1?; break
                                      ¬ R1 # => @@ [R0 # => RS! , R0?; break
                                                       ¬ R0 # => null; break
                                                       ];break
                                      ];break
                     ];break
    ];loop
]
end;
```

Figure 5-7 : modèle CHP d'un arbitre "daisy chain"

Le schéma de priorité est séquentiel. Le circuit est par défaut au repos et se réveille chaque fois qu'il détecte une activité sur au moins un des canaux. La fonction de sonde, ou *probe* (le symbole dièse # en syntaxe CHP), détecte l'activité d'un canal. La priorité entre les canaux d'entrée est simplement modélisée en testant séquentiellement les requêtes sur les canaux. La plus haute priorité est le canal R3 testé en premier (*R3#*). S'il est actif, la ressource commune RS est attribuée à R3 en écrivant sur RS (commande CHP d'écriture : *S!*). Parallèlement (opérateur de composition parallèle ","), R3 est servi en acquittant le canal (*R3?*). Si R3 n'est pas actif (*¬R3#*) le canal suivant est analysé de la même façon que R3 et ainsi de suite pour les autres canaux.

La stabilité des gardes #R3 et #¬R3 n'est pas garantie. Le niveau du signal de requête de R3 peut changer au moment où il est évalué. Dans ce cas, un choix non déterministe est utilisé en CHP par le double @@. Au niveau matériel il correspond à l'utilisation d'un synchroniseur pour résoudre l'éventuel problème de métastabilité qui peut se produire pendant l'évaluation sur le canal R3.

Schéma de l'arbitre

La Figure 5-8 présente la structure de l'arbitre à priorité en chaîne.

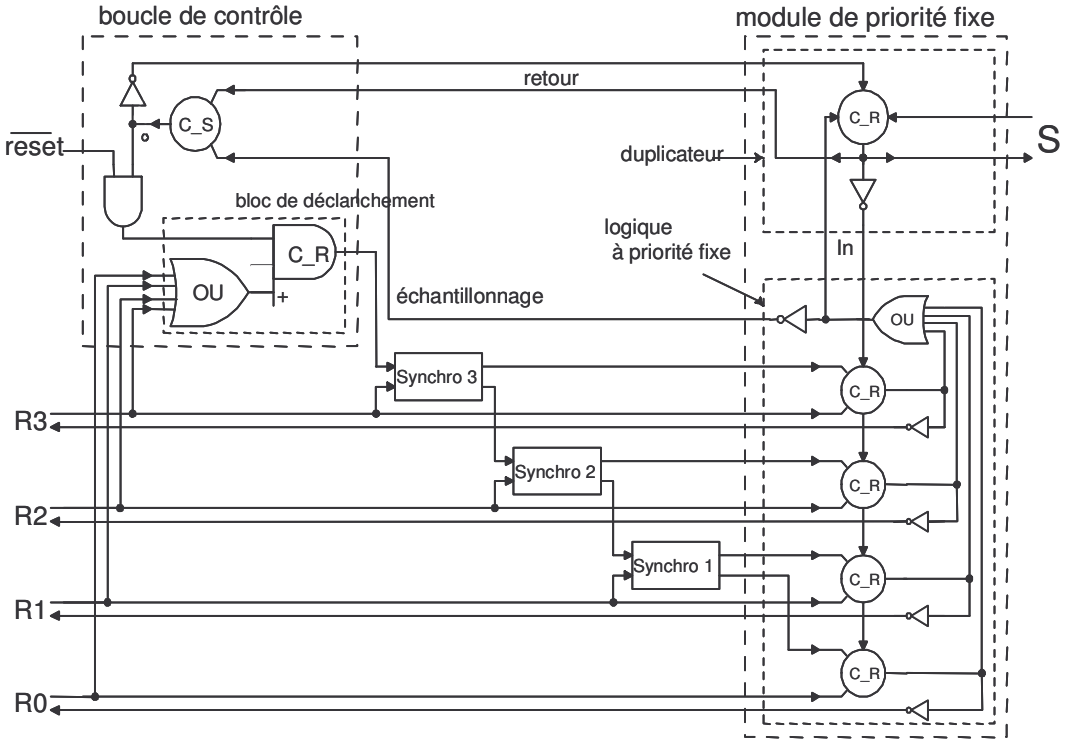


Figure 5-8 : Structure de l'arbitre à priorité en chaîne

Cet arbitre est composé de trois blocs :

- La boucle de contrôle : elle doit réactiver l'arbitre après qu'une requête entrante ait été servie et que la ressource commune ait été accédée. Elle se compose du bloc de déclenchement qui détecte toute activité sur les canaux demandeurs et qui maintient l'arbitre au repos tant qu'il n'y a pas de nouvelles requêtes.
- Le deuxième bloc est la succession de trois synchroniseurs qui échantillonnent séquentiellement les requêtes entrantes.
- Le bloc de logique à priorité fixe : il détermine le canal d'entrée à servir en fonction de l'ordre de priorité choisi. On remarque que dans cette structure, on n'a pas besoin de synchroniseur pour R0. En effet, si ni R3, ni R2 ou R1 n'est actif ce sera forcement R0. Le duplicateur contrôle la ressource partagée S.

5.2.2. Arbitre à échantillonnage parallèle

Principe et spécification

La modélisation en CHP de cette famille d'arbitres et leur synthèse en technologie asynchrone a été présenté dans [RIG 01]. Ces arbitres découplent l'échantillonnage des signaux de requête et la résolution des contentions entre les demandeurs. De telles structures dépassent les architectures précédemment proposées en terme de complexité et de latence. En plus, elles sont fortement modulaires (la fonction de priorité peut être facilement modifiée en ne changeant que le module de priorité) et aisément extensibles (les synchroniseurs sont en parallèle). Ces

arbitres réalisent des fonctions de priorité indépendantes de l'ordre d'arrivée des requêtes en attente.

La Figure 5-9 présente le code CHP qui modélise un arbitre quatre voix à priorité fixe et à échantillonnage parallèle. La structure de l'arbitre comprend un module de détection parallèle des requêtes à base de synchroniseurs insensibles au délai (Cf. §5.1.2) et un module qui implémente la fonction de priorité à résoudre.

```
[[ R0# ∨ R1 # ∨ R2 # ∨ R3 #                          -- boucle de contrôle
→ @@[       #R3 → s3 := 1; break                     -- module de détection parallèle
           ]#R3 → s3 := 0; break                     -- de requêtes : synchroniseur 3
        ] ,
   @@[      #R2 → s2 := 1; break                     -- synchronizer 2
           ]#R2 → s2 := 0; break
        ] ,
    @@[     #R1 → s1 := 1; break                     -- synchronizer 1
           ]#R1 → s1 := 0; break
        ] ,
     @@[    #R0 → s0 := 1; break                     -- synchronizer 0
           ]#R0 → s0 := 0; break
        ] ; break
 ] ;
 @[  s3 = 1 → R3 ? , S! ;loop                        -- module de priorité
      s3 = 0 → @[  s2 = 1 → R2 ? , S! ;break
                   s2 = 0 → @[  s1 = 1 → R1 ? , S! ;break
                                s1 = 0 → R0 ? , S! ;break
                               ] ;break
                  ] ;break
   ] ;
] ;loop
End;
```

Figure 5-9 : modèle CHP d'un arbitre à échantillonnage parallèle

La première ligne du code a toujours la charge de détecter l'activité sur les canaux entrants. Ensuite, quatre sous parties identiques modélisent l'échantillonnage parallèle des signaux de requêtes (commenté dans le code CHP synchroniseur 3 à 0). Les variables s3 à s0 sont utilisées pour mémoriser les échantillons (activité du canal). Le codage de la donnée est en double rails. Les échantillons sont ensuite exploités dans le module de priorité pour déterminer quel sera le canal à servir.

Schéma bloc de l'arbitre à échantillonnage parallèle

Les quatre synchroniseurs de la Figure 5-10 fonctionnent de façon concurrente pour échantillonner les signaux de requête. Le module de priorité détermine le demandeur à servir en accord avec la fonction de priorité codée dans ce bloc. En conservant le même ordre de priorité que les exemples précédents, la Figure 5-11 présente un circuit QDI du bloc de priorité.

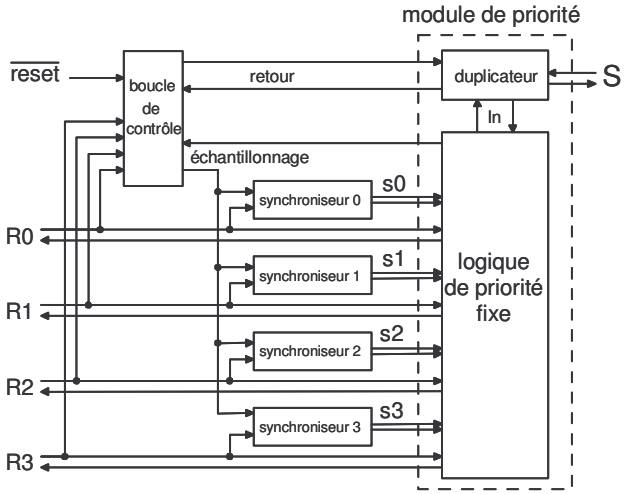


Figure 5-10 : Schéma bloc de l'arbitre à priorité avec échantillonnage parallèle

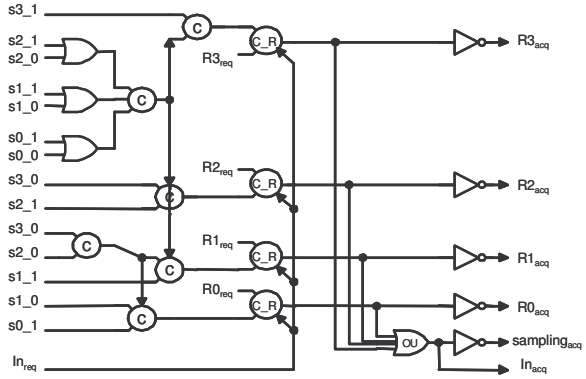


Figure 5-11 : exemple de logique de priorité fixe de l'arbitre à échantillonnage parallèle

5.2.3. Arbitre à priorité dynamique avec échantillonnage parallèle

Dans les arbitres à priorité dynamique, les canaux d'entrée transportent en plus de la requête une information de priorité. Ici le schéma de fonctionnement est différent des autres. L'arbitre à priorité fixe effectue les comparaisons des requêtes actives et sert celle qui à le niveau de priorité le plus haut. L'entrée qui aura l'index le plus haut est, par convention, sélectionné en cas de niveaux de priorité identiques. On présente ici la même structure que l'arbitre avec échantillonnage parallèle mais ici l'algorithme de résolution mis en œuvre possède une priorité dynamique.

5.2.3.1. Spécification CHP de l'arbitre

La Figure 5-12 présente le code CHP qui modélise un arbitre quatre voix à priorité dynamique qui fonctionne suivant le principe de l'arbitre à échantillonnage parallèle.

Chaque entrée Ri est maintenant codée grâce à un canal double rails : deux niveaux de priorité sont seulement considérés pour des raisons de simplicités mais il n'y a pas de limitations à ce niveau la dans la modélisation du programme. Ce canal double rails code à la

fois la requête et la valeur de priorité : zéro (« 01 » en double rails) qui est plus faible qu'une priorité de un (« 10 » en double rails).

```
* [ #( R0 ∨ R1 ∨ R2 ∨ R3 )                          -- boucle de contrôle
→ [  [      #R3 → s3 := 1 , [  #R3 = 0 → r3 := 0  -- synchroniseur 3
                              @ #R3 = 1 → r3 := 1 ]
     @@ ⌉#R3 → s3 := 0
     ] ,
     [      #R2 → s2 := 1 , [  #R2 = 0 → r2 := 0 -- synchroniseur 2
                              @ #R2 = 1 → r2 := 1 ]
     @@ ⌉#R2 → s2 := 0
     ] ,
     [      #R1 → s1 := 1 , [  #R1 = 0 → r1 := 0  -- synchroniseur 1
                              @ #R1 = 1 → r1 := 1 ]
     @@ ⌉#R1 → s1 := 0
     ] ,
     [      #R0 → s0 := 1 , [  #R0 = 0 → r0 := 0   -- synchroniseur 0
                              @ #R0 = 1 → r0 := 1 ]
     @@ ⌉#R0 → s0 := 0
   ] ] ;
-- module de priorité dynamique
-- 1st étage: analyseur de requêtes deux voies
-- et comparaison de priorité
   [ [   (s0, s1) = (0, 0) → v1_0 := 0
     @ (s0, s1) = (1, 0) → v1_0 := 1
     @ (s0, s1) = (0, 1) → v1_0 := 2
     @ (s0, s1) = (1, 1) → [   r1 ≥ r0 → v1_0 := 2
                             @ r1 < r0 → v1_0 := 1 ]
     ] ,
     [   (s2, s3) = (0, 0) → v3_2 := 0
     @ (s2, s3) = (1, 0) → v3_2 := 1
     @ (s2, s3) = (0, 1) → v3_2 := 2
     @ (s2, s3) = (1, 1) → [   r3 ≥ r2 → v3_2 := 2
                             @ r3 < r2 → v3_2 := 1 ]
     ] ] ;
-- 1er étage d'acquittement
   [   (v1_0, v3_2) = (1, 0) → comp := 0 , R0? , S!
   @ (v1_0, v3_2) = (2, 0) → comp := 0 , R1? , S!
   @ (v1_0, v3_2) = (0, 1) → comp := 0 , R2? , S!
   @ (v1_0, v3_2) = (0, 2) → comp := 0 , R3? , S!
 -- multiplexeur
   @ (v1_0, v3_2) = (1, 1) → comp := 1 , C0 := r0, C1 := r2
   @ (v1_0, v3_2) = (1, 2) → comp := 1 , C0 := r0, C1 := r3
   @ (v1_0, v3_2) = (2, 1) → comp := 1 , C0 := r1, C1 := r2
   @ (v1_0, v3_2) = (2, 2) → comp := 1 , C0 := r1, C1 := r3
   ] ;
-- 2ième étage: comparateur de priorité deux voies
   [  comp = 1 → [   C1 ≥ C0 → out := 1
                   @ C1 < C0 → out := 0 ]
   @ comp = 0 → skip
   ] ;
 -- 2ième étage d'acquittement
   [  comp = 1  → [   out = 1 → [   (v1_0, v3_2) = (1, 1) → R2? , S!
                                  @ (v1_0, v3_2) = (1, 2) → R3? , S!
                                  @ (v1_0, v3_2) = (2, 1) → R2? , S!
                                  @ (v1_0, v3_2) = (2, 2) → R3? , S!
]
                  @ out = 0 → [   (v1_0, v3_2) = (1, 1) → R0? , S!
                                  @ (v1_0, v3_2) = (1, 2) → R0? , S!
                                  @ (v1_0, v3_2) = (2, 1) → R1? , S!
                                  @ (v1_0, v3_2) = (2, 2) → R1? , S!
]
   @ comp = 0 → skip
 ] ] ;
```

Figure 5-12 : Spécifications CHP d'un arbitre à priorité dynamique quatre voies.

La première partie est très similaire à celle de l'arbitre à priorité fixe précédent. Elle correspond aux étiquettes synchroniseurs 3 à 0 dans le programme ci-dessus. Par contre, le module de priorité est beaucoup plus compliqué parce qu'il doit effectuer la comparaison des priorités des requêtes qui sont en compétition. Le programme possède une structure de comparaison des priorités à deux étages. D'autres structures pourraient être envisagées entraînant différents compromis entre complexité, vitesse et consommation.

Le premier étage du module de priorité analyse de façon concurrente, les entrées par paires et si nécessaires effectue la comparaison des priorités. La variable v1_0 signifie : pas de requête (valeur 0) ou une requête sur R0 (valeur 1) ou une sur R1 (valeur 2). La requête avec la plus forte priorité est envoyée à l'étage suivant. La variable v3_2 joue le même rôle pour les canaux R2 et R3. Ces deux variables ne peuvent pas être ensemble à zéro car les comparateurs de priorités sont déclenchés sur la détection d'au moins une requête. Si une seule requête franchit le premier étage, le canal correspondant est immédiatement servi et peut accéder à la ressource commune. Si plusieurs requêtes se présentent, elles sont traitées par le deuxième étage de comparaison. Un multiplexeur identifie les requêtes qui sont en comparaison et envoie alors leur propre valeur de priorité dans le deuxième comparateur de priorités. Finalement le résultat de cette comparaison avec le couple d'entrées en contention permet d'identifier le vainqueur (deuxième étage d'acquittement).

5.2.3.2. Structure de l'arbitre à priorité dynamique

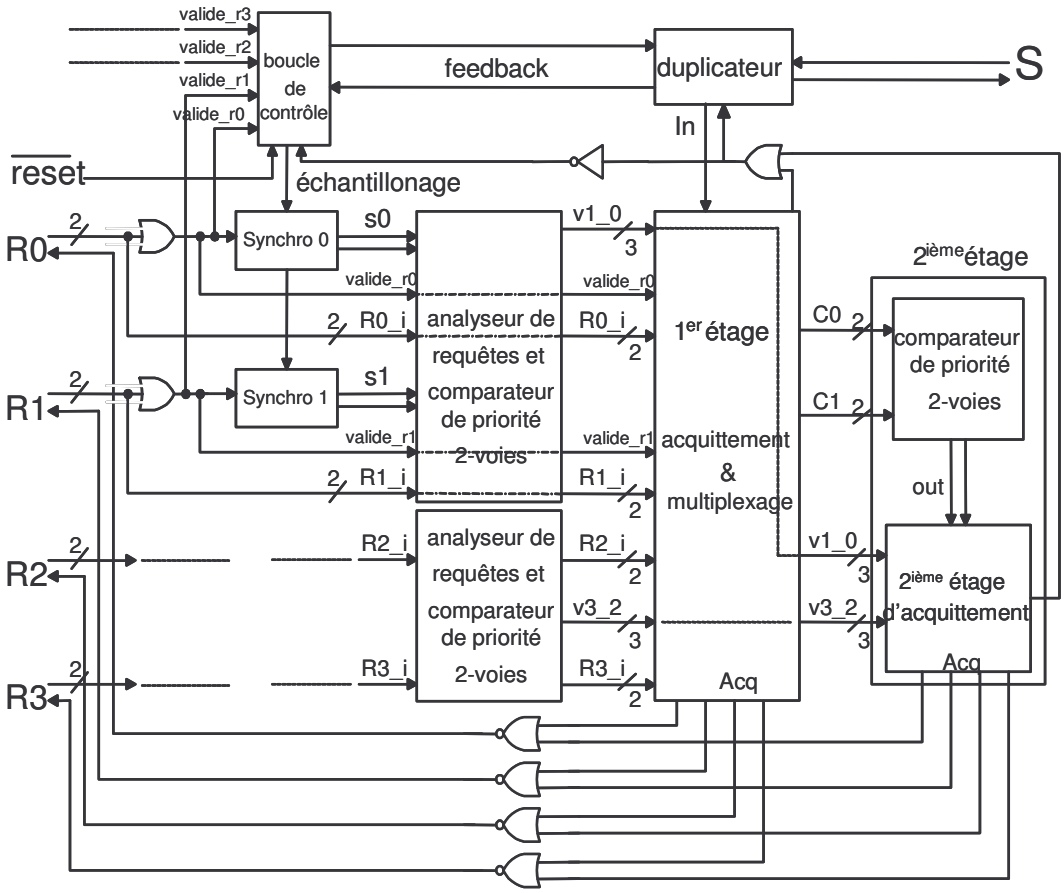


Figure 5-13 : Schéma bloc de l'arbitre à priorité dynamique.

Une description plus détaillée des blocs du circuit est maintenant abordée.

L'analyseur de requêtes et le comparateur de priorités à deux voies

Ils sont au nombre de deux et chacun compare une paire de requêtes. Le circuit synthétisé est Figure 5-14 :

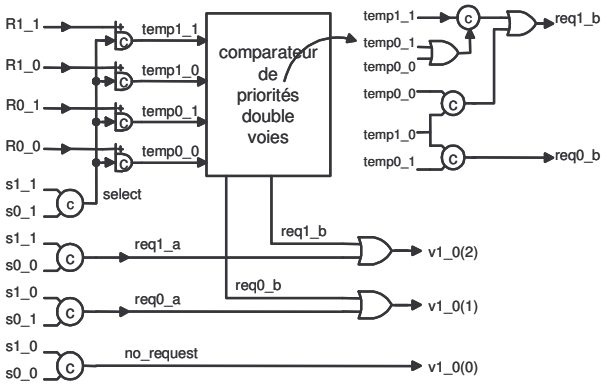


Figure 5-14 : Analyseur de priorités et comparateur de priorité.

Ce circuit a pour fonction de libérer le demandeur de priorité le plus fort. On remarque que le comparateur de priorité est activé seulement si il y a deux requêtes présentes dans le bloc (Muller (s1_1, s0_1).

Premier étage d'acquittement et multiplexeurs

Ce bloc détecte si une seule requête est active à la sortie des deux blocs précédents. Dans ce cas favorable le demandeur est immédiatement acquitté et peut accéder en parallèle à la ressource commune. Voici une représentation du circuit :

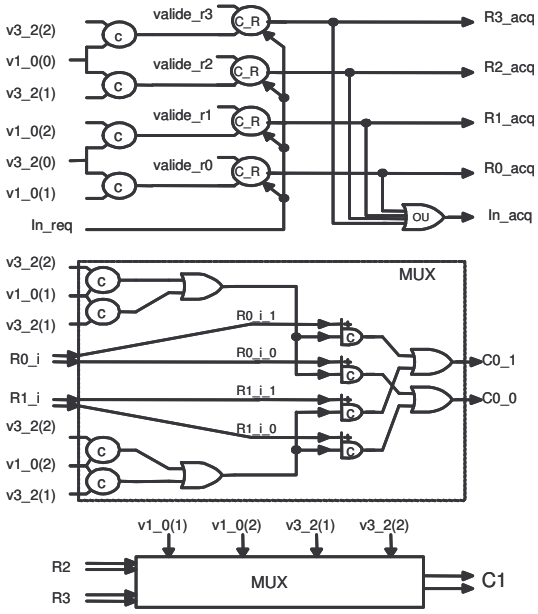


Figure 5-15 : Multiplexeurs et premier étage d'acquittement.

Dans le cas où plusieurs requêtes sont en contention après ce premier étage, deux multiplexeurs propagent le niveau de priorité au comparateur de priorités final.

Deuxième étage d'acquittement pour servir le canal vainqueur.

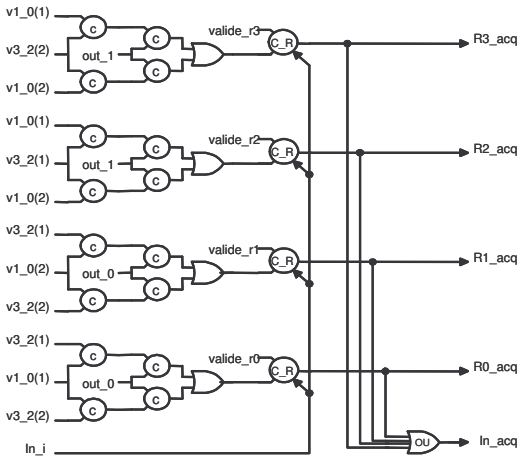


Figure 5-16 : Schéma du deuxième étage d'acquittement.

5.2.4. Conclusion

Cette section a présenté la modélisation en CHP d'arbitres à priorité, qui sont des structures non-déterministes, ainsi que l'étude des circuits à base de synchroniseurs issus du résultat de leur synthèse.

En particulier une famille d'arbitres à aéchanitllonnage parallèle a été présenté dans [RIG 01]. Ces arbitres découplent l'échantillonnage des signaux de requête et la résolution des contentions entre les demandeurs. De telles structures dépassent les architectures précédemment proposées en terme de complexité et de latence. En plus, elles sont fortement modulaires (la fonction de priorité peut être facilement modifiée en ne changeant que le module de priorité) et aisément extensibles (les synchroniseurs sont en parallèle). Ces arbitres réalisent des fonctions de priorité indépendantes de l'ordre d'arrivée des requêtes en attente.

La section suivante présente comment ces arbitres à échantillonnage parallèle s'intègrent dans les modules d'arbitrage des nœuds d'un réseau de communication sans horloge.

5.3. Construction d'un nœud de contrôle : arbitrage et routage

Les modules d'Arbitrage *PRSA* et de Routage *IRR* constituent le cœur du réseau de communication sans horloge. La philosophie de leur conception est présentée aux §4.3.2.1 et §4.3.2.2. Leur place dans une architecture de réseau de communication sans horloge est illustrée par l'exemple du réseau FAN (Cf. la Figure 4-31 qui détaille les étages de communication de ce réseau). Les modules *PRSA* et *IRR* assurent respectivement les services d'arbitrage et de transaction. La spécification de l'ensemble de ces services est celle du réseau FAN introduit au §4.3.3.2. Cette spécification va permettre de réaliser le routage de l'information à travers la topologie d'interconnexion du réseau de communication, grâce aux modules d'Emission et de Réception (respectivement *Initiator component* et *Target component* de la Figure 4-31) qui sont détaillés dans la suite de ce chapitre.

Les modules d'Arbitrage et de Routage sont distribués à chaque nœud du réseau, mais le choix d'un réseau d'interconnexion totale de n vers n-1 tel que FAN simplifie l'implémentation de ces ressources. Les nœuds du réseau étant tous connectés à un périphérique, l'arbitrage se fait uniquement dans les modules de Réception.

Ces arbitres asynchrones sont non déterministes et sont construits sur le principe des arbitres à échantillonnage parallèle *Parallel-Request-Sampling Arbiter* présentés aux §5.2.2 et §5.2.3. Le bloc d'arbitrage permet à chaque nœud d'implémenter localement son propre algorithme de priorité statique ou dynamique sur l'exemple de ceux présentés au §2.3.3.1.

5.3.1. Maître/Esclave et Emission/Réception

Une remarque à propos des modules d'Emission et de Réception (ou *Initiator component* et *Target component*): ces modules existent dans l'étage de communication "Routage et Arbitrage" de la Figure 4-31 pour tous les périphériques. Le réseau ne fait pas de distinction

entre les périphériques Maître et Esclave, ce qui améliore sa modularité et son extensibilité. Un périphérique Maître est un périphérique capable d'initier une transaction sur le réseau de communication sans répondre à une précédente communication. Un périphérique Esclave initie une transaction uniquement en réponse à une précédente communication. Par exemple dans le système WUCS (Cf. §4.3.3.1), le MIPS1 Maître initie au moyen de son module d'Emission une communication de son propre chef, pour demander le cryptage d'une donnée au DES Esclave. Le DES reçoit la demande par son module de Réception, traite la donnée à crypter puis initie à son tour au moyen de son module d'Emission une communication sur le réseau vers le module de Réception du MIPS1.

5.3.2. Le module d'Emission (ressource de Routage IRR)

Le module d'Emission, illustré par le composant *Initiator* de la Figure 5-17, prend la décision de routage de l'information dans le réseau de communication sans horloge.

5.3.2.1. Schéma de Principe

La Figure 5-17 présente l'architecture du module d'Emission (*Initiator component*). Le bloc *Target_Add ress Decoder* contrôle les deux démultiplexeurs qui pilotent les canaux c*omplexifie Control* et *Packet Data* vers le bon destinataire de la transaction. Ce bloc *Target_Address Decoder* lit un canal MR[5] à chaque transaction, même en mode rafale, évitant ainsi une mémorisation d'adresse ou un décodage des champs du canal de contrôle.

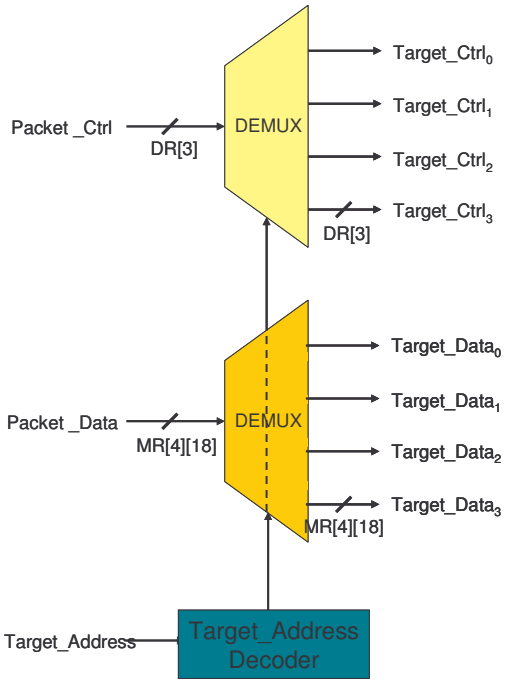


Figure 5-17 : architecture du composant Initiator (ressource de Routage)

5.3.2.2. Format des canaux de communication

Présentation

Le principe ici est de séparer complètement les informations émises dans le réseau en plusieurs canaux qui seront "consommés" par des modules différents au sein du réseau. Ces canaux correspondent aux champs du mot de la Figure 2-7 qui est rappelée ici :

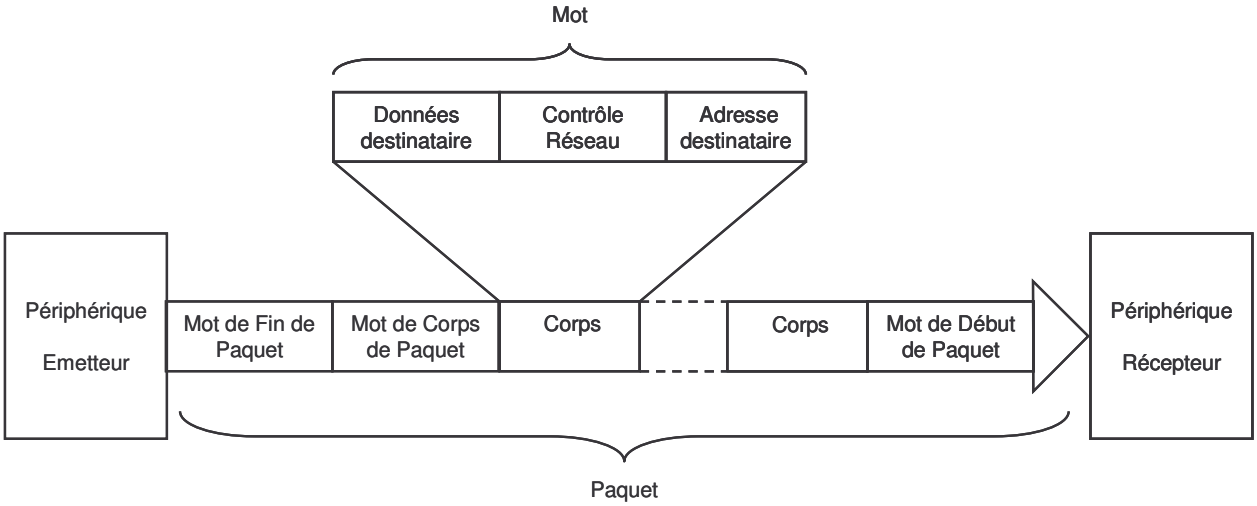


Figure 5-18 : format d'un mot de communication dans une transaction par paquet

Ces trois champs deviennent les trois canaux d'entrée du composant *Initiator* de la Figure 5-17. Le champ Adresse Destinataire devient le canal d'adresse *Target Address* de la Figure 5-17 et fournit l'adresse du périphérique récepteur de la transaction. Le deuxième champ Contrôle Réseau devient le canal de contrôle P*acket Control* de la Figure 5-17 et contient toutes les propriétés de services nécessaires à la bonne gestion des mots et du paquet par le réseau d'interconnexion. Le troisième champ Données Destinataire devient le canal de données *Packet Data* de la Figure 5-17 et contient l'ensemble des informations destinées au périphérique récepteur de la transaction.

Le Tableau 5-1 récapitule les noms des trois canaux de communication du module d'Emission, leurs différents champs d'attributs et leur taille pour l'architecture FAN.

Channel ID	Field Name	Summary
Packet_Ctrl DR[3]	Priority level DR Packet_Control DR[2]	0: High_level Priority 1: Low_level Priority Transaction Mode : 00 Burst Packet 01 Start Packet 10 End Packet 11 Single Packet
Packet_Data MR[4][18]	Data MR[4][16] Next Target Address MR[4][2]	Field for Target Peripheral Reply Address
Target_Address MR[5]		Routing in NoC

Tableau 5-1 : spécification des canaux de communication de l'architecture FAN pour le module d'Emission.

Le canal de données

Le canal de données *Packet Data* de la Figure 5-17 contient tous les champs destinés au périphérique cible de la transaction. Il s'agit des données proprement dites mais également de toutes les informations nécessaires à ce périphérique pour la gestion de ces données.

Le réseau fournit un service particulier que nous appelons Réponse Indirecte : lors d'une transaction, un périphérique maître A peut signifier à un périphérique B, esclave ou maître, de répondre à l'adresse du périphérique C, esclave ou maître. En reprenant l'exemple du §5.3.1, le MIPS1 Maître initie une communication pour demander le cryptage d'une donnée au DES Esclave en spécifiant dans le champ *Next Target Address* (Cf. Tableau 5-1) du canal de donnés non pas son adresse mais celle d'un banc de mémoire RAM. Le DES reçoit la commande, traite la donnée à crypter puis initie à son tour une communication vers la RAM dont il possède l'adresse. Seul un périphérique maître peut spécifier une adresse de réponse différente de la sienne. Ce service permet une très grande flexibilité de communication, car une chaîne de communication du type Maître1→Esclave1→M2→E2→M3… peut être configurée dynamiquement en cours de fonctionnement. La gestion de ce service est faite au niveau des blocs Adaptateurs de Services, au sein de l'Interface de Communication Matérielle du périphérique.

Le canal de données est codé en MR[4][18] : un champ *Data* de 16 digits pour obtenir 32 bits de données[11] et le champ *Next Target Address* de deux digits pour coder l'adresse de réponse éventuelle (Cf. Tableau 5-1).

Le canal de données n'est jamais exploité par le réseau qui se "contente" de le transmettre point à point de l'initiateur vers la cible. A ce canal sont associés deux autres canaux exploités par le réseau : le canal d'adresse et le canal de contrôle.

Le canal d'adresse

Le canal d'adresse *Target Address* de la Figure 5-17 fournit les informations pour router au sein du réseau les canaux de données et de contrôle associés à la même transaction. Il est codé en MR[5] car le système WUCS possède six périphériques (soit cinq destinataires possibles pour chaque *Initiator*). Ce canal n'est pas transmis dans le réseau de communication. Pour connaître l'adresse de réponse éventuelle, le périphérique récepteur utilise le champ *Next Target Address* du canal de données.

Le canal de contrôle

Le canal de contrôle *Packet Control* contient toutes les informations de services d'arbitrage et de transaction que peut exploiter le réseau (Cf. §2.3.3 sur ces deux familles de services offerts par un réseau sur silicium).

---

[11] Il s'agit ici de données au sens large et qui regroupent toutes les informations nécessaires au fonctionnement du périphérique . Par exemple pour un banc de mémoire : données proprement dites, adresse de lecture ou d'écriture en mémoire, mot de commande pour la lecture/écriture,…

Ce canal va donc spécifier pour l'architecture FAN les services précédemment identifiés au chapitre 4 (Cf. §4.3.3.2) :

- Le degré de priorité associé à la transaction. Les MIPS sont de priorité statique la plus élevée, les autres composants de priorité faible.
- L'indication du mode de transaction : en mode rafale (sans préemption) avec une taille de paquet quelconque, ou en mode simple (paquet de un mot).

L'ensemble de ces services fournit un canal codé en DR[3] (ou MR[2][3]). Le digit de poids fort *Priority Level (Packet Control(2))* spécifie une requête de transaction ainsi que son niveau de priorité associé. Les deux autres digits spécifient le mode de transaction du paquet en rafale (*burst*) ou simple. Ce qui donne quatre états possibles pour un mot :

- *Start Packet* : le mot est le début d'un nouveau paquet
- *Burst Packet* : le mot est un mot de corps de paquet
- *End Packet* : mot de fin de paquet
- *Single Packet* : le paquet est constitué d'un seul mot

Ces différents modes de transaction sont rappelés par le Tableau 5-2.

		Packet Control (0)	
		0 (continue burst)	1 (end burst)
Packet Control (1)	0 (no sync)	Burst Packet (Burst Mode)	Start Packet (Burst Mode)
	1 (start sync)	End Packet (Burst Mode)	Single Packet (Single Mode)

Tableau 5-2 : modes de transfert par paquet des données dans l'architecture FAN

Ces deux digits ne sont pas codés en MR[4] mais DR[2] pour éviter une opération de décodage dans le module de Réception (Cf. §5.3.3.4).

5.3.3. Le module de Réception (ressources d'Arbitrage PRSA et de Routage IRR)

Le module de Réception, illustré par le composant *Target* de la Figure 5-19, assure deux services essentiels (Cf. §2.3.3) :

- un service d'arbitrage : il est en charge de résoudre les conflits d'accès au périphérique (autrement dit la ressource) auquel il est connecté ;
- un service de transaction : il assure que la transaction se déroule selon le mode choisi.

5.3.3.1. Schéma de Principe

La Figure 5-19 présente l'architecture du module de Réception (*Target component*) constitué de quatre blocs présentés par la suite. Seules quatre interconnexions point à point issues du réseau son représentés sur la Figure 5-19 pour des raisons de lisibilité. Les principes présentés ici restent cependant valables et facilement extensibles au réseau FAN qui implémente des modules de Réception avec cinq interconnexions parallèles.

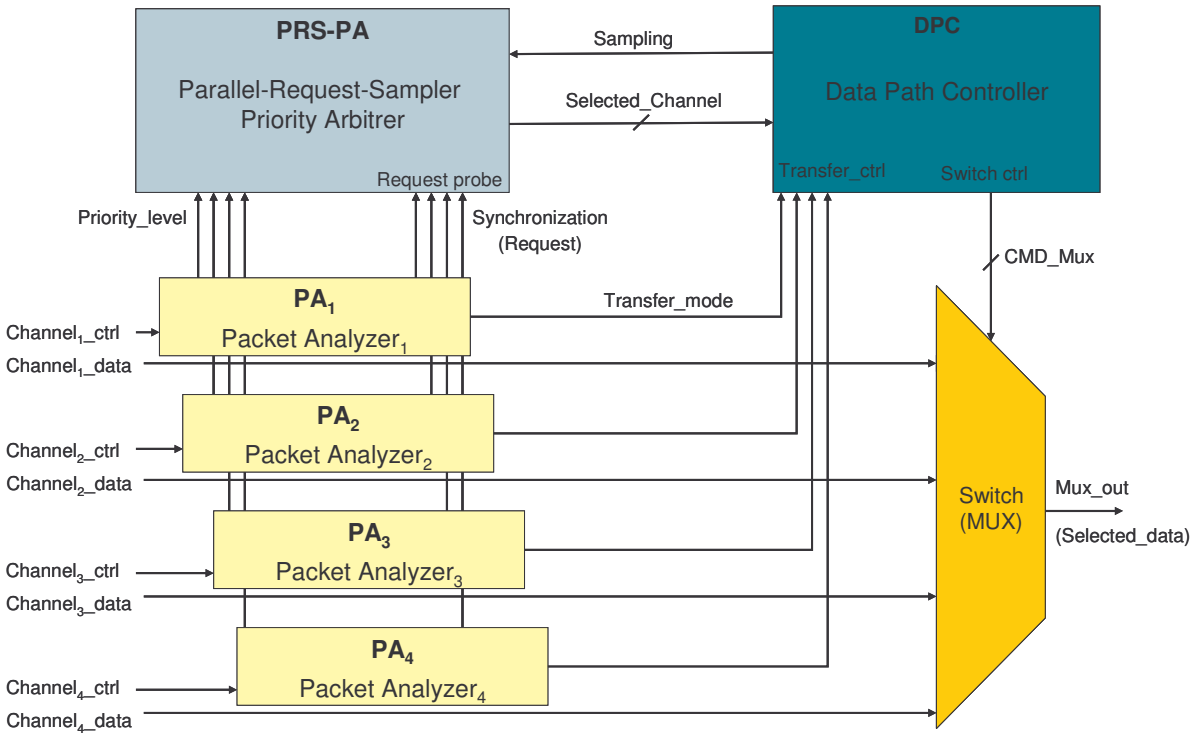


Figure 5-19 : architecture du composant Target (ressource de Routage et d'Arbitrage)

5.3.3.2. Bloc d'Analyse des paquets

Le bloc *Packet Analyzer (PA)* (Cf. §5.3.2.2) est chargé de décoder les champs du canal de contrôle, issu du réseau et appelé dans la Figure 5-19 *Channel$_i$_ctrl*.

Si le champ *Packet_Control* (Cf. Tableau 5-1) du canal de contrôle indique un début de paquet, le bloc PA envoie une requête au bloc PRS-PA à l'aide du signal *Synchronization* simple rail (SR), ainsi que le niveau de priorité *Priority level* associé. Il transmet au bloc DPC le champ *Packet_Control* qui spécifie le mode de transaction (Cf. Tableau 5-2). La Figure 5-20 présente la modélisation en CHP du bloc *Packet Analyzer*. L'Annexe B présente le résultat de synthèse de ce bloc au niveau DTL (la spécification DTL (*Data Transfert Level*) [DIN 03] fournit un ensemble de règles qui garantissent que le réseau de Pétri et les graphes de données sont synthétisables vers des circuits asynchrones (Cf. §3.2.2)).

```
constant Start : DR := '1';
constant Not_Start : DR := '0';

process PACKET_ANALYZER_process
     port(
     Channel_Ctrl : in DI DR[3];
     Synchronization : out DI SR;
     Priority_Level : out DI DR;
     Transfert_Mode : out DI DR
     )
variable Channel_Ctrl_var : DR[3];
begin
[
     *[
     Channel_Ctrl ? Channel_Ctrl_var;
     Transfert_Mode ! Channel_Ctrl_var[0],
             [
             Channel_Ctrl_var[1] = Start =>
                         Synchronization ! ,                   -- Transaction request
                         Priority_Level ! Channel_Ctrl_var[2] -- Transaction priority level
             @Channel_Ctrl_var[1] = Not_Start =>
                         skip
             ]
     ]
]
end ;
```

Figure 5-20 : spécification CHP du bloc PA (décodeur des champs du canal de contrôle)

5.3.3.3. Bloc d'Arbitrage

Le bloc *Parallel-Request-Sampler Priority Arbiter* (*PRS-PA)* est un arbitre à cinq voies à priorité fixe et à échantillonnage parallèle. Seules quatre voies sont représentées sur la Figure 5-19 pour des raisons de lisibilité. L'arbitre échantillonne de manière parallèle les requêtes (canaux SR *Synchronization*) et résout une fonction de priorité fixe en tenant compte des niveaux de priorité associés aux requêtes valides (canaux DR *Priority level*). Il fournit alors au bloc *DataPath Controller* (*DPC)* le canal de données élu (*Selected Channel* sur la Figure 5-19) pour être multiplexée vers le périphérique cible.

Soulignons que les éléments synchroniseurs implémentés dans le bloc d'arbitrage *PRS-PA* fonctionnent en environnement totalement asynchrone. En cas d'éventuelle métastabilité, ces circuits répondent en un temps non borné mais néanmoins fini. Or, dans les systèmes insensibles aux délais, la correction fonctionnelle du bloc d'arbitrage *PRS-PA*, contenant ces synchroniseurs, est indépendante des temps de traitement (Cf. 3.1.5), et donc de la durée de la réponse à l'état métastable. Le bloc d'arbitrage *PRS-PA* est donc totalement fiable (Cf. §4.2.2.3 sur le coût de synchronisation d'un arbitre).

La modélisation en CHP du bloc *PRS-PA* pour le réseau FAN à cinq voies est présentée en Annexe B. Elle est faite sur le principe des arbitres à priorité fixe et à échantillonnage parallèle présentés au §5.2.2. Simplement l'algorithme de priorité implémente une fonction à deux niveaux de priorité.

5.3.3.4. Bloc de Contrôle du chemin de données et Multiplexage

Le bloc *DataPath Controller* (*DPC)* pilote le multiplexeur de sortie en fonction du canal demandeur élu par l'arbitre PRS-PA et du mode de transaction du mot en cours de transfert (Cf. Tableau 5-1). Le multiplexeur (*MUX*) fourni en direction du périphérique destinataire le canal de données issu du réseau et élu par l'arbitre.

Seulement à l'initialisation ou lorsque un mot de fin de paquet est détecté (*End Packet* ou *Single Packet*), le DPC active l'arbitre PRS-PA au moyen du canal SR *Sampling* pour que ce dernier soit prêt à échantillonner des requêtes, nouvelles ou en attente, au prochain transfert. Ainsi en mode rafale, l'arbitre est mis en veille jusqu'au début du prochain paquet de communication.

L'Annexe B présente la modélisation en CHP du bloc *DataPath Controller* pour le réseau FAN à cinq voies.

5.3.4. Conclusion : modularité des composants du réseau de communication sans horloge

Ce paragraphe en forme de conclusion réfléchit sur les choix de conception que nous avons fait et insiste en particulier sur la très forte modularité de la méthodologie de conception des réseaux de communication sans horloge présentée en section §4.3.

5.3.4.1. Le choix d'une interconnexion totale

Le choix d'une interconnexion totale permet d'implémenter un module d'Emission le plus simple et efficace possible en terme de complexité de routage, de performances de latence et de débit, ainsi qu'en terme de coût en logique de contrôle. Le module de réception est également très efficace et simple, avec cependant un potentiel d'extension des services très important grâce à son découpage très modulaire qui sépare les fonctions d'arbitrage (bloc PRS-PA), de décodage du canal de contrôle (bloc PA) et de contrôle du chemin de données (bloc DPC).

Le choix d'une interconnexion totale répond à un besoin de flexibilité élevé à court et moyen terme, tout en conservant un coût de complexité faible. Nous n'utilisons bien sûr pas de bus centralisé difficilement extensible et évolutif en services. La méthodologie est donc d'explorer les architectures de WUCS sur une plateforme de communication générique[12] distribuée très simple. Lorsqu'une architecture de WUCS est figée, il suffit alors de spécialiser et de simplifier le réseau d'interconnexion en fonction des besoins de l'application et de chaque liaison Emetteur/Récepteur (E/R). L'interconnexion totale permet en effet d'adapter efficacement les tailles des canaux de chaque liaison point à point E/R ainsi que la fonction d'arbitrage de chaque module de Réception. Elle a d'ailleurs la capacité de devenir non totale en supprimant certaines connexions point à point inutiles ou supportant suffisamment peu de trafic

[12] interconnexion totale de tailles des canaux de communication standards et algorithme d'arbitrage unique.

pour autoriser des communications indirectes par passage en mémoire par exemple, ou encore en utilisant le service de Réponse Indirecte (Cf. §4.3.3.2 et §5.3.2.2).

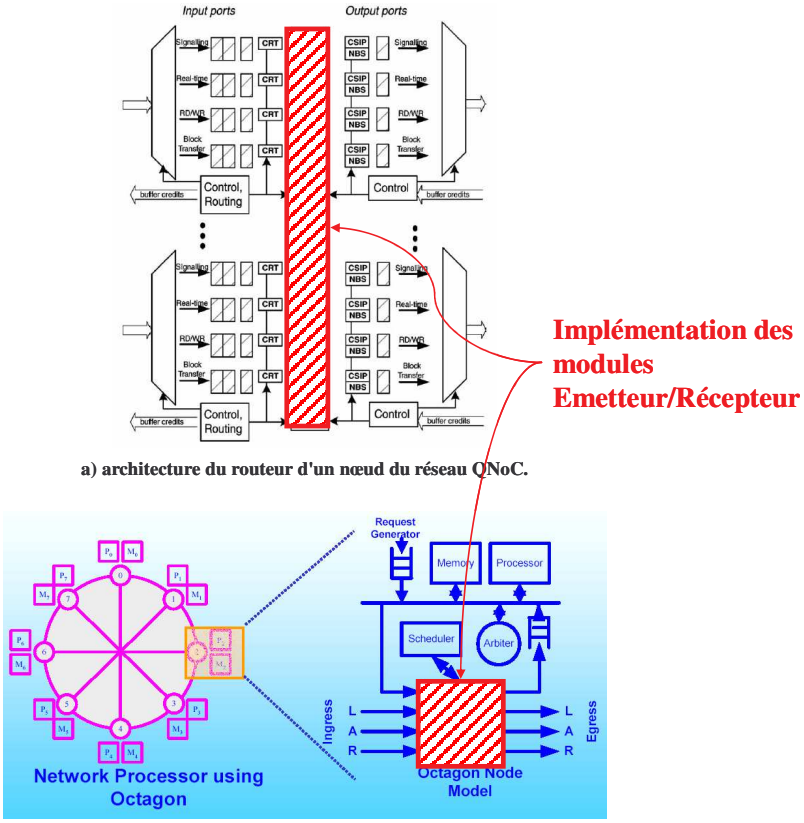
Cependant, l'interconnexion totale, même optimisée sur chaque ligne de communication point à point, coûte chère en connexions physiques. La complexité également devient moins avantageuse lorsque le nombre de périphériques du système augmente (Cf. §4.3.3.2), nécessitant de se tourner vers d'autres architectures de réseaux de communication.

5.3.4.2. Utilisation des modules Emetteur/Récepteur au sein d'autres routeurs de réseaux de communication

L'avantage principal des modules E/R présentés ci-dessus est leur extrême modularité et extensibilité. Ces modules peuvent donc servir de composants de base pour la création de nouvelles architectures distribuées de réseaux de communication fournissant des services plus étendus, de routage notamment lorsque les systèmes sont complexes.

Ces réseaux de communication distribués sont tous, dans le principe, constitués de nœuds de communication intégrant des routeurs plus ou moins sophistiqués. Une utilisation performante de nos modules E/R est de servir de composant de base à ces routeurs au sein de réseaux de communication. En effet, tous ces routeurs nécessitent une matrice d'interconnexion centrale point à point, généralement de type crossbar ou interconnexion plus ou moins totale, avec un degré de service interne relativement simple. Ces routeurs de nœuds de communication profitent alors des performances élevées et des faibles coûts des modules E/R, au prix d'un faible coût d'adaptation.

Par exemple, parmi les nombreuses architectures de réseaux distribués intégrés très performantes qui existent, les modules E/R peuvent fournir le crossbar central d'interconnexion du nœud d'un réseau à maille régulier ou irrégulier comme le QNoC [BOL 03], ou encore du Mux/Demux des noeuds de l'Octagon. Ces deux architectures sont rappelées su la figure suivante. Pour des systèmes de complexité supérieure à celle de WUCS, la réalisation de versions asynchrones des réseaux du type Octagon ou QNoC nous paraissent en effet des architectures très performantes.



a) architecture du routeur d'un nœud du réseau QNoC.

b) modèle architectural de l'Octagon

Figure 5-21 : utilisation des module E/R dans des réseaux d'interconnexion complexes

Les modules E/R peuvent bien sûr également être adaptés à d'autres architectures classiques de routeurs comme les crossbars (Cf. §2.4.2.3) ou les commutateurs Batcher-Banyan (Cf. §2.4.2.4 et [FESQ 97]) particulièrement adaptés aux protocoles asynchrones.

Chapitre 6 : Les enjeux de l'interfaçage : synchronisation et FIFOs.

Ce chapitre traite du problème de la synchronisation aux interfaces entre les domaines de fonctionnement synchrones et asynchrones. La section 4.1 du chapitre 4 a mis en relief les difficultés pour un circuit synchrone de traiter un signal asynchrone : le comportement du circuit devient non déterministe, le risque d'erreur par métastabilité apparaît et sa résolution impose des coûts élevés en latence. La section 4.2 détaille le coût de cette synchronisation pour une architecture globalement asynchrone et localement synchrone. Nous avons montré en particulier l'avantage considérable que procure l'utilisation d'un réseau de communication asynchrone dans une architecture GALS, pour améliorer le coût de cette synchronisation en terme de potentiel de non-déterminisme, de fiabilité, de vitesse, de consommation et de surface. Malgré tout, l'utilisation d'Interfaces de Synchronisation sur le principe de celles présentées dans la méthodologie de conception de réseaux de communication sans horloge du §4.3, reste indispensable. D'une part pour permettre aux périphériques synchrones d'échantillonner de la manière la plus fiable possible les signaux asynchrones issus du réseau et d'autre part pour adapter les vitesses d'exécution, et donc de communication, entre les domaines de fonctionnement du réseau asynchrone et des périphériques synchrones.

Bien sûr, il existe d'autres méthodes de synchronisation et d'adaptation de fréquence de domaines de fonctionnement différents, que l'ensemble synchroniseur plus FIFO mixte synchrone/asynchrone que nous proposons au §4.3.3.3. C'est pourquoi ce chapitre commence par passer en revue les principales techniques de synchronisation entre modules d'une architecture GALS : les horloges interruptibles, les systèmes rendus déterministes et les ensembles synchroniseur plus FIFOs asynchrones. La deuxième partie présente en détail une instance de FIFO asynchrone existante, pour laquelle nous proposons une amélioration d'architecture pour adapter les domaines de fonctionnement synchrone vers asynchrone et réciproquement, réalisant ainsi les Interfaces de Synchronisation du réseau sans horloge FAN.

Toutes les solutions d'architectures présentées dans ce chapitre proposent une implémentation des données en codage groupé nécessitant des contraintes temporelles sur les signaux de contrôle de requête et d'acquittement, mais permettant d'être compatible avec les protocoles de communication des périphériques synchrones (Cf. §4.3.3.3).

6.1. Les techniques de synchronisation des composants synchrones

La contrainte de conception d'une architecture GALS vient de sa spécificité : la synchronisation par communication de domaines d'horloges indépendantes. Tout au long du manuscrit nous avons insisté sur l'importance de cette synchronisation pour la fiabilité des systèmes globalement asynchrones et localement synchrones et sur son coût élevé (Cf. §4.2). Cette section présente les différentes méthodologies de synchronisation entre modules d'une architecture GALS et justifie en conclusion notre choix de l'ensemble synchroniseur plus FIFO mixte synchrone/asynchrone comme solution.

6.1.1. Prévention de la métastabilité : les systèmes à horloges interruptibles

6.1.1.1. Les circuits synchrones : petite histoire

Les réseaux d'interconnexion du système sur silicium *Shir Khan* [VILL 03] présenté au §2.6.1.3 utilisent des méthodes de synchronisation par interruption et extension d'horloge dites respectivement *Stoppable (or Pausable)* et *Stretchable Clock*. Ces méthodes cherchent à prévenir le risque de métastabilité, plutôt que de devoir le résoudre une fois qu'il s'est produit, au moyen de synchroniseurs dont la fiabilité non nulle ne peut être garantie [PEC 76]. Dans cette même référence l'auteur montre justement que l'utilisation de systèmes dont l'horloge peut être interrompue ou dont la phase peut être étendue permet d'améliorer la fiabilité des systèmes (entièrement synchrones à l'époque).

Cette technique est employée par Chapiro dans sa thèse [CHAP 84], mais son approche Addressait des systèmes synchrones de petites dimensions et n'est plus compatible avec les méthodologies de conception actuelle. En effet les circuits synchrones modernes tels que ceux présentés en section §1.1 utilisent difficilement ces techniques d'horloge interruptible qui font perdre le verrouillage des circuits à boucles de phase (*PLLs*).

6.1.1.2. Les architectures GALS

Ce problème ne se pose pas dans les architectures GALS de contrôle local. Ainsi Yun et al. [YUN 96b] définissent véritablement le concept actuel de *Pausable Clocking Control* (PCC)[13] pour synchroniser des horloges de fréquences différentes. Leur méthode revient à injecter un arbitre à base d'élément d'exclusion mutuelle dans la boucle de l'oscillateur générant le signal de l'horloge, sur le principe de la Figure 6-1. L'horloge est interrompue tant que l'élément d'exclusion mutuelle est métastable ou pendant la phase d'acquittement du protocole.

13 Ou SCC pour *Stretchable Clocking Control*, sachant que les deux méthodes sont bien souvent indifféremment appelées par l'un ou l'autre terme.

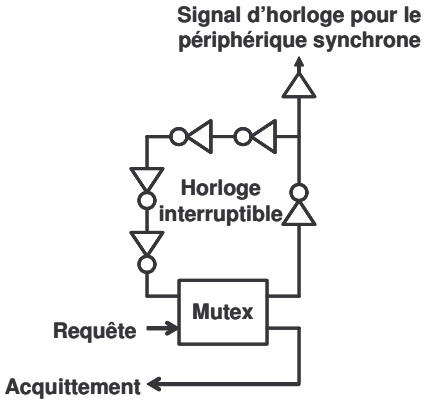


Figure 6-1 : principe de contrôleur d'horloge interruptible (PCC) à base de MUTEX

Mais leur méthode présente une latence d'au moins trois cycles d'horloge par transaction et l'implémentation des blocs d'arbitre des requêtes concurrentes sur l'horloge est irréalisable en raison des capacités de sortance et d'entrance (*fan-in* et *fan-out*) trop élevées. En outre, un taux d'erreur non nul existe toujours si les horloges des modules Emetteur et Récepteur de la communication sont corrélées [SEM 03].

Leur concept est donc améliorer par Bormann et Cheung [BORM 97] qui introduisent la notion d' adaptateurs asynchrones (*Self-Timed Wrapper*) rendant les systèmes PCC et SCC préventifs de métastabilité et donc entièrement fiables. Muttersbach, Villiger et al. perfectionnent encore les concepts PCC et SCC et fournissent les premiers circuit PCC/SCC avec adaptateurs asynchrones, sur un circuit de cryptographie tout d'abord [MUTT 99] puis sur le circuit d'étude de topologies *Shir Khan* [VILL 03].

L'ensemble des topologies de *Shir Khan* utilisant des adaptateurs asynchrones (*Self-Timed Wrapper*) est présenté en détails au §2.6.1.3. Nous rappelons donc simplement que la technique d'interruption de l'horloge consiste à ne pas déclencher le front montant suivant de l'horloge après avoir reçu le signal de son interruption, cela jusqu'à décision nouvelle de reprise de l'horloge, comme l'illustre la Figure 6-2.

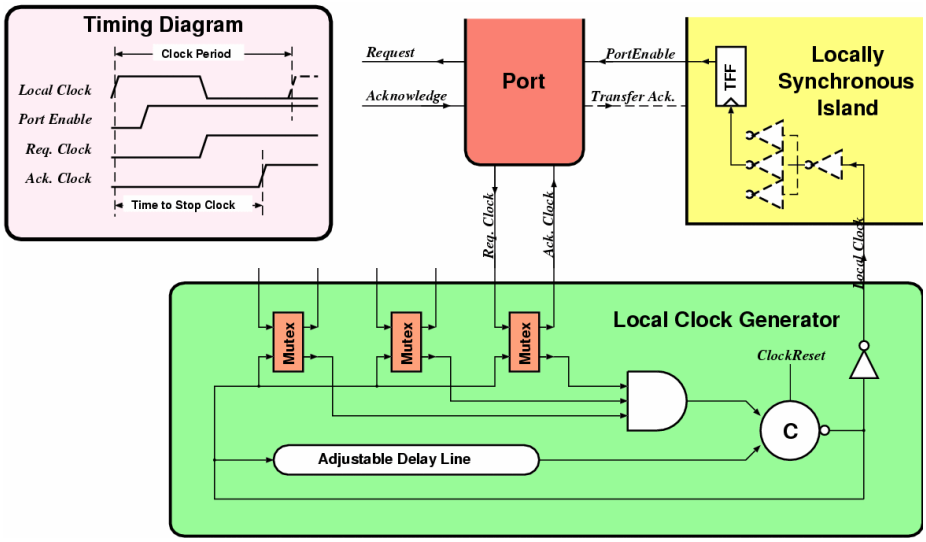


Figure 6-2 : diagramme temporel du contrôleur d'horloge interruptible (PCC) des circuits de Shir Khan [VILL 03]

La technique voisine d'extension de l'horloge consiste quand à elle à maintenir le signal d'horloge dans la phase où il se trouve jusqu'à complétion du protocole de communication. L'objectif est d'assurer que l'ensemble des signaux lus par les périphériques synchrones soit stable lorsque ces signaux sont échantillonnés, prévenant ainsi de toute métastabilité.

Ces méthodes PCC souffrent cependant d'attentes, qui peuvent être importantes, de complétion du protocole de rendez-vous avant de pouvoir redémarrer les horloges des périphériques synchrones Emetteur et Récepteur de la communication. En outre le module récepteur nécessite des modifications d'architecture pour être adapté au système PCC, ce qui dégrade son potentiel de réutilisation et de modularité. Enfin, si l'arbre d'horloge du périphérique synchrone est grand, sa gigue de la phase – variation temporelle de l'horloge par rapport à un front de référence (*jitter*) (Cf. §1.1.1) – peut provoquer une métastabilité dans l'échantillonnage des requêtes et générer une erreur [DOB 04].

6.1.1.3. Les détecteurs de métastabilité et les architectures GSLA

Certaines architectures utilisent des détecteurs de métastabilité, autrement dit de délai des signaux, qui déterminent si l'occurrence des données est trop proche de celle de l'horloge et risquent de violer ses contraintes temporelles (fenêtre de danger de la Figure 4-8 du §4.1.2.2). Dans ce cas soit l'horloge soit les données sont retardées de manière à retrouver les spécifications temporelles.

Ainsi une solution proposée dans [SJO 00] consiste à réaliser une architecture GLSA (globalement synchrone et localement asynchrone) où des modules asynchrones sont insérés dans un pipeline synchrone très haut débit, comme l'illustre la Figure 6-3. Cette solution permet de rendre le système insensible aux variations d'environnement *PVT* (*Process Voltage Temperature*) tout en étant totalement fiable. De telles solutions permettent en fonctionnement normal, soit la plus grande partie du temps, de ne pas interrompre l'horloge. Mais de tels systèmes souffrent de latences très élevées en cas d'interruption des horloges et nécessitent en outre une horloge globale de contrôle des transactions alors que l'utilisation de *PLLs* est contrainte par l'horloge interruptible justement.

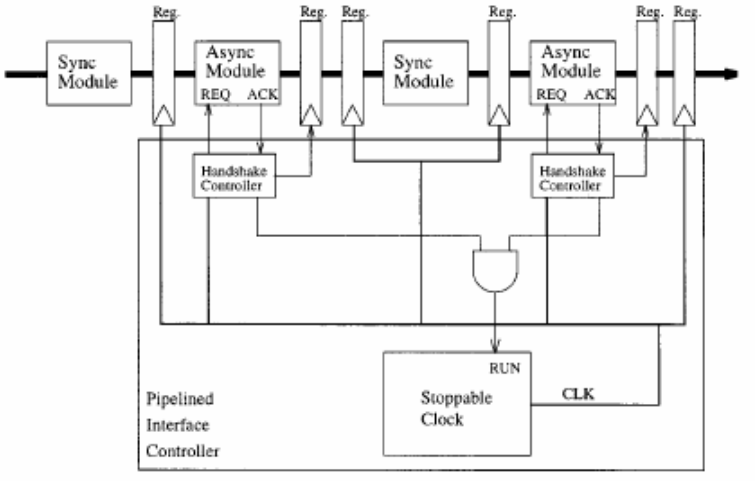


Figure 6-3 : pipeline mixte haute performance synchrone/asynchrone avec horloge interruptible

6.1.1.4. Les Systèmes rendus déterministes

Les solutions précédentes préviennent la métastabilité, en rendant le système parfaitement fiable grâce à des adaptateurs asynchrones. Mais elles restent non déterministes car le cycle d'horloge auquel le signal de requête asynchrone se produit reste imprédictible. Une solution, toujours basée sur l'interruption de l'horloge, consiste à rendre l'architecture GALS déterministe directement par construction. Le principe général est de rendre un bloc synchrone déterministe en contraignant la séquence des données d'entrée à se produire à un moment déterminé, toujours en contrôlant l'horloge.

Système à faible bande passante

Nilsson et Torkelson [NIL 96] présentent une méthode de synchronisation pour des applications de traitement de signal (*DSP* pour *Digital Signal Processing*) basses performances qui reçoivent des données à faible fréquence, mais dont l'architecture GALS est capable de fournir des performances plus élevées. La Figure 6-4 présente le principe de communication asynchrone et d'horloge locale mutuellement exclusives. L'horloge de chaque bloc synchrone est interrompue de manière synchrone par un évènement déterministe, car local, qui se produit après un nombre fixe de cycles de fonctionnement. Elle est ensuite redémarrée de manière asynchrone par l'arrivée d'un signal de requête asynchrone.

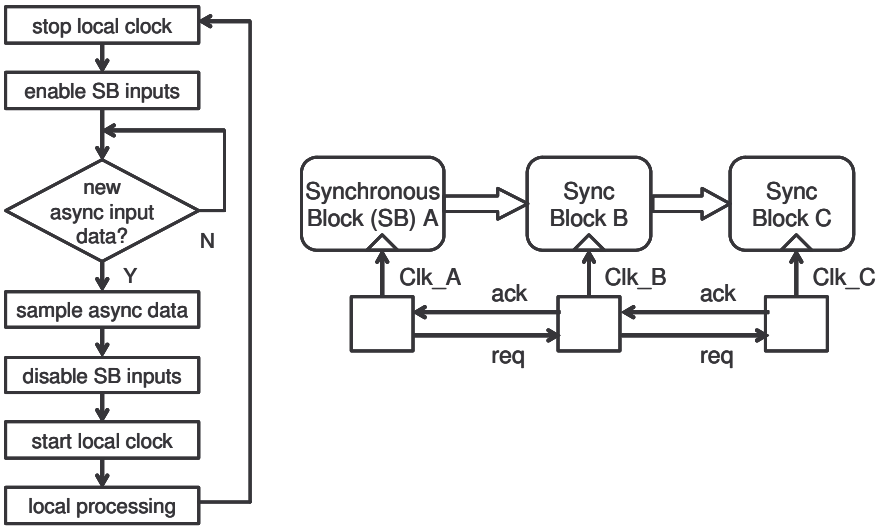


Figure 6-4 : synchronisation déterministe de systèmes à faible bande passante par exclusion mutuelle de l'horloge locale et de la communication asynchrone

Système à passage de jetons

Heath et Harris utilisent dans [HEATH 03] une combinaison de l'interface asynchrone (*Self-Timed Wrapper*) présentée au §6.1.1.2, notamment dans [BORM 97] et [MUTT 99], avec le principe d'interruption déterministe de l'horloge de [NIL 96] présenté ci-dessus.

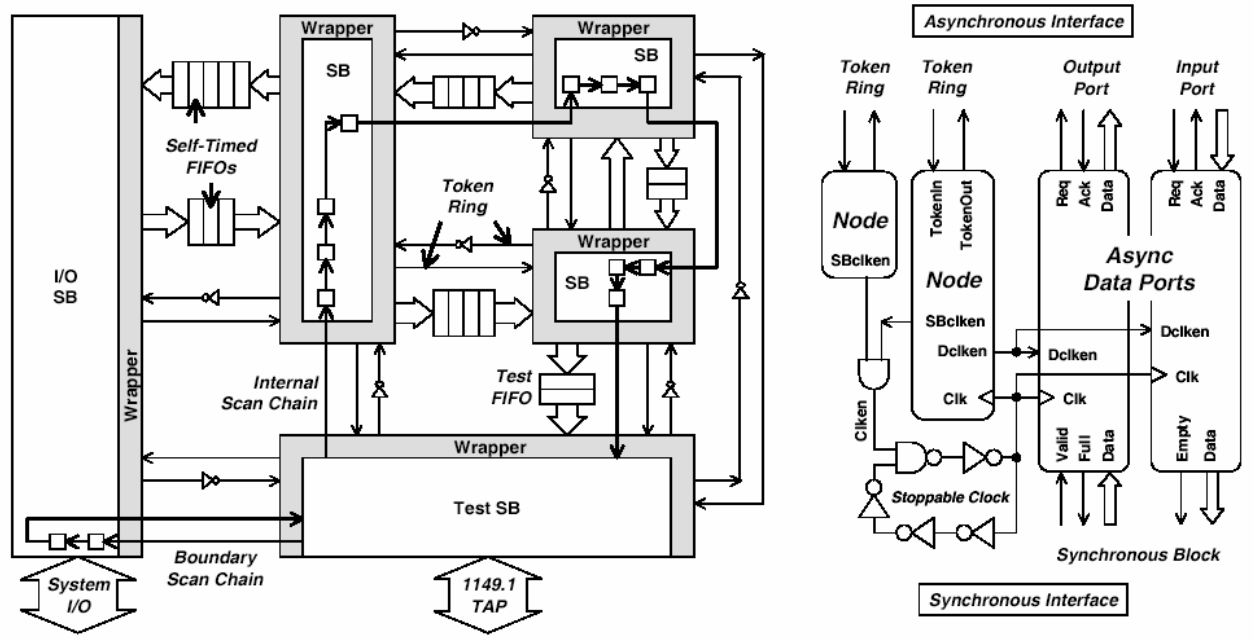


Figure 6-5 : architecture de système GALS déterministe par passage de jeton avec le détail de l'interface asynchrone

Le principe est illustré par la Figure 6-5. Chaque canal de communication point à point entre deux périphériques synchrones (*SB* pour *Synchronous Block* , Figure 6-5) est constitué :

- d'un canal de données groupées, éventuellement accéléré par une FIFO asynchrone sans synchroniseur, sur le principe de l'Interface de Performance présentée au §4.3.3.4) ;
- d'un canal simple rail deux phases qui forme un anneau de deux nœuds (*Node* sur le détail de l'interface asynchrone de la Figure 6-5) répartis dans chacun des deux périphériques.

Le système GALS est rendu déterministe par l'échange d'un jeton entre les nœuds de l'anneau. Ces deux nœuds se transmettent le jeton de manière déterministe ce qui permet de contrôler à la fois l'horloge et les ports de communication. La Figure 6-6 montre les opérations de la machine à états du nœud.

Possession du jeton

Lorsque le nœud reçoit le jeton (A) et que le compteur *Recycle Counter (RC)* atteint zéro (B), le nœud autorise le port asynchrone à compléter le protocole de communication en validant le signal *DClken (C)*, sachant qu'à ce moment tous les signaux en entrée sont valides. Par exemple pour un port de réception de communication, les signaux de requête et de données sont lus tandis que l'acquittement est émis. Ces opérations sont synchrones sur l'horloge *Clk* du périphérique, donc déterministes. Leur durée d'opération est donc décomptée (D) par le compteur *Hold Counter (HC)* du nœud à chaque cycle de l'horloge *Clk*. Ces opérations peuvent être suspendues (M) par un autre nœud, contrôlant lui aussi l'horloge *Clk*, d'un autre canal de communication point à point de ce périphérique.

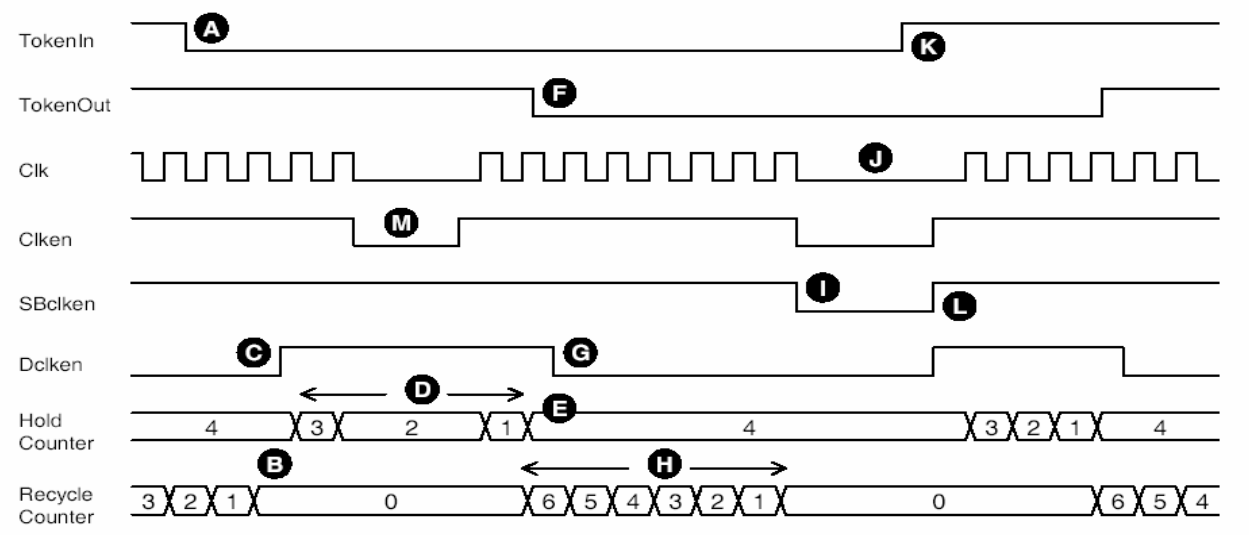


Figure 6-6 : fonctionnement de la machine à états du nœud de l'interface asynchrone contrôlant le passage de jeton

Emission du jeton

Lorsque le compteur HC atteint zéro (E), le port asynchrone est invalidé (G), le jeton est rendu (F) à l'autre nœud de la connexion point à point et le compteur RC est déclenché (H). Ce compteur est paramétré pour estimer la durée probable de réponse de l'autre noeud engagé dans la communication. Si ce dernier ne renvoie pas le jeton dans le décompte de temps du compteur RC, l'horloge est suspendue (I-J) jusqu'à la réception du jeton, qui garanti que les signaux d'entrée sont à nouveau stables et parvenus dans un délai fixe de nombre de cycles d'horloge, déterminé par le compteur RC.

6.1.1.5. Conclusion

La dernière solution intéressante présentée ici cherche à combiner les avantages de toutes les solutions : FIFOs de performance uniquement sans les synchroniseurs, adaptateurs asynchrones pour encapsuler les périphériques synchrones, interruption déterministe et réduite au mieux des horloges. Cette solution est très intéressante dans un objectif de test et de validation d'un système, facilité par une réalisation déterministe (Cf. §4.2.1.4 sur le coût de la synchronisation). Mais elle impose une estimation a priori des temps de communication difficile dans un environnement GALS (sans compter en outre les variations PVT de l'oscillateur !) : il faut être capable d'estimer finement l'occurrence de chacune des communications point à point et la durée d'une transaction pour éviter un mauvais dimensionnement du compteur RC qui interrompt l'horloge. L'arbitrage, par essence non-déterministe, n'est pas envisagé et ainsi seules des communications point à point sont autorisées. Cela convient pour des architectures hétérogènes, spécifiquement adaptées à une solution avec peu de composants, mais interdit les topologies régulières distribuées nécessaires à la création de systèmes complexes (Cf. §2.2.1 sur l'extensibilité et §2.7 sur l'automatisation de la conception des réseaux). Enfin, de manière générale, toute cette complexité des systèmes à horloge interruptible est mise en œuvre pour compenser, souvent sans succès, le risque de métastabilité d'un synchroniseur comparativement beaucoup plus petit et d'un rapport fiabilité/performances encore efficace à suffisamment long terme, comme nous allons le voir à présent.

6.1.2. Synchronisation par ensemble synchroniseur plus FIFO mixte synchrone-asynchrone

L'adaptation des domaines d'horloge mutuellement asynchrones nécessite une synchronisation pour garantir le bon fonctionnement du système. L'approche classique pour synchroniser les signaux reçus en entrée d'un périphérique synchrone est d'utiliser un circuit synchroniseur sur le principe de celui présenté au §4.1.3. La fonction de ce circuit est de résoudre l'éventuelle métastabilité qui pourrait se produire si le signal d'entrée ne respecte pas les contraintes temporelles de l'horloge (Cf. §4.1.2).

Mais si les vitesses de fonctionnement relatives de deux composants synchrones communicants sont très différentes, la communication sera souvent bloquée par l'un des deux qui sera trop long à répondre aux sollicitations. L'élément synchroniseur est donc la plupart du temps associé à un bloc de cellules FIFOs (*First-In First-Out*) destiné à améliorer le débit de la communication en tamponnant les vitesses relatives des deux composants communicants. Ainsi notre méthodologie de conception de réseaux de communication sans horloge de la section 4.3 utilise des Interfaces de Synchronisation à base de synchroniseurs et de FIFOs asynchrones. D'une part pour permettre aux périphériques synchrones d'échantillonner de la manière la plus fiable possible les signaux asynchrones issus du réseau et d'autre part pour adapter les vitesses d'exécution, et donc de communication, entre les domaines de fonctionnement du réseau asynchrone et des périphériques synchrones.

Ce paragraphe présente et étudie séparément ces deux composantes des Interfaces de Synchronisation. Le §6.1.2.1 présente rapidement les synchroniseurs et le §6.1.2.2 compare différentes instances de FIFOs asynchrones.

6.1.2.1. Résolution de la métastabilité : les synchroniseurs

Présentation

Nous parlons ici de synchroniseurs en environnement synchrone, qui nécessitent de résoudre la métastabilité en un temps borné de nombre de cycles d'horloge, au contraire des synchroniseurs en environnement insensible au délai des arbitres asynchrones (Cf. chapitre 5 précédent), autorisés à répondre en un temps fini mais non borné, et donc parfaitement fiables.

Nous avons vu au §4.1.3 que le circuit le plus simple permettant de réaliser un synchroniseur est de placer entre le signal de contrôle asynchrone et le bloc synchrone deux bascules en série synchronisées sur l'horloge du bloc [SEM 03]. Ce circuit est robuste car d'une fiabilité la plupart du temps suffisante [GIN 02] [SEM 03], mais il impose un coût en latence de deux cycles d'horloge à chaque extrémité de l'interconnexion point à point pour réaliser une transaction complète. La Figure 6-7 rappelle l'implémentation d'un synchroniseur double flip-flop pour échantillonner les signaux d'un canal données groupées.

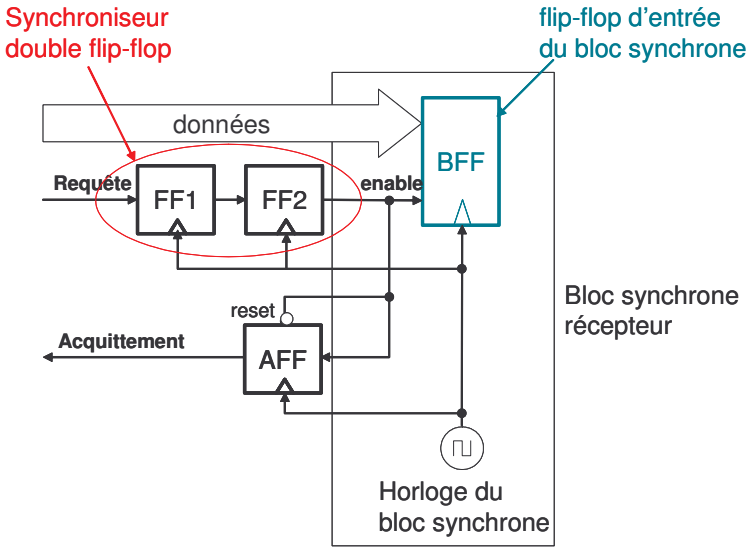


Figure 6-7 : synchroniseur double flip-flop (*DFF Synchronizer*)

Améliorations du synchroniseur

De nombreuses optimisations de l'élément synchroniseur existent pour améliorer soit sa latence soit sa fiabilité, la plupart offrant des solutions spécialisées pour des domaines de fonctionnement d'horloge particuliers (Cf. §1.1). En effet, il existe des architectures dont la synchronisation se fait par communication mais dont les composants ont des horloges de fréquences multiples multi-synchrones, plésiochrones ou hétérochrones[14]. Cela permet de se dispenser des contraintes de conception d'une horloge globale, tout en facilitant la synchronisation des composants grâce aux fréquences multiples des horloges.

Ainsi pour un système plésiochrone, Xanthopoulos et al. [XAN 01] expliquent que le synchroniseur est chargé de compenser les possibles légères différences de fréquence de fonctionnement entre deux domaines[15], qui pourraient conduire à échantillonner deux fois ou à rater un transfert de données. Dally et Poulton présentent dans [DAL 98] un synchroniseur pour des systèmes hétérochrones qui utilisent un comparateur de phases qui compare au niveau du bloc récepteur les horloges des deux blocs communicants. Si l'écart de phase est jugé trop faible, la décision d'échantillonner les données n'est pas prise et celles-ci sont perdues (d'un point de vue matériel en tout cas, une solution logicielle étant toujours possible pour réémettre les données).

Franck et Ginosar présentent dans [FRAN 04] un synchroniseur très performant dit adaptatif car il n'est pas conçu pour des systèmes dont les horloges des composants sont corrélées d'une quelconque manière. Il permet un transfert de données par cycle d'horloge du périphérique d'horloge la plus lente. En outre, il prévient du double échantillonnage ou du ratage d'un transfert de données.

---

[14] Pour le cas particulier des systèmes mésochrones existent aussi des synchroniseurs spécialisés, qui permettent de relâcher les contraintes de conception sur l'arbre d'horloge [DAL 98].

[15] Les sous-systèmes ont des générateurs d'horloge distincts destinés à fonctionner à la même fréquence mais soumis aux aléas de variations de procédés de fabrication et d'environnement de fonctionnement (température, alimentation…).

Ce synchroniseur convient donc parfaitement pour des architectures globalement asynchrones et localement synchrones, à deux conditions que:

- le système exige de hautes performances, car ce synchroniseur impose de recevoir l'horloge du module émetteur (périphérique ou nœud du réseau de communication) et demande un investissement de conception important ;
- le réseau soit de séquencement synchrone. Car si le réseau de communication est asynchrone ce synchroniseur n'est pas envisageable : les noeuds d'interconnexion devraient prendre en charge le routage de l'horloge !!

De nombreuses autres améliorations de la latence ou de la fiabilité du synchroniseur classique DFF pour des architectures GALS ont été proposées. Mais Ginosar montre dans [GIN 03] que les solutions d'amélioration de la latence dégradent toutes la fiabilité, tandis que les solutions duales offrent un très faible gain en robustesse pour un coût très élevé en latence.

Conclusion : méthode de conception du synchroniseur

Ginosar conclue que la meilleure solution consiste en une rigoureuse méthodologie de conception qui identifie tous les chemins de communication point à point reliant deux domaines d'horloge différents. Cette analyse appelée croisement de domaines d'horloge (*clock domain crossing analysis*) est réalisée au moyen d'outils d'analyse de chemins d'interconnexion du type synthétiseurs logiques et analyseurs statiques de temps de propagation, qui sont capables de générer un avertissement chaque fois qu'un signal traverse des domaines d'horloge différents. Le degré de recouvrement possible des fronts d'horloge et le risque de métastabilité doit alors être étudié pour chaque couple de domaines d'horloge différents.

La robustesse du synchroniseur classique DFF de la Figure 6-7 convient donc parfaitement pour la conception du système WUCS, à condition de respecter cette analyse des croisements de domaines d'horloge. Le synchroniseur DFF reste encore à moyen terme la meilleure solution pour la plupart des architectures globalement asynchrones et localement synchrones, dont les composants ont des horloges complètement décorrélées. Si les horloges des périphériques possèdent une quelconque corrélation, alors des synchroniseurs spécifiques peuvent être employés.

La solution la plus fiable serait bien sûr de réaliser un système GALA (Globalement Asynchrone et Localement Asynchrone) exempt de tout synchroniseur.

6.1.2.2. Adaptation des domaines de fonctionnement : les FIFOs mixtes synchrone-asynchrone

L'élément synchroniseur est associé à un bloc de cellules FIFOs (*First-In First-Out*) destiné à améliorer le débit de la communication en tamponnant les vitesses relatives des deux composants communicants. En effet si les vitesses de fonctionnement relatives de ces deux composants sont très différentes, la communication sera souvent bloquée par l'un des deux, qui sera trop long à répondre aux sollicitations. En outre, une FIFO périphérique permet de cacher la latence d'un synchroniseur. La latence pleine du synchroniseur ne sera visible que lorsque la FIFO est vide ou pleine [DAL 98]. De plus, les périphériques émettent, en fonctionnement

normal et pour la plupart des protocoles de communication des réseaux de communication (Cf. §2.5 et §2.6), une donnée par cycle d'horloge. La plupart des FIFOs utilisent alors une technique dite *lookahead* pour annoncer leur état plein ou vide, en avance de leur état réel respectivement de remplissage ou de famine, afin de prévenir le périphérique en tenant compte de la latence du synchroniseur.

Une différence existe aussi sur l'architecture des interfaces de synchronisation (synchroniseur plus FIFO) des périphériques synchrones selon que le réseau de communication soit synchrone ou asynchrone. Chelcea et Nowick montrent dans [CHEL 04] qu'une interface de synchronisation mixte entre un périphérique synchrone et un réseau de communication sans horloge (on parle de "FIFO synchrone-asynchrone") est implémentée avec moins de logique qu'une interface de synchronisation entre deux composants synchrones d'horloges différentes ("FIFO synchrone-synchrone").

FIFOs double horloges : adaptation des domaines d'horloge

En relation avec les synchroniseurs spécifiques présentés dans le paragraphe précédent pour des systèmes plésiochrones ou hétérochrones, il existe de nombreuses implémentations de FIFOs double horloges (*Dual-Clock FIFO*) qui adaptent deux domaines d'horloge. L'émetteur écrit les données dans une mémoire RAM double ports à sa fréquence de fonctionnement et le récepteur lit les données à sa propre vitesse de fonctionnement, sur le principe de la Figure 6-8.

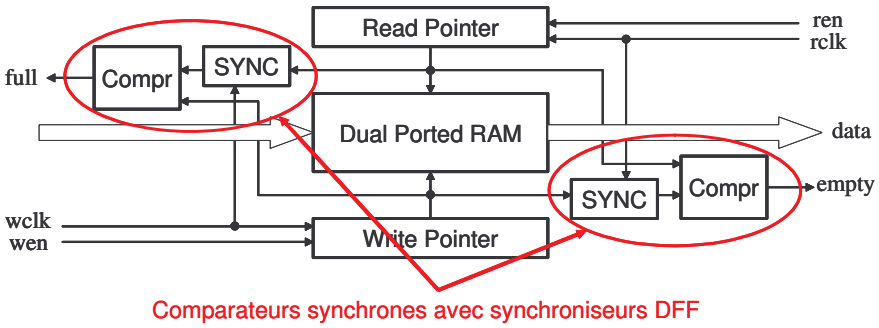


Figure 6-8 : principe de la FIFO double horloge avec synchroniseur [GIN 02]

Ces FIFOs double ports nécessitent un contrôle du flot des données très important pour écrire et lire concurrement dans la mémoire, ainsi que pour la gestion des signaux de contrôle de l'état de la mémoire (pleine, presque pleine, presque vide, vide, à moitié vide ou pleine) comme le montre l'interface de la FIFO double ports *DW_fifo_s2_sf* de Synopsis [SYN 01] illustrée par la Figure 6-9.

La synchronisation entre les domaines de fonctionnement n'est nécessaire que pour gérer les états plein et vide de la FIFO grâce à un contrôleur, ou comparateur, qui peut être synchrone comme sur les Figure 6-8 et Figure 6-9 ou asynchrone sur l'exemple de la FIFO double ports de Cummings et Alfke [CUM 02] présentée Figure 6-10. Ces FIFOs sont donc parfois appelées des FIFOs asynchrones, bien que la conception soit entièrement réalisée en technique synchrone. Si les différences de fonctionnement des deux domaines sont très différentes, elles seront souvent pleines ou vides, selon que le récepteur sera plus lent ou plus rapide, nécessitant de nombreuses synchronisations.

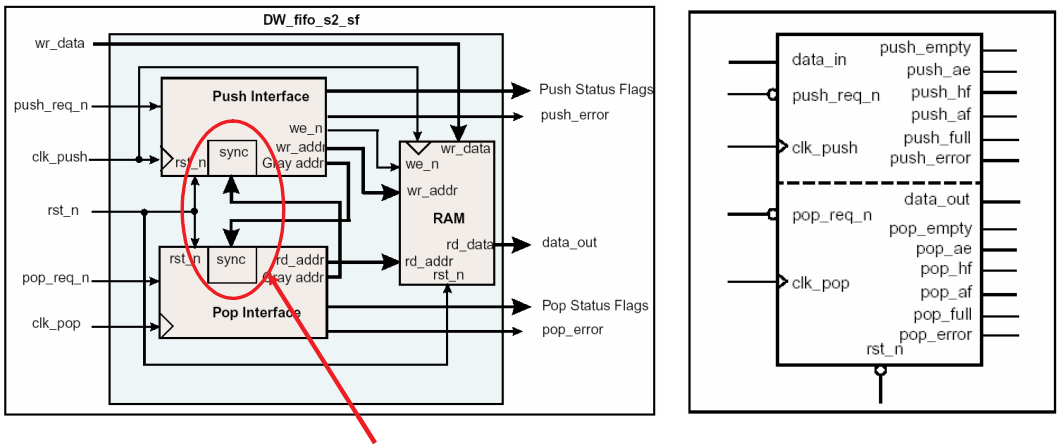


Figure 6-9 : architecture globale et interface de la FIFO double ports DW_fifo_s2_sf de Synopsis

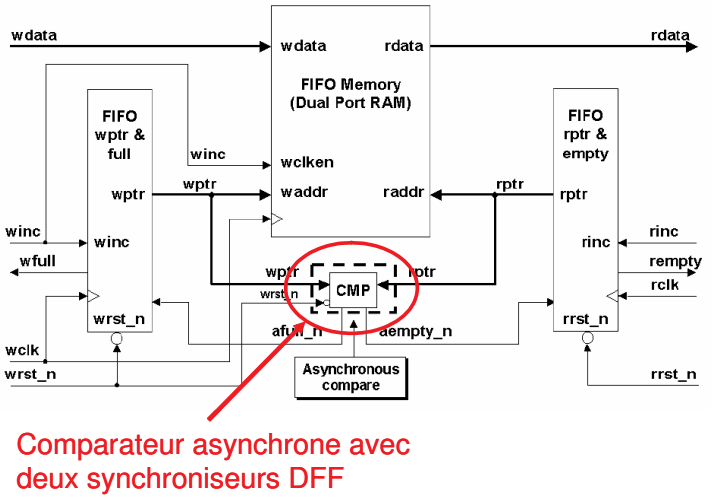


Figure 6-10 : FIFO double ports avec comparateur asynchrone

Ces FIFOs, en particulier [CUM 02] et [SYN 01], ont été étudiées dans le cadre de la collaboration sur le projet Prepor de STMicroelectronics. Nous espérions pouvoir intégrer d'un point de vue méthodologique l'instance de FIFO dans la conception de l'Interface de Communication des périphériques synchrones, entre les blocs d'Adaptation de Services et d'Adaptation de Protocole (Cf. §4.3.3.3). L'objectif était de conserver la même Interface de Synchronisation pour les deux versions de Prepor, avec un réseau de séquencement synchrone et un réseau de séquencement asynchrone. Mais ces FIFOs double port ne conviennent pas pour des architectures GALS car elles n'utilisent pas de protocole de communication par rendez-vous. En outre le contrôle de l'horloge de la FIFO du côté du réseau s'avérait complexe à mettre en œuvre et inefficace en terme de performances.

Synchronisation par pipeline

Le défaut de la FIFO est qu'elle reste située physiquement proche du périphérique. Si les lignes d'interconnexion sont longues l'utilisation de répéteurs à base de buffers de mémorisation sera nécessaire (Cf. §7.2.1), accroissant encore davantage la latence de

communication. Seizovic propose dans [SEI 94] non pas une FIFO de synchronisation mais un pipeline de synchronisation, sur le principe de la Figure 6-11. Un étage de FIFO élastique et un synchroniseur sont connectés en parallèle respectivement sur le bus de données et sur la ligne de requête ou d'acquittement. Mais cela oblige à utiliser un synchroniseur par étage de FIFO et la latence de cette solution devient proportionnelle au nombre d'étages de la FIFO. En outre l'émetteur est tenu de fournir des données à un taux constant.

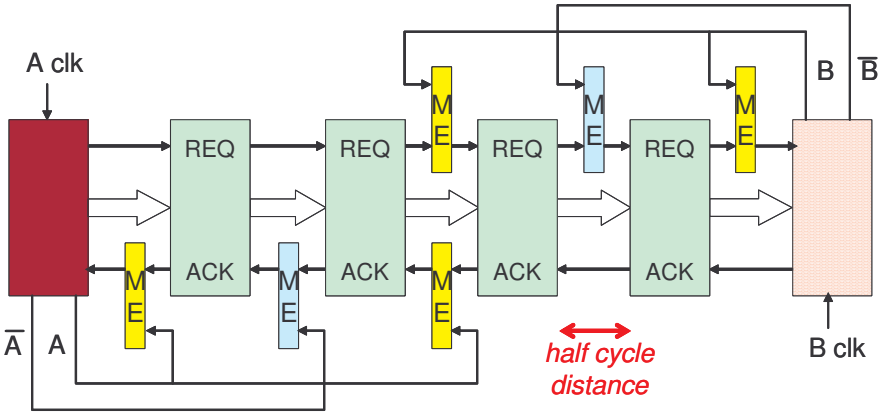


Figure 6-11 : synchronisation par pipeline

De nombreuses réalisations reprennent le principe de synchronisation par pipeline présenté ci-dessus, mais en conservant le contrôle par l'horloge uniquement aux extrémités du pipeline, sur le principe de la Figure 6-12, de manière à s'affranchir du synchroniseur [SUTH 01] [BRU 95]. Ces solutions conservent cependant le défaut de déplacer les données, ce qui induit une latence et une consommation électrique élevées. Or le pipeline n'est utile que lorsque les lignes d'interconnexion au sein du réseau sont longues. Dans ce cas là des répéteurs logiques et électriques peuvent être employés (Cf. §7.2.1) et la fonction d'adaptation des vitesses des domaines de fonctionnement reste dévolue à la FIFO proche du périphérique.

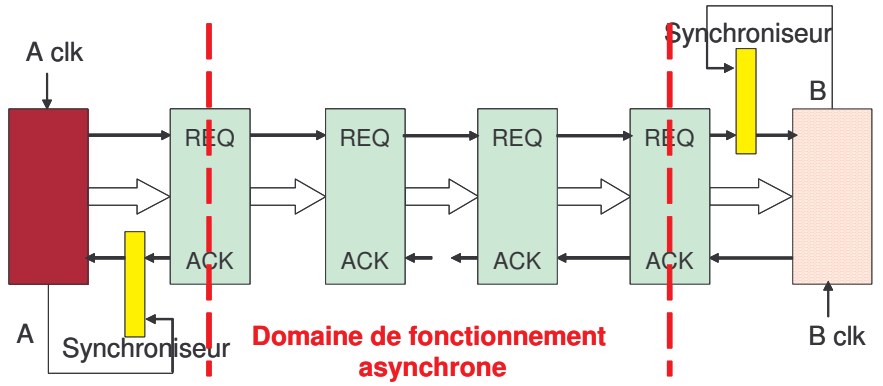


Figure 6-12 : FIFOs mixtes et asynchrones pour une synchronisation par pipeline

En outre ces solutions sont entièrement réalisées en technique de conception "données groupées". Dans notre cas cela signifierait transformer, entre le premier et le dernier étage du pipeline "données groupées", le format des canaux de communication en protocole QDI. Sans compter la complexité de gestion de l'arbitrage et du routage qui devraient concilier proximité avec les périphériques synchrones et une profondeur de pipeline suffisante pour absorber leurs

différences de vitesses de fonctionnement avec les périphériques. C'est pourquoi nous préférons rester sur la solution présentée au §4.3.3.3 : des synchroniseurs associés à des FIFOs mixtes synchrone-asynchrone en codage données groupées qui adaptent les vitesses de fonctionnement entre l'Interface de Communication du périphérique synchrone et le réseau de communication sans horloge. Dans le réseau, le protocole est rendu insensible au délai avant d'être dirigé vers sa destination. Si la ligne est longue, des répéteurs et une adaptation en protocole deux phases sont ajoutés.

FIFOs mixtes synchrone-asynchrone : adaptation des domaines de fonctionnement

Chelcea et Nowick présentent dans [CHEL 01] une FIFO circulaire mixte synchrone-asynchrone faible latence, destinée à interfacer des composants de vitesses de fonctionnement très différentes. Leur architecture est très modulable, la conception des interfaces de chaque côté de la FIFO étant parfaitement découplée. Ils peuvent ainsi fournir :

- des FIFOs mixtes asynchrone vers synchrone et réciproquement synchrone vers asynchrone pour interfacer un composant synchrone et un composant asynchrone ;
- des FIFOs double horloges pour interfacer deux composants synchrones ;
- des FIFOs entièrement asynchrones pour interfacer deux composants asynchrones ;
- ces FIFOs entièrement asynchrones peuvent servir de répéteurs pour les lignes d'interconnexion longues.

La Figure 6-13 présente les interfaces des FIFOs mixtes synchrone vers asynchrone et réciproquement. Le protocole est choisi à quatre phases données groupées.

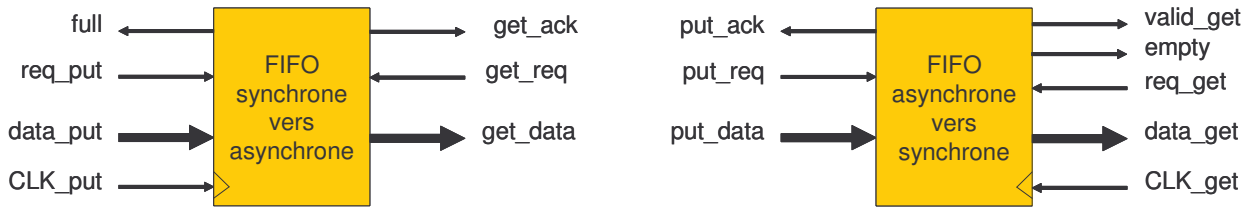


Figure 6-13 : interfaces des FIFOs mixtes synchrone vers asynchrone et asynchrone vers synchrone

Les signaux *full* et *empty* indiquent respectivement les états plein et vide de la FIFO, en avance de l'état réel de la FIFO, afin de prévenir le périphérique synchrone en tenant compte de la latence du synchroniseur (technique dite *lookahead)*. Le signal *valid_get* est un signal de contrôle supplémentaire qui indique au périphérique que la FIFO est déjà vide :

- *empty* = 0 et *valid_get* = 1 : la FIFO contient encore des données
- *empty* = 1 et *valid_get* = 1 : la FIFO délivre la dernière donnée
- *empty* = 1 et *valid_get* = 0 : la FIFO est vide

L'interface asynchrone n'a pas besoin de ces signaux de contrôle supplémentaires. Par exemple si la FIFO est pleine, le signal d'acquittement *put_ack* est simplement maintenu invalide jusqu'à libération d'une place dans la FIFO.

La FIFO mixte asynchrone vers synchrone (FIFO AS) est associée à deux synchroniseurs DFF pour échantillonner de manière robuste les signaux *valid_get* et *empty*. La FIFO mixte synchrone vers asynchrone (FIFO SA) est associée à un synchroniseur DFF pour échantillonner le signal *full*. La FIFO garantit par construction que le risque de métastabilité n'existe que lorsqu'elle passe de l'état plein à non plein ou de l'état vide à non vide.

Le cœur de la FIFO est constitué d'un buffer circulaire de cellules asynchrones élémentaires répétées autant de fois que la profondeur de FIFO désirée. Cette cellule est ensuite entourée de la logique de contrôle nécessaire en fonction de la FIFO désirée. La Figure 6-14 illustre l'architecture des FIFOs mixtes a) AS, b) SA. Par exemple pour la FIFO AS, le côté asynchrone n'a pas besoin de logique supplémentaire. Le côté synchrone requiert un détecteur d'état vide de la FIFO (*lookahead empty detector*) et un contrôleur de lecture de la FIFO (*get controler*). Pour une FIFO SA de manière symétrique on obtient un détecteur d'état plein (*lookahead full detector*) et un contrôleur d'écriture de la FIFO (*put controler*).

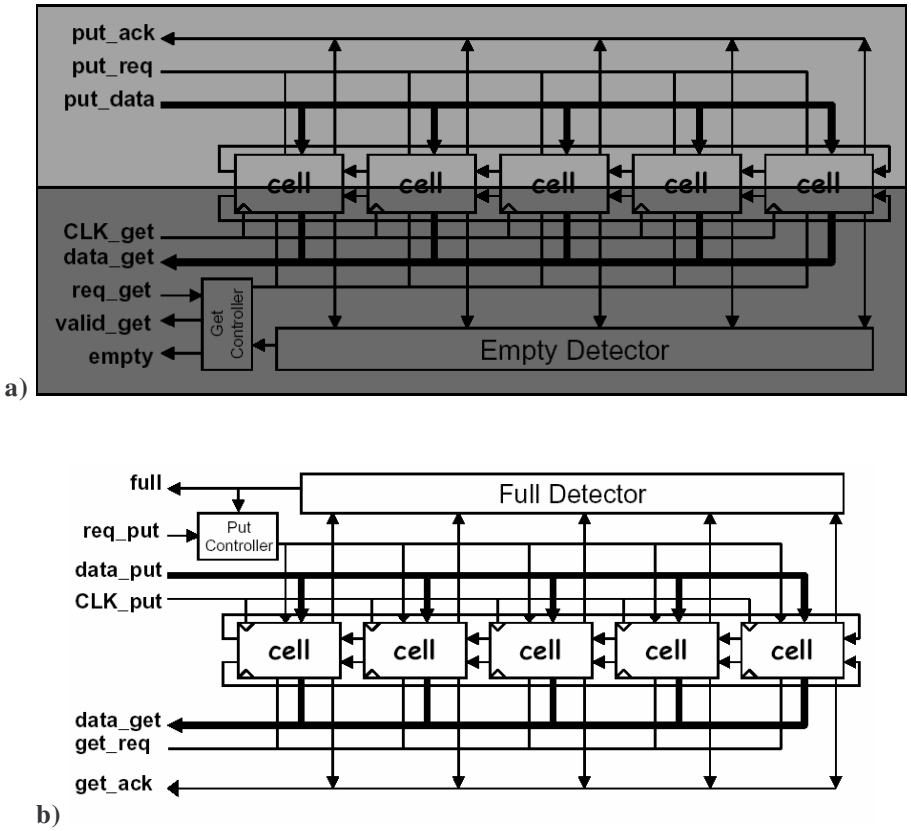


Figure 6-14 : architecture de la FIFO mixte a) asynchrone vers synchrone, b) synchrone vers asynchrone

Le contrôle de chaque cellule de la FIFO circulaire est local et autorise des accès concurrents de chaque côté des interfaces, avec une transaction par cycle garantie en fonctionnement normal (FIFO ni vide ni pleine) du côté de l'interface synchrone, à condition que les vitesse de fonctionnement des deux domaines soient proches (vitesses plésiochrones, Cf. §1.1). Ce sont deux jetons circulaires qui contrôlent le comportement de la FIFO : un jeton *Put Token (PT)* circule de cellule en cellule pour remplir la FIFO et le jeton dual *Get Token (GT)* circule pour autoriser la lecture dans les cellules de la FIFO.

Les données ne se déplacent pas : elles sont écrites et lues dans la même cellule. La FIFO offre ainsi un potentiel de faible consommation : les données sont immobiles et seuls des jetons de contrôle se déplacent. Elle offre aussi un potentiel de faible latence : la donnée peut être lue aussi tôt qu'elle est écrite sans transfert supplémentaire. En outre ces architectures sont facilement extensibles grâce au contrôle distribué des cellules : le rajout de cellules pour augmenter la taille de la FIFO et l'augmentation des tailles des bus de données se font avec très peu de modifications de l'architecture.

Nous avons donc décidé de nous inspirer de ces circuits pour construire nos Interfaces de Synchronisation. Nous avons cependant adapté uniquement les FIFOs entièrement asynchrones car :

- les FIFOs double horloges ne correspondent pas aux besoins de nos architectures avec un réseau localement sans horloge ;
- leurs FIFOs mixtes présentent une grande complexité (Cf. §6.2.2) et une latence élevée lorsque la FIFO est pleine ou vide (deux cycles d'attente avec les synchroniseurs DFF des détecteurs d'état plus une bascule supplémentaire dans la cellule de FIFO du côté synchrone), ce qui se produit souvent lorsque les composants communicants sont de vitesses très différentes ;
- nos répéteurs, au contraire de ceux présentés par Chelcea et Nowick, devront prendre en charge des canaux QDI codés en MR[4] (Cf. §5.3.2.2).

6.1.3. Conclusion

Les méthodes d'interruption d'horloge, y compris les solutions rendant le système déterministe, cherchent à prévenir la métastabilité pour fournir un circuit entièrement fiable. Le problème est que la complexité, la latence et la consommation d'énergie de ses solutions sont proportionnelles au nombre de canaux de communication point à point établis par chaque module synchrone. En outre, un nombre élevé de canaux concurrents tous capables d'interrompre l'horloge, sans arbitrage ni préemption, engendre des temps de résolution très longs, voir des blocages. De plus, avec la réduction d'échelle des composants et l'augmentation de la surface des circuits, les périphériques synchrones deviennent de plus en plus grands et l'arrêt et le redémarrage de leur horloge locale devient très problématique, en particulier pour la gestion des problèmes de déphasage (Cf. §1.1), sans compter que les solutions obtenues sont difficilement réutilisables. Enfin seules les solutions des §6.1.1.3 et §6.1.1.4 sont parfaitement fiables, mais très limitantes dans leurs applications possibles.

Les méthodes utilisant des synchroniseurs pour résoudre la métastabilité plus des FIFOs pour adapter les vitesses de fonctionnement des domaines d'horloge différents, même si ces solutions n'empêchent pas le système d'être sensible aux erreurs, permettent de meilleures performances en adaptant convenablement la latence des FIFOs et des synchroniseurs. La robustesse du synchroniseur classique DFF reste encore suffisante pour la majorité des systèmes, sous condition d'une rigoureuse méthodologie de conception : c'est l'analyse du croisement des domaines d'horloge (*clock domain crossing analysis*). Le degré de recouvrement possible des fronts d'horloge et le risque de métastabilité doit être étudié attentivement pour chaque couple de domaines d'horloge différents.

L'utilisation de FIFOs permet d'absorber la latence du synchroniseur en fonctionnement normal (FIFO ni vide ni pleine). Les FIFOs double horloge permettent de synchroniser deux composants proches et d'horloges différentes, mais parviennent difficilement à conserver un taux de remplissage efficace de la FIFO lorsque les vitesses de fonctionnement sont très différentes, car aucun des deux côtés de la FIFO synchrone n'est élastique, contrairement à une FIFO asynchrone. Les FIFOs utilisées comme pipeline déplacent les données et sont donc coûteuses en latence et en consommation d'énergie. Elles doivent être utilisées pour synchroniser deux composants synchrones avec des interconnexions longues.

Les FIFOs asynchrones permettent d'adapter de manière élastique différents domaines de fonctionnement synchrones ou asynchrones. En particulier Chelcea et Nowick dans [CHEL 01] proposent une solution complète, à partir de cellules de FIFO circulaire asynchrone, de FIFOs double horloges, mixtes ou asynchrones, pour une architecture globalement asynchrone et localement synchrone exigeant une faible latence et une faible consommation. Nous présentons à présent une étude détaillée de ces travaux et une amélioration.

6.2. Choix et étude d'une FIFO asynchrone

Cette section présente tout d'abord l'architecture des cellules de la FIFO asynchrone qui servent d'éléments de base à la construction des FIFOs mixtes précédemment introduites. Nous proposons une modélisation en CHP qui modélise l'architecture des cellules de FIFO améliorant le parallélisme des opérations. Nous proposons également une architecture de l'environnement qui pilote les cellules de FIFO, à base d'arbres de multiplexeurs/démultiplexeurs pour remplacer l'utilisation de bus de communication trois états coûteux en énergie. Dans le deuxième paragraphe nous revenons sur le fonctionnement des FIFOs mixtes SA et AS et justifions le choix de conserver plutôt une FIFO entièrement asynchrone.

6.2.1. FIFO en environnement entièrement asynchrone

6.2.1.1. Principe de fonctionnement

A l'origine de la construction des FIFOs mixtes se trouve la réalisation d'une FIFO circulaire asynchrone proposée dans [CHEL 00a]. L'architecture de cette FIFO est illustrée par la Figure 6-15, proposant une FIFO de quatre cellules.

Le *Starter* est une cellule spéciale qui permet d'initialiser la FIFO en injectant deux jetons dans l'anneau des cellules. Les canaux *put* et *get* sont des canaux quatre phases données groupées. Leurs ports de réception sont passifs (rond blanc sur la Figure 6-15, les ronds noirs indiquant le port du côté actif), c'est-à-dire que c'est toujours l'environnement qui initie le protocole de communication. Les jetons se déplacent dans le sens trigonométrique.

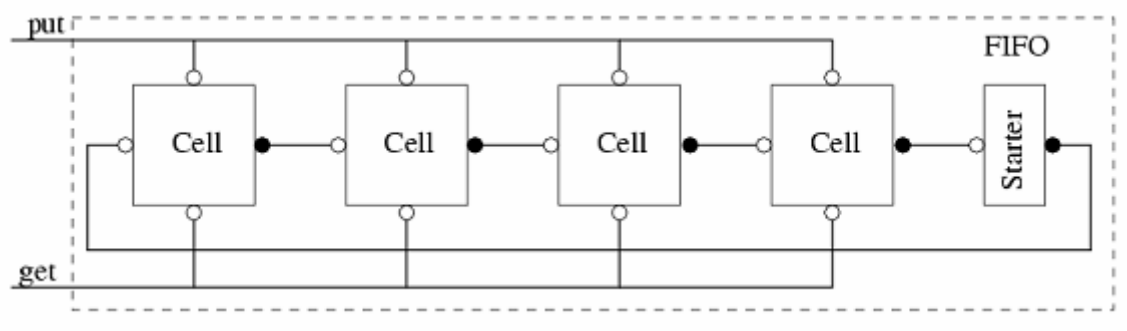


Figure 6-15 : architecture de la FIFO circulaire asynchrone

La Figure 6-16 détaille les canaux de communication et l'interface d'une cellule pour une FIFO circulaire asynchrone de deux cellules.

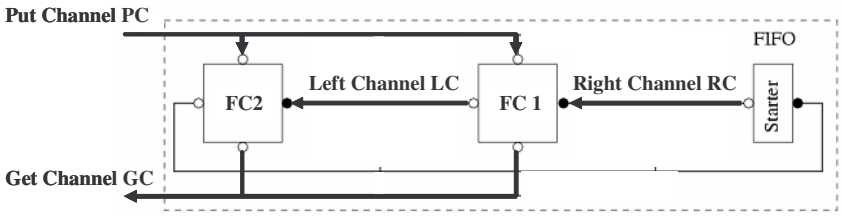


Figure 6-16 : détail des canaux de communication d'une cellule de FIFO

L'interface de chaque cellule FC de la FIFO, sur l'exemple de la cellule FC1 (*FIFO Cell 1*), est constituée de quatre ports :

- le port passif *Put Channel PC* reçoit la requête d'écriture des données dans la cellule ;
- le port passif *Get Channel GC* fournit les données en réponse à une demande de l'environnement ;
- le port actif *Right Channel RC* reçoit de la cellule précédente les deux jetons circulaires, un pour écrire l'autre pour lire ;
- le port passif *Left Channel LC* émet les jetons à la cellule suivante.

La modification de l'activité des ports (actif/passif) ne change pas le protocole de fonctionnement de la FIFO.

Ce sont deux jetons circulaires qui contrôlent le comportement de la FIFO : un jeton *Put Token (PT)* circule de cellule en cellule pour remplir la FIFO et le jeton dual *Get Token (GT)* circule pour autoriser la lecture dans les cellules de la FIFO. Le jeton PT précède toujours le jeton GT et les deux jetons ne peuvent être dans la même cellule. C'est le fonctionnement normal d'une FIFO : la cellule doit être remplie avant d'être vidée.

La Figure 6-17 présente la modélisation fonctionnelle en CHP a) d'une cellule FC et b) du Starter.

a)

```
process FIFO_cell
port (        -- channels for token passing
              RC : in bd passive SR
              LC : out bd active SR
               -- channels for data passing
              PC : in bd active BD[data_bus_width]
              GC : out bd active BD[data_bus_width]
      )
variable data : mr[data_bus_width];

begin
              [ [  -- sequence to enqueue data
                   RC? ; PC?data ; LC! ;
                   -- sequence to dequeue data
                   RC? ; GC!data ; LC! ; break
                ]; loop
              ]
end;
```

b)

```
process Starter
port (        -- channels for token passing
              RC : in bd passive SR
              LC : out bd active SR
      )

begin
              [ [  -- token init
                   LC! ; LC! ; break
                ];
                [   -- token passing
                   RC? ; LC! ; loop
                ]; break
              ]
end;
```

Figure 6-17 : modélisation fonctionnelle en CHP a) d'une cellule FC et b) du Starter

Les canaux LC et RC sont des canaux simple rail car le jeton n'a pas besoin de contenir d'information. Les canaux simple rail sont des canaux de contrôle typiquement utilisé pour le passage de jetons. Les canaux RC et GC sont des canaux données groupées (*bundle data BD*).

Initialement l'anneau est vide et le *Starter* possède les deux jetons. A l'initialisation, il met en circulation tout d'abord le jeton PT puis lorsque celui-ci a été consommé par la première cellule, le jeton GT peut lui être envoyé. C'est la séquence *[LC! ; LC! ; break]* de la Figure 6-17 b). Ensuite le *Starter* se contente de faire circuler les jetons.

Le comportement des cellules est identique pour toutes. Les opérations suivantes sont réalisées en boucle : le canal RC est lu une première fois (*RC?*), ce qui correspond à l'obtention du jeton PT. Ce jeton autorise la complétion du protocole sur le canal PC et donc l'écriture des données dans la cellule (*PC?data*). Le canal LC transmet le jeton PT à la cellule suivante (*LC!*). Le canal RC attend cette fois le jeton GT pour autoriser la lecture des données (*GC!data*) et ensuite transmettre le jeton à la cellule suivante.

6.2.1.2. Avantages et inconvénients

Le comportement de la FIFO garantit une séquence correcte : la cellule de droite empile et dépile une donnée toujours avant celle de gauche. En outre la libre circulation des jetons dans l'anneau garanti l'absence de blocage. L'intérêt de cette architecture est double :

- le contrôle distribué permet une excellente modularité et extensibilité ;
- les données sont immobiles, offrant un potentiel de faibles latence et consommation.

On peut cependant lui faire deux reproches :

- l'architecture présentée exécute les opérations sur les canaux dans une stricte cas Une possibilité de parallélisme est proposée en optimisant le protocole. Mais cette optimisation se limite à autoriser le passage du jeton PT (ou GT) juste après le début de l'opération d'empilement (ou dépilement) des données, permettant de compléter les protocoles sur les canaux LC et PC (ou GC) de manière concurrente.

- la communication avec l'environnement se fait au moyen de bus partagés trois états qui se révèlent pénalisants en consommation lorsque le nombre de cellules augmente (Cf. §2.4.2.1). Le buffer de ces bus, qui pilote les cellules, doit être redimensionné pour chaque taille de FIFO.

Nous proposons deux améliorations simples à ces limites : une modélisation en CHP par machines à états qui permet de paralléliser les opérations en garantissant un comportement toujours correct ; et l'utilisation d'arbres de multiplexage un vers deux sans contrôle pour remplacer la ligne de bus partagé. Les détails de conception au niveau portes logiques des éléments de cette FIFO sont donnés en Annexe C.

6.2.1.3. Première amélioration : optimisation de la séquence d'opérations

Les Figure 6-18 et Figure 6-19 présentent la modélisation CHP, sous forme de machine à états synthétisable, respectivement pour la cellule Starter et pour une cellule FC de la FIFO circulaire asynchrone.

```
process Starter
port (        -- channels for token passing
              RC : in bd passive SR
              LC : out bd active SR
              -- channel for fsm state update
              CS : in bd passive SR ;
              NS : out bd active SR
      )

begin
              [  [CS?cs ;
                 @ [        cs = 0 => LC! , NS!0 ; break
                            cs = 1 => LC! , NS!1 ; break
                            cs = 2 => RC? ; LC! , NS!1 ; break
                    ]
                 ]; loop
              ]
end;


process Starter_ctrl
port (        NS : in di passive dr ;
              CS : out di active dr
      )
begin
              [  [  CS!0 ;
                    NS?ns ;
                    @ [  ns = 0 => CS!1 ; break
                         ns = 1 => CS!2 ; break
                       ]
                 ] ; loop
              ]
end;
end process Starter_ctrl;
```

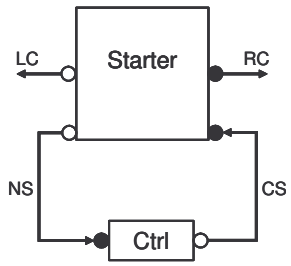


Figure 6-18 : modélisation CHP de la cellule Starter

Le comportement de la cellule Starter reste identique aux précédentes explications. Simplement sa modélisation respecte ici la spécification DTL (*Data Transfert Level*) [DIN 03], qui fournit un ensemble de règles garantissant que le réseau de Pétri et les graphes de données du modèle décrit sont synthétisables vers des circuits asynchrones (Cf. §3.2.2). L'utilisation de machines à états permet de découpler complètement les opérations, car les opérateurs de parallélisme (,) et de séquentialité (;) imposent des synchronisations entre les instructions, et donc des dépendances.

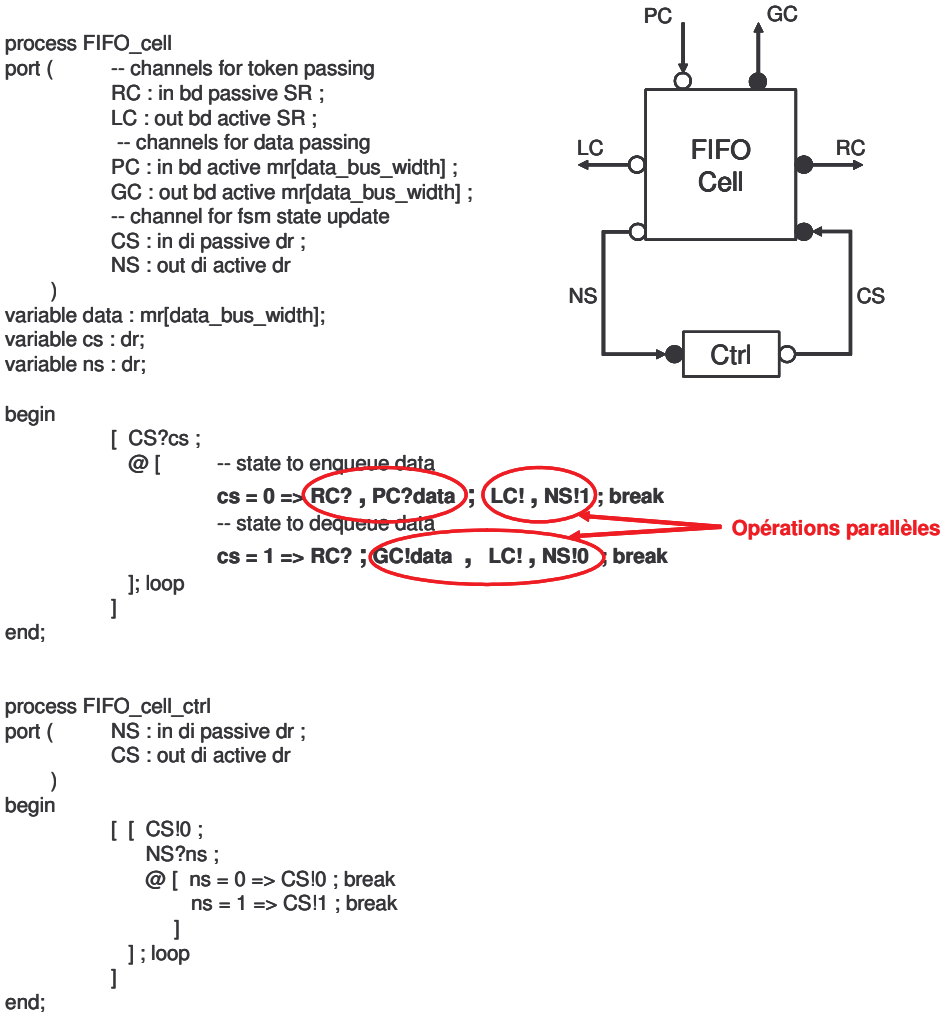


Figure 6-19 : modélisation CHP de la machine à états d'une cellule FC de la FIFO circulaire asynchrone, avec parallélisation des opérations.

Ainsi les opérations d'empilage et dépilage d'une cellule de la FIFO sont-elles complètement séparées dans la modélisation de la Figure 6-19. L'état courant *cs=0* (*cs* pour *current_state*) gère l'empilement de la donnée dans la FIFO et le passage du jeton PT, tandis que l'état courant *cs=1* gère le dépilement de la donnée et le jeton GT. Le contrôleur de cellule *FIFO_cell_ctrl* initie correctement la cellule (*CS!0*) puis se contente de mettre à jour les états de la machine. L'amélioration des opérations dans la FIFO vient de la parallélisation des tâches suivantes :

- Dans l'état courant *cs=0* :
  - les opérations de lecture du jeton PT (*RC?*) et d'empilement des données (lecture du canal *PC?data*) se font en parallèle ;
  - les opérations de passage du jeton PT (*LC!*) et de passage à l'état suivant (*NS!1*) se font en parallèle ;
- Dans l'état courant *cs=1* : une fois le jeton GT reçu (*RC?*), les opérations de libération des données (écriture dans le canal *GC!data*), de passage du jeton PT (*LC!*) et de passage à l'état suivant (*NS!1*) se font en parallèle.

Un strict parallélisme est cependant interdit pour une des opérations. Le protocole de lecture du canal *PC?data* ne peut se terminer avant le protocole de réception du jeton PT. En effet, la fin du protocole autorise le canal PC à initier une nouvelle transaction et à mettre à jour ses données, alors que le jeton PT, et donc l'autorisation d'écriture dans la cellule, n'a peut-être pas été reçu. La cellule risque alors de mémoriser dans ses latchs asynchrones une nouvelle donnée et de perdre la précédente.

La solution est apportée par les options de synthèse qu'offre l'outil TAST. La parallélisation des tâches nécessite un point de synchronisation (représenté par l'opérateur de séquentialité ";" dans la modélisation CHP). Dans la Figure 6-20, cette synchronisation entre les entrées parallèles RC et PC se fait au niveau de la dernière cellule de Müller C5, générant les signaux de requête sur les canaux LC et NS. Cette figure illustre la synthèse DTL de l'état courant *cs=0* pour une cellule de FIFO modélisée selon la précédente Figure 6-19. Cette figure est incomplète – mais néanmoins suffisante pour notre propos – car elle montre uniquement la synthèse de la partie contrôle (structures de contrôle et de choix, séquentialité, parallélisme) et non le chemin de données associé, en particulier la mémorisation de ces dernières dans les latchs asynchrones.

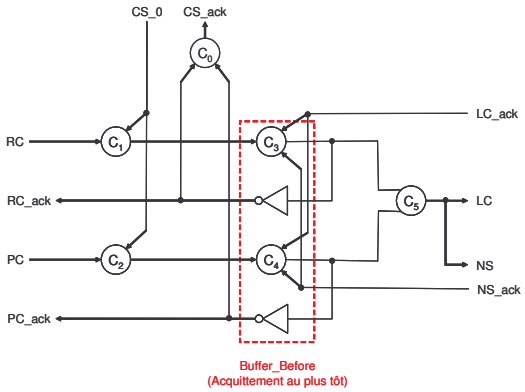


Figure 6-20 : le contrôle parallèle d'une cellule de FIFO avec acquittement au plus tôt

Les requêtes des canaux RC et PC sont synchronisés avec le signal *CS_0* de contrôle de l'état courant *cs=0* par les cellules de Müller C1 et C2 (principe de réalisation du rendez-vous entre les blocs *FIFO_cell* et *FIFO_cell_ctrl*). Les cellules de Müller suivantes C3 et C4 permettent de réaliser le rendez-vous (ou synchronisation) avec l'étage suivant de *FIFO_cell*. Ce rendez-vous permet de bufférisér les étages de fonctionnement.

Cette bufferisation se fait de manière à acquitter les canaux au plus tôt dans la séquence du circuit. C'est une option de synthèse appelée *Buffer_Before* pour signifier qu'on acquitte les canaux d'entrée avant d'émettre les requêtes en sortie. Elle a pour effet de découpler complètement les protocoles sur les canaux d'entrées PC et RC, les rendant parfaitement parallèles. Or ce n'est pas ce qui est souhaité ici. Il est donc possible de modifier les choix de synthèse de manière à synchroniser les acquittements des canaux RC et PC. La Figure 6-21 illustre la même séquence de code (synthèse DTL de l'état courant *cs=0)* que pour la Figure 6-20, mais cette fois avec le choix de réaliser les acquittements au plus tard (*Buffer_After*). Ce choix de synthèse permet de synchroniser les requêtes RC et PC avant de les acquitter (cellule C3), garantissant la correction du comportement.

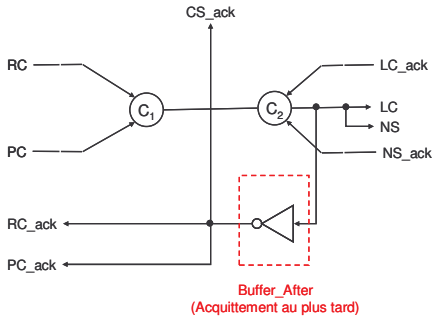


Figure 6-21 : le contrôle parallèle d'une cellule de FIFO avec acquittement au plus tard

6.2.1.4. Deuxième amélioration : suppression du bus partagé

Présentation

Pour remplacer les lignes longues avec des charges élevées du bus trois états, nous utilisons deux composants appelés *One-to-Two Sequential switch* (OTS) et *Two-to-One Sequential switch* (TOS). Ces composants sont assemblés en arbres binaires sur le principe de la Figure 6-22 qui présente une FIFO asynchrone circulaire. Chaque composant OTS est un démultiplexeur de un vers deux, sans contrôle. A l'initialisation, il aiguille la première communication à droite (sortie marquée par un jeton sur la Figure 6-22), puis bascule ensuite automatiquement d'une sortie sur l'autre à chaque communication. Le composant TOS fournit la fonctionnalité réciproque de multiplexeur deux canaux vers un, sans contrôle, en commençant de même toujours par la droite.

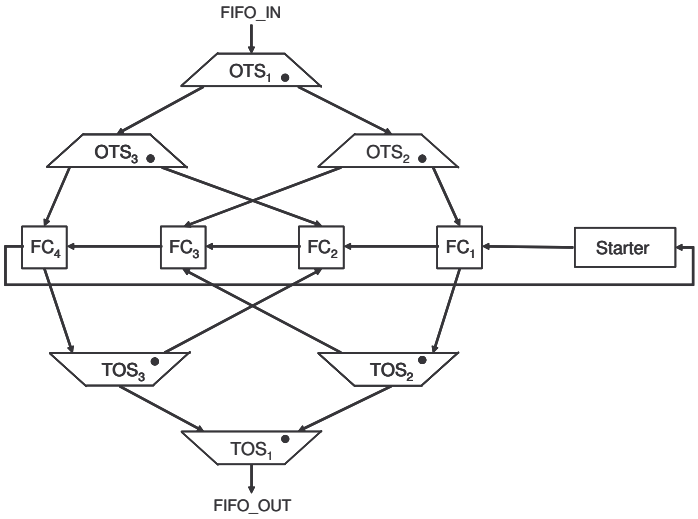


Figure 6-22 : connexion aux cellules de la FIFO par des arbres binaires de multiplexeurs/démultiplexeurs

Cette architecture toute simple présente les intérêts suivants :

- la séquence reste garantie correcte et sans blocage : l'ordre d'arrivée des données sur les cellules suit exactement celui du jeton PT. Il en va de même pour l'ordre de multiplexage des données par le réseau de composants TOS qui suit exactement l'ordre de circulation du jeton GT. Cela autorise, si on le souhaite, la complétion

parallèle vraie des canaux PC et RC avec synthèse de l'acquittement au plus tôt (Cf. §6.2.1.3).

- Le contrôle distribué et les composants parfaitement identiques permettent une excellente modularité et extensibilité.

A noter que, du fait de la commutation systématique des composants OTS et TOS, cette solution ne peut être utilisée que pour un nombre de cellules de FIFO multiple d'une puissance de deux pour que la séquence reste fonctionnellement correcte. Nous avons bien sûr la possibilité d'utiliser des sélecteurs de 1 vers 3 et 3 vers 1 fonctionnant sur le même principe et combinables avec les précédents sélecteurs 1 vers 2 ou 2 vers 1. Des FIFOs de profondeur multiple de trois peuvent ainsi être également pilotées par des arbres de multiplexeurs/démultiplexeurs.

Comparaison de performances

Nous envisageons l'utilisation de ces arbres de multiplexeurs/démultiplexeurs pour leur facilité d'extensibilité et de conception en regard d'un bus distribué. La consommation de ces arbres paraît également plus avantageuse lorsque les cellules de FIFO sont nombreuses, c'est-à-dire lorsque la charge à piloter est élevée. En revanche, la latence va être proportionnelle à la profondeur logique de l'arbre, alors que la porte qui pilote le bus sera dimensionnée pour répondre en un temps constant. Une comparaison des coûts doit être menée de manière détaillée pour mesurer l'intérêt qu'apporte l'arbre, dans le compromis entre facilité de conception et coûts. Une étude préliminaire du premier ordre peut cependant permettre une comparaison a priori entre les deux implémentations, sous certaines hypothèses simplificatrices :

- La capacité d'entrée C_{cell_in} d'une cellule de FIFO est équivalente à la capacité d'entrée d'un composant OTS C_{OTS_in} : les latchs pour mémoriser les données qui sont identiques, plus une cellule de Müller pour le signal de requête du canal PC. Soit C_{ref} cette capacité.
- La capacité d'entrée C_{NoC_in} du démultiplexeur du réseau de communication qui lit le contenu des cellules de la FIFO, est équivalente à la capacité d'entrée d'un composant TOS $C_{TOS\ in}$: les latchs pour mémoriser les données qui sont identiques, plus une cellule de Müller pour le signal de requête du canal GC. Les capacités d'entrée des composants TOS et OTS sont donc équivalentes, on conserve C_{ref}.
- Soit $T_{réf}$ le temps de réponse du bus pour charger la capacité d'entrée de toute la FIFO, quelle que soit sa profondeur N (nombre de cellules), la porte qui pilote le bus étant dimensionnée pour soutenir la charge. Le composant OTS est capable de charger une cellule de FIFO, mais aussi un autre composant OTS, sous le même temps de réponse : le latch et la cellule de Müller en sortie du composant OTS attaquent respectivement des composants identiques, que ce soit en entrée de la cellule de la FIFO ou en entrée d'un autre composant OTS. On a donc $T_{OTS\_FIFO} = T_{OTS\_OTS} = T_{réf}$.

Performances de l'arbre d'entrée de la FIFO

Energie

L'énergie nécessaire pour charger l'ensemble de N cellules de la FIFO est calculée à partir de la formule E = ½ CV². Soit L_D la profondeur logique de l'arbre de démultiplexeurs. On se place dans un cas défavorable pour l'arbre en ne tenant pas compte de la longueur de fils :

Energie du bus partagé : $E_{bus\_in} = \frac{1}{2} V_{dd}^2 \sum_{i=1}^{N} C_{cell\_in} = \frac{1}{2} V_{dd}^2 . N . C_{réf}$

Energie de l'arbre de démultiplexeurs :

$$E_{arbre\_in} = \frac{1}{2} V_{dd}^2 (C_{cell\_in} + L_D . C_{OTS\_in}) = \frac{1}{2} V_{dd}^2 C_{réf} (1 + L_D)$$

En normalisant par ½ $C_{ref} V_{dd}^2$, cela revient à comparer les coûts en énergie relative illustrés par la Figure 6-23 :

Energie relative du bus partagé : $Er_{bus\_in} = N$

Energie relative de l'arbre de démultiplexeurs: $Er_{arbre\_in} = 1 + L_D$ **avec** $L_D = \log_2 N$

Les arbres de composants OTS et TOS ne peuvent être utilisés que pour un nombre de cellules de FIFO multiple d'une puissance deux, c'est pourquoi les points calculés pour l'arbre binaire sur la Figure 6-23 ne sont pas reliés. Cependant d'autres tailles de FIFO peuvent être atteintes avec des sélecteurs 1 vers 3 et 3 vers 1 : FIFOs de taille 3, 6, 9… et cela suivant la courbe non tracée reliant les points calculés pour l'arbre binaire. Ces points n'ont pas été représentés pour ne pas surcharger les graphiques.

Latence

Le délai du bus centralisé correspond à $T_{réf}$. Le délai de l'arbre dépend de sa profondeur, chaque composant OTS devant charger un autre composant OTS sauf le dernier qui charge une cellule de FIFO, de la manière suivante :

Délai de l'arbre de démultiplexeurs: $T_{arbre\_in} = (L_D - 1) T_{OTS\_OTS} + T_{OTS\_FIFO} = L_D . T_{réf}$

En normalisant par $T_{réf}$, cela revient à comparer les coûts en énergie relative donnés illustrés par la Figure 6-23 :

Délai relatif du bus partagé : $Tr_{bus\_in} = 1$

Délai relatif de l'arbre de démultiplexeurs : $Tr_{arbre\_in} = L_D$

Tableaux comparatifs des coûts

La Figure 6-23 présente la comparaison des coûts en énergie et en latence comparés entre un arbre binaire de démultiplexeurs et un bus partagé pilotant l'entrée de la FIFO circulaire asynchrone.

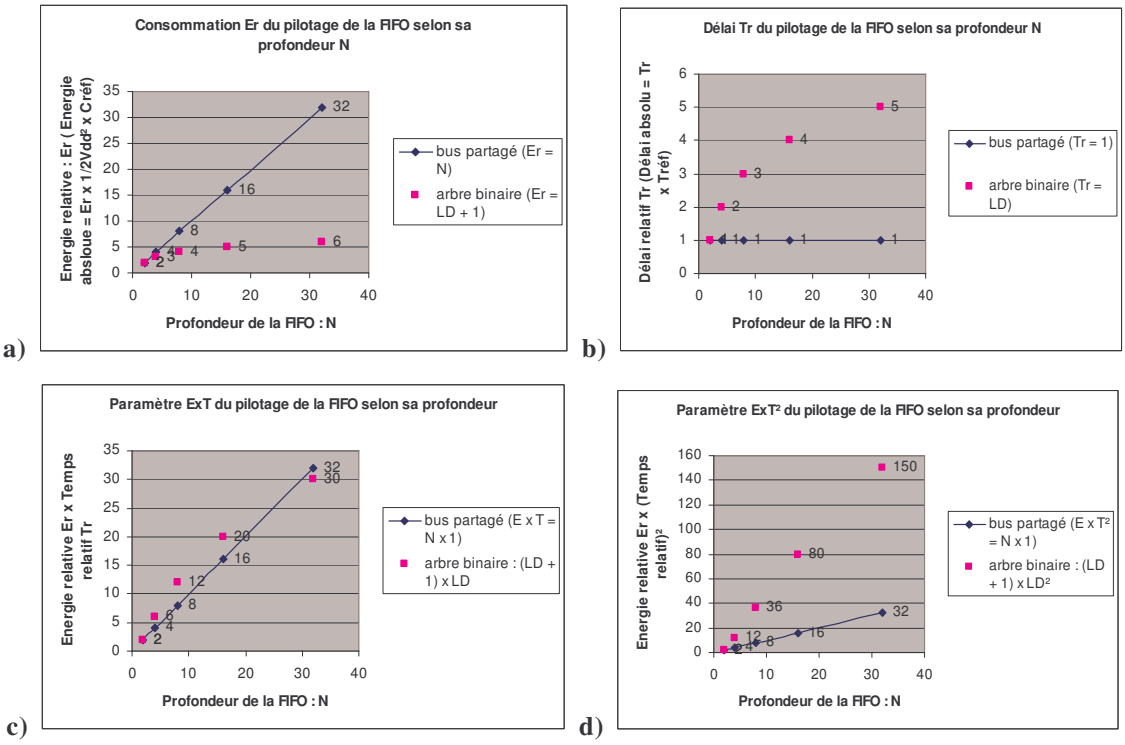
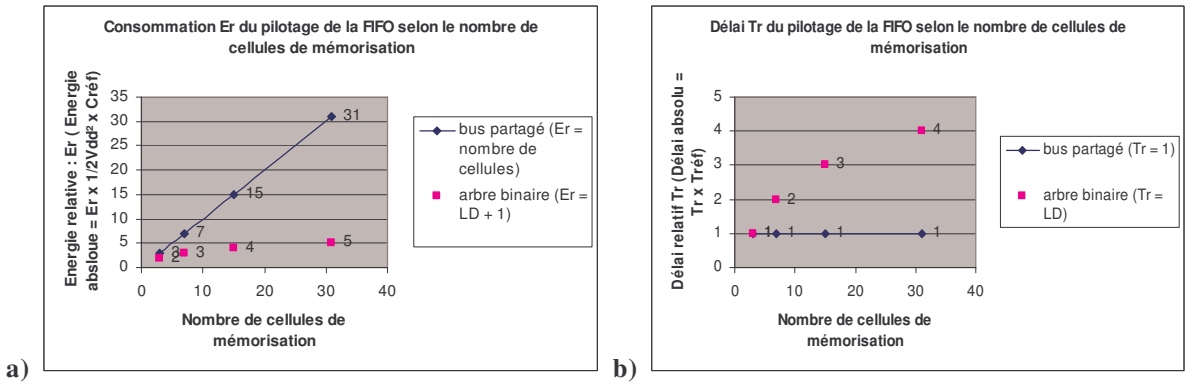
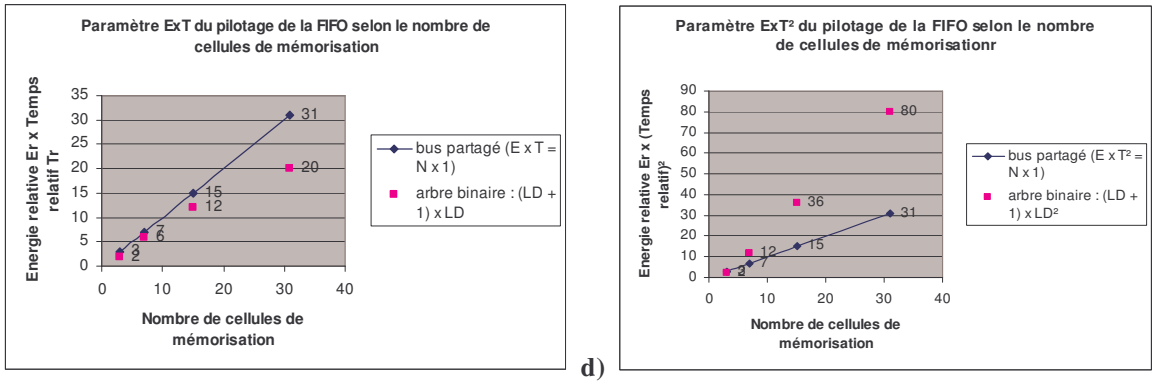


Figure 6-23 : comparaison de coûts entre un arbre binaire de démultiplexeurs et un bus partagé pilotant l'entrée de la FIFO circulaire asynchrone, en fonction de la profondeur de la FIFO : a) consommation d'énergie, b) délai, c) paramètre ET (énergie x délai), d) paramètre (énergie x délai²)

Ces calculs de première approximation confirment l'avantage énergétique de l'arbre de démultiplexage (binaire ou composé) mais aussi sa latence très élevée. Ce mauvais résultat doit cependant être compensé par le fait que les étages de l'arbre sont utilisés comme un pipeline pour compenser les différences de vitesses de fonctionnement entre le périphérique synchrone et le réseau asynchrone. On peut considérer qu'un arbre binaire de profondeur logique $L_D = 2$ rajoute trois cellules en entrée à la profondeur de la FIFO, initialement de quatre. En terme de latence, cela ne change rien à la comparaison, mais il est possible de mesurer la consommation à nombre égal de cellules de mémorisation. C'est-à-dire les cellules de la FIFO plus les sélecteurs des arbres binaires d'un côté et de l'autre côté un nombre équivalent de cellules de mémorisation par la seule profondeur de la FIFO. On obtient alors les résultats de la Figure 6-24, où le coût de l'arbre de démultiplexage s'avère compétitif avec l'arbre binaire, sans compter le gain en facilité de conception et d'extensibilité.





c) d)

Figure 6-24 : comparaison de coûts entre un arbre binaire de démultiplexeurs et un bus partagé pilotant l'entrée de la FIFO circulaire asynchrone, à nombre égal de cellules de mémorisation : a) consommation d'énergie, b) délai, c) paramètre ET (énergie x délai), d) paramètre (énergie x délai²)

Performances de l'arbre de sortie de la FIFO

Energie du bus partagé de sortie

Le coût de pilotage par le bus partagé va dépendre de la capacité de la longueur de fil plus la capacité d'entrée C_{NoC_in} du démultiplexeur du réseau de communication. On peut considérer que cette capacité de longueur de fil peut être ramenée à une capacité de longueur unitaire de bus multipliée par le nombre de cellules N de la FIFO, car la capacité totale du fil est égale à la somme des capacités de chaque segment de fil. On obtient :

Energie du bus partagé : $$E_{bus\_out} = \frac{1}{2} V_{dd}^2 . (C_{NoC\_in} + C_{bus\_out}) = \frac{1}{2} V_{dd}^2 . (C_{réf} + N . C_{bus\_out\_unit})$$

On peut également poser que cette capacité de longueur unitaire de bus est un multiple k facteur de la capacité de référence $C_{réf}$. En énergie relative on obtient alors :

Energie relative du bus partagé : $$Er_{bus\_out} = 1 + N.k$$

Energie de l'arbre des multiplexeurs de sortie

La consommation en énergie de l'arbre dépend de sa profondeur, chaque composant TOS devant charger un autre composant TOS sauf le dernier qui charge la capacité d'entrée C_{NoC_in} du démultiplexeur du réseau de communication, de la manière suivante.

Energie de l'arbre de multiplexeurs :

$$E_{arbre\_out} = \frac{1}{2} V_{dd}^2 (C_{NoC\_in} + (L_D - 1) . C_{TOS\_in}) = \frac{1}{2} V_{dd}^2 . C_{réf} . L_D$$

En énergie relative on obtient alors :

Energie relative de l'arbre de multiplexeurs : $$Er_{arbre\_out} = L_D$$

Latence

La comparaison pour les délais est identique à celle de l'arbre binaire et du bus partagé d'entrée de la FIFO, sauf que cette fois-ci ce sont les portes de sortie des cellules de la FIFO qu'il faut redimensionner pour permettre de conserver un délai de pilotage par le bus partagé constant quelle que soit la charge, proportionnelle au nombre de cellules N, de celui-ci.

Comparaison de coûts

La comparaison des coûts pour le pilotage de la sortie de la FIFO est similaire à celle réalisée pour le pilotage de l'entrée de la FIFO. Simplement l'intérêt d'utiliser un arbre va dépendre du rapport k entre la capacité de longueur unitaire de bus et la capacité d'entrée des composants TOS, qui revient à la capacité de référence $C_{réf}$. Le Tableau 6-1 montre que si ce rapport k est supérieur, en première approximation, à 0,2 alors l'avantage ira vers l'arbre de multiplexage. Ce qui revient à dire plus simplement et intuitivement que plus la capacité de longueur unitaire de fil est grande devant la capacité d'entrée des portes, plus l'utilisation de l'arbre est avantageuse sur le bus partagé.

Consommation Er du pilotage en sortie de la FIFO selon sa profondeur N et en fonction du rapport k

	k=0,1	k=0,2	k=0,3	
Profondeur de la FIFO N	Consommation du bus partagé en fonction de k : Er = 1 + k.N			Consommation de l'arbre binaire
	Er = 1 + 0,1N	Er = 1 + 0,2N	Er = 1 + 0,3N	Er = LD
2	1,2	1,4	1,6	1
4	1,4	1,8	2,2	2
8	1,8	2,6	3,4	3
16	2,6	4,2	5,8	4
32	4,2	7,4	10,6	5

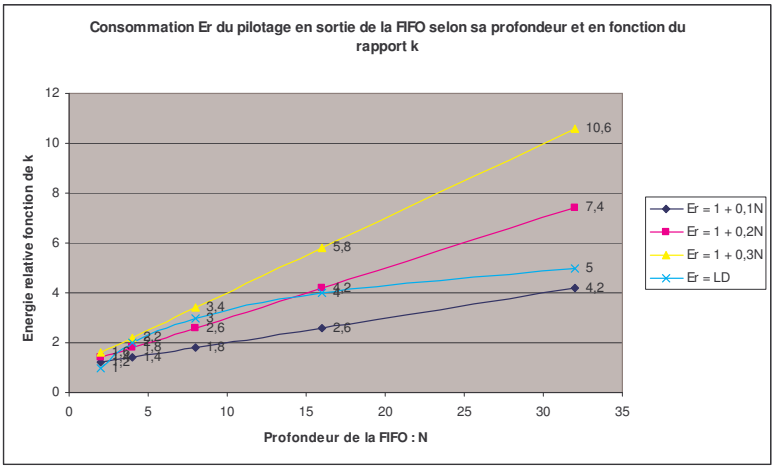


Tableau 6-1 : Comparaison du coût de consommation d'énergie en sortie de la FIFO en fonction du rapport k entre la capacité d'entrée des portes et la capacité de longueur de fil unitaire

Conclusion

Cette étude de première approximation permet d'estimer le coût comparatif des arbres de multiplexage/démultiplexage par rapport aux bus partagés en entrée et en sortie de la FIFO asynchrone circulaire. En première approximation cette solution d'arbres de pilotage, qui vise à réduire la consommation et de faciliter la conception, semble de coûts compétitifs par rapport au bus partagé, malgré un coût en latence élevé lorsque le nombre de cellules de la FIFO augmente. Il est bien sûr nécessaire de compléter cette étude en première approximation par des simulations électriques détaillées des performances des bus et des arbres de multiplexage/démultiplexage pour déterminer de manière précise les avantages et intérêts relatifs de chaque solution.

6.2.1.5. Conclusion

A partir de l'architecture de cellules de FIFO asynchrone circulaire existantes, nous proposons une modélisation en CHP qui améliore le parallélisme des opérations. Nous proposons également une construction modulaire de l'environnement de ces cellules de FIFO à base d'arbres binaires de multiplexeurs/démultiplexeurs pour améliorer la facilité de conception et la consommation pour des FIFOs de grande taille. Dans le deuxième paragraphe nous présentons le fonctionnement des FIFOs mixtes SA et AS et identifions les adaptations que nous envisageons de réaliser.

6.2.2. FIFO mixte synchrone/asynchrone

6.2.2.1. Rappel de la solution de Chelcea et Nowick

Avantages

A partir des cellules de FIFO asynchrones précédemment présentées au §6.2.1.1, Chelcea et Nowick [CHEL 01] construisent les différentes instances de FIFOs circulaires déjà énumérées et permettent d'interfacer des couples de composants périphériques de n'importe quel type de synchronisation :

- des FIFOs entièrement asynchrones ;
- des FIFOs mixtes asynchrone/synchrone ;
- des FIFOs double horloges ;

Ces architectures sont très modulaires, car la conception des interfaces de chaque côté de la FIFO est parfaitement découplée. Les interfaces du côté synchrone autorisent une transaction par cycle d'horloge en fonctionnement normal (FIFO ni vide ni pleine) du côté de l'interface synchrone, à condition que les vitesses de fonctionnement des deux domaines soient proches. Les données ne se déplacent pas. La FIFO offre ainsi un potentiel de faible consommation et de faible latence (Cf. §6.1.2.2). En outre ces architectures sont facilement extensibles grâce au contrôle distribué des cellules : le rajout de cellules pour augmenter la taille de la FIFO et l'augmentation des tailles des bus de données se font avec très peu de modifications de l'architecture. Se reporter à la Figure 6-14 qui illustre l'architecture des FIFOs mixtes a) asynchrone vers synchrone (FIFO AS), b) synchrone vers asynchrone (FIFO SA).

Inconvénients

La gestion de l'état vide ou plein de la FIFO pour le côté synchrone, complexifie de manière notable l'architecture de la FIFO. Le côté synchrone de la FIFO AS requiert un détecteur d'état vide de la FIFO avec une cellule d'avance (*lookahead empty detector*) et un contrôleur de lecture de la FIFO (*get controler*). Pour une FIFO SA de manière symétrique on obtient un détecteur d'état plein avec une cellule d'avance (*lookahead full detector*) et un contrôleur d'écriture de la FIFO (*put controler*). La détection des états vide et plein en avance

d'une cellule, fait qu'une FIFO de taille N est vue par les composants qui lui sont connectés comme une FIFO de taille N-1.

Cette détection pose également le problème d'une possibilité de blocage de la FIFO, dans le cas où la FIFO est vue vide par le récepteur alors qu'il reste une donnée dans une cellule. Pour y remédier, les concepteurs conçoivent l'implémentation suivante de la Figure 6-25 a). Ce circuit combine la détection de la FIFO réellement vide (signal *oe*) avec au plus une cellule écrite (signal *ne*). Les signaux *f_i* sont des signaux d'indication de remplissage issus de chaque cellule de la FIFO : s'il n'y a pas deux cellules consécutives pleines, la FIFO est considérée vide. Le circuit de détection d'état plein est présenté Figure 6-25 b). Les signaux e_*i* sont des signaux indiquant l'état vide, issus de chaque cellule de la FIFO : s'il n'y a pas deux cellules consécutives vides, la FIFO est considérée pleine.

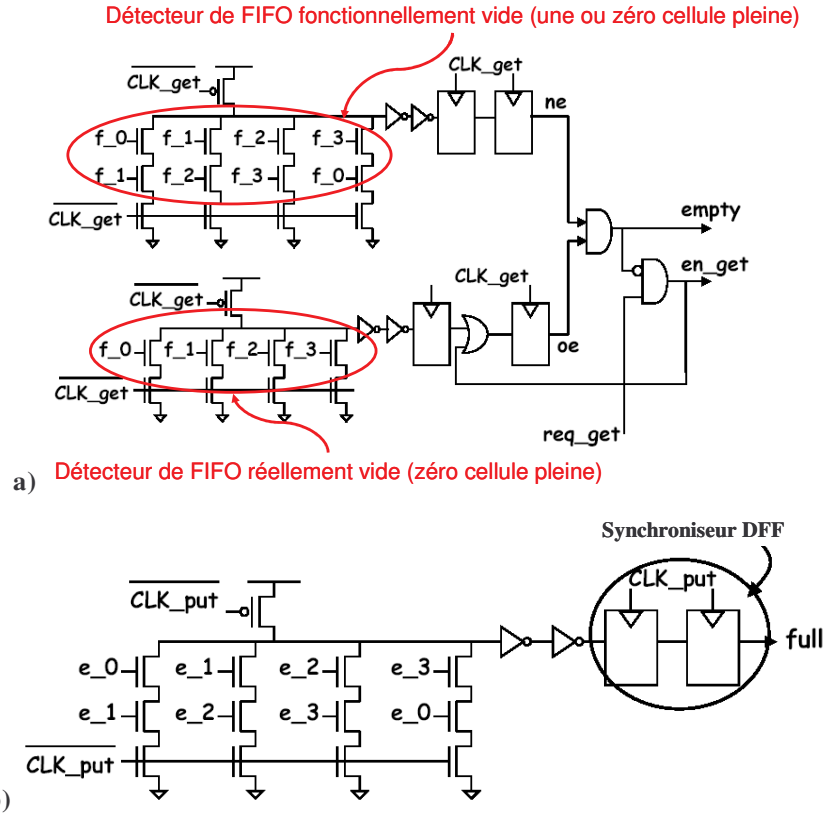


Figure 6-25 : circuits de détection a) de la FIFO vide b) de la FIFO pleine

Le synchroniseur DFF du bloc de détection d'une cellule réellement vide pose un problème de fiabilité [FRAN 04], car une cellule combinatoire OU, indispensable fonctionnellement, est insérée entre les bascules. En cas de conflit de délai horloge/donnée sur la première bascule, les contraintes de la fenêtre temporelle de la seconde bascule peuvent ne pas être respectées (Cf. §4.1.2.2). La solution consiste à insérer une deuxième bascule devant la porte OU [FRAN 04], de manière à obtenir un synchroniseur DFF complet. Mais alors il faut rajouter cette même bascule du côté du détecteur de cellule fonctionnellement vide pour que le circuit reste fonctionnel. Et si les vitesses de remplissage/famine de la FIFO veulent rester symétriques lorsque les émetteurs et récepteurs ont des vitesses de fonctionnement semblables, il faut aussi rajouter une bascule au circuit de détection d'état plein.

La latence est donc augmentée d'un cycle d'horloge, à ajouter aux deux cycles d'attente des bascules déjà présentes dans le détecteurs, plus une bascule supplémentaire dans la cellule de FIFO du côté synchrone.

Conclusion

Chelcea et Nowick proposent une FIFO mixte modulaire et de faible consommation, destinée à interfacer des périphériques synchrones et asynchrones avec une faible latence. La complexité des détecteurs d'état est justifiée par la garantie de soutenir une transaction de lecture ou d'écriture par cycle d'horloge du côté de l'interface synchrone, en fonctionnement normal (FIFO ni vide ni pleine) et à condition que les vitesses de fonctionnement des deux domaines soient proches (vitesses plésiochrones, Cf. §1.1). Or cette situation est impossible à garantir pour une FIFO mixte interfaçant un domaine d'horloge synchrone avec un domaine asynchrone auto séquencé. Ces domaines risquent au contraire de présenter des vitesses de fonctionnement très différentes. Dans une telle situation la FIFO sera souvent vide ou pleine, et la latence de trois cycles d'horloge (quatre si l'on veut corriger le défaut de fiabilité) sera alors pleinement ressentie.

6.2.2.2. FIFO circulaire asynchrone pour le système WUCS

Principe et justification

La FIFO précédemment étudiée est spécifiée dans l'objectif d'interfacer, sans pénalité de coût, des périphériques avec un protocole natif synchrone ne respectant pas le principe de synchronisation par communication des architectures GALS. Nous venons de voir que les performances de cette solution sont très pénalisées si, comme dans notre cas, nous interfaçons ce périphérique synchrone avec un réseau de communication sans horloge. En outre, le paradigme des architectures GALS (Cf. §1.2.2.1) postule que tous les composants se synchronisent par canaux de communication et doivent donc être capables de supporter un protocole de communication, qu'il soit à quatre ou deux phases et en données groupées ou insensible aux délais.

Nous décidons donc de ne pas utiliser ces interfaces synchrones de FIFO mais de conserver l'utilisation de la précédente FIFO entièrement asynchrone, même en environnement mixte, supprimant ainsi la gestion des états plein et vide de la FIFO. L'interface asynchrone n'a en effet pas besoin de ces signaux de contrôle supplémentaires. Par exemple si la FIFO est pleine, le signal d'acquittement *put_ack* est simplement maintenu invalide jusqu'à libération d'une place dans la FIFO. Ainsi la communication avec le périphérique synchrone est faite directement par protocole de communication quatre phases données groupées, le périphérique étant mis en attente de la complétion du protocole à chaque communication. La synchronisation est faite au niveau du périphérique synchrone grâce à des synchroniseurs DFF, sur les signaux d'acquittement pour une communication périphérique vers réseau, et sur les signaux de requête pour une communication duale.

Ainsi deux synchroniseurs sont utilisés, au lieu de trois synchroniseurs pour la FIFO mixte précédente. L'objection qui peut être faite est que cette FIFO mixte garantit par

construction que le risque de métastabilité n'existe que lorsqu'elle passe de l'état plein à non plein ou de l'état vide à non vide (au contraire de notre solution où le risque existe à chaque transition). D'une part ce risque se produit souvent lorsque les vitesses de fonctionnement sont différentes, d'autre part la fiabilité du synchroniseur testant les cellules réellement vides présente un défaut de fiabilité.

Illustration

Pour la collaboration avec STMicroelectronics sur le projet Prepor, tous les périphériques synchrones devaient offrir une interface intégrée de communication avec le réseau par protocole quatre phases données groupées. Il en est de même pour les composants synchrones du système WUCS. Ainsi l'adaptation du protocole du processeur MIPS synchrone est réalisée de la manière suivante, illustrée par la Figure 6-26.

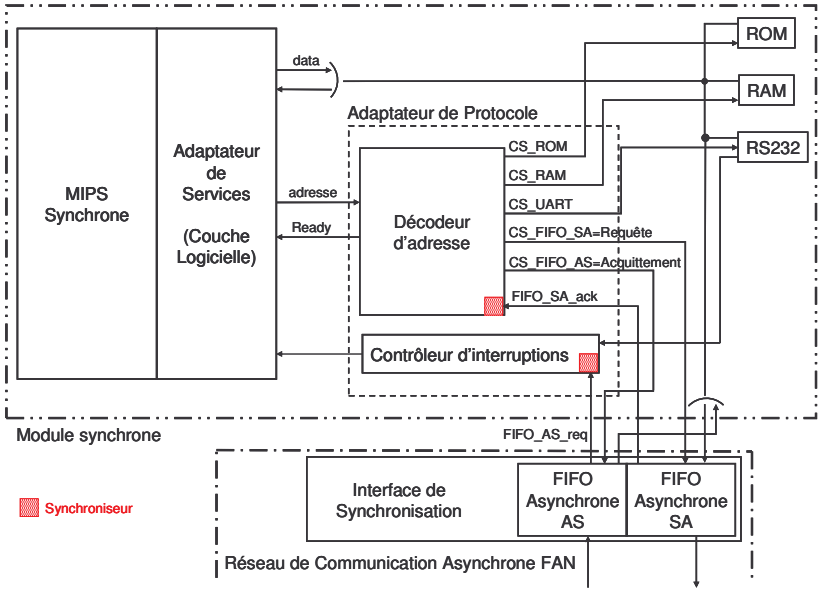


Figure 6-26 : Interfaces de Communication et de Synchronisation entre le réseau sans horloge FAN et le processeur synchrone MIPS

L'adaptation des protocoles de Service offerts par le réseau FAN est réalisée de manière logicielle par le processeur. Cette interface logicielle code (en émission) ou décode (en réception) sur le bus *data* les données en identifiant :

- trois champs qui représentent la séparation des futurs canaux *Target Address*, P*acket Control* et *Packet Data* du réseau (Cf. §5.3.2) ;
- Au sein du canal P*acket Control*, les informations de priorité ;
- Au sein du canal *Target Address*, l'adresse du destinataire ;
- Au sein du canal P*acket Data*, les données et adresses nécessaires au périphérique destinataire pour exécuter sa tâche (en particulier l'adresse de retour éventuellement indirect du service Réponse Indirecte, cf. §5.3.2.2)

Le décodeur d'adresses et le contrôleur d'interruption réalisent l'adaptation de protocole. Lors d'une émission, l'adresse du réseau FAN est envoyée par le MIPS au décodeur, qui envoie une requête respectant le protocole quatre phases grâce au signal *cs_FIFO_SA* (*Chip Select FIFO Synchronous to Asynchronous*). Le MIPS est mis en attente de l'acquittement par le signal *Ready*. Lors d'une communication initiée par le réseau, la FIFO AS envoie sa requête au contrôleur d'interruption. Le décodeur d'adresses transforme l'adresse de sélection de la FIFO AS en signal d'acquittement.

6.2.2.3. Conclusion

Nous venons d'étudier la possibilité d'utiliser une FIFO mixte synchrone/asynchrone pour adapter les périphériques synchrones à un réseau de communication sans horloge, quelque soit leur technique de synchronisation, et qui garantisse un débit d'une donnée par cycle en fonctionnement normal. Nous sommes cependant revenus à une solution plus simple de FIFO entièrement asynchrone adaptée à des périphériques acceptant une synchronisation par protocole de communication. Cette solution s'avère tout aussi robuste et de performances équivalentes lorsque les vitesses de fonctionnement sont très différentes. Elle est en outre plus flexible et facile à mettre en œuvre. Combinée à un circuit synchroniseur robuste, elle fournit le module d'Interface de Synchronisation de nos réseaux de communication sans horloge.

6.3. Conclusion

Ce chapitre traite du problème de la synchronisation aux interfaces entre les domaines de fonctionnement synchrones et asynchrones. Les méthodes d'interruption d'horloge, y compris les solutions rendant le système déterministe, cherchent à prévenir la métastabilité pour fournir un circuit entièrement fiable. Le problème est que ses solutions restent complexes, sont difficilement réutilisables et ne sont pas forcément totalement fiables, à moins d'être très limitantes dans leurs applications possibles. Les méthodes utilisant des synchroniseurs, pour résoudre la métastabilité, plus des FIFOs pour adapter les vitesses de fonctionnement des domaines d'horloge différents, même si ces solutions n'empêchent pas le système d'être sensible aux erreurs, permettent de meilleures performances en adaptant convenablement la latence des FIFOs et des synchroniseurs. La robustesse du synchroniseur classique DFF reste encore suffisante pour la majorité des systèmes. L'utilisation de FIFOs permet d'absorber la latence du synchroniseur en fonctionnement normal.

En particulier les FIFOs asynchrones adaptent de manière élastique différents domaines de fonctionnement synchrones ou asynchrones. A partir d'une solution proposée dans [CHEL 01], nous explorons des améliorations possibles. Nous améliorons de manière automatique le degré de parallélisme entre les différentes opérations des cellules de la FIFO circulaire asynchrone, grâce aux possibilités de synthèse de l'outil TAST. Nous proposons une alternative au bus partagé pour connecter la FIFO avec son environnement, en vue d'améliorer la facilité de conception, l'extensibilité et la consommation : un arbre de multiplexeurs et de démultiplexeurs sans contrôle (routage systématique). Mais la mesure comparative précise des performances de ces deux possibilités doit encore être menée. Enfin, nous avons étudié en détail

la possibilité d'utiliser une FIFO mixte synchrone/asynchrone pour adapter les périphériques synchrones à un réseau de communication sans horloge, quelque soit leur technique de synchronisation, et qui garantisse un débit d'une donnée par cycle en fonctionnement normal. Nous sommes cependant revenus à une solution plus simple de FIFO entièrement asynchrone adaptée à des périphériques acceptant une synchronisation par protocole de communication. Cette solution s'avère tout aussi robuste et de performances équivalentes lorsque les vitesses de fonctionnement sont très différentes. Elle est en outre plus flexible et plus facile à mettre en œuvre.

Chapitre 7 : Les contraintes de réalisation du chemin de données : les lignes d'interconnexion, l'adaptation de protocoles.

Ce dernier chapitre n'apporte pas d'autre contribution que d'introduire l'importance et les enjeux d'une réalisation performante des lignes physiques du réseau sur silicium. Aujourd'hui les technologies submicroniques profondes de conception de circuits intégrés voient leurs performances et leurs coûts en énergie contraints non pas par les transistors mais bien par les lignes de connexion. L'interconnexion physique est aujourd'hui un défi majeur des systèmes sur silicium modernes et vraisemblablement la clé de la réussite de ceux de demain.

D'ailleurs, pour bien comprendre l'importance égale de l'étude de l'intégrité du signal avec l'analyse de délai, de surface ou de consommation d'énergie du système sur silicium, la Figure 7-1 [SIA 04] rappelle l'illustration d'un flot de conception moderne en mettant en relief les outils qui doivent maintenant tenir compte de paramètres physiques d'interconnexion, rompant les distinctions traditionnelles entre les différents niveaux de conception.

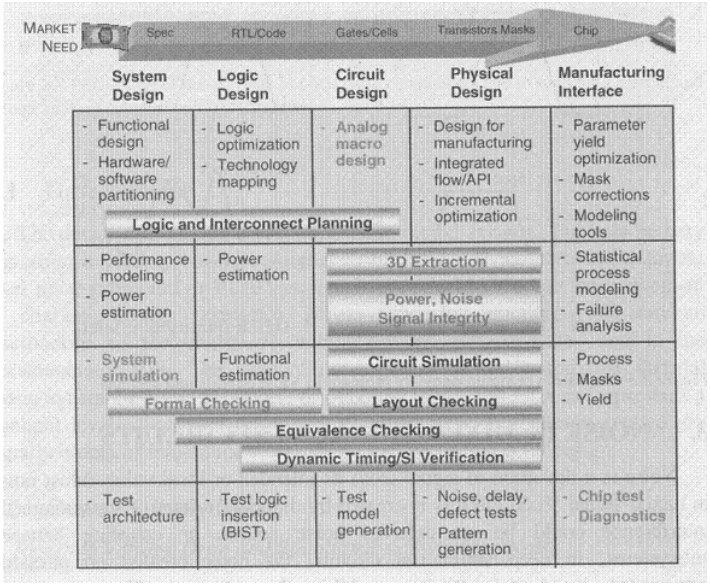


Figure 7-1 : flot de conception moderne montrant la nécessité de prendre en compte des paramètres physiques de plus en plus en amont [SIA 04]

L'émergence des techniques de conception centrées sur le réseau d'interconnexion (*Interconnect-centric Design Methodologies*), en fournissant des structures régulières de

réseaux sur silicium, permet d'accommoder au plus tôt les contraintes spécifiques à l'interconnexion et facilite ainsi la conception globale du système sur silicium.

Ce chapitre commence par rappeler au 7.1 les caractéristiques physiques et électriques des lignes, et le coût pour un système d'interconnexion des délais de propagation. La section suivante 7.2 traite de l'arrangement possible des lignes physiques et des protocoles de niveau signal pour réduire ces délais. La section 7.3 présente le chemin complet d'une ligne d'interconnexion du réseau FAN depuis le démultiplexeur du module d'émission jusqu'aux multiplexeur et bloc d'analyse des paquets (*Packet Analyzer*) du module de Réception. A la part de ce délai d'une communication dû au chemin des données, doit encore s'ajouter la part de délai dû au contrôle de la communication et celle dûe aux attentes actives de communication.

7.1. Caractéristiques physiques et électriques des lignes

Les paramètres qui permettent de mesurer l'intégrité d'un signal sont le délai d'interconnexion, le couplage électromagnétique des lignes, la diaphonie ou "*crosstalk*", les effets de transmission des lignes (adaptation d'impédance et réflexions), le couplage de substrat, l'intégrité de l'alimentation électrique du circuit et les effets du bruit sur le délai (*noise-on-delay effects*).

Il ne s'agit pas ici d'étudier tous ces effets ni de reprendre toute la théorie des lignes, qui d'ailleurs se voit réévaluée à chaque nouvelle génération de circuits. Le lecteur intéressé trouvera d'excellentes informations sur la modélisation des lignes, et sur la conception des couches physiques plus généralement, dans [SRI 94]. Il s'agit dans cette section de fournir des clefs d'accès aux dernières recherches sur les problèmes d'intégrité du signal, mais uniquement liés aux lignes d'interconnexion dans les réseaux de communication. Une équipe de conception désireuse de réaliser des systèmes sur silicium performants et fiables ne peut plus aujourd'hui contourner ces problèmes.

7.1.1. Le délai d'interconnexion

7.1.1.1. Modèles de délai d'interconnexion

Alors que les dimensions de l'interconnexion diminuent avec l'avancée des technologies, paradoxalement le délai d'interconnexion augmente considérablement. Son importance croissante nécessite des modèles de délai d'interconnexion de plus en plus précis. Le modèle de délai le plus célèbre est le modèle capacitance/résistance RC basé sur la constante de temps de Elmore [ELM 48]. Des contributions plus récentes sont venues enrichir et raffiner ce modèle en prenant notamment en compte les effets inductifs [ISM 00], de manière à obtenir des modèles RLC des lignes de transmission. En effet, les technologies de génération 0,18 µm et au delà nécessitent des modèles RLC d'ordre second ou plus [ISM 01], que les anciens modèles classiques RC ne peuvent atteindre. La Figure 7-2 présente le modèle par morceau RLC simplifié d'une ligne de longueur infinie.

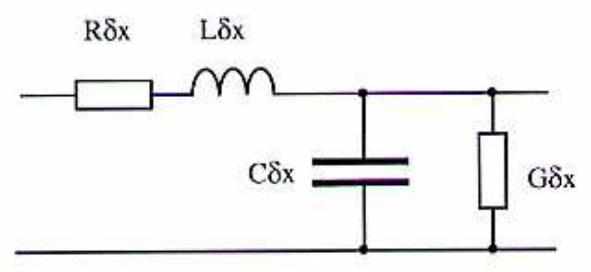


Figure 7-2 : modèle par morceau simplifié d'un ligne de longueur infinie

Les modèles de délai les plus largement utilisés actuellement sont basés sur des techniques d'approximation par ordre réduit comme l'évaluation d'ondes asymptotiques (*AWE* pour *Asymptotic Waveform Evaluation*), dont une revue est faite dans [SRI 94].

7.1.1.2. Le coût des délais d'interconnexion

Horowitz et al. présentent dans [HO 01] le problème du délai d'interconnexion dans les systèmes sur silicium modernes. La Figure 7-3 illustre le calcul du délai dans les portes et les interconnexions – pour une capacité en sortie de porte, appelée sortance, de FO4 (*FanOut of four*) – en fonction de l'avancée dans les technologies submicroniques profondes. Les lignes locales conservent une évolution de performance compatible avec celle des portes. En revanche les lignes longues d'interconnexion globale accusent un réel retard dans l'évolution de performance, cela de manière de plus en plus critique et malgré l'utilisation de répéteurs (Cf. §7.2.1).

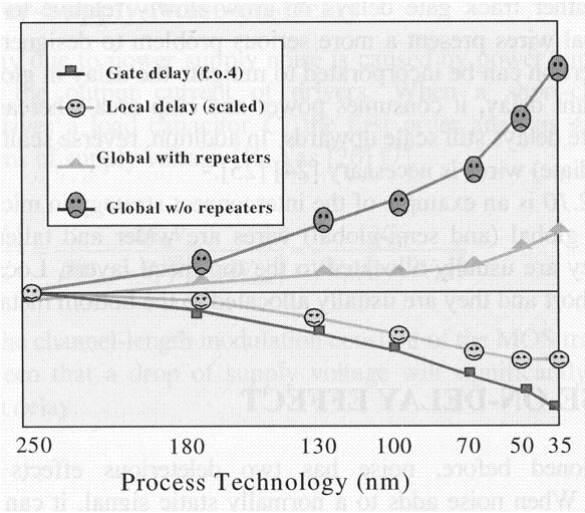


Figure 7-3 : calcul du délai dans les portes et les interconnexions avec l'avancée des technologies submicroniques profondes

7.1.2. Intégrité du signal

7.1.2.1. Illustration de l'intégrité du signal

Les complexes modèles de délai RLC permettent de référer et de calculer les délais de transfert d'un signal sur une ligne de communication. Dans un cas idéal, le Récepteur reçoit un signal identique en caractéristiques au signal émis à l'origine par l'Emetteur. En réalité, le signal reçu est souvent dégradé en raisons de perturbations telles que le couplage électromagnétique des lignes, la diaphonie ou *crosstalk*, les aléas de l'alimentation (*power supply glitches*) ou des phénomènes de dispersion du signal (perte d'énergie le long de la ligne). La figure suivante illustre ce problème d'intégrité du signal le long d'une ligne de communication.

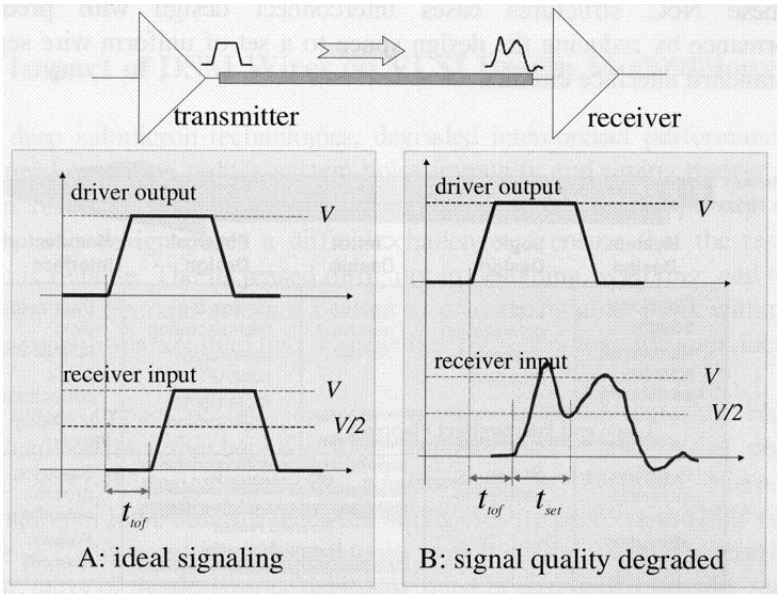


Figure 7-4 : Illustration du problème d'intégrité du signal le long d'une ligne de communication

Si le signal reçu est trop dégradé pour être correctement évalué par le Récepteur, une erreur dans le fonctionnement du circuit peut survenir. La section §7.2 étudie l'arrangement des lignes pour assurer une intégrité convenable du signal.

7.1.2.2. Perturbations dans les lignes

Le bruit électromagnétique a deux effets négatifs sur les performances d'un circuit. Un bruit ajouté sur une valeur statique de signal peut fausser l'information portée par cette valeur et provoquer une erreur de fonctionnement. Le bruit ajouté lorsqu'un signal commute provoque des erreurs sur les paramètres temporels du signal, comme du *jitter* ou du *skew* (Cf. §1.1). Ce paragraphe présente les sources de bruit significatives pour les lignes de communication.

Le couplage électromagnétique des lignes

Le couplage électromagnétique d'une ligne A avec une ligne B est une source majeure de perturbation d'un signal sur une ligne de communication. Ces phénomènes de couplage peuvent se produire de différentes manières présentées par la Figure 7-5.

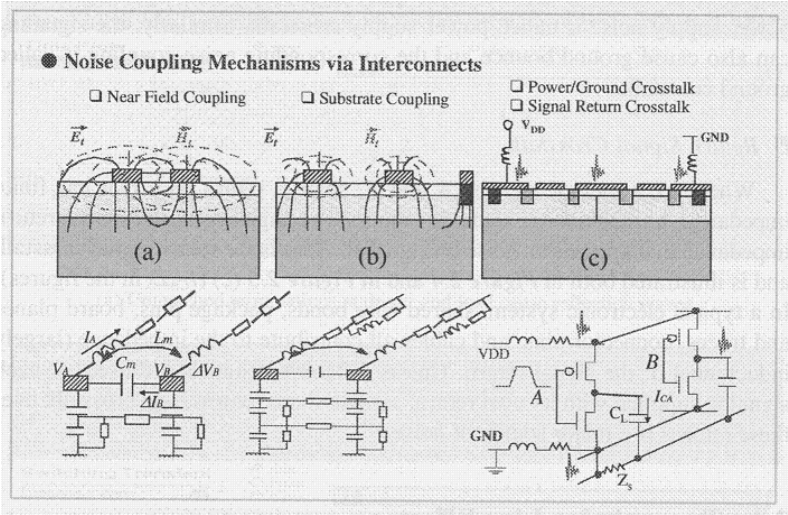


Figure 7-5 : mécanismes de couplage de bruit par les interconnexions

Le cas a) illustre le couplage de lignes proches par champ électromagnétique. Les couplages par champ électrique ou magnétique sont appelés respectivement couplage capacitif et couplage inductif. Le cas b) illustre le couplage indirect de lignes éloignées par le substrat. Ce dernier peut en effet servir de média de transport passif du champ électromagnétique d'une ligne à l'autre. Le cas c) illustre les phénomènes de couplage qui peuvent survenir par le transport de champs électromagnétiques à travers les couches de transport de l'alimentation.

Effet du couplage sur le délai des signaux

La Figure 7-6 illustre les effets du crosstalk sur le délai des signaux. Le délai de crosstalk peut provoquer le non respect des contraintes temporelles d'un circuit synchrone et par conséquent une erreur de fonctionnement, voir un phénomène de métastabilité si la transition du signal perturbé se produit dans la fenêtre de danger du circuit qui échantillonne ce signal (circuit synchroniseur) (Cf. §4.1.2.2). En revanche ce retard de signal dans un circuit insensible au délai n'aura aucune incidence.

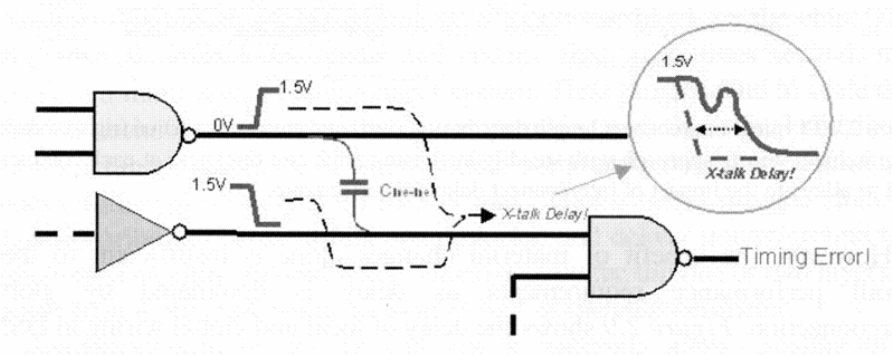


Figure 7-6 : exemple d'influence de couplage électromagnétique sur le délai d'un signal

7.1.3. Conclusion

Les paramètres essentiels de caractérisation d'une ligne d'interconnexion sont son modèle de délai RLC et le respect de l'intégrité du signal. La section suivante présente comment l'arrangement des lignes d'interconnexion au moyen de répéteurs asynchrones garantit une meilleure intégrité du signal.

7.2. Arrangements des lignes

Les lignes physiques d'interconnexion d'un circuit obéissent à des règles très strictes d'arrangement spatial. Ainsi leur espacement doit respecter les contraintes de couplage électromagnétique capacitif et inductif, comme nous venons de le voir. De plus, les lignes trop longues doivent être découpées en sections plus courtes de manière à pouvoir d'une part régénérer électriquement le signal et d'autre part à respecter les contraintes de temps de propagation vis-à-vis de l'horloge dans le cas d'un circuit synchrone (Cf. §1.1). De manière plus globale, à bande passante donnée, les concepteurs cherchent à obtenir les largeurs de bus physiques (nappes de fils d'interconnexion) les plus petites possibles afin de réduire les ressources en surface d'interconnexion. Malgré tout, l'évolution des systèmes sur silicium vers des architectures de plus en plus complexes, intégrant des composants toujours plus nombreux sur des surfaces de circuits toujours plus grandes, impose l'utilisation de lignes globales de plus en plus longues et de plus en plus nombreuses malgré les difficultés de délai que cela provoque (Cf. §7.1.1.2)

L'objectif de cette section est de réfléchir au potentiel de la technologie asynchrone pour améliorer l'arrangement de ces lignes d'interconnexion longues entre les composants du système sur silicium. Le premier paragraphe présente l'utilisation de répéteurs asynchrones pour améliorer les performances des lignes, le paragraphe suivant 7.2.2 réfléchit sur une possible amélioration du protocoles de niveau signal quatre phases vers deux phases, en terme de consommation d'énergie et de performance, et le dernier paragraphe discute de l'espacement des lignes du bus physique.

7.2.1. Les répéteurs électriques et logiques

Comme nous venons de le voir au §7.1.1.2, la nécessité de connecter les cœurs de composants par des lignes d'interconnexion globale de plus en plus longues et de plus en plus nombreuses pose un véritable problème dans les technologies submicroniques avancées. Un moyen de freiner la dégradation exponentielle des délais de propagation sur ces lignes longues est d'utiliser des répéteurs, comme le montre la Figure 7-3. La fonction première de ces répéteurs est de régénérer le niveau électrique du signal le long de la ligne, ce qui permet d'améliorer la latence. En effet la résistance de cette ligne, et par conséquent le temps de propagation du signal (par unité de longueur), augmente avec la longueur physique du fil. La Figure 7-7 présente quelques circuits de répétition du signal le long des lignes d'interconnexion.

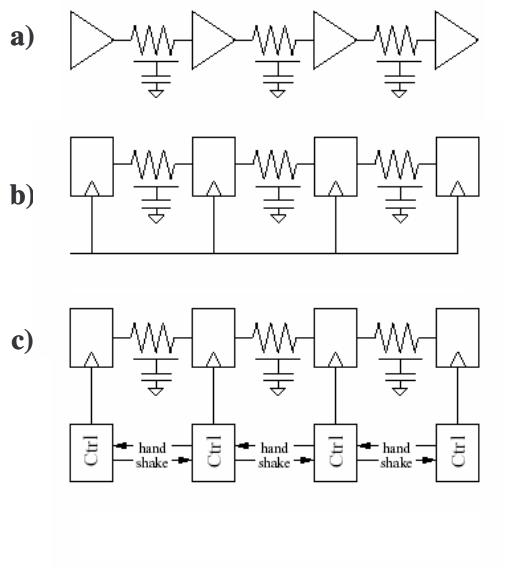


Figure 7-7 : comparaison de circuits pour améliorer la performance des lignes globales

a) Les répéteurs simples améliorent la latence de manière très significative (quelques dizaines de fois le délais d'une porte au lieu de quelque centaines [HO 01]). Ils ne permettent pas d'améliorer le débit en revanche.

b) Un pipeline de registres synchrones peut améliorer les performances de latence et débit mais au prix d'une fréquence d'horloge de fonctionnement très élevée. Sans compter les difficultés d'une synchronisation globale par horloge sur des lignes de longue distance : complexité d'intégrité du signal d'horloge (Cf. chapitre 1) et nécessité d'interfaces de synchronisation aux extrémités du pipeline (Cf. §4.2) au sein même du réseau d'interconnexion. En outre, cela réduit fortement le potentiel de modularité d'une architecture GALS.

c) Ho et al. viennent de montrer dans [HO 04] que des répéteurs asynchrones permettent d'améliorer à la fois les performances de latence et de débit, en outre pour un faible coût de logique de contrôle pour des répéteurs micropipeline (codage "données groupées").

Ces répéteurs asynchrones permettent de conserver les propriétés de localité des architectures GALS, s'affranchissent des problèmes de synchronisation au sein du réseau d'interconnexion et leur conception est indépendante de la technologie (principe de Design Reuse en introduction du chapitre 2). La Figure 7-8 présente la méthode de calcul pour déterminer la longueur optimale de ligne entre deux répéteurs asynchrones en fonction de la technologie. Les explications et la méthode détaillées sont données dans [HO 04]. La méthode cherche en priorité à améliorer la latence (couple l_{opt}, w_{opt}). A partir des modèles RC (jugés suffisants) du répéteur et de la ligne globale sont déterminés :

- La longueur l_{opt} optimale de séparation entre deux répéteurs ;
- La dimension optimale w_{opt} du buffer de sortie du répéteur ;
- le délai total de la ligne d'interconnexion ainsi obtenu.

On s'autorise ensuite à faire varier le couple longueur/dimension du buffer autour du point d'optimum (l_{opt}, w_{opt}) pour tenter d'améliorer le débit, sachant que ce dernier s'améliore en proportion du nombre de répéteurs.

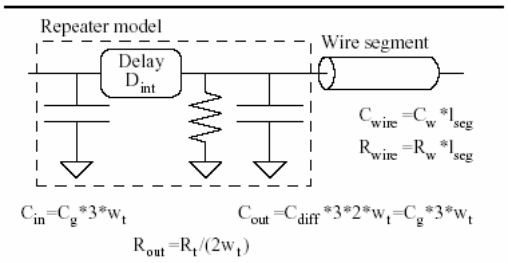


Figure 3. A linear model of the datapath latch

$$l_{opt} = 9.33\sqrt{\frac{R_t C_g}{R_w C_w}}$$

$$w_{opt} = \frac{1}{\sqrt{6}}\sqrt{\frac{R_t C_w}{R_w C_g}}$$

$$Delay = \left(\frac{total\ length}{segment\ length}\right)\cdot\left[R_{out}\left(C_{cout}+C_{in}+C_{wire}\right)+R_{wire}\left(\frac{C_{wire}}{2}+C_{in}\right)+D_{out}\right]$$

Figure 7-8 : méthode de calcul pour déterminer la longueur optimale de ligne entre eux répéteurs asynchrones

Les répéteurs présentés dans [HO 04] sont basés sur l'architecture micropipeline de FIFO minimale GasP [SUTH 01]. Cette architecture offre des performances très élevées, notamment de débit, mais nécessite l'insertion de délais dans la partie contrôle et d'hypothèses temporelles qui doivent être vérifiées assez tard dans le flot de conception, en phase de conception du *layout* du circuit. Des répéteurs réalisés de manière insensible au délai (QDI) s'avèrent en revanche plus robustes mais aussi plus coûteux en surface. Les lignes d'interconnexion globales longues de nos réseaux de communication sans horloge sont point à point avec un codage insensible au délai (généralement MR[4]) et les répéteurs devront donc être adaptés à un protocole QDI quatre phases ou deux phases selon la longueur de la ligne, comme nous en discutons à présent.

7.2.2. Adaptations de protocole quatre phases ←→ deux phases

Nous avons vu au §7.1.1.2 que les lignes globales du réseau d'interconnexion sont très pénalisantes pour les performances du système. En outre ces lignes longues vont consommer davantage d'énergie que les lignes locales courtes. L'activité sur ces lignes doit donc être déterminée pour être minimale.

L'architecture du réseau de communication sans horloge FAN spécifie un codage des données insensible au délai en protocole quatre phases. Le défaut principal du protocole quatre phases est la phase de remise à zéro (RZ pour *Return to Zero*) du canal. Le codage des données est choisi en MR[4] autant que possible plutôt qu'en MR[2], pour diviser par deux le nombre de transitions (Cf. §3.1.3.2). La phase de RZ nécessite donc, pour un signal sur quatre de chaque digit, une nouvelle transition de valeur qui doit parcourir toute la ligne globale d'interconnexion.

Pour remédier à cela, il est possible de choisir un protocole deux phases ou NRZ (*Non Return Zero*) qui dispense de la phase de RZ du canal, divisant ainsi par deux le nombre de

transitions par transaction et par digit, apportant un gain en consommation d'énergie et en vitesse. Ainsi Stevens présente dans [STEV 03] les coûts comparatifs de plusieurs protocoles de communication synchrones et asynchrones sur une ligne très longue en fonction du degré de pipeline de la ligne. Il montre que les protocoles asynchrones nécessitent plus de répéteurs que les protocoles synchrones pour soutenir les mêmes performances de débit et latence dans les conditions de fonctionnement pire cas mais obtiennent une consommation d'énergie équivalente à performance égale. La limite de cette comparaison vient des répéteurs asynchrones qu'il a conçu à partir de cellules standard de bibliothèque de composants destinés à des circuits synchrones. Or les cellules spécifiquement asynchrones, comme les portes de Müller, trouvent leurs performances très dégradées par une telle conception.

Son étude confirme cependant que le protocole deux phases est plus avantageux que le protocole quatre phases pour des lignes très longues, en termes à la fois de performances et de consommation d'énergie. Cependant cette comparaison[16] porte sur des protocoles données groupées et sur des lignes toujours très longues. Or l'avantage n'est pas systématique, car si le protocole quatre phases est a priori plus lent et consomme plus d'énergie, le protocole deux phases nécessite une logique de contrôle beaucoup plus importante que le protocole quatre phases car il doit détecter des transitions et non des niveaux logiques. Il faut donc déterminer à partir de quelle longueur de ligne d'interconnexion le gain en transitions du protocole deux phases compense son coût de logique supplémentaire des répéteurs, mais aussi des interfaces pour réaliser l'adaptation de protocole quatre phases/deux phases au niveau des modules Emetteur/Récepteur (E/R) présentés aux §5.3 et §4.3.3.3. Même si le choix d'un codage deux phases s'avère de plus en plus favorable car avec la réduction d'échelle, le coût en énergie de la communication devient de plus en plus élevé par rapport au coût respectif de la logique de contrôle. Ce codage étant pertinent sur des lignes longues, il est réalisé au plus tard pour éviter d'implémenter les étages de communication précédents (Routage et Arbitrage des modules E/R) en logique deux phases très complexe.

Une meilleure étude comparative pourra donc être envisagée pour nos protocoles insensibles au délai (codage 1 parmi N et de manière plus étendue N parmi M), en s'appuyant sur les travaux d'optimisation d'insertion de répéteurs présentés au §7.2.1 [HO 04]. D'une part pour déterminer, en fonction de la technologie, à partir de quelle longueur de ligne le protocole deux phases devient meilleur en terme de performances et de consommation d'énergie que le protocole quatre phases. D'autre part pour comparer la performance des protocoles synchrones et des protocoles insensibles au délai réalisés avec des composants en technologie asynchrone *full custom*.

16 En outre, l'étude est faite sur une seule technologie et la méthode de détermination des valeurs des paramètres de coûts et de performance se trouve être semi empirique et dépendante de la technologie. Elle n'est pas donc extensible à d'autres technologies ni même paramétrable en fonction de la longueur de ligne.

7.2.3. Espacement des lignes

Pour réduire l'effet des phénomènes de couplage par inductance essentiellement, mais aussi par capacitance, présentés au §7.1.2.2, les bus physiques de communication des technologies récentes imposent un certain arrangement des lignes [ZHE 99] :

- Des lignes de plus en plus larges en fonction de leur longueur pour mieux résister au bruit ;
- Un espacement des lignes contraint pour minimiser le couplage direct sans perdre trop d'espace ;
- L'insertion régulière de lignes "boucliers" (*shields*) entre les lignes pour minimiser le couplage indirect par le substrat. La Figure 7-7 présente le regroupement de lignes de données (généralement par paquets de taille inférieure à huit lignes) séparées par des boucliers connectés alternativement à chaque borne d'alimentation.

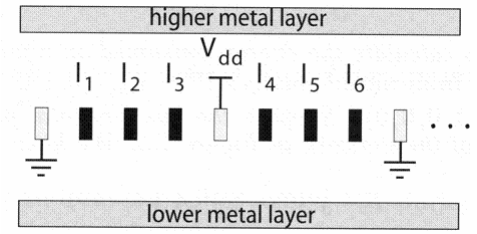


Figure 7-9 : arrangement des lignes de bus physique pour réduire le couplage

Cet arrangement des lignes pour la réduction de couplage est inutile en codage insensible au délai du type 1 parmi N. D'une part parce que les effets du *crosstalk* sur le délai des signaux dans un circuit insensible au délai (QDI) n'aura aucune incidence. D'autre part parce que le codage 1 parmi N (ou Multi-Rail MR) ne "génère" pas de couplage, puisqu'une seule ligne parmi quatre commute à chaque transaction. Un bus physique QDI s'affranchit donc des lignes de bouclier et autorise un resserrement des lignes de données.

Cette optimisation possible de l'espacement des lignes QDI en phase finale de conception du circuit permet donc de réduire l'occupation surfacique que nécessite un bus physique asynchrone QDI (deux fils par bit en codage MR[4] ou DR plus une ligne d'acquittement par canal).

7.3. Le chemin de données complet de la Ressource de Transport. Le coût d'une communication

Cette dernière section illustre comment les précédentes réflexions participent à la conception d'une ligne point à point du réseau de communication sans horloge FAN et de l'estimation du coût de la communication de périphérique à périphérique.

7.3.1. Le chemin de données complet sur une ligne E/R du réseau FAN

La Figure 7-10 présente le chemin de données complet d'une liaison point à point entre un module Emetteur et un module Récepteur du réseau FAN, représentant ainsi la ressource de Transport. Le chemin de données complet inclut :

- les Démultiplexeurs (*DEMUX* de la Figure 7-10) des canaux de contrôle et de données du composant Initiateur,
- l'éventuel Adaptateur de protocole quatre phases/ deux phases selon la longueur de la ligne (*2φ* de la Figure 7-10),
- les Répéteurs QDI deux phases ou quatre phases selon l'adaptation ou non en protocole deux phases (*R* de la Figure 7-10),
- l'éventuel Adaptateur de protocole deux phases/quatre phases complémentaire du précédent (*4φ* de la Figure 7-10),
- enfin au sein du composant Récepteur le bloc Multiplexeur pour le canal de données et le bloc Packet Analyzer pour le canal de contrôle.

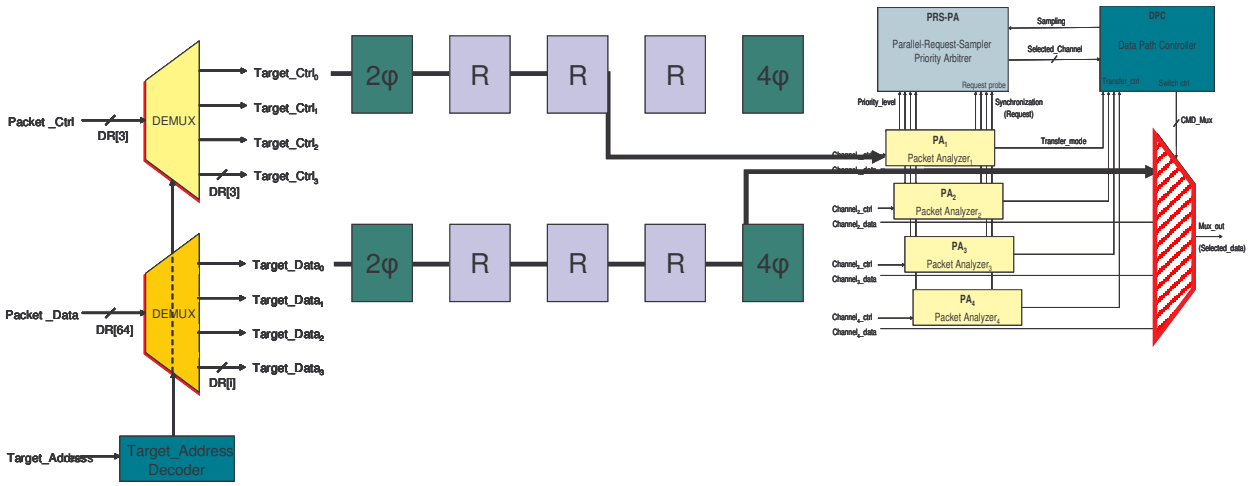


Figure 7-10 : le chemin de données complet sur une ligne E/R du réseau FAN

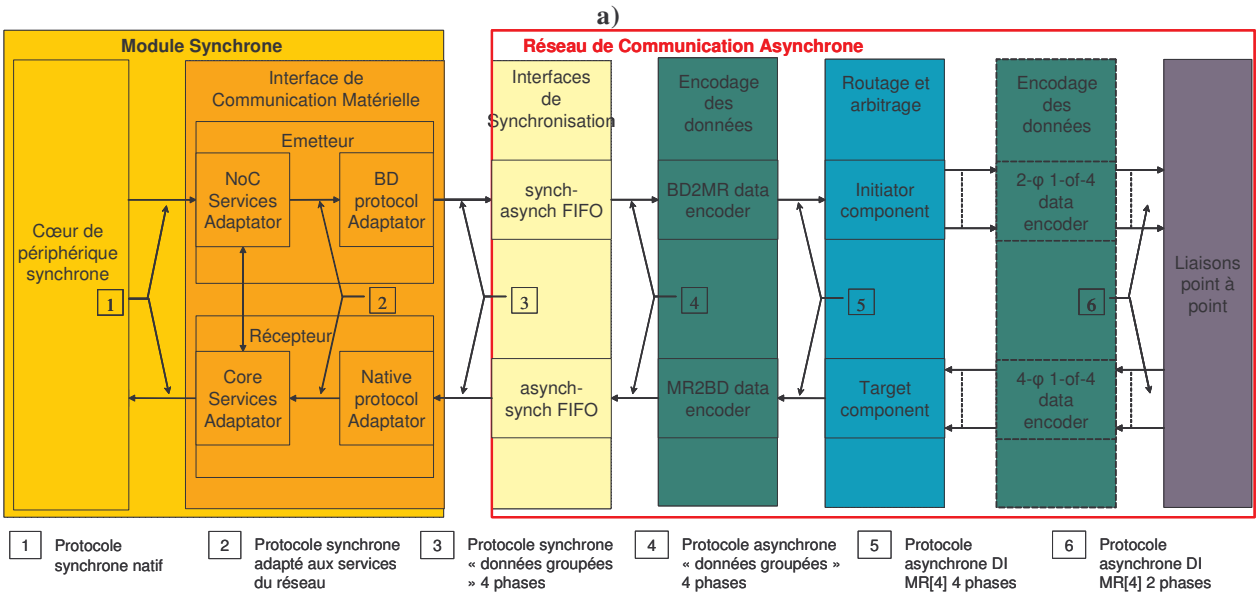
7.3.2. Le coût total de communication

La section §7.1 présente le modèle RC ou RLC de délai d'une ligne d'interconnexion qui intervient dans le calcul du délai total d'une transaction de communication sur le réseau. Mais il faut aussi tenir compte de tous les précédents étages d'adaptation de protocole de service ou de synchronisation (niveau signal), sur l'exemple de la Figure 4-31 du §4.3.3.3, ainsi que sur les latences de gestion des services eux-mêmes, à savoir le contrôle d'Arbitrage et de Routage au sein du réseau de communication.

La durée totale d'une communication à travers le réseau d'interconnexion peut alors être divisée en trois parties de la manière suivante :

1. Le Contrôle : c'est le délai nécessaire à l'exécution de toutes les instructions de contrôle des procédures de communication, sans les temps d'attente s'ils existent. Ce délai est la partie fixe du délai de communication car il est le même pour toutes les instances de communication d'un même protocole. Ce délai dépend seulement du type de contrôle et de son implémentation. Il ne dépend ni des données ni de l'environnement.
2. La Transmission de données : c'est la partie de la communication qui dépend de la taille et du format des données. Ce délai est réparti dans les multiplexeurs des différents modules du réseau ainsi que dans les modules de codage des données (mise au format multi-rail ou données groupées, adaptation de protocole deux phases ou quatre phases), les FIFOs et les fils physiques des lignes de communication.
3. L'Attente active : cette partie contient tous les délais de blocage du processus tout au long de la communication, comme les attentes de rendez-vous. Cette partie très variable dépend de l'exécution dynamique du système et rend difficile l'estimation précoce fiable du délai des communications.

La Figure 7-11 rappelle l'architecture détaillée des étages de communication du réseau FAN pour un module périphérique localement a) synchrone b) asynchrone.



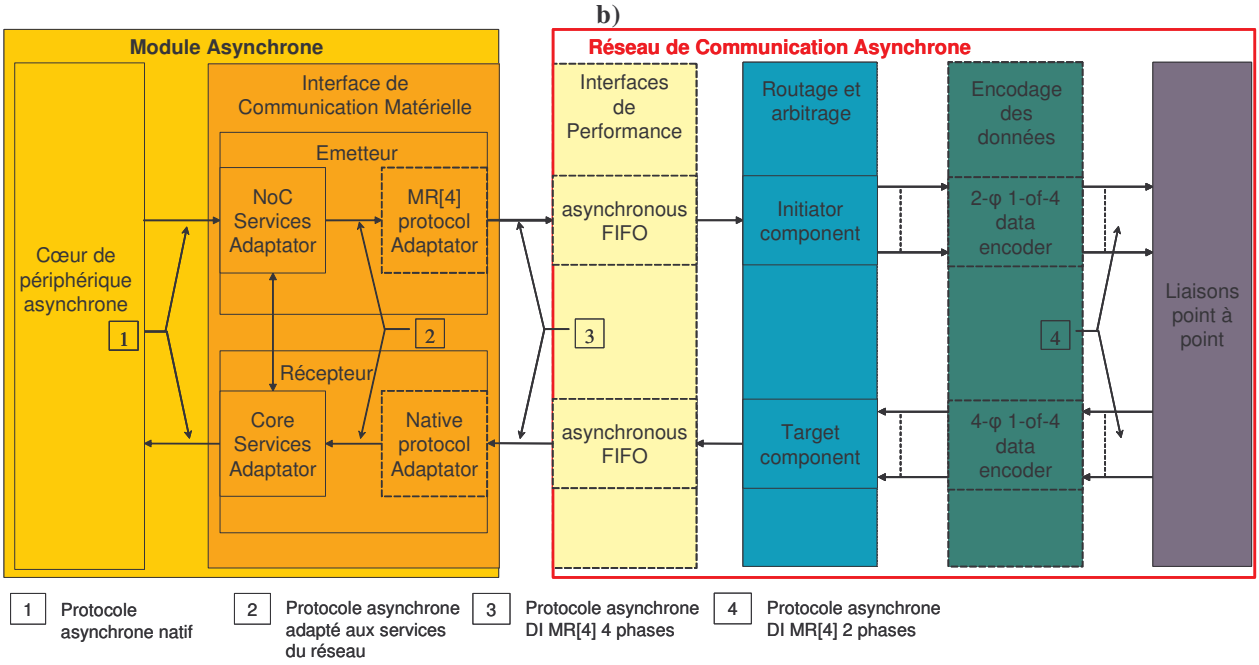


Figure 7-11 : architecture détaillée des étages de communication du réseau FAN pour un module périphérique localement a) synchrone b) asynchrone

Il est important de connaître pour une liaison point à point la part des coûts de Contrôle, de Transmission de Données et d'Attente active dans une communication. Ainsi dans le réseau FAN le coût total de la communication est réparti de la manière suivante.

7.3.2.1. Le Contrôle

Le réseau FAN a été conçu pour offrir un contrôle des procédures de communication simple mais également aisément extensible dans les services de transaction et d'arbitrage.

1. L'Interface de Communication du périphérique comprend deux blocs. Le bloc d'Adaptation des Services et un bloc de Contrôle qui met en forme les protocoles de service entre le réseau FAN et le périphérique. Sa machine à états dépend du degré de complexité des protocoles de service respectifs et représente le coût de Contrôle le plus élevé de la liaison point à point.
2. Le Routage et l'Arbitrage des modules E/R présentés au chapitre 5 ont été spécifiés pour être les plus simples et efficaces possibles en terme de latence de contrôle. Le coût du Contrôle vient donc essentiellement de l'Interface de Communication.

7.3.2.2. La Transmission de données

La Transmission de données va représenter le coût réel de communication, essentiellement dans les Interfaces de Communication et dans une moindre mesure dans les lignes globales longues d'interconnexion.

1. L'Interface de Communication du périphérique comprend deux blocs. Le bloc d'Adaptation de protocole est un bloc de Transmission de données qui n'est nécessaire que pour un périphérique synchrone. Il est en général très simple et de faible coût.
2. L'Interface de Synchronisation présentée au chapitre 6 est spécifiée de latence faible. Cependant elle exige une certaine profondeur de FIFO pour adapter de manière performante les périphériques de haute et très haute vitesse de fonctionnement. Elle présente un coût de lecture et d'écriture du côté synchrone relativement élevé par rapport au côté asynchrone et de manière plus générale par rapport aux autres composants de la Transmission de données.
3. Le chemin de données tel que présenté au §7.3.1 est dimensionné pour le plus faible coût possible de latence mais le §7.1.1.2 montre que les lignes globales longues d'interconnexion représentent un coût élevé dans une communication. Les étages d'Encodage des données (mise au format multi-rail ou données groupées, adaptation de protocole deux phases ou quatre phases) ont une part négligeable dans ce coût.

7.3.2.3. L'Attente active

Cette partie très variable dépend de l'exécution dynamique du système et rend difficile l'estimation précoce fiable du délai des communications.

1. L'Arbitre peut générer une Attente active sur une communication s'il n'élit pas cette communication dans l'arbitrage d'accès à une ressource ou dans la résolution d'une éventuelle métastabilité par les synchroniseurs asynchrones ;
2. L'Interface de Synchronisation peut générer une Attente active dans la gestion de Lecture/Ecriture dans la FIFO ;
3. Le réseau de communication sans horloge FAN présente une possibilité d'Attente active tout au long de la communication, du fait des attentes de rendez-vous de synchronisation.

7.4. Conclusion

Ce chapitre a présenté l'importance croissante du coût des lignes physiques dans le délai d'une communication. Ce délai de propagation des signaux est particulièrement critique pour les lignes d'interconnexion globales longues, de plus en plus nombreuses avec l'évolution des systèmes sur silicium vers des architectures de plus en plus complexes.

La technologie asynchrone présente un potentiel pour améliorer l'arrangement deces lignes. Un pipeline de répéteurs asynchrones permet d'améliorer à la fois les performances de latence et de débit des lignes. Ces répéteurs asynchrones permettent de conserver les propriétés de localité des architectures GALS, s'affranchissent des problèmes de synchronisation au sein du réseau d'interconnexion et leur conception est indépendante de la technologie.

Une étude comparative doit être envisagée pour nos protocoles insensibles au délai (codage 1 parmi N et de manière plus étendue N parmi M). D'une part pour déterminer, en fonction de la technologie, à partir de quelle longueur de ligne le protocole deux phases devient meilleur en terme de performances et de consommation d'énergie que le protocole quatre phases. D'autre part pour comparer la performance des protocoles synchrones et des protocoles insensibles au délai réalisés avec des composants en technologie asynchrone *full custom*.

L'arrangement des lignes pour la réduction de couplage est inutile en codage insensible au délai : une optimisation de l'espacement des lignes QDI est donc possible en resserrant les fils de données et en supprimant les lignes de boucliers.

La section 7.3 a présenté le chemin complet d'une ligne d'interconnexion du réseau FAN depuis le démultiplexeur du module d'émission jusqu'aux multiplexeur et bloc d'analyse des paquets (*Packet Analyzer*) du module de Réception. La Transmission de données va représenter le coût réel de communication, essentiellement dans les Interfaces de Communication et de Synchronisation, et dans une moindre mesure dans les lignes globales longues d'interconnexion.

Conclusion – Directions futures de Recherche

Conclusion

La conception de réseaux de communication intégrés est un problème complexe qui recouvre un large champ de compétences et de savoirs. Au cours de nos travaux, nous avons cherché à explorer et à évaluer le plus grand nombre possible de domaines de connaissances et de savoir-faire techniques nécessaires à la modélisation et la réalisation d'architectures de communication. Nos études visaient trois objectifs :

1. répondre aux problèmes de conception des systèmes sur silicium par le choix de d'architectures GALS ;
2. répondre aux exigences spécifiques des réseaux d'interconnexion grâce à la technologie asynchrone ;
3. permettre la synthèse automatique de structures non déterministes, d'arbitres et des réseaux de communication par l'outil TAST.

L'analyse des méthodes de synchronisation d'un système par horloge globale, ou par un réseau d'horloges connectées, avait pour objectif de nous familiariser avec les techniques de conception synchrone, en vue de maîtriser la compréhension et la réalisation de réseaux de communication synchrones.

L'étude détaillée comparative de nombreuses architectures de communication intégrées nous a permis de recenser les grandes familles de topologies régulières d'interconnexion, en identifiant pour chacune ses critères d'utilisation efficace. En particulier les atouts des architectures distribuées du type anneau, crossbar ou maille, à la fois pour être conçues en technologie asynchrone, et pour interconnecter des systèmes globalement asynchrones et localement synchrones.

Cette étude détaillée était en outre indispensable pour dégager, parmi les nombreux critères recensés de spécification d'un réseau de communication, trois paramètres qui nous paraissent essentiels pour concevoir des systèmes de communication performants : la flexibilité, les protocoles de synchronisation, ou de communication de niveau signal, et les protocoles de services. La conception asynchrone, absente de tout signal global, offre une modularité très élevée qui rend les réseaux d'interconnexion asynchrones très flexibles car fortement extensibles et facilement interfaçables avec des composants périphériques, qu'ils soient synchrones ou asynchrones. Les protocoles de communication de niveau signal implémentés en logique

insensible au délai, permettent de réaliser un réseau de communication sans horloge très robuste et d'un degré de fiabilité très élevé.

Pour les protocoles de niveau signal, nous avons comparé le coût d'une synchronisation par communication, aux interfaces entre les composants périphériques et le réseau d'interconnexion d'une architecture GALS, selon que ce réseau soit synchrone ou asynchrone. Nous comparons de même le coût de l'arbitrage en environnement asynchrone (requêtes mutuellement asynchrones), selon que l'arbitre soit synchrone ou asynchrone. Nous montrons que le réseau et l'arbitre asynchrone résolvent les problèmes de synchronisation de manière plus fiable, plus performante et moins coûteuse.

Cette synchronisation à l'interface entre deux domaines de fonctionnement de vitesses différentes, dont au moins un domaine est synchrone, est une source de non-déterminisme et de risque de métastabilité, donc d'erreur, pour l'ensemble de l'architecture GALS. Il en est de même pour l'arbitrage de signaux mutuellement asynchrones.

Pour améliorer la qualité de ces circuits générateurs d'un risque de métastabilité, nous proposons une classe d'arbitres asynchrones à échantillonnage parallèle parfaitement fiables, qui réalisent des fonctions de priorité indépendantes de l'ordre d'arrivée des requêtes. Ces arbitres offrent une excellente modularité, sont aisément extensibles et améliorent la latence de réponse aux requêtes.

Pour répondre au problème de la synchronisation à l'interface des domaines synchrones et asynchrones, nous proposons l'utilisation d'une FIFO circulaire asynchrone par passage de jeton. Ce circuit offre une très grande modularité et un degré élevé de parallélisme des opérations. Le choix d'un bus partagé pour connecter la FIFO avec son environnement offre un potentiel de faible latence à ce circuit, tandis que le choix d'un arbre de multiplexeurs et de démultiplexeurs sans contrôle (routage systématique) offre un potentiel de faible consommation et de modularité (facilité de conception et extensibilité) renforcé.

Pour réaliser une interface de synchronisation mixte synchrone/asynchrone complète, nous associons cette FIFO asynchrone à la robustesse du synchroniseur classique DFF, qui reste encore suffisante pour nos besoins, et d'ailleurs pour la majorité des systèmes. L'interface de synchronisation ainsi réalisée permet d'adapter, de manière élastique et performante, différents domaines de fonctionnement synchrones ou asynchrones, de vitesses relatives très différentes.

Ces composants d'arbitrage et de synchronisation s'intègrent dans une méthodologie de conception d'une architecture globalement asynchrone et localement synchrone utilisant un réseau de communication sans horloge modulaire, flexible (en terme de services, d'extensibilité et d'interfaçage), robuste et performant. Cette méthode repose sur l'utilisation de cinq composants modulables construits pour répondre chacun à des critères de spécification bien déterminés du réseau de communication : la synchronisation et la mise en forme des communications aux interfaces avec les périphériques, l'arbitrage et le routage aux noeuds du réseau, le transport physique des données au cœur du réseau.

Cette méthodologie de conception s'est attachée à formaliser la conception des réseaux de communication sans horloge, en particulier la modélisation du non-déterminisme. L'objectif est de permettre la génération automatique par l'outil TAST de structures non-déterministes.

L'ensemble de ce manuscrit a donc présenté un nombre certain d'avantages de la conception asynchrone pour la réalisation de réseaux sur silicium au sein d'un système globalement asynchrone et localement synchrone : meilleure modularité et localité au chapitre 1, meilleure flexibilité et fiabilité au chapitre 2. Le chapitre 4 prouve la meilleure fiabilité et performance de synchronisation et d'arbitrage des réseaux de communication sans horloge, toujours pour une architecture GALS. Le chapitre 5 présente une classe d'arbitres parfaitement fiables et très performants. Le chapitre 6 illustre la plus grande facilité et flexibilité d'interfaçage de composants au moyen d'interfaces asynchrones. Nous avons aussi parlé d'atouts pour la consommation, d'émissions électromagnétiques moins bruyantes et de performance moyenne meilleure. Ces avantages potentiels vont dépendre en partie de l'arrangement judicieux des lignes du réseau d'interconnexion et du protocole implémenté pour réaliser le transport physique des données au cœur du réseau, points abordés au chapitre 7.

Ce manuscrit représente pour le groupe CIS la première étude dans le domaine des architectures globalement asynchrones et localement synchrones et des réseaux de communication. Il s'insère parfaitement dans les domaines de recherche en modélisation, vérification et synthèse de systèmes asynchrones, en se focalisant sur les réseaux sur silicium. Il doit donc servir de référence pour les travaux futurs : réalisation de systèmes sur silicium intégrant à la fois des composants synchrones et asynchrones, modélisation des systèmes en langage SystemC, vérification formelle en collaboration avec le groupe VDS et synthèse automatique des réseaux de communication, y compris les structures non-déterministes.

Perspectives

Réalisation de systèmes GALS

La continuité immédiate des travaux présentés ici est la réalisation du système globalement asynchrone et localement synchrone WUCS développé par le groupe CIS. Nous disposons sous forme de description structurelle en portes logiques, de l'ensemble des modules assemblés composant le réseau FAN. Pour l'interface de synchronisation, nous avons choisi d'utiliser les arbres de multiplexeurs/démultiplexeurs en enlevant les latchs dans les cellules de la FIFO asynchrone. Celle-ci devient donc uniquement un contrôleur local qui autorise le passage des données entre les terminaisons de composants OTS de l'arbre d'entrée et les terminaisons de composants TOS de l'arbre de sortie. Les données se trouvent ainsi mémorisées dans les fils entre les deux arbres d'entrée et de sortie, sous le contrôle du passage des deux jetons.

La description structurelle au niveau portes logiques de l'ensemble des périphériques constituants le système est également disponible pour le processeur MIPS synchrone, le DES asynchrone, la liaison série RS232 et les bancs de mémoire. La validation du système complet est en cours, par prototypage sur le FPGA Stratix d'Altera.

Automatisation de la conception

Ce prototype matériel constitue un démonstrateur de la méthodologie de conception mise en place pour les réseaux d'interconnexion.

Cette méthodologie doit prendre en compte, au plus tôt dans la conception, les contraintes spécifiques à l'interconnexion. Ainsi le flot de conception TAST intègre à présent un estimateur d'activité et un estimateur de la consommation d'énergie qui permettent au concepteur de collecter des informations pertinentes sur la répartition de l'activité et de la consommation d'énergie d'un circuit lors d'une simulation donnée. L'avantage de ces estimateurs est que les informations sont obtenues très tôt dans le flot de conception, au niveau de la spécification CHP d'un circuit.

Dans la perspective d'aboutir à la synthèse entièrement automatique de réseaux sur silicium asynchrones, il faut à présent intégrer dans le flot de conception TAST la synthèse automatique des structures non déterministes, en particulier grâce aux techniques de vérification formelles.

Vérification formelle

Ces procédés de vérification permettent en effet d'analyser si les structures de choix des processus CHP satisfont l'exclusion mutuelle des gardes, ou non. L'impact de cette analyse sur le circuit implémenté est essentiel : si les choix sont mutuellement exclusifs, le circuit ne nécessite pas d'implémenter des éléments de résolution spécifiques. En revanche les structures de choix où plusieurs gardes peuvent être vraies enemble doivent être protégés par des éléments d'exclusion mutuelle ou des synchroniseurs.

Ces outils et méthodologies de validation formelle, dont l'étude se poursuit en collaboration avec le groupe VDS du laboratoire TIMA, vont permettre de modéliser formellement les services offerts par le réseau de communication, afin d'être en mesure de prouver sa correction fonctionnelle.

Bibliographie

[ABR 01] **A. Abrial, J. Bouvier, M. Renaudin, P. Senn et P. Vivet,** "*A New Contactless Smart Card IC using On-Chip Antenna and Asynchronous Microcontroller*", Journal of Solid State Circuits, vol. 36, no. 1101-1107, 2001.

[AHO 04] **T. Ahonen, D. A. Sigüenza-Tortosa, H. Bin et J. Nurmi,** "*Topology optimization for application-specific networks-on-chip*", Proceedings of the International Workshop on System-Level Interconnect Prediction, Paris, France, 2004.

[ALL 03] **E. Allier, G. Sicard, L. Fesquet et M. Renaudin,** "*A New Class of Asynchronous A/D Converters Based on a Time Quantization*", Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'03, Vancouver, CA, Canada, 12-16 May 2003.

[ARM 99] **ARM Ltd.,** *Advanced Microcontroller Bus Architecture (AMBA) Specification*, Rev 2.0, http://www.arm.com/products/solutions/AMBAHomePage.html, May 1999.

[ARM 02a] **ARM Ltd.,** *AMBA Multi-layer AHB Overview*, Rev http://www.arm.com/miscPDFs/1745.pdf, June 2002.

[ARM 02b] **ARM Ltd.,** *AMBA AHB-Lite Overview*, Rev http://www.arm.com/miscPDFs/1744.pdf, June 2002.

[ARM 04] **ARM Ltd.**, "*AMBA Advanced eXtensible Interface (AXI)*", http://www.arm.com/products/solutions/AMBAAXI.html, 2004.

[ART 04a] **Arteris**, "*The Network-on-Chip Company*", http://www.arteris.net/overview.html, 2004.

[ART 04b] **Arteris**, "*Enabling Network-on-Chip for SOC Design*", http://www.arteris.net/documents/Arteris_backgrounder_2.8.pdf, 2004.

[BAG 02] **A. Baghdadi,** "*Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2002. Directeur de Thèse: A. A. Jerraya.

[BAIN 00] **W. J. Bainbridge,** "*Asynchronous System-on-Chip Interconnect*", Ph. D. thesis, University of Manchester, 2000. Directeur de Thèse: S. Furber.

[BAIN 02] **W. J. Bainbridge et S. Furber,** "*CHAIN: A Delay Insensitive CHip Area INterconnect*", IEEE Micro, vol. 22, no. 5, pp. 16-23, September/October 2002.

[BAI 03] **R. Baines et D. Pulley**, "*The picoArray and Reconfigurable Baseband Processing for Wireless Basestations*", picoChip Designs Ltd,,http://www.picochip.com/downloads/IEEE_ComSoc_Jan2003.pdf, 2003.

[BEN 00] **M. Benes,** "*Design and Implementation of Communication and Switiching Techniques for the Pleiades Family of Processors*", Master of Science, University of California at Berkeley, 2000. Directeur de Thèse: J. M. Rabaey.

[BER 04] **R. A. Bergamaschi,** "*Early and Accurate Analysis of SoCs: Oxymoron or Reality?*" Proceedings of the International Workshop on System-Level Interconnect Prediction (SLIP), Paris, France, February 2004.

[BOL 03] **E. Bolotin, E. Cidon, R. Ginosar et A. Kolodny,** "*QNoC : QoS architecture and design process for network on chip*", Journal of Systems Architecture, no. June 2003.

[BORM 97] **D. S. Bormann et P. Cheung,** "*Asynchronous wrapper for heterogeneous systems*", Proceedings of the Proceedings of International Conf. Computer Design (ICCD'97), Oct. 1997.

[BOR 03b] **D. Borrione, M. Boubekeur, L. Mounier, M. Renaudin et A. Sirianni,** "*Validation of asynchronous circuit specifications using IF/CADP*", Proceedings of the VLSI-SOC'03, 13th IFIP International Conference on Very Large Scale Integration, Darmstad, Germany, Dec. 2003.

[BOR 03a] **D. Borrione, M. Boubekeur, L. Mounier, M. Renaudin et A. Sirianni,** "*Modeling CHP descriptions in Labeled Transitions Systems for an efficient formal validation of asynchronous circuit specifications*", Proceedings of the Proc FDL'03, Frankfort, Germany, 23-26 September 2003.

[BOU 04a] **F. Bouesse, M. Renaudin et F. Germain,** "*Asynchronous AES Crypto-processor Including Secured and Optimized Blocks*", Journal of Integrated Circuits and Systems (JICS), vol. 1, no. March 2004.

[BOU 04b] **F. Bouesse, G. Sicard, A. Baixas et M. Renaudin,** "*Quasi Delay Insensitive Asynchronous Circuits for low EMI*", Proceedings of the 4th International Workshop on Electromagnetic Compatibility of Integrated Circuits (EMC COMPO 2004), Angers, France, March 31st, 2004.

[BRINI 03] **S. Brini, D. Benjelloun et F. Castanier,** "*A Flexible Virtual Platform for Computational and Communication Architecture Exploration of DMT VDSL Modems*", Proceedings of the Design Automation and Test in Europe (DATE), Munich, Germany, March 3-7, 2003.

[BRU 95] **E. Brunvand,** "*Low latency self-timed flow-through FIFOs*", Proceedings of the 16th Conference on Advanced Research in VLSI, Chapel Hill, NC, March 1995.

[CHAN 01] **A. Chandrakasan, W. Bowhill et F. Fox,** "*Design of High-Performance Microprocessor Circuits*", IEEE Press, 2001,

[CHAP 84] **D. Chapiro,** "*Globally-Asynchronous Locally-Synchronous Systems*", PhD Thesis, Stanford University, 1984. Directeur de Thèse:

[CHAT 03] **A. Chattopadhyay et Z. Zilic,** "*A globally asynchronous locally dynamic system for ASICs and SoCs*", Proceedings of the 13th ACM Great Lakes symposium on VLSI, Washington, D. C., USA, 2003.

[CHEL 00a] **T. Chelcea et S. M. Nowick,** "*Low-Latency Asynchronous FIFO's Using token Rings*", Proceedings of the Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'00, Eilat, Israel, 4-6 April 2000.

[CHEL 01] **T. Chelcea et S. M. Nowick,** "*Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols*", Proceedings of the 38th Conference on Design Automation (DAC'01), Las Vegas, Nevada, United States, 13-16 March 2001.

[CHEL 04] **T. Chelcea et S. M. Nowick,** "*Robust Interfaces for Mixed-Timing Systems*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 8, pp. August 2004.

[CUM 02] **C. E. Cummings et P. Alfke,** "*Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons*", Proceedings of the Synopsis User Group (SNUG) Papers and Presentations, San Jose, California, 2002.

[DAL 98] **W. J. Dally et J. W. Poulton,** "*Digital Systems Engineering*", Cambridge University Press, 1998,

[DIK 99] **C. Dike et E. Burton,** "*Miller and Noise Effects in a Synchronizing Flip-flop*", IEEE Journal of Solid-State Circuits, vol. 34, no. 6, pp. 849-855,

[DIN 03] **A. V. Dinh Duc,** "*Synthèse Automatique de Circuits Asynchrones QDI*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2003. Directeur de Thèse: M. Renaudin.

[DOB 04] **R. Dobkin, R. Ginosar et C. P. Sotiriou,** "*Data Synchronization Issues in GALS SoCs*", Proceedings of the Asynch'04,

[DUM 03] **E. Dumitrescu et D. Borrione,** "*Symbolic simulation as a simplifying strategy for SoC verification with symbolic model checking*", Proceedings of the Proc. IWSOC2003, Calgary, Canada, July 2003.

[BARD 02] **D. Edwards et A. Bardsley,** "*Balsa: An asynchronous hardware synthesis language*", The Computer Journal, vol. 45(1), no. 1, pp. 12-18, 2002.

[EDW 04] **D. A. Edwards et W. B. Toms,** "*Design, Automation and Test for Asynchronous Circuits and Systems*", Rapport IST-1999-29119, Working Group on Asynchronous Circuit Design, 2004.

[HAS 95] **B. El Hassan,** "*Architecture VLSI asynchrone utilisant la logique différentielle à précharge : application aux opérateurs arihtmétiques*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 1995. Directeur de Thèse: A. Guyot.

[ELM 48] **W. C. Elmore,** "*The transient response of damped linear networks with particluar regard to wideband amplifiers*", Journal of Applied Sciences, vol. 19, no. 56-63, Janv. 1948.

[ESS 02] **M. Essalhiene, L. Fesquet et M. Renaudin,** Dynamic Voltage Scheduling for Real Time Asynchronous Systems, in *12th International Workshop on Power and Timing, Modeling, Optimization and Simulation (PATMOS)*. Sevilla, Spain, 2002.

[FESQ 97] **L. Fesquet,** "*Intégration de sous-systèmes photoniques dans les architectures de communication multiprocesseurs*", Electronique, Paul Sabatier, Toulouse, 1997. Directeur de Thèse: J. Collet.

[FRAG 03] **J. Fragoso, G. Sicard et M. Renaudin,** "*Power/Area trade-offs in 1-of-M parallel-prefix asynchronous adders*", Proceedings of the PATMOS'03 - 13th International Workshop on Power and Timing Modeling, Optmization and Simulation, Torino, Italy, September 10-12, 2003.

[FRAN 04] **U. Frank et R. Ginosar,** "*A Predictive Synchronizer for Periodic Clock Domains*", Proceedings of the 14th International Workshop on Power and Timing, Modeling, Optimization and Simulation (PATMOS'04), 2004.

[FUL 04a] **Fulcrum MicroSystems**, "*Clockless switch-based systems*", http://www.fulcrummicro.com/, 2004.

[FUL 04b] **Fulcrum MicroSystems**, "*PivotPoint FM1010: a Six-interface SPI-4.2 Switch Chip*", http://www.fulcrummicro.com/products/fm1010.htm, 2004.

[FURB 00] **S. Furber,** "*ARM system-on-chip architecture*", Addison Wesley, 2000, 0-201-67519-6.

[GIN 02] **R. Ginosar,** "*Synchronization and Arbitration*", Proceedings of the ACiD Summer School on Asynchronous Circuit Design, Grenoble, France, July 15-19 2002.

[GIN 03] **R. Ginosar,** "*Fourteen ways to fool your Synchronizer*", Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'03, Vancouver, Canada,

[GUE 00a] **P. Guerrier,** Un réseau d'interconnexion pour systèmes intégrés, in *U.F.R. d'Informatique*, A. Greiner, Ed.: Université Paris VI, 2000.

[GUE 00b] **P. Guerrier et A. Greiner,** "*A Generic Architecture for On-Chip Packet-Switched Interconnections*", Proceedings of the Design Automation and Test in Europe (DATE), Paris, France, 2000.

[GUE 00b] **P. Guerrier et A. Greiner,** A Generic Architecture for On-Chip Packet-Switched Interconnections, in *Design Automation and Test in Europe (DATE)*. Paris, France, 2000.

[GUT 00] **V. Gutnik et A. Chandrakasan,** "*Active GHz Clock Network Using Distributed PLLs*", Proceedings of the International Solid-State Circuits Conference, 2000.

[HAU 95] **S. Hauck,** "*Asynchronous design methodologies: An Overview*", Proceedings of the IEEE, vol. 83, no. 1, pp. 69-93, January 1995.

[HAU 95] **S. Hauck,** Asynchronous Design Methodologies: an Overview, in *Proceedings of the IEEE*, vol. 83, 1995, pp. 66-93.

[HEALD 00] **R. Heald,** "*Implementation of a 3rd-Generation SPARC v9 64b Microprocessor*", Proceedings of the International Solid-State Circuits Conference, 2000.

[HEATH 03] **M. Heath et I. Harris,** "*A Deterministic Globally Asynchronous Locally Synchronous Microprocessor Architecture*", Proceedings of the 4th International Workshop on Microprocessor Test and Verification (MTV), Austin, TX, USA, May 29-30, 2003.

[HEATH 04] **M. Heath,** "*Synchro-Tokens: Eliminating Nondeterminism to Enable Chip-Level Test of Globally Asynchronous Locally Synchronous SoC's*", Proceedings of the Design, Automation, and Test in Europe (DATE), Paris, France, February 16-20, 2004.

[HEM 98] **A. Hemani et al.,** "*A methodology and algorithms for efficient interprocess communication synthesis from system description in SDL*", Proceedings of the Proceedings of VLSI Design, Chennai, India, January 1998.

[HEM 99] **A. Hemani et al.,** "*Lowering Power Consumption in Clock by Using Globally Asynchronous Locally Synchronous Design Style*", Proceedings of the Design Automation Conference, 1999.

[HIN 01] **G. Hinton and al.,** "*A 0.18-mm CMOS IA-32 Processor With a 4 -GHz Integer Execution Unit*", IEEE Journal of Solid State Circuits, vol. 36, no. 11, pp. 1617-1627, November 2001.

[HO 01] **R. Ho, K. W. Mai et M. K. Horowitz,** "*The future of Wires*", Proceedings of the IEEE, vol. 89, no. 4, pp. 490-504, April 2001.

[HO 04] **R. Ho, J. Gainsley et R. Drost,** "*Long wires and asynchronous control*", Proceedings of the Asynch'04, 2004.

[T. Q. HO 02] **T. Q. Ho, J. B. Rigaud, M. Renaudin, L. Fesquet et R. Rolland,** "*Implementing Asynchronous Circuits on LUT Based FPGAs*", Proceedings of the Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, 12th International Conference, FPL 2002, Montpellier, France, September 2-4, 2002.

[HOAR 78] **C. A. R. Hoare,** "*Communicating Sequential Processes*", Communications of the ACM, vol. 8, no. 666-677, April, 1978.

[IBM 99] **IBM Corporation**, "*The CoreConnect™ Bus Architecture*", http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF77852569910050C0FB/$file/crcon_wp.pdf, 1999.

[INTEL 99] **Intel Corporation,** *Pentium III Xeon Processor at 500 and 550 MHz*, Rev February 1999.

[ISM 00] **Y. I. Ismail, E. G. Friedman et J. L. Neves,** "*Equivalent Elmore delay for RLC trees*", IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 1, pp. January 2000.

[ISM 01] **Y. I. Ismail et E. G. Friedman,** On-Chip Inductance in High Speed integrated Circuits: Kluwer Academic Publisher, 2001.

[KAR 01] **F. Karim, A. Nguyen, S. Dey et R. Rao,** "*On-chip communication architecture for OC-768 network processors*", Proceedings of the 38th ACM IEEE Design Automation Conference, Las Vegas, Nevada, United States, 2001.

[KAR 02] **F. Karim, A. Nguyen et S. Dey,** "*An Interconnect Architecture for Networking Systems on Chips*", IEEE Micro, vol. 22, no. 5, pp. 36-45, September/October 2002.

[KOND 02] **A. Kondratyev et K. Lwin,** "*Design of Asynchronous Circuits using CAD Tools*", IEEE Design and Test of Computers, vol. 19, no. 4, pp. 107-117, July/August 2002.

[KUR 01] **N. Kurd, J. Barkatullah, R. Dizon, T. Fletcher et P. Madland,** "*Multi-GHz Clocking Scheme for Intel Pentium 4 Microprocessor*", Proceedings of the IEEE International Solid-State Circuits Conference, 2001.

[LEI 85] **C. Leiserson,** "*Fat-Trees: Universal Networks for Hardware- Efficient Supercomputing*", IEEE Transactions on Computers, vol. C- 34, no. 10, pp. 892-901, octobre 1985.

[LOV 02] **W. O. Lovett,** "*CHip Area Network Simulation*", Master of Science, University of Manchester, 2002. Directeur de Thèse: S. Furber.

[LYO 03] **D. Lyonnard,** "*Approche d'assemblage systématique d'éléments d'interface pour la génération d'architecture multiprocesseur*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2003. Directeur de Thèse: A. A. Jerraya.

[YAK 03] **A. Madalinski, A. Bystrov, V. Khomenko et A. Yakovlev,** "*Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design*", Proceedings of the IEE Proceedings: Computers & Digital Techniques: Special Issue on Best Papers from DATE'2003, 2003.

[MAR 93] **A. J. Martin,** "*Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits*", tiré de l'ouvrage "*Developments in Concurrency and Communication"*, C. A. R. Hoare (Ed.), Addison Wesley, chapitre, pp.1-64, 1993.

[MES 90] **D. Messerschmitt,** "*Synchronization in Digital System Design*", IEEE Journal on Selected Areas in Communications, vol. 8, no. 8, pp. 1404-1419, October 1990.

[MON 04] **Y. Monnet, M. Renaudin et R. Leveugle,** "*Asynchronous circuits sensitivity to fault injection*", Proceedings of the 10th IEEE International On-Line Testing Symposium, Madeira Island, Portugal, July 12-14, 2004.

[iHPerf 00] **T. Monteil, J.-M. Garcia et O. Brun,** "*Observation et gestion de ressources pour des grappes homogènes et hétérogènes de calculateurs*", Proceedings of the Ecole d'Hiver iHPerf'2000 : Applications Parallèles Hautes Performances : Analyse, Conception et Utilisation de Grappes Homogènes ou Hétérogènes de Calculateurs, Aussois, France, 4 au 8 décembre 2000.

[MUTT 99] **J. Muttersbach and al.,** "*Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems*", Proceedings of the Twelvth IEEE International Conference on ASIC/SOC, 1999.

[NIL 96] **P. Nilsson et M. Torkelson,** "*A Monolithic Digital Clock Generator for On-Chip Clocking of Custom DSP's*", IEEE Journal of Solid-State Circuits, vol. 31, no. 5, pp. 700-706, May 1996.

[OCP 01] **OCP International Partnership,** *Open Core Protocol*, Rev http://www.ocpip.org/data/ocp_data_sheet.pdf, September, 26 2001.

[OCP-IP 04] **OCP International Partnership**, "*Open Core Protocol International Partnership*", http://www.ocpip.org/about/welcome, 2004.

[OC 04] **Open Cores**, "*SoC Interconnection: Wishbone*", http://www.opencores.org/projects.cgi/web/wishbone/wishbone, 2004.

[OMI 95] **Open Microprocessor Systems Initiative (OMI)**, "*PI-Bus Systems Toolkit*", http://www.cordis.lu/esprit/src/results/pages/infoind/infind24.htm,

[OMI 98] **Open Microprocessor Systems Initiative (OMI)**, dans le cadre d'Esprit, programme européen des technologies de l'information,http://www.cordis.lu/esprit/src/omi-home.htm,

[PAN 04] **D. Panyasak,** "*Réduction des Emissions Electromagnétiques des Circuits Intégrés : l'alternative Asynchrone*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2004. Directeur de Thèse: M. Renaudin.

[MPSoC 00] **P. Paulin,** "*Trends and Requirements for Network Processor SoC tools*", Proceedings of the MultiProcessor System-on-Chip Summer School (MP SoC 2002), France, July 2002.

[PEC 76] **M. Pechoucek,** "*Anomalous response Times of Input Synchronizers*", IEEE Transactions on Computers, vol. C-25, no. 2, pp. 133-139, February 1976.

[QLF 03] **QLF**, "*QuaLification of Circuits Group, TIMA Laboratory*", http://tima.imag.fr/qlf/index.html, 2003.

[REN 99] **M. Renaudin, P. Vivet et F. Robin,** "*ASPRO: an Asynchronous 16-bit RISC Microprocessor with DSP Capabilities*", Proceedings of the ESSCIRC, Duisburg, 21-23 September, 1999.

[REN 00] **M. Renaudin,** "*Asynchronous Circuits and Systems: a Promising Design Alternative*", Journal of Microelectronics, vol. 54, no. 133-149, 2000.

[REN 04b] **M. Renaudin et A. Yakovlev,** "*Arbiters*", tiré de l'ouvrage "*Synchronization (à paraître)"*, R. Ginosar (Ed.), KLUWER academic publishers, chapitre 5, 2004.

[REST 02] **P. Restle et al.,** "*The clock distribution of the Power4 microprocessor*", Proceedings of the ISSCC Digest of Tehnical Papers, 2002.

[RIG 01] **J. B. Rigaud, J. Quartana, L. Fesquet et M. Renaudin,** Modeling and design of asynchronous priority arbiters for on-chip communication systems, in *VLSI-SOC'01, 11th IFIP International Conference on Very Large Scale Integration*. Montpellier, France, 2001, pp. 424-429.

[RIG 01] **J. B. Rigaud, J. Quartana, L. Fesquet et M. Renaudin,** "*Modeling and design of asynchronous priority arbiters for on-chip communication systems*", Proceedings of the VLSI-SOC'01, 11th IFIP International Conference on Very Large Scale Integration, Montpellier, France, 3-5 Dec. 2001.

[RIG 02a] **J. B. Rigaud,** "*Spécification de Bibliothèques pour la Synthèse de Circuits Asynchrones*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2002. Directeur de Thèse: M. Renaudin.

[RIGA 02] **J. B. Rigaud,** Spécification de Bibliothèques pour la Synthèse de Circuits Asynchrones: Institut National Polytechnique de Grenoble, 2002.

[RIG 02b] **J. B. Rigaud, J. Quartana, L. Fesquet et M. Renaudin,** "*Modeling and design of asynchronous priority arbiters for on-chip communication systems*", tiré de l'ouvrage "*SOC Design Methodologies"*, Hardbound M.-L. Flottes (Ed.), KLUWER academic publishers, chapitre 5, 2002.

[RIG 02c] **J. B. Rigaud, J. Quartana, L. Fesquet et M. Renaudin,** "*High-level modeling and design for on-chip communication systems*", Proceedings of the Design Automation and Test in Europe (DATE'02), Paris, France, March 4-8, 2002.

[RUSU 00] **S. Rusu et S. Tam.,** "*Clock Generation and Distribution for the First IA -64 Microprocessor*", Proceedings of the International Solid-State Circuits Conference, 2000.

[SCH 04] **J. Schmaltz et D. Borrione,** "*A functionnal approach to the formal specification of networks on chip*", Rapport TIMA--RR-04/04-01--FR, TIMA Laboratory, Grenoble, 2004.

[SEITZ 80] **C. Seitz,** "*System Timing*", tiré de l'ouvrage "*Introduction to VLSI Systems"*, C. Mead et L. Conway, chapitre 7, 1980.

[SEI 94] **J. Seizovic,** "*Pipeline Synchronization*", Proceedings of the IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 1994), 1994.

[SIA 04] **SEMATECH Inc. (http://sematech.org),** International Technology Roadmap for Semiconductors (ITRS), http://public.itrs.net/Files/2003ITRS/Home2003.htm: Semiconductor Industry Association, 2004.

[SIA 04] **SEMATECH Inc. (http://sematech.org),** "*International Technology Roadmap for Semiconductors (ITRS), http://public.itrs.net/Files/2003ITRS/Home2003.htm*", Rapport Semiconductor Industry Association, 2004.

[SEM 03] **Y. Semiat et R. Ginosar,** "*Timing Measurements of Synchronization Circuits*", Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'03, Vancouver, Canada,

[SIL 03] **Silicore Corporation**, "*Summary of SoC Interconnection Buses*", http://www.silicore.net/uCbusum.htm, 24 Sep 2003.

[SIL 04] **Silicore Corporation**, "*The WISHBONE Service Center*", http://www.silicore.net/wishbone.htm, 4 Oct 2004.

[SJO 00] **A. E. Sjogren et C. J. Myers,** "*Interfacing Synchronous and Asynchronous Modules within a High-Speed Pipeline*", IEEE Transactions on VLSI Systems, vol. 8, no. 5, pp. 573-583, October 2000.

[SLI 02] **K. Slimani and al.,** "*Ultra Secure SmartMIPS Design Report*", Rapport Projet Européen G3CARD, 2002.

[SLI 04] **K. Slimani, J. Fragoso, M. Essalhiene, L. Fesquet et M. Renaudin,** "*Low-Power Asynchronous Processors*", tiré de l'ouvrage "*Low-Power Electronics Design"*, C. Piguet (Ed.), CRC Press, chapitre 22, 2004.

[SRI 94] **M. Sriram et S. M. Kang,** "*Physical Design for Multichip Modules*", kluwer Academic Publishers, 1994,

[STEV 03] **K. S. Stevens,** "*Energy and Performance Models for Clocked and Asynchronous Communication*", Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'03, Vancouver, CA, Canada, 12-16 May 2003.

[SUTH 89] **I. E. Sutherland,** Micropipelines, in *Communications of the ACM*, vol. 32, 1989, pp. 720-738.

[SUTH 01] **I. E. Sutherland et S. Fairbanks,** "*GasP: A Minimal FIFO Control*", Proceedings of the Seventh International Symposium on Advanced Research on Asynchronous Circuits and Systems, Asynch 2001, Salt Lake City, Utah, USA, March 2001.

[SYN 01] **Synopsys Inc.**, "*DW_fifo_s2_sf : Synchronous (Dual-Clock) FIFO with Static Flags*", http://www.synopsys.com/products/designware/docs/doc/dwf/datasheets/dw_fifo_s2_sf.pdf, August 17, 2001.

[TANEN 00] **A. S. Tanenbaum et M. van Steen,** "*Synchronization*", tiré de l'ouvrage "*Distributed Systems, Principles and Paradigms"*, P. I. Edition (Ed.), Prentice Hall, chapitre 5, pp.241-290, 2000.

[TRI 01] **C. Tristam,** "*It's Time for Clockless Chips*", Technology Review, no. 36-41, October 2001.

[BERK-91] **K. van Berkel, J. Kessels, M. Roncken, R. Saeijs et F. Schalij,** "*The VLSI-programming language Tangram and its translation into handshake circuits*", Proceedings of the European Conference on Design Automation (EDAC), 1991.

[VILL 02] **T. Villiger and al.,** "*Multi-point Interconnect for Globally-Asynchronous Locally-Synchronous Systems*", Proceedings of the Handouts of the second Asynchronous Circuit Design Workshop, ACiD 2002, Münich, Germany, January 28-29, 2002.

[VILL 03] **T. Villiger, H. Kaeslin, F. Gurkaynak, S. Oetiker et W. Fichtner,** "*Self-timed Ring for Globally-Asynchronous Locally-Synchronous Systems*", Proceedings of the Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'03, Vancouver, CA, Canada, May, 12-16 2003.

[VIV 01] **P. Vivet,** "*Une Méthodologie de Conception de Circuits Intégrés Quasi-Insensibles aux Délais: Application à l'Etude et à la Réalisation d'un Processeur RISC 16-bits Asynchrone*", Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2001. Directeur de Thèse: M. Renaudin.

[VSI 04a] **VSI Alliance**, "*Welcome to the VSI Alliance*", http://www.vsi.org/, 2004.

[VSI 04b] **VSI Alliance**, "*VSI Alliance Specifications*", http://www.vsi.org/library/specs/summary.htm#dd, 2004.

[WES 93] **N. H. E. Weste et K. Estraghan,** "*Principles of CMOS VLSI Design, A Systems Perspective*", Addison Wesley, 1993,

[WU 00] **Q. Wu, M. Pedram et X. Wu,** "*Clock Gating and Its Application to Low Power Design of Sequential Circuits*", IEEE Transactions on Circuit and Systems, vol. 47, no. 3, pp. 415-420, Mar. 2000.

[XAN 01] **T. Xanthopoulos and al.,** "*The Design and Analysis of the Clock Distribution Network for a 1.2 GHz Alpha Microprocessor*", Proceedings of the IEEE International Solid-State Circuits Conference, 2001.

[XAN 01] **X. Xanthopoulos and al.,** The Design and Analysis of the Clock Distribution Network for a 1.2 GHz Alpha Microprocessor, in *IEEE International Solid-State Circuits Conference*, 2001, pp. 402-403.

[YUN 96a] **K. Yun, P. Beerel et J. Arceo,** "*High Performance Asynchronous Pipeline*", Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'96), Aizu-Wakamatsu, Japan, 17-28 April 1996.

[YUN 96b] **K. Yun et D. Dill,** "*Pausible clocking: a first step toward heterogeneous systems*", Proceedings of the Proceedings of International Conf. Computer Design (ICCD'96), Oct. 1996.

[ZHE 99] **L. R. Zheng et H. Tenhunen,** "*Noise margin constraints on interconnectivity for low power and mixed-signal VLSI circuits*", Proceedings of the Advanced Research in VLSI : proceeedings of 20th Anniversary Conference, 1999.

[ZHU 02] **S. Zhuang, W. Li, J. Carlsson, K. Palmkvist et L. Wanhammar,** "*An Asynchronous Wrapper with Novel Handshake Circuits for GALS Systems*", Proceedings of the The IEEE International Conference on Communication Circuits And Systems (ICCCAS'02), Cheungdu, China, August 2002.

Annexes

Annexe A – Syntaxe et sémantique du langage CHP

A1. Syntaxe CHP

A1.1 Le fichier source

Le fichier source d'un programme écrit en langage CHP est un simple fichier texte dont l'extension est par convention « .chp ».
Un programme écrit en langage CHP est composé d'un ou de plusieurs composants. Chaque composant est constitué d'un ensemble de processus parallèles et/ou d'instances de composants qui communiquent par des canaux. Les processus contiennent des instructions qui peuvent être composées en séquence ou en parallèle. Structures d'un composant et d'un process CHP :

COMPONENT <*component name* >
<*port declarations*>
<*component, channel and constant declarations*>
BEGIN
<*processes and component instances*>
END;

PROCESS <process name>
<port declaration>
<variable declaration>
BEGIN
Instructions
END;

A1.2 Conventions lexicales

Typologie

Le CHP est un langage insensible à la casse (case insensitive). Cela signifie qu'il n'y a pas de différence entre les caractères majuscules et minuscules. De ce fait, écrire BEGIN, BeGiN ou begin est strictement équivalent.

Commentaires

L'ajout d'un commentaire dans un programme CHP s'effectue après le signe « -- » et doit finir avant la nouvelle ligne. L'ajout de lignes de commentaires se fait en utilisant les balises « /* » et « */ ».

Caractères permis

[\t\r\n ;:\[\]\.\ »\'\(\)\-\+\ !\ ?*@,/#<>=_a-zA-Z0-9]

Identificateurs
Les identificateurs commencent par une lettre et sont suivis par une liste de lettres et/ou de chiffres. Comme le CHP étendu peut être compilé en VHDL, en C, ou autre, les mots clé du langage cible sont interdits. Ce point de contrôle est laissé au bon soin du concepteur pour le moment. Tous les objets CHP (composants, variables, canaux, …) sont repérés par un unique identificateur.

Mots clés CHP

ABS	ACTIVE	AND	BD	BEGIN	BIT
BOOLEAN	CHANNEL	COMPONENT	CONSTANT	DI	DR
END	FALSE	IN	INTEGER	MAP	MOD
MR	NAND	NATURAL	NOR	NOT	OR
OTHERS	OUT	PASSIVE	PORT	PROCESS	ROL
ROR	SDR	SIGNED	SKIP	SLA	SLL
VARIABLE	XNOR	XOR			

A1.3 Type de données
Un entier N est représenté en précision arbitraire par ″d_{L-1} , d_{L-2} , .., d_0″[B][L]. Cette écriture signifie vecteur de « L » digits représentés dans la base « B ». La longueur L et la base B sont des nombres naturels. Le type de données peut être signé ou non signé.
Type de données non signé
La valeur d'un entier non signé selon l'écriture définie précédemment est donnée par

$$N = \sum_{i=0}^{L-1} d_i B^i$$

Les types sont présentés selon l'ordre de complexité (valeur simple, vecteur, ou tableau). Deux types de base existent en CHP étendu, un type simple et un type vecteur.

Valeur simple
La valeur simple est équivalente à un tableau dont la longueur et la dimension valent 1.
MR[B] : type de Multi-Rails en base B représentant un nombre entre 0 et (B-1), codé en « 1 parmi n ».

Exemple :

```
VARIABLE  b : MR[B] ;     -- déclaration d'une variable b avec le type MR
          b := "x"[B];    -- affectation de b avec 0 <= x < B
```

Vecteur
Le type vecteur est équivalent à un tableau dont la dimension vaut 1. Sa syntaxe se traduit comme suit :

```
a[i]      -- le ième élément de a
          --« a » est un vecteur (contient plus d'un élément).
```

Pour accéder à un élément du vecteur « a », il faut préciser l'index de l'élément entre crochets après l'identificateur de la variable. « i » est une expression qui peut être évaluée statiquement et dont la valeur doit être comprise dans l'intervalle [0 , L-1], si L est la longueur du vecteur « a ». La variable a[i] est du même type que a, excepté que sa longueur est « 1 ».

MR[B][L] : Vecteur de L éléments de type Multi-Rails en base B. Une variable de ce type correspond à un nombre entre 0 et (B^L - 1).

Exemple :

```
VARIABLE  b : MR[B][3] ;    -- déclaration d'un vecteur de 3 éléments en base B
          b := "x.y.z"[B];  -- affectation du vecteur b avec  0 <= x , y, z< B
```

D'autres types de données basés sur les types non-signés précédents ont été définis. Ils introduisent plus de facilité pour les concepteurs. Ils se déclinent comme suit :

Valeurs simples

DR Type de double-rails - équivalent au type MR[2]
BIT Nombre binaire (0 ou 1) - équivalent au type MR[2]
BOOLEAN Valeur booléenne (vraie ou fausse) - équivalent au type MR[2]
SR Type de mono-rail - équivalent à MR[1]
Sert uniquement à synchroniser des processus communicants (donc il n'est pas utilisé pour des variables)

Vecteurs

DR[L] Vecteur de L éléments de type double-rails - équivalent au type MR[2][L]
BIT[L] Vecteur de nombres binaires - équivalent au type MR[2][L]
BOOLEAN[L] Vecteur de booléens - équivalent au type MR[2][L]
NATURAL[Max] Nombres naturels appartenant à l'intervalle [0, Max] – équivalent au type MR[2][L] , L étant la partie entière supérieure de $Log_{2}[Max+1]$. Max est un entier naturel qui doit être spécifié lors de la déclaration de l'élément du type NATURAL.

Tableaux

Le type "tableau" peut être utilisé uniquement dans la déclaration de variables et de constantes. Les ports et les canaux ne peuvent être déclarés en type tableau pour des raisons technologiques (le protocole de communication implique 1 seul acquittement par canal de communication, or si un canal est déclaré en tableau et que ses différents éléments sont connectés à différents ports, quel port devra alors arbitrer l'acquittement et que feront les autres ports en parallèle ?). Tous les types de CHP sont représentables en tableau, excepté SR.

MR[B][L][D]

Dans tous les cas D est la longueur du tableau. C'est à dire que le tableau est constitué de D éléments de type MR[B][L] ou SMR[B][L] (tous les autres types étant des dérivés de ces deux types).
Un tableau ne peut pas être utilisé en tant que tel dans un programme CHP. Seuls les éléments d'un tableau peuvent être utilisés dans les expressions.

Pour accéder à un élément d'une variable tableau, la syntaxe est la même que celle utilisée pour accéder a un digit d'un type vecteur.

```
a[n]  -- le nième élément du tableau « a »
      -- « n » est une expression
```

Si n est une expression évaluable statiquement, n doit être compris dans l'intervalle [0, D-1]. Le type de la variable a[n] est soit MR[B][L] ou SMR[B][L] selon la déclaration de « a ».
Si les éléments du tableau sont des vecteurs alors, les digits des ces éléments peuvent être accédés comme suit :

```
a[i][n]      -- le ième élément de a[n]
a[i..j][n]   -- tous les éléments de a[i][n]  à a[j][n]
```

Les autres types non signés de tableau dérivent du type de base MR[B][L][D] :

DR[L][D] Tableau de D vecteurs de L éléments de type double-rails - équivalent au type MR[2][L][D]
BIT[L][D] Tableau de D vecteurs de nombres binaires - équivalent au type MR[2][L][D]
BOOLEAN[L][D] Tableau de D vecteurs de booléens - équivalent au type MR[2][L]

NATURAL[Max][D] Tableau de D nombres naturels appartenant à l'intervalle [0, Max] – équivalent au type MR[2][L] , L étant la partie entière supérieure de $Log_{2^{[Max+1]}}$. Max est un entier naturel qui doit être spécifié lors de la déclaration de l'élément du type NATURAL.

Type de données signé
La valeur d'un entier signé selon l'écriture définie précédemment est donnée par

$$N = \begin{cases} \sum_{i=0}^{L-1} d_i B^i & \text{si } d_{L-1} < B/2 \\ (d_{L-1} - B)\ B^{L-1} \sum_{i=0}^{L-2} d_i B^i & \text{si } d_{L-1} >= B/2 \end{cases}$$

Cette écriture signifie que seul le digit de poids fort (le plus à gauche) porte le signe. Les autres digits se comportent comme des digits non signés.

En outre tous les types de données non signés présentés plus haut (sauf le type SR) possèdent leurs équivalents en type signé. Les deux types de base (1 type simple et 1 vecteur) sont donc :

Valeur simple
SMR[B] Type de Multi-Rails signé en base B. Si B est pair, ce type représente un nombre compris dans l'intervalle [–(B/2), (B/2)-1]. Si B est impair, il représente un nombre compris dans l'intervalle [-(B-1)/2 , (B-1)/2].

Exemple:
```
VARIABLE   b : SMR[4];      -- déclaration d'une variable
           b := "2"[4];     --affectation de b :  b = 2 – 4 = -2
```

Vecteur
SMR[B][L] Vecteur de L éléments « Multi-Rail » signés en base B.

Exemple :
```
VARIABLE   b : SMR[3][3] ;    -- déclaration de b
           b := "2.2.2"[3];    -- affectation : b=(-1)*3² + 2* 3¹ + 2* 3⁰  = -1
```

Les autres types de données signés définis sont basés sur les types signés de base précédents. Ils introduisent plus de facilité pour les concepteurs. Ces types sont les suivants :

Valeur simple
SDR Type double-rails signé - équivalent à SMR[2]
Vecteur
SDR[L] Vecteur de L éléments double-rails signés - équivalent à SMR[2][L]
INTEGER[Max] Nombre de type SMR[2][L] compris dans l'intervalle [–Max+1),Max] - équivalent à SMR[2][L]
Tableau
SMR[B][L][D] Tableau de D vecteurs de L éléments en base B
SDR[L][D] Tableau de D vecteurs de L éléments de type double-rails - équivalent au type MR[2][L][D]
INTEGER[Max][D] Tableau de D nombres de type SMR[2][L] compris dans l'intervalle [–Max+1),Max] - équivalent à SMR[2][L]

A1.4 Conversion de types de données

Les données manipulées en CHP sont typées. Cela signifie que pour chaque donnée que l'on utilise il faut préciser son type, ce qui permet de connaître l'occupation mémoire de la donnée ainsi que sa représentation. Les deux opérandes d'une opération logique, arithmétique ou encore d'une action de communication doivent être du même type. L'affectation est la seule opération qui permet de convertir les types de données et les rendre compatibles. Le type de données de la partie droite de l'affectation est toujours converti en le type de données de la partie gauche. A titre d'illustration si une variable « x » est déclarée avec le type DR et une variable « y » avec le type MR[3] alors l'affectation x :=y appelle automatiquement la procédure de conversion de type et « y » est converti en DR.

A1.5 Opérations

On note les opérations par ordre de priorité décroissante de haut en bas et de droite à gauche.

()
- (unaire) not abs
*** / mod + - and nand or nor xor xnor sll sla srl sra rotl rotr**
= /= < <= > >=
 # ! ?
:=

Opérations de conversion de signe

Une conversion de signe ne se fait pas automatiquement mais explicitement. Le passage d'un type non signé à signé s'effectue par l'opérateur SIGNED(< expression non signée>). A l'inverse le passage d'un type signé à non signé s'obtient par l'opérateur UNSIGNED(<expression non signée>). Il est à noter que l'opération de conversion de signe ne change pas la structure du type (base et longueur) mais uniquement le signe. Par ailleurs, la valeur arithmétique n'est conservée que si elle appartient à l'intervalle du type de destination ($< (B^L+1)/2$ si la conversion est vers signé, >0 si la conversion est vers non signée). A titre d'exemple un MR[3][3] qui vaut ″2.2.2″[3]= ″26″[10] s'il est converti en signé vaudra ″2.2.2″[3]= ″-1″[10].

Opérations de comparaison

« =, /=, <, >, <=, >= ». Si les deux opérandes comparés sont des variables, alors ils doivent être du même type (même signe, même base, même longueur).Si l'un des deux opérandes comparés est une expression constante et l'autre une variable, alors le type de l'expression est converti pour être adapté au type de la variable.

Opérations logiques

« not, nand, and, nor, or, xnor, xor ». Ils sont identiques aux opérateurs logiques classiques définis pour les valeurs booléennes à ceci près qu'en CHP ils sont étendus aux autres types. Dans une base donnée, le résultat d'une opération logique est le même que celui d'une opération booléenne effectuée bit par bit avec la représentation binaire de chaque digit de la valeur. Les deux opérandes doivent cependant être du même type (base, longueur, signe). Pour effectuer une opération logique entre opérandes de différents types une affectation doit être explicitée.

Opérations arithmétiques

« +, -, *, mod, abs ». La restriction pour les opérateurs arithmétiques est que les deux opérandes doivent être du même type (base, longueur, signe). Dans le cas contraire, une affectation / conversion est nécessaire. Les deux opérateurs de négation unaire « - » et abs s'appliquent uniquement pour le type signé.

Opérations de décalage

« sll, sla, srl, sra, rotl, rotr ». L'opérande qui exprime le nombre digits à décaler doit être non signé. L'opérande sur lequel s'effectue le décalage est de type quelconque, SR exclu.

A1.6 Instructions

Les instructions impliquent une vérification de type, mais tous les types CHP s'expriment en fonction des 2 types de base MR[B][L] (type non signé) et SMR[B][L] (type signé). Une conversion de type est effectuée seulement lorsque les types des opérandes sont structurellement différents. A titre d'exemple les opérations impliquant les types BOOLEAN, BIT et DR ne nécessitent pas de conversion puisque ils sont tous équivalent au type MR[2][1].

Instruction d'affectation / conversion

« := ». L'instruction d'affectation / conversion peut être utilisée avec tous les types du CHP, le type « SR » mis à part. Dans l'écriture « Var := Expression » l'opérande Var doit obligatoirement être une variable (pas de constante, canal, port ou expression). L'affectation est utilisée entre différents types à condition qu'ils soient de signe identique. Dans ce cas on doit obligatoirement recourir à l'opérateur de conversion de signe. Cet opérateur est le seul qui permet la conversion de type.

Instructions de communication

Les canaux et les ports sont utilisés pour communiquer entre processus et composants. Pour qu'un canal ou un port de type « td » réceptionne ou émette une donnée à travers une variable, il faut que cette variable soit également de type « td » excepté pour le signe. Aucune conversion n'est effectuée à ce niveau vu que seul l'opérateur d'affectation le permet. Les opérateurs assurant les actions de communications sont au nombre de 3 : le probe « # », l'opérateur de réception de données « ? », et l'opérateur d'émission de données « ! ».

- Le probe « # » : Le probe permet de vérifier si une donnée est prête sur un port. Cet opérateur retourne un booléen et ne peut être utilisé que pour les ports passifs (un port d'entrée est passif par défaut, un port de sortie doit par contre être explicité passif avant d'être probé). En fait, il n'est utilisé que dans les expressions de gardes car seules les gardes retournent un booléen. L'écriture « #port_name » teste si une donnée est présente dans le canal relié au port port_name . Si le port « probé » est un port d'entrée, on peut également comparer la valeur de Cette donnée à une valeur explicite grâce à l'écriture « #port_name = expression ».

- Réception de données « ? » :

La syntaxe « <nom_de_port> ? ; » permet de recevoir une donnée sans la mémoriser et la syntaxe « <nom_de_port> ? <nom_de_variable> ; » permet de lire une donnée en la stockant dans une variable. Seuls les ports d'entrée peuvent être utilisés à gauche de l'opérateur « ? » et seules les variables sont utilisées pour recevoir les données émises par ces ports.

- Emission de données « ! » :

La syntaxe « <nom_de_port> ! ; » permet d'émettre un acquittement et n'est permise qu'avec le type SR. La syntaxe « <nom_de_port> ! <expression> ; » permet d'émettre une donnée contenue dans une expression à travers un port. Seuls les ports de sortie peuvent être utilisés à gauche de l'opérateur « ! » et toutes les expressions peuvent être émises les données émises à travers ces ports. Les types des opérandes doivent être strictement identiques.

Instructions de choix et répétitions

Une commande gardée est composée d'une liste d'instructions précédée par une condition booléenne. Si la condition est vraie la liste d'instructions est exécutée.

- Structure de contrôle déterministe

La structure de contrôle déterministe attend jusqu'à ce qu'une garde et une seule garde soit vraie pour exécuter l'instruction correspondante.

@[
guarded commands (BREAK|LOOP)
guarded commands (BREAK|LOOP)
{guarded commands (BREAK|LOOP)…}
];

- Structure de contrôle indéterministe

La structure de contrôle indéterministe attend jusqu'à ce que au moins une garde soit vraie. Pour cette structure, il est possible que plusieurs gardes soient vraies en même temps. Dans ce cas, la structure fait un tirage aléatoire d'une garde parmi l'ensemble des gardes vraies et exécute l'instruction correspondante.

@@[
guarded commands (BREAK|LOOP)
guarded commands (BREAK|LOOP)
{guarded commands (BREAK|LOOP)…}
];

A1.7 Opérateurs

Opérateur de séquentialité
« ; ». Sémantiquement, l'écriture « inst1 ; inst2 » signifie que l'instruction 2 n'est exécutée que lorsque l'instruction « inst1 » a été elle même exécutée. Dès la fin de l'exécution de l'instruction « inst2 » on passe à la suite du programme.

Opérateur de parallélisme
« , ». Sémantiquement, l'écriture « inst1 , inst2 » signifie que l'instruction « inst1 » s'exécute parallèlement que l'instruction « inst2 ». On ne passe à la suite du programme que lorsque l'exécution des 2 instructions est terminée.

Annexe B – Dérivation systématique des circuits d'arbitrage indéterministe en Réseaux de Petri

B.1. L'élément d'exclusion mutuelle

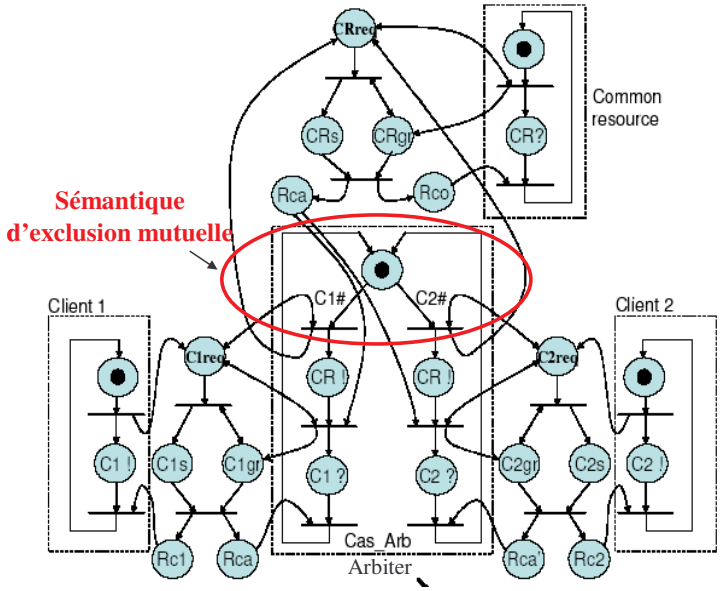


Figure B-1 : dérivation d'un circuit à base d'exclusion mutuelle (exemple du §4.1.4.2.).

B.2. Le circuit synchroniseur

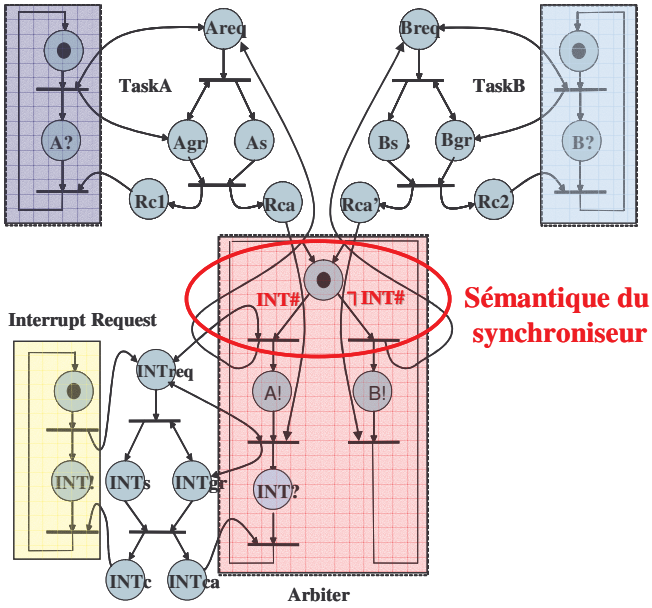


Figure B-2 : dérivation d'un circuit à base de synchroniseur (exemple du §4.1.4.3.).

Annexe C – Architecture de la FIFO asynchrone

C.1. Le Starter

C.1.1. Un exemple de Starter généré à la main, correspondant à la modélisation fonctionnelle de la Figure 6.17

```
[LC ! ; LC ! ;
        [ RC ? ; LC ! ; loop] ;
Break
]
```

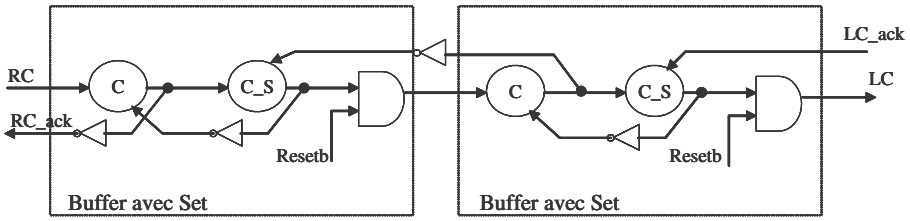


Figure C-1 : synthèse manuelle du Starter de la FIFO circulaire par passage de jeton

C.1.2. Le Starter correspondant à la modélisation CHP sous forme de machine à états synthétisable de la Figure 6.18

```
[CS ? cs;
        @ [ cs = 0 => LC !, NS !0 ; break
            cs = 1 => LC !, NS !1 ; break
            cs = 2 => RC?; LC !, NS !1 ; break
           ] ;
loop
]
```

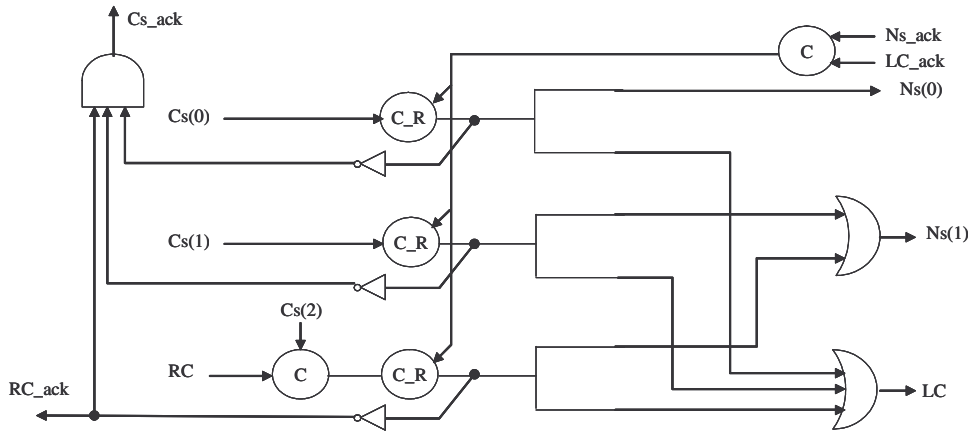


Figure C-2 : synthèse du Starter (machine à état) de la FIFO circulaire par passage de jeton

```
[CS ! 0 ;
    [NS ? ns;
          @ [ ns = 0 => CS !1 ; break
              ns = 1 => CS !2 ; break
            ] ;
    loop] ;
break
]
```

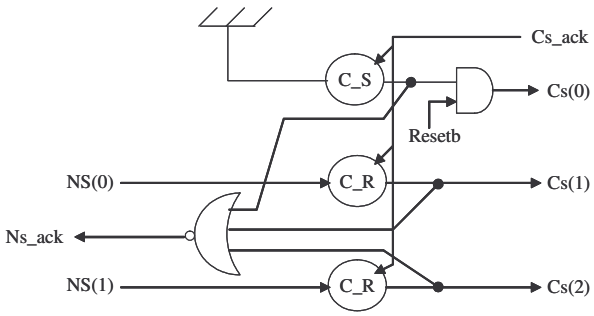
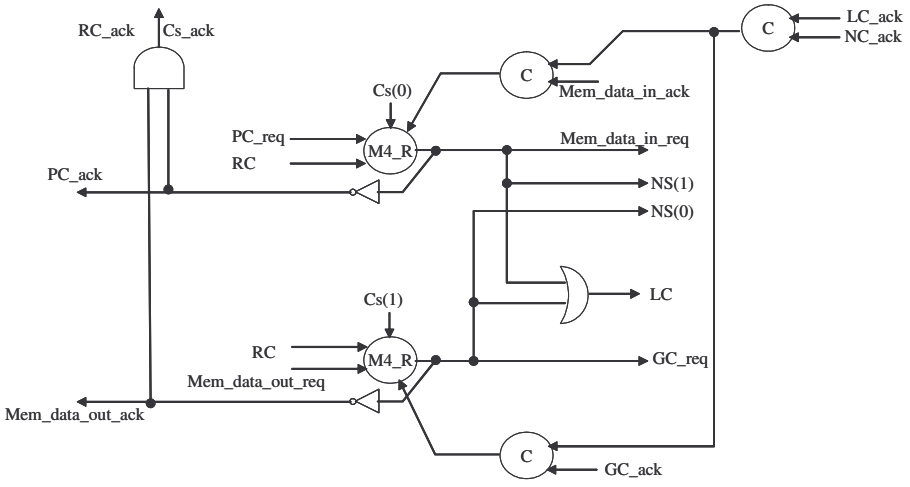


Figure C-3 : synthèse du contrôleur du Starter (machine à état) de la FIFO circulaire

C.2. La cellule de FIFO

```
[CS ? cs;
          @ [ cs = 0 => RC?, PC?data; LC !, NS !1, mem_data_in ! data ; break
              cs = 1 => RC?, mem_data_out? data; GC ! data, LC !, NS ! 0 ; break
            ] ;
loop
]
```



[Mem_data_in? Data ; mem_data_out ! Data ; loop]

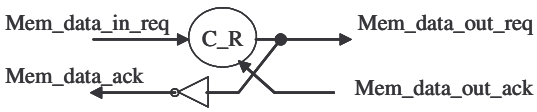


Figure C-4 : synthèse de la cellule de FIFO (machine à état) avec acquittement au plus tard

```
[CS ! 0 ;
    [NS ? ns; CS ! ns ; loop ] ;
break
]
```

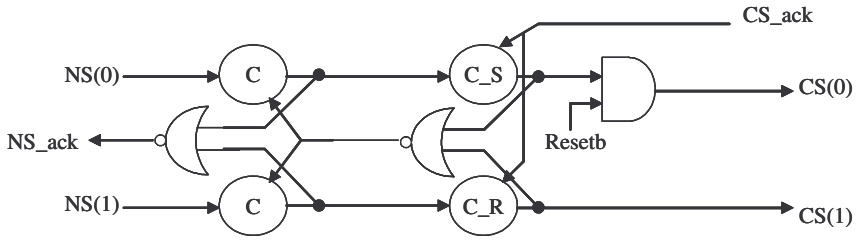


Figure C-5 : synthèse du contrôleur de la cellule de FIFO circulaire avec acquittement au plus tard

C.3. Démultiplexeur 1 vers 2: One-to-Two Sequential Switch (OTS)

```
OTS

[ CS ? cs
    @ [ cs =0 => In ? X ; NS ! '1', Out0 ! X ; break
        cs =1 => In ? X ; NS ! '0', Out1 ! X ; break
      ]:
loop
]
```

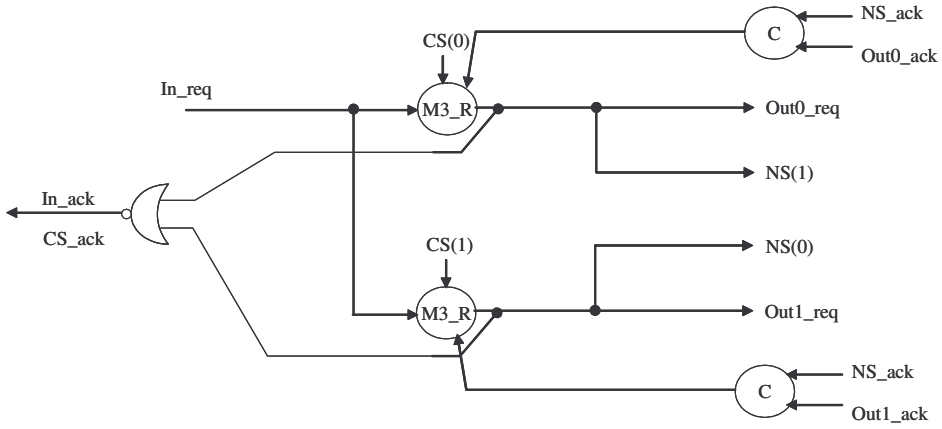


Figure C-6 : synthèse du démultiplexeur 1 vers 2 (One-to-Two Sequential Switch (OTS))

```
OTS_ctrl

[CS ! 0 ;
    [NS ? ns; CS ! ns ; loop ] ;
break
]
```

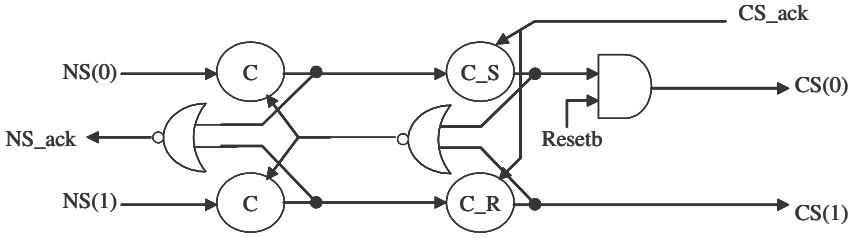


Figure C-7 : synthèse du contrôleur du démultiplexeur 1 vers 2 (One-to-Two Sequential Switch (OTS))

C.4. Two-to-One Sequential Switch (TOS)

```
TOS
[ CS ? cs
    @ [ cs =0 => In0 ? X ; NS ! '1', Out ! X ; break
        cs =1 => In1 ? X ; NS ! '0', Out ! X ; break
      ]:
loop
]
```

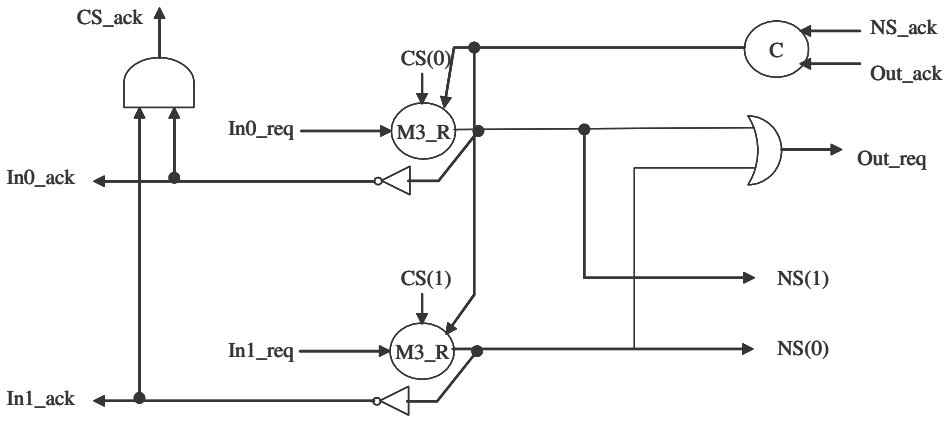


Figure C-8 : synthèse du multiplexeur 2 vers 1 (Two-to-One Sequential Switch (TOS))

```
TOS_ctrl

[CS ! 0 ;
    [NS ? ns; CS ! ns ; loop ] ;
break
]
```

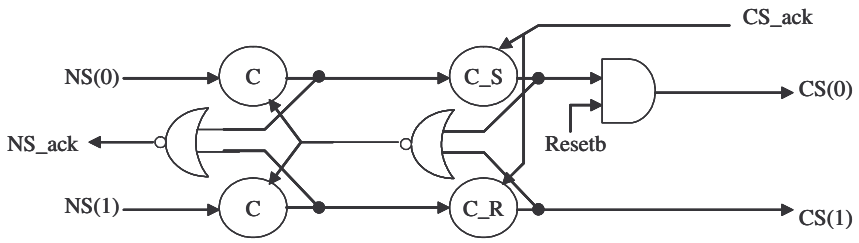
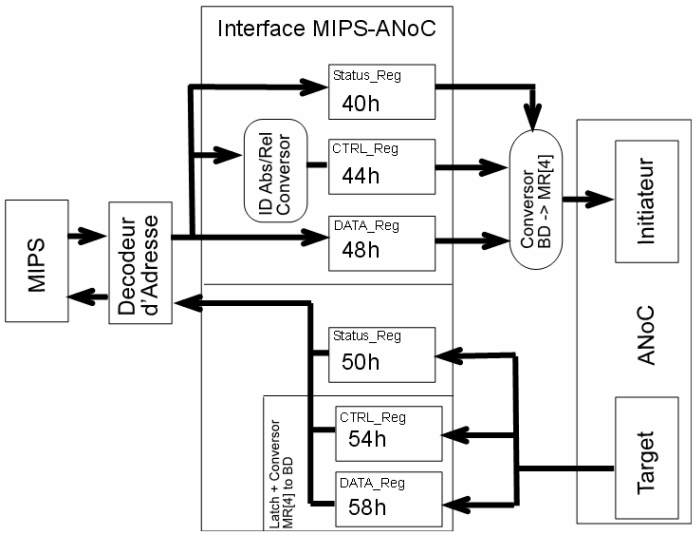


Figure C-9 : synthèse du contrôleur du multiplexeur 2 vers 1 (Two-to-One Sequential Switch (TOS))

Annexe D – Système WUCS : détail de l'interface MIPS-ANOC



Résumé

Cette thèse porte sur l'étude d'architectures de communication sans horloge pour la conception de réseaux de communication asynchrones appliqués aux systèmes globalement asynchrones et localement synchrones. Elle s'intègre également dans le cadre du développement de l'outil de conception automatique de circuits asynchrones TAST (TIMA Asynchronous Synthesis Tool).

L'importance des besoins de communication au sein des systèmes intégrés modernes fait du réseau d'interconnexion un acteur majeur de la complexité et des performances de ces systèmes. Parmi les nombreuses méthodologies existantes adressant le problème de synchronisation au sein des systèmes sur silicium, nous montrons l'intérêt de choisir un réseau d'interconnexion sans horloge pour la communication des systèmes globalement asynchrones et localement synchrones. Nous développons dans ce manuscrit une méthodologie de conception d'un réseau d'interconnexion qui utilise les propriétés d'excellente modularité des circuits sans horloge. Nous découpons la construction de nos réseaux sur silicium asynchrones en quatre modules majeurs : arbitrage, transport, routage et synchronisation. L'objectif de ce découpage simple est de permettre à terme la synthèse automatique d'arbitres et de réseaux de communication sans horloge dans l'outil de conception TAST. Les modules du réseau sont spécifiés en CHP, un langage de modélisation de haut niveau adapté à la description et à la synthèse de circuits asynchrones. A travers ces modélisations, nous mettons en relief l'importance des problèmes d'arbitrage et de synchronisation entre les blocs du système. Nous présentons un système de communication qui illustre cette méthodologie de construction de réseau par assemblage de modules et son degré d'automatisation actuel.

Title: Design of Asynchronous Network on Chip: application to GALS systems

Abstract

This thesis tackles a research on self-timed communication architectures for the design of asynchronous Networks-on-Chip (NoCs), dedicated to Globally-Asynchronous Locally-Synchronous (GALS) systems. This study also takes part within the framework of the TIMA Asynchronous Synthesis Tool (TAST) suite.

The needs for communication within modern Systems-on-Chip (SoCs) turns the interconnect network into a major contributor of complexity and performances of these systems. Among many existing methodologies adressing the problem of synchronization within SoCs, this work demonstrates the advantages to choose an asynchronous interconnect network for the communication of GALS systems. This manuscript puts forward a design methodology for interconnect networks which uses the modularity property of asynchronous circuits. We cut out the construction of our asynchronous NoCs in four major modules: arbitration, transport, routing and synchronization. The aim of this classification is to help the automatic synthesis of arbiters and of asynchronous interconnect networks using TAST. The basic modules of these communication networks are specified in CHP (Communicating Hardware Processes) language. CHP is a high-level modelling language adapted to describe and to synthesize asynchronous circuits. Through these modelling, the proposed methodology throws into relief arbitration and synchronization problems between concurrent elements of the system. Also, a communication system case-study is presented to illustrate the asynchronous NoC design methodology and its current automation level.

Laboratoire TIMA, 46 avenue Félix Viallet.
38031 Grenoble Cedex, France.

ISBN : 2-84813-042-3
ISBNE : 2-84813-043-1