Plate-forme de prototypage rapide fondée sur la synthèse de haut niveau pour applications de radiocommunications.

Pierre Bomel

N° d'ordre : 45

THESE

Pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITE DE BRETAGNE SUD

Spécialité : Sciences de l'ingénieur

Mention : Electronique et Informatique Industrielle

PIERRE BOMEL

Plate-forme de prototypage rapide fondée sur la synthèse de haut niveau pour applications de radiocommunications

Soutenue publiquement le 21 décembre 2004

COMPOSITION DU JURY

E. Boutillon	LESTER, Université de Bretagne Sud	Directeur de thèse
M. Jezequel	TAMCIC, ENST Bretagne	
J.-D. Legat	DICE, Université Catholique de Louvain	Rapporteur
E. Martin	LESTER, Université de Bretagne Sud	
F. Nouvel	IETR, INSA	
M. Renaudin	TIMA, INPG de Grenoble	Rapporteur
P. Tortelier	France Telecom R&D	

Laboratoire d'Electronique des Systèmes Temps Réel
Université de Bretagne Sud

Remerciements

Je remercie tout particulièrement Monsieur Eric Martin, Professeur des Universités à l'Université de Bretagne Sud, Directeur du LESTER et Président de l'Université de Bretagne Sud pour m'avoir accueilli dans son équipe en tant que doctorant et ingénieur de recherche. Je lui exprime toute ma reconnaissance et mon estime pour son dynamisme, sa persévérance, et son altruisme.

Je remercie Monsieur Emmanuel Boutillon, Professeur des Universités à l'Université de Bretagne Sud, pour son encadrement de thèse et la gestion quotidienne des contrats qui ont permis le financement de mon travail d'ingénieur de recherche ces quatre dernières années.

Je remercie Monsieur Marc Renaudin, Professeur des Universités à l'Institut National Polytechnique de Grenoble, pour avoir accepté d'être rapporteur de ce manuscript et pour m'avoir fait l'honneur d'être président du jury.

Je remercie Monsieur Jean-Didier Legat, Professeur à l'Université Catholique de Louvain, pour avoir accepté d'être rapporteur de ce manuscript.

Je remercie Madame Fabienne Uzel-Nouvel, Maître de Conférences à l'Institut National des Sciences Appliquées de Rennes, Monsieur Michel Jezequel, Directeur d'Etudes à l'Ecole Nationale Supérieure des Télécommunications de Bretagne et Monsieur Patrick Tortelier, ingénieur à France Telecom R&D, pour avoir accepté d'examiner ces travaux de recherche.

Je tiens à remercier Monsieur Philippe Coussy, Maître de Conférences à l'Université de Bretagne Sud, pour sa collaboration sans failles et son amicale critique des manuscrits.

Je tiens à exprimer ma gratitude à tous les membres du LESTER, pour leur disponibilité, leur capacité d'écoute et leurs conseils qui ont contribué à nourrir aussi bien mes doutes que mes espoirs et m'ont finalement aidé à mieux comprendre ce qu'est la recherche.

Enfin, je remercie mes collègues de travail Madame Florence Palin, notre secrétaire, et Messieurs Thierry Gourdeaux et Dominique Heller, ingénieurs de recherche, pour leur bonne humeur et leur soutient permanent.

A mes enfants, Lucie et Paul.

Résumé

L'avènement des technologies sub-microniques profondes de fabrication des semi-conducteurs et l'accroissement de la complexité des systèmes intégrables sur une seule puce ont pour conséquence de faire apparaître de nouveaux défis méthodologiques en conception de circuits au niveau système. La réutilisation intensives de composants pré-développés, ou synthétisés à la demande, permet de réduire les temps de développement et donc le coût de conception. Malheureusement, cette réutilisation fait aussi apparaître des chemins critiques sur les pistes métalliques de grandes longueurs qui connectent les composants entre eux. L'optimisation locale des fréquences de fonctionnement de chacun des blocs peut alors être réduite à néant par les mauvaises performances du réseau de communication inter-composants. C'est dans ce contexte que la théorie des systèmes insensibles à la latence (LIS) propose une solution très prometteuse fondée sur un réseau de communication pseudo-asynchrone et des modèles de wrappers de synchronisation qui encapsulent les composants pour les rendre insensibles aux asynchronismes des communications. On doit néanmoins constater que les différentes propositions actuelles d'architectures de wrappers ne sont pas suffisamment performantes en surface et en vitesse pour être exploitées dans toutes les conditions. Cela est particulièrement vrai lorsque les composants ont des latences de calcul importantes et de grandes quantités de données à traiter comme on en trouve communément en radiocommunications numériques.

Nous proposons dans ce mémoire une architecture de plate-forme de prototypage rapide, nommée PALMYRE, pour applications de radiocommunications numériques qui intègre dans sa composante système une nouvelle version de l'outil de synthèse de haut niveau GAUT. Pour cela, nous étudions tout d'abord les contraintes en terme de puissance de calcul et de communication des applications de type DVB-DSNG, puis les différentes méthodologies de prototypage actuellement pratiquées et enfin inventorions les plates-formes de prototypage les plus récentes. Nous retenons la méthodologie de conception/prototypage orientée plate-forme comme fondement et nous nous appuyons sur sa décomposition en plates-formes matérielle, logicielle et système pour guider la conception de notre plate-forme de prototypage rapide. La plate-forme matérielle que nous proposons est constituée de nœuds de calcul de type DSP C6x, de composants programmables de type Virtex et de liaisons point à point capables d'atteindre des débits de l'ordre de 3 Gbit/s. Nous concevons en C++ une interface logicielle (API) pour DSP et en VHDL RTL des interfaces matérielles pour FPGA qui permettent à une architecture mixte DSP/FPGA de communiquer efficacement. Nous caractérisons notre API et proposons une méthodologie de mesure de performances dont le but est de déterminer les conditions optimales (allocation mémoire, taille des paquets, mode de programmation synchrone/asynchrone) de fonctionnement d'un système qui exploite au mieux la plate-forme matérielle.

L'intégration de l'outil GAUT à la plate-forme système permet de synthétiser semi-automatiquement des composants de niveau algorithmique, ou IPs virtuels, qui s'interfacent naturellement au travers de notre API et des interfaces matérielles. Cette intégration dans un flot CAO de niveau système est rendue possible grâce a deux contributions distinctes. Tout d'abord, l'introduction de la théorie des LIS dans l'unité de communication des circuits synthétisés par GAUT autorise la synthèse de composants rapides dont la fréquence n'est pas pénalisée par le réseau de communication. Pour cela, nous proposons un nouveau modèle de wrapper que nous nommons *processeur de synchronisation* et nous prouvons par l'expérience ses meilleures performances en surface et en vitesse par rapport aux meilleures architectures à base de machines d'états finis. Ensuite, nous concevons une nouvelle unité de mémorisation multi-bancs dont le principal bénéfice est le support du pipelining d'algorithme que l'outil GAUT est susceptible de mettre en œuvre lorsque la contrainte de temps est telle que la simple mise en parallèle de plus de matériel ne suffit plus pour tenir une cadence applicative. Cette unité de mémorisation assure, pour les diverses tranches du pipeline, le calcul d'adresse dynamique lors des accès mémoires aux multiples instances des variables qui nécessitent une duplication.

Grâce aux nouvelles unités de communication et de mémorisation, l'outil GAUT est mis en œuvre avec succès dans le contexte de conception du modem DVB-DSNG du projet RNRT ALIPTA, mené conjointement par les sociétés Arexsys, Sacet, Thales Communications, Turboconcept ainsi que l'ENSTB et le LESTER. L'étude approfondie des résultats de synthèse prouve que des gains importants en surface de l'ordre de 90 % et des gains en vitesse de l'ordre de 10 à 30% sont obtenus pour les wrappers grâce à leur implantation sous la forme du *processeur de synchronisation* que nous proposons. Dans le cadre d'une méthodologie de réutilisation intensive d'IPs virtuels, l'optimisation de la surface, la préservation des fréquences optimales des blocs, la composition aisée de chaînes de traitements à base de blocs synchrones et la possibilité de migration vers une solution de type multi-puces (Multi Chip Module) sont les quatre principaux avantages qu'illustre l'intégration de GAUT dans le flot CAO de la plate-forme système PALMYRE.

Abstract

Semi-conductor very deep sub-micron technologies available today and single-die system integration complexity increase raise new methodological challenges in system design activities. Intensive reuse of pre-developed, or synthesized on demand, components reduce development time and thus design cost. Unfortunately, this reuse paradigm creates critical paths on long metallic wires between components. Local frequency optimization of each bloc can be lost when the inter-component communication network has poor performances. The theory of latency insensitive systems (LIS) recommends in this context a very promising solution based on a pseudo-asynchronous communication network and on synchronization wrapper models which encapsulate components and make them robust (insensible) to the communication asynchronisms. Nevertheless, one must state that the different wrapper architecture proposals are not speed and area efficient enough to be deployed in all conditions. This is particularly true when components have long computation latencies and process huge amount of data as we commonly find in digital radio-communications.

We propose in this work a rapid prototyping platform architecture named PALMYRE. It is dedicated to digital radio-communications and integrates into its system platform part a new version of the high-level synthesis tool GAUT. We first study computing and communication constraints for DVB-DSNG applications. Secondly, we survey the most recent prototyping methodologies and we take a count of the current prototyping platforms in use. We retain the platform based prototyping/design methodology as the most sound basis and rely on its tree steps architecture (hardware, software and system platforms) to guide our platform design. The platform we propose is composed of computing nodes (C6x DSPs and VirtexE FPGAs) and point to point communication links able to reach a sustained bandwidth of 3 Gbit/s. We develop a C++ API for the DSPs and VHDL hardware interfaces which allow a mixed DSP/FPGA prototype to efficiently communicate between nodes. We also propose an API performances characterization method enabling to determine best running conditions in term of memory, packet size and communication programming style (synchronous vs asynchronous).

The integration of GAUT into the system platform allows to semi-automatically synthesize components specified at the algorithmic level. These are also called virtual IPs. They naturally communicate through our API and hardware interfaces and exploit the computing and communication resources from the hardware and software platforms. This integration into a CAD flow is possible thanks to two distinct contributions. First, we introduce the theory of latency insensitive systems inside the communication units synthesized by GAUT. It allows to preserve the local frequency optimizations of components when designing a whole system with synthesized IPs. To reach this objective we present a new wrapper model and call it a

synchronization processor. We prove experimentally its better speed and area performances compared to the current best finite state machines architectures of wrappers. Then, we design a new multi-banks memory unit which main benefit is to support the algorithmic-level pipelining introduced by GAUT when simple hardware parallelism is not sufficient to sustain an applicative sampling cadency. This memory unit handles data transfers for all pipeline stages and dynamic address computation while accessing the different instances of duplicated variables.

Thanks to these new communication and memory units, GAUT is successfully used in a project targeting the design of a DVB-DSNG modem. This is the RNRT ALIPTA project. The companies Arexsys, Sacet, Thales Communications and Turboconcept whith the ENSTB and the LESTER have worked on several digital IPs and validated their integration into an existing DVB-DSNG processing chain. A close study of synthesis results proves that up to 90% of area savings and from 10 to 30% of frequency increase can be obtained when the wrappers are implemented with our *synchronization processors*. We conclude that, in the context of a system design methodology based on intensive virtual IP reuse, area reduction, optimal frequencies preservation, easier composition of processing chains based on synchronous blocs and possibility to migrate to multi-chips modules solutions are four key advantages enabled by the integration of GAUT in the PALMYRE system platform.

Table des matières

Table des figures

Table des tableaux

Introduction

Actuellement, des systèmes complets, contenant une partie logicielle et une partie matérielle, sont intégrés sur une même puce nommée système sur silicium (SoC ou System on a Chip). Ce haut degré d'intégration est permis par l'évolution des technologies sub-microniques profondes (VDSM ou Very Deep Sub-Micron) de fabrication des semi-conducteurs qui, sans cesse, repoussent les limites de finesse de gravure des transistors et de vitesse de fonctionnement. Ainsi, quelques dizaines de millions de portes logiques, contenant des transistors capables de commuter à des fréquence de l'ordre du GigaHertz, sont-elles intégrables sur des puces qui n'excèdent pas 100 millimètres carrés. La contrepartie de cette intégration à l'échelle nano-métrique est que la complexité des systèmes qui sont maintenant concevables croît exponentiellement. En conséquence, la maîtrise de la vitesse de fonctionnement, de la consommation et du temps de développement des SoCs sont devenus plus hasardeux et beaucoup plus complexes. De nouvelles méthodologies de conception apparaissent donc. Elles sont d'un niveau d'abstraction plus élevé que la conception dite de niveau transfert de registres (RTL ou Register Transfert Level). Il s'agit de la conception au niveau système qui préconise la réutilisation intensive de composants pré-développés (blocs durs techno-dépendants, netlist logiques, ou description RTL) ou synthétisés à haut niveau (IP virtuels de niveau algorithmique) plutôt que leur développement.

Dans ce contexte méthodologique, dont le principal but est de réduire les temps de conception de nouveaux systèmes complexes monopuces, la réutilisation de composants pré-développés sur une même puce nécessite de mettre en place des réseaux de communications inter-composants de grandes tailles. Les interconnexions physiques entre ces composants dans les SoCs sont la cause d'une dégradation de la vitesse maximale de fonctionnement : les temps de propagation des signaux sur les pistes métalliques introduisent des chemins critiques. L'optimisation locale des composants est donc réduite à néant par le seul fait de les réutiliser sur des puces de grandes dimensions. De nombreuses méthodologies d'optimisation de la fréquence des circuits synchrones existent à différents niveaux d'abstraction qui vont du niveau technologique au niveau système. Nous en faisons un inventaire critique. C'est au niveau système que la méthodologie des systèmes globalement asynchrones et localement synchrones (GALS), et en particulier son dérivé pseudo-asynchrone la méthodologie des systèmes insensibles à la latence (LIS ou Latency Insensitive Systems), apportent de nouvelles solutions pour briser les chemins critiques inter-composants. La preuve formelle de l'équivalence fonctionnelle entre circuits synchrones et circuits LIS permet en outre de mettre en œuvre une méthodologie de conception au niveau système qui repose, d'une part, sur l'insertion d'unités de mémorisation le long des interconnexions afin de pipeliner le transit des données entre les blocs et ainsi briser les chemins critiques et, d'autre part, sur l'encapsulation des composants synchrones dans des wrappers de synchronisation qui doivent les rendre insensibles aux latences de transfert introduites ainsi qu'aux irrégularités applicatives des flots de données. Nous étudions les propositions actuelles

qui sont fondées sur des implantations à base de logique combinatoire, de machines d'états finis et de registres à décalages et montrons que les meilleures solutions actuelles d'encapsulation des composants dans des wrappers ne sont pas suffisamment efficace en terme de surface et de vitesse.

Nous proposons, dans le cadre du projet PALMYRE inscrit au contrat de plan Etat/Région, une architecture de plate-forme de prototypage rapide qui permet de valider les circuits ainsi obtenus à l'aide de prototypes d'applications de radiocommunications numériques de nouvelle génération.

Pour cela, nous étudions les contraintes en terme de puissance de calcul et de communication des applications de type DVB-DSNG (Digital Video Broadcast-Digital Satellite News Gathering) et concluons que des nœuds de calcul d'une puissance de l'ordre de 1 Gops et des canaux de communication inter-noeuds de calcul d'un débit de l'ordre de 1 Gbit/s sont nécessaires pour bâtir une plate-forme matérielle susceptible d'exécuter un prototype en temps-réel. Nous répertorions les méthodologies de prototypage rapides : prototypage virtuel, prototypage fondé sur l'émulation, prototypage dédié, prestations de fonderie rapides, prototypage analogique, prototypage semi-dédié et prototypage/conception orienté plate-forme. Nous retenons la méthodologie de prototypage orientée plate-forme comme fondement et nous appuyons sur sa taxonomie en trois volets (plate-forme matérielle, plate-forme logicielle et plate-forme système) pour guider la conception de notre plate-forme de prototypage rapide. Nous inventorions alors les plates-formes de prototypage actuelles et faisons une analyse critique de leur qualités avant de proposer la nôtre et de justifier ses meilleures performances. Notre plate-forme de prototypage est constituée de nœuds de calculs de type DSP C6x de Texas Instruments et de FPGA de Xilinx connectables entre eux à l'aide de liens de communication ayant des bandes passantes qui vont de 160 Mbit/s à 3,2 Gbit/s. Nous concevons et développons une API (Application Programming Interface) qui permet de construire une architecture mixte DSP/FPGA dans laquelle tout type de nœud de calcul peut communiquer avec ses voisins. Nous caractérisons cette API et proposons une méthodologie de test d'API qui permet de déterminer les conditions optimales en terme de place mémoire, de tailles de paquets et de style de programmation (API synchrone / asynchrone) pour tout circuit synthétisé dont seul la latence de traitement serait connue.

Afin d'obtenir des circuits plus performants en vitesse qui s'auto-adaptent naturellement aux asynchronismes applicatifs, nous proposons aussi dans ce mémoire d'intégrer la théorie des systèmes insensibles à la latence dans l'outil de synthèse de haut niveau GAUT (Générateur Automatique d'Unité de Traitement) du LESTER. GAUT est un outil de synthèse d'unité de traitement sous contraintes de cadence applicative, de surface et de consommation qui s'intègre dans la plate-forme système de notre plate-forme PALMYRE. Pour cela nous avons ajouté à la synthèse de GAUT deux unités nommées unité de communication et unité de mémorisation.

Nous proposons d'intégrer dans l'unité de communication un modèle de wrapper de synchronisation dont l'implantation RTL est une machine d'états finis concurrente avec chemin de données (CFSMD ou Concurrent Finite State Machine with Datapath). Cette machine de trois états a un comportement qui dépend de la lecture d'une séquences d'opérations stockée dans une mémoire ROM. Nous nommons ce wrapper *processeur de synchronisation* par analogie avec l'interprétation d'un micro-code par un micro-processeur. Nous étudions la

complexité des scénarios de communications que l'on retrouve lors de l'exécutions d'algorithmes de traitement du signal et de l'image tels les transformées de Fourier discrètes, décimées en temps et en base 2 (DFT, DIT radix 2) et les transformées de cosinus discrètes (DCT 8x8). Nous vérifions par synthèse la synthétisabilité des meilleurs wrappers de synchronisation proposés actuellement ainsi que leurs surfaces et vitesses. Nous comparons alors les performances de notre processeur de synchronisation et observons que des gains de 90% en surface et 40% en fréquence peuvent être obtenus. Nous validons ce dernier point avec des expériences de synthèse de processeurs de synchronisation pour circuits de type décodeurs de Viterbi 64 états et Reed-Solomon pour chaînes de réception DVB-DSNG.

Nous proposons aussi d'intégrer une unité de mémorisation multi-bancs qui supporte le pipelining au niveau algorithmique que l'outil GAUT est susceptible de mettre en œuvre. Cette unité de mémorisation permet, à chaque cycle, de faire transiter des données d'un banc mémoire vers un à plusieurs bus et d'un bus vers un à plusieurs bancs mémoires. A l'aide d'un formalisme simple nous décrivons quelles sont les situations particulières concernant la durée de vie des variables en mémoire introduites par le pipelining d'algorithme. Ce sont la duplication de variables, la duplication de variables rebouclées et l'allongement des vecteurs vieillissants. Nous spécifions alors une architecture d'unité de mémorisation à base de DPRAM pilotées par un unique générateur d'adresses qui tient compte de la tranche de pipeline pour chaque instance de l'algorithme en cours d'exécution dans le circuit et l'intégrons à GAUT.

Enfin nous appliquons l'outil GAUT, agrémenté de la synthèse des nouvelles unités de communication et de mémorisation, à la synthèse d'IPs dans le cadre du projet RNRT ALIPTA. Ce projet concerne la conception, l'intégration et la validation des blocs d'une chaîne de réception DVB-DSNG auxquels ont participé les sociétés Arexsys, Turbo-Concept, Sacet et Thales Communications ainsi que l'ENSTB. Ces expériences démontrent l'intérêt et l'applicabilité de nos travaux dans un cadre préindustriel qui nécessite la coopération de chercheurs, de concepteurs d'IP virtuels et d'intégrateurs systèmes.

Plan du mémoire

Le premier chapitre est consacré à la problématique générale de conception des SoCs avec les dernières technologies VDSM. Il met en évidence le formidable défi qui consiste à maîtriser la « giga-complexité » des nouveaux circuits à l'échelle nano-métrique de fabrication des transistors.

Dans le deuxième chapitre, nous dressons un inventaire complet des méthodologies d'optimisation de la vitesse de fonctionnement des circuits numériques. Nous explorons les différents niveaux d'abstraction en partant du niveau technologique pour aboutir à un niveau système, qui se révèle alors être le meilleur candidat pour apporter des solutions novatrices et susceptibles de briser les chemins critiques introduits par la réutilisation de composants pré-développés. Ce sont les méthodologies GALS et LIS dont nous faisons une analyse critique des dernières propositions.

Les troisième et quatrième chapitres présentent respectivement notre contribution en terme, d'une part, de conception d'une plate-forme de prototypage rapide d'applications de radiocommunications numériques et, d'autre part, d'intégration de l'outil GAUT dans la plate-forme système. Une étude des méthodologies de prototypage nous conduit à sélectionner la méthodologie de prototypage/conception orientée plate-forme et à la doter d'une API dont nous caractérisons les performances. L'intégration de l'outil GAUT repose sur l'insertion de la théorie des LIS dans l'unité de communication et sur l'ajout du support du pipelining d'algorithmes dans l'unité de mémorisation.

Le cinquième chapitre est une présentation de la mise en œuvre de ces travaux dans le cadre de la conception du modem DVB-DSNG du contrat ALIPTA. Nous concluons que l'inclusion de la théorie des LIS et le support du pipelining d'algorithme en synthèse de haut niveau permettent de concevoir, de valider et d'intégrer, dans un contexte de développement de produits de la gamme des SoCs, des circuits numériques complexes par réutilisation d'IPs virtuels de niveau algorithmique.

Enfin, dans un dernier chapitre, nous concluons ce mémoire en présentant les apports de nos travaux et dégageons les perspectives qu'ils suggèrent.

Chapitre 1

Méthodologies de conception des SoCs

1. Introduction

L'évolution des technologies de fabrication des semi-conducteurs a atteint le stade dit sub-micronique profond (à partir de 0,18 µm), ou VDSM (Very Deep Sub-Micron), et permet un degré d'intégration encore jamais atteint. Les circuits ainsi conçus sont nommés SoC (System on a Chip) ce qui signifie systèmes monopuces. Ils contiennent de 5 à 10 millions de portes logiques, leur conception dure de l'ordre de 18 à 24 mois, coûte de 10 à 20 millions d'euros (ou de dollars) [ARV04] et leur marché est en pleine expansion.

Les technologies VDSM ont l'inconvénient d'introduire dans la conception de circuits complexes deux nouvelles contraintes technologiques : 1) le temps de traversée des signaux peut largement dépasser le temps de calcul combinatoire et allonger les périodes d'horloges et 2) l'arbre d'horloge devient le principal consommateur d'énergie.

Enfin, de nouvelles méthodologies de conception au niveau système sont nécessaires pour dominer la complexité et l'hétérogénéité des SoC, conserver à peu près constant le temps de développement des nouveaux produits [GAR03] et stopper l'accroissement du différentiel de productivité en conception (design gap). Ce « design gap » est dû à une augmentation du nombre de transistors disponibles de 58% par an alors que la productivité des ingénieurs n'augmente que de 21 % pendant la même durée [KHN01][MAT01].

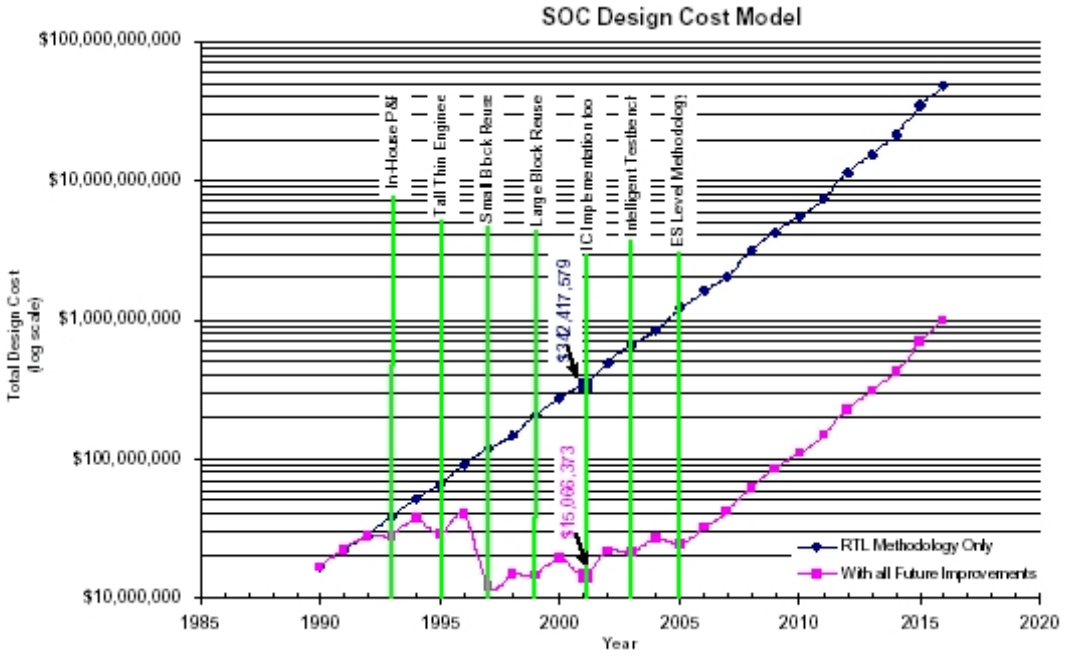


Figure 1-1 : Coût de développement en fonction de la méthodologie [KHN02]

La Figure 1-1 représente l'évolution du coût de développement d'un SoC en fonction de l'année et de la méthodologie de conception employée. Les deux courbes illustrent les évolutions prévues avec et sans améliorations de ces dernières : on observe sur la courbe du haut la tendance inflationniste du coût lorsque la méthodologie RTL est employée seule alors que la courbe du bas présente l'évolution avec l'adjonction régulière de nouvelles méthodologies. Les

méthodologies successives permettent de maintenir le coût de développement en dessous de quelques dizaines de millions de dollars jusqu'en 2005, puis la tendance à la hausse reprend à partir de 2005 si aucune amélioration n'est apportée au niveau système (ESL ou Electronic System Level).

Les méthodologies de conception successives présentées par [KHN02] sont :

- *RTL methodology* : est la méthodologie RTL qui a supplanté la méthodologie de conception au niveau portes (*gate level design methodology*).

- *In-house place and route* : est le rapatriement des services de placement et routage (synthèse physique) chez le concepteur au détriment du fondeur afin de réduire les durées d'itération entre le client et le fournisseur.

- *Tall thin engineer* : est le remplacement du découpage des équipes par spécialité « outil » par un nouveau découpage par fonction « produit ». L'ingénieur est qualifié de grand (*tall*) quand il couvre toutes les étapes de conception (top-down) et fin (*thin*) quand il couvre une part fonctionnelle réduite du produit.

- *Small block reuse (2500 à 25000 portes) et large block reuse (25000 à 100000 portes)* : sont la banalisation de la réutilisation de blocs durs. Une méthodologie *very large block reuse (au-delà de 100000 portes)* est attendue vers 2010.

- *IC implementation tools* : est l'automatisation complète du flot de conception qui va du RTL aux masques (GDSII).

- *Intelligent test bench* : est l'automatisation du flot de vérification fonctionnelle à partir de rétro-annotations issues de la synthèse physique.

- *ESL, Electronic System Level* : est la future méthodologie de niveau système. Elle intègre les outils de co-conception, de co-vérification et de raffinement logiciel et matériel. Cette méthodologie permettra un bon quantitatif comparable a celui déjà fait par la méthodologie RTL par rapport à la synthèse au niveau portes.

Avant de présenter en détails les nouvelles contraintes technologiques et les besoins méthodologiques introduits par les technologies VDSM, nous nous ressituons dans le contexte économique et caractérisons la demande du marché.

2. Les objectifs commerciaux

La croissance de la demande en produits nouveaux amène les concepteurs de circuits à intégrer de plus en plus de transistors sur une même puce. Il est ainsi prévu que les ventes de SoCs, de l'ordre de 160 millions de pièces en 1998 et de 345 millions en 1999, dépassent 1,3 milliards de pièces vendues en 2004. La progression future des ventes devrait atteindre 30% par an. Elle sera essentiellement soutenue par les segments de marché communication (téléphonie mobile, vidéo mobile, PDA, pagers, …), ordinateurs (ordinateurs personnels, ordinateurs portables, …) et domestique (TV numériques, décodeurs, consoles de jeux, électroménager intelligent, HIFI et

ses dérivés portables, …). Le prix moyen (ASP ou Average Selling Price) se situe entre 10,80 et 10,50 dollars par SoC et devrait ne pas baisser grâce à l'augmentation des fonctionnalités intégrées par système. La Figure 1-2 illustre la segmentation actuelle du marché des SoC. Selon la définition retenue des segments « communication », « computer » et « consumer » les pourcentages peuvent différer selon les organismes de prévisions [INS04a][GAR03]. Cependant, toutes les prévisions s'accordent sur le fait que ces trois segments sont déjà, et resteront, les plus importants générateurs de volumes de ventes.

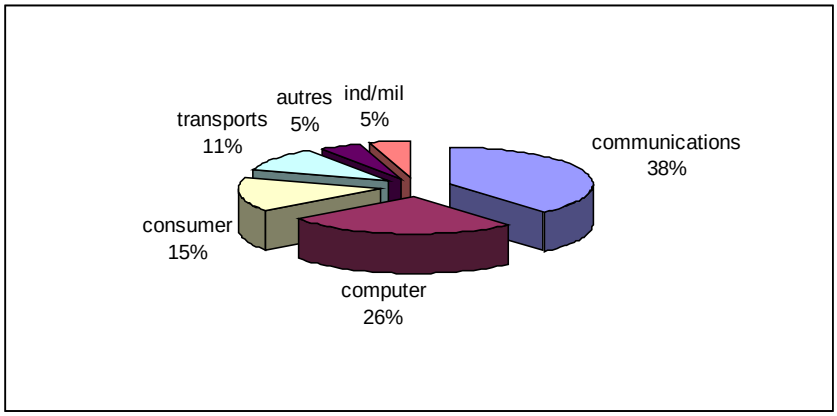


Figure 1-2 : Répartition des segments de marché des SoC

Retenons enfin que le point fondamental pour l'industrie des semi-conducteurs est que l'augmentation des fonctions intégrées sur un SoC est le meilleur moyen de lutte contre l'érosion du prix de vente de ses produits. Les SoCs permettront à l'industrie des semi-conducteurs de préserver ses marges, à condition que le coût et la durée de développement des SoCs restent à peu près constants.

3. Evolution et contraintes technologiques

Depuis quarante ans l'industrie des semi-conducteurs s'est particulièrement distinguée par son aptitude à améliorer continûment ses produits. Les principaux types d'améliorations sont l'augmentation du niveau d'intégration, de la rapidité, de la compacité et des fonctionnalités ainsi que la réduction du coût et de la consommation par fonction. La plupart de ces tendances fortes sont le résultat de la réduction de la finesse de gravure des nouvelles technologies de fabrication des circuits intégrés. La conséquence la plus visible de cette réduction de surface des composants électroniques élémentaires sur silicium (réduction des dimensions W et L du canal des transistors et de la distance minimale entre deux pistes de même niveau de métal) est que le niveau d'intégration double approximativement tous les vingt-quatre mois. Cette observation, nommée « loi de Moore[1] », date de décembre 1975 et reste d'actualité dans les prévisions de l'ITRS [ITR03] pour les quinze prochaines années. Les courbes de la Figure 1-3, extraites de ces prévisions, montrent la tendance de l'évolution des technologies de fabrication des DRAM et des MPU (Multi Processors Unit). La courbe du haut est la tendance décroissante de l'écart

[1] La loi de Moore affirme que le niveau d'intégration est multiplié par 2 tous les 18 à 24 mois. D'où ses différentes formulations possibles « x4 tous les 3 ans » et « x2 tous les deux ans ».

minimal entre deux lignes de métal adjacentes au premier niveau de métallisation (M1 ½ pitch) dans les MPU tandis que la courbe du bas est la tendance décroissante de la distance minimale entre deux colonnes mémoires (DRAM ½ pitch) adjacentes. Des finesses de gravure inférieures à 0,10 µm sont prévues en production à partir de 2004-2005 et dix années plus tard les 0,03-0,02 µm de finesse devraient être atteints.

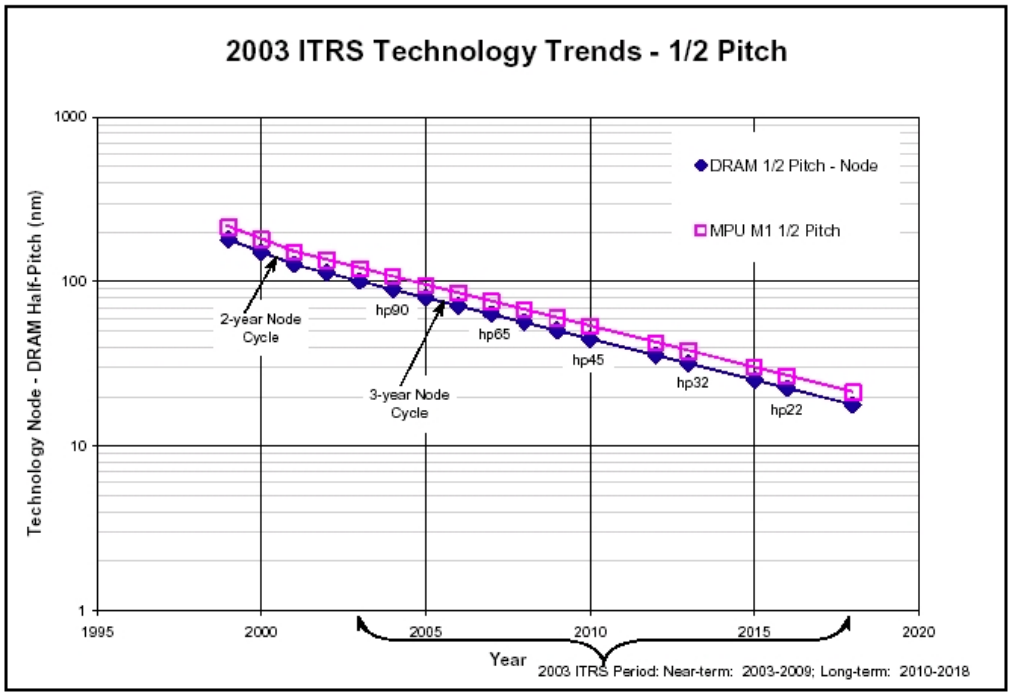


Figure 1-3 : Evolution prévisionnelle des technologies [ITR03]

Il faut néanmoins savoir que la loi de Moore n'est réaliste que si le marché peut « consommer » ce qui est concevable et ensuite produit. C'est d'ailleurs en ce sens qu'il faut l'interpréter. Ainsi cette loi pragmatique soutient qu'il y a une corrélation forte entre l'augmentation du marché des semi-conducteurs de 17% par an et la réduction du coût par fonction de 25% sur la même durée. La réduction de la surface des composants électroniques élémentaires serait donc une condition « sine qua non » du maintient du marché des semi-conducteurs.

Brièvement présenté ainsi, l'évolution des semi-conducteurs serait un sujet clos et circonscrit au domaine exclusif des technologues du silicium. Il n'en est rien car, du transistor au SoC, l'augmentation de l'intégration possible et donc du nombre de fonctionnalités sur une puce rend le flot de conception (spécification, raffinement, vérification) de plus en plus long et complexe. En l'absence de nouvelles méthodologies et d'outils de conception de SoCs les temps de développement et de vérification de ces derniers augmentent avec la complexité des applications et de leurs scénarios de mise en œuvre.

Compte tenu de l'allongement des temps de conception, la mise en production d'une nouvelle technologie ne garantit nullement une mise sur le marché de nouveaux produits issus de cette dernière avant qu'elle ne soit dépassée par une nouvelle encore plus performante. En effet, la durée inter-technologies pendant laquelle la dernière technologie domine les précédentes n'est que de l'ordre de deux années à court terme (2003 à 2009) et de l'ordre de trois années à long terme (2010 à 2018). Dans ces conditions, les temps d'étude de marché, d'étude de faisabilité,

de conception, de réalisation, de caractérisation, de validation, de première introduction sur le marché, puis de production en volume (le Time to Market) sont contraints d'autant si l'on doit absolument affirmer que l'on possède la technologie la plus agressive pour vendre ses produits.

L'augmentation de la durée de développement d'un SoC est-elle un frein définitif à l'exploitation et donc à l'apparition de nouvelles technologies de fabrication des semi-conducteurs ? Probablement pas car on constate que les procédés de fabrication de semi-conducteurs ont une durée de vie très supérieure au délai d'apparition d'une nouvelle technologie, comme l'indique la Figure 1-4 extraite d'une présentation de la SeMaTech [SEM04]. La durée de vie d'une technologie est de l'ordre de neuf à douze années, soit le temps d'introduction de trois à quatre nouvelles technologies avant obsolescence.

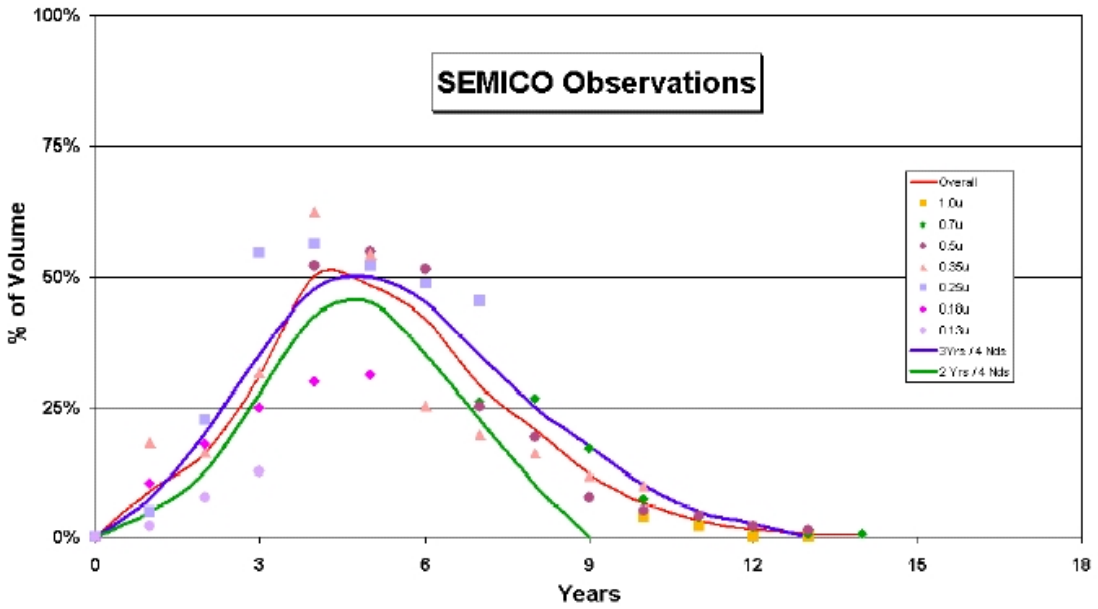


Figure 1-4 : Durée de vie moyenne des technologies [SEM04]

La Semiconductor Industry Association [SIA03] (SIA) à la disposition du public des statistiques détaillées au sujet des capacités des usines de semi-conducteurs. Ce sont les Semiconductor Industry Capacity Supply Statistics (SICAS). Le dernier rapport « Statistic Report – 3rd Quarter 2003 » [SIC03] illustre très bien ce fait en décrivant un groupe de technologies MOS de taille supérieure ou égale à 0,7 µm alors que les technologies les plus récentes sont de 0,12 µm pour les processeurs et les ASIC et de 0,10 µm pour les DRAM. La Figure 1-5 montre que les technologies supérieures ou égales à 0,7 µm représentent un volume exprimé en tranches de silicium traitées par semaine (Wafer Starts per Week) sensiblement supérieur à celui de toutes les technologies plus agressives. A titre d'exemple de cohabitation de plusieurs technologies dans une même « fab », nous citons l'usine Nantaise de la société Atmel [ATM04] dont les cinq technologies actuellement en cours de production vont du BICMOS 1 µm, au CMOS 0,35 µm.

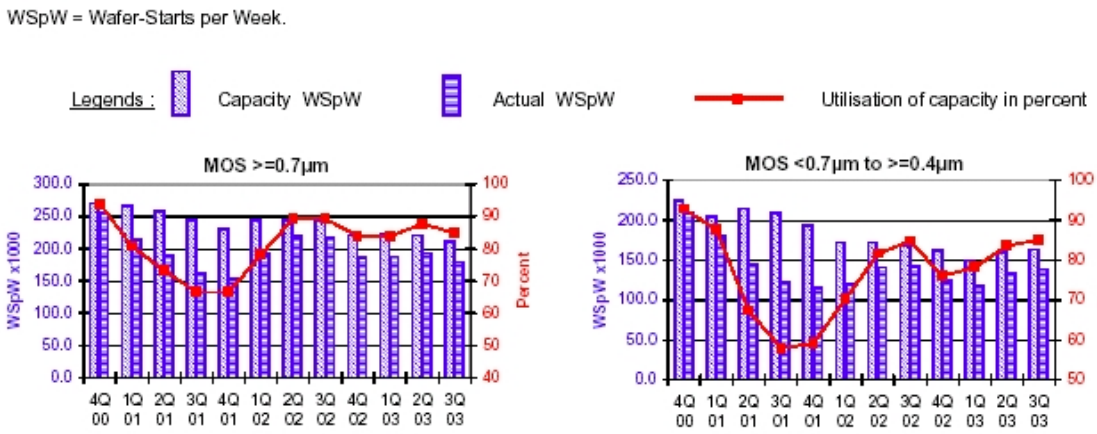


Figure 1-5 : Capacités et volumes de wafers produits par technologies [SIC03]

Compte tenu de l'évolution des technologies de fabrication, les nouvelles contraintes sont maintenant dues à la prédominance des temps de propagation des signaux. L'impact est double. Tout d'abord, la fréquence maximale est limitée par le routage des plus longues pistes métalliques et, enfin, la synthèse d'arbres d'horloges est fortement handicapée par un décalage (skew) qui peut devenir supérieur à la période de l'horloge.

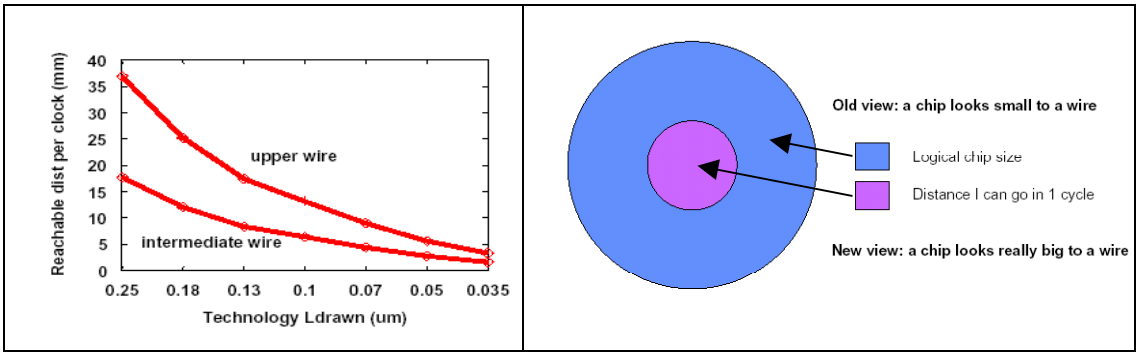


Figure 1-6 : Distance parcourue par un signal en un cycle d'horloge [HO99]

La Figure 1-6 illustre ce phénomène :

- quantitativement à l'aide de courbes qui représentent l'évolution des temps de propagation sur des lignes « moyennes » (intermediate wires) intra-composants et « longues » (upper wire) inter-composants

- et qualitativement à l'aide de cercles concentriques qui représentent les zones accessibles à un signal qui serait issu du centre d'une puce circulaire en un seul coup d'horloge.

La puce « standard » a une surface de référence de 140 mm2 [ITR03], soit environ 12 mm de côté. Depuis 1999 [HO99] constate qu'en technologie 0,6 µm la distance parcourue par un signal pendant une durée correspondant à la commutation d'une porte CMOS est de l'ordre de 5 mm, soit moins de la moitié du coté de la puce. Les prévisions, pour une technologie de 0.06 µm, sont que la distance sera de l'ordre de 1,5 mm. Le rapport entre la longueur du coté de

la puce et le temps de traversée pendant une commutation d'une porte logique ne fait que s'accroître. Pour garantir qu'un signal puisse traverser la totalité de la puce pendant un cycle d'horloge il faut alors diminuer la fréquence de l'horloge. *Cette conséquence est en totale opposition avec les capacités en fréquence des nouvelles technologies.* Une autre métrique fréquemment employée pour qualifier ce phénomène est le nombre de millimètres qu'un signal pourra parcourir par période de l'horloge la plus rapide. Pour parcourir la totalité du circuit, la période d'horloge minimum ne pourra donc pas être inférieure à plusieurs dizaines (voire centaines) de commutations de portes logiques pour pouvoir parcourir la totalité de la puce [MAT97] [HO01].

Quand à la consommation de l'arbre d'horloge, de nombreuses études sur les processeurs hautes performances chez Intel ont prouvé très tôt [TIW98] que plus de la moitié de la consommation du système peut lui être imputée. A nouveau, tout comme pour les temps de propagation, les solutions ne tenant pas compte du routage ne suffisent plus. Il faut reprendre le sujet à la base et supprimer le problème dans son ensemble : c'est à dire ne plus utiliser d'arbre d'horloge unique pour l'ensemble du circuit.

Ayant présenté l'évolution et les contraintes des nouvelles technologies de fabrication des circuits intégrés, nous présentons maintenant les méthodologies qui ont, jusqu'à présent, lutté contre l'affaiblissement de la fréquence d'horloge lorsque la taille du circuit augmente.

4. Solutions méthodologiques établies

Bien sûr, il existe déjà des méthodologies qui luttent contre les chemins critiques combinatoires à différents niveaux d'abstraction. Les plus novatrices d'entre-elles ne se limitent pas à l'optimisation des synthèses logique et physique : elles introduisent l'optimisation en fréquence au niveau architectural ou système. Elles appartiennent à quatre classes distinctes : la conception de circuits traditionnelle, la conception de SoCs numériques, la conception au niveau architecturale et l'exploitation du pipelining.

Dans la classe « conception de circuits traditionnelle », on trouvera l'optimisation de la topologie d'interconnexion, le dimensionnement et l'insertion de répéteurs (buffers), le dimensionnement des pistes de métal (wires) et des transistors (drivers) qui maintiennent les valeurs logiques en sortie de portes logiques [CON97a]. Des travaux de recherche en synthèse logique on amélioré les méthodes antérieures qui négligeaient le temps de propagation au profit du temps de commutation. Par exemple, les méthodologies de synthèse logique de [GOS98] et [OTT98] exploitent le temps de propagation et considère comme négligeable le temps de commutation.

Dans la classe « conception de SoCs numériques », de nouvelles approches commerciales de synthèse combinée logique et physique ont fait leur apparition. Elles exploitent l'extraction et la remontée des informations RC des pistes après routage (la rétro-annotation) ou après placement (informations RC d'estimation d'un routage global) pour piloter une à plusieurs phases de re-routage partiel (une sorte de mode ECO automatisé, ou Engineering Change Order). Ce sont les outils Physical Compiler [SYN04], Precision Physical Synthesis (pour FPGA seulement) [MEN04], PKS [CAD04], BLAST [MAG04] et ZenTime [ZEN04]. Ils permettent à des sociétés

telle Tensilica [TEN04] de proposer des IP (physiquement synthétisable avec Physical Compiler) de type processeurs paramétrables VLIW/SIMD DSP pour SoC dont la taille est de l'ordre de 200.000 portes logiques, qui fonctionnent à une fréquence de 250 MHz, et sont produits en technologie 0,13 μm.

Dans la classe « conception au niveau architectural » de nombreux auteurs proposent d'intégrer le coût des interconnexions à la synthèse d'architecture. [MOS96] et [XU97] partitionnent l'ensemble des opérations et effectuent un préplacement avant d'allouer les ressources matérielles et de les interconnecter entre elles. L'estimation de la longueur des interconnexions sans pré-placement est un point délicat addressé par [MEC96] et [HAL98]. Ces méthodes, pour être applicables, surestiment le coût des interconnexions et [JEG00] propose d'estimer plus finement le coût des interconnexions en fonction de la caractérisation de la localité des données pour un modèle d'architecture cible particulier.

Dans la classe « pipelining » on retrouve les architectures de processeurs pipeliné (jusqu'à 5 étages comme les architecture SPARC et MIPS [HNN94]), super-pipeliné ou hyper-pipelinés (c'est à dire au-delà de 5 étages selon la terminologie d'Intel). Pour briser les longues équipotentielles ce sont, par exemple, les 31 étages du pipeline de la micro-architecture Netburst hyper-pipelinée du pentium 4 [GLA00]. Au niveau comportemental, le pipelining est soit fonctionnel (algorithme pipeliné), soit structurel (opérateurs pipelinés). Très tôt, SEHWA [PAR88] fut un des premiers outils de synthèse qui partitionne un graphe acyclique en étages séparés par des barrières de registres. Enfin, au niveau système, l'introduction de pipeline dans la communication permet de briser les chemins critiques issus de la complexité des SoCs. Ce sont les travaux orientés NoC [BNI02a/b] et les systèmes insensibles à la latence [CAR02].

Les trois premières classes de méthodologies de réduction des chemins critiques sont, tôt ou tard, limitées par la longueur des longues pistes métalliques que l'on trouve dans les SoCs. C'est donc dans la quatrième catégorie (pipelining) que se situent les évolutions dont le but ne consiste plus à optimiser (avec ou sans estimations et/ou rétro-annotation) mais à supprimer les chemins critiques. Une nouvelle approche consiste donc à découper le système en blocs synchrones localement plus performants et à éliminer les longues lignes de métal qui les relient entre eux et qui sont la cause des chemins critiques désastreux dans le système.

Enfin, remarquons que ces phénomènes sont présents aussi bien sur les puces des FPGA que dans les ASIC. Alors que les concepteurs d'ASIC disposent de méthodologies de réduction des chemins critiques en conception, les concepteurs de circuits sur FPGA subissent la plus ou moins grande aptitude à préserver la fréquence maximale de fonctionnement des outils de configuration du réseau de communication pré-routé. Les sociétés Xilinx et Altera proposent des architectures de réseau de communication très différentes : une architecture hiérarchique à trois niveaux chez Altera et une architecture dite « segmentée » chez Xilinx. Avec chacune de ces architectures, la capacité des outils à rapprocher physiquement sur la puce les éléments logiques qui communiquent le plus entre eux est déterminante.

Type de connexion	**Pourcentage**
Local	45
Saut de 2 CLB	30
Saut de 6 CLB	20
Lignes longues	5

Tableau 1-1 : Répartition moyenne des types de connexions

Par exemple, les expériences faites par [SHA02] lui permettent de constater avec des VirtexII (2v3000 à 2v6000 remplis au maximum) que chez Xilinx 1) la localité est bien préservée par l'outil de placement-routage, 2) les temps de propagation d'un même composant sont beaucoup moins dispersés que chez Altera et 3) le nombre d'arbres d'horloges « locaux » est moindre. Sur le Tableau 1-1 on constate que 75 % des connexions sont faites en local (CLB adjacents) ou sont peu distantes (saut de 2 CLB). Cela prouve que l'outil de synthèse XST a su préserver la localité d'au moins 75 % du circuit.

5. Les nouveaux besoins méthodologiques

Ainsi, le besoin d'outils et de méthodes de conception de SoC, au niveau système est bien réel et l'ITRS les recense régulièrement. La dernière version (2003) de ces prévisions montre que ces besoins sont, à court terme, plus liés à l'accroissement de la complexité, de la rapidité et de la réduction de la consommation des SoCs qu'à la diminution des tailles des structures physiques. Ces besoins sont listés dans le Tableau 1-2. De même, les besoins à long terme sont :

- Modéliser la sensibilité au bruit : les tensions diminuent, donc le niveau de bruit relatif augmente. En particulier le bruit de commutation et les variations de tension associées aux brusques appels de courant.

- La probabilité d'obtenir des circuits au fonctionnement parfait décroît. La finesse de gravure de la technologie rend le système plus sensible à la diaphonie et à l'électro-migration. Il faut donc développer des techniques de détection et d'auto-correction d'erreurs. La reconfigurabilité est une piste envisagée.

[JER02a, JER02b] (un ouvrage auquel nous avons participé pour le compte de l'un des auteurs), présente un état de l'art actualisé et structuré autour de la conception de circuits électroniques monopuces. Les co-auteurs confirment que la tendance SoC multi-processeurs ne cesse d'augmenter (à partir de 2005 70% des ASIC comporteront au moins une CPU) et expliquent que « le goulet d'étranglement du processus de conception devient maintenant la réalisation du réseau de communication des composants embarqués sur la puce et la validation globale de l'ensemble du système avant sa réalisation ». De multiples points de cassure apparaissent, ce sont l'assemblage de composants hétérogènes, l'élargissement des métiers de conception des

circuits à l'ingénierie du logiciel de base et la nécessité de la synthèse système. Ce sont, à nouveau, les mêmes besoins que ceux de la catégories « orientés conception » de l'ITRS avec en plus l'adjonction du besoin de fourniture des logiciels de base (compilateur, assembleur, linker, drivers, moniteurs embarqués, …) qui permettront d'exploiter les processeurs et les périphériques d'une même puce à un niveau d'abstraction (tel un langage impératif et structuré du type C) autre que l'assembleur et les registres.

Catégorie	Niveaux	**Sujet**
Consommation	Techno	Réduction des courants de fuites des transistors pour les applications majoritairement en attente (la plupart du temps en « stand by »).
	Techno et Système	Contrôle et réduction de la consommation pour les applications faible consommation (sur batteries).
Conception	Architecture ou Système	Amélioration de la productivité en conception par élévation du niveau d'abstraction.
	Système	Banalisation de la réutilisation de circuits tierce partie.
	Système	Conception orientée « plate-forme » (platform-based design).
	Architecture	Synthèse logicielle et matérielle de l'interface entre logiciel et matériel.
Simulation	Techno	Meilleurs modèles électriques aux fréquences élevées de la bande 5-40 GHz.
	Système	Vérification de systèmes de plus en plus complexes.
Test	Physique	Réduction du nombre de broches pour le test.
Boîtier	Physique	Nouveaux boîtiers ayant un très grand nombre d'entrées-sorties et un très grand nombre de broches d'alimentation.
	Physique	Réduction du coût des gros boîtiers qui ne suivent pas la loi de Moore. Si le coût des boîtiers ne diminue pas il peut ruiner les efforts faits pour réduire le coût par fonction sur le silicium.
	Physique	Développement de nouveaux boîtiers dissipant mieux ainsi que de leurs modèles pour estimer leur impact en dissipation, fréquence, consommation, etc …

Tableau 1-2 : Besoins méthodologique, ITRS 2003

L'assemblage de composants hétérogènes est une nécessité pour les méthodologies qui prônent la réutilisation d'IP préconçus. Les IP (Intellectual Property) sont des composants virtuels (ou VC pour Virtual Component) dont le but est de favoriser leur réutilisation intensive. [COU03] présente leur historique, leur évolution, ainsi que la problématique de leur intégration dans un

système complet. La notion de connecteur (wrapper) est utilisée par les méthodologies de raffinement des communications et représente la « colle logique » (logic glue) permettant d'assembler des composants qui sont hétérogènes en termes de niveaux d'abstractions ou de protocoles. Les wrappers sont utilisés dans le cadre de la (co-)simulation (éventuellement multi-niveaux d'abstractions) et de la synthèse système des communications.

L'hétérogénéité de la communication rassemble :

- La notion de canal abstrait [DAV96] qui spécifie des liens de communication monodirectionnels, avec ou sans protocoles, et avec types de données génériques.

- La notion d'interface fonctionnelle (FI ou Functional Interface) [VSI04] qui est un ensemble de transactions du type lire/écrire spécifiant l'interface entre comportement et communication.

- La notion d'interface de composant virtuel (VCI ou Virtual Component Interface) [VSI04] qui propose un jeu de signaux pour un protocole flexible et extensible au niveau cycle.

- La notion de BFM (Bus Functional Model) [RAW94] qui encapsule la spécification d'un comportement dans une enveloppe précise au niveau cycle (CA ou Cycle Accurate).

- La notion de BCASH (Bus Cycle Accurate Shell) [SYS04] [COW04] qui adapte l'appel de procédure à distance (RPC, ou remote Procedure call) à la synchronisation CA.

Afin de simplifier cette taxonomie, [YOO01] limite les niveaux d'abstraction à trois : le niveau système (SL, ou System level) qui représente la communication de paquets de données via des canaux abstraits et des interfaces fonctionnelles, le niveau architecture (AL, ou Architectural Level) qui représente les divers protocoles (FIFOs, files d'attentes, poignée de main, …) et leurs paramètres (taille des FIFOs), et le niveau transfert de registres (RTL, ou Register Transfer level) qui représente l'implantation des protocoles sur un medium de communication particulier. Le niveau SL se concentre sur la plus-value « applicative » de la communication, le niveau AL se concentre sur l'impact du choix du protocole, et le niveau RTL se concentre sur l'implantation physique de ces derniers.

6. Bilan

Nous avons successivement présenté la problématique de conception des SoCs en termes d'objectifs commerciaux, de nouvelles contraintes apportées par l'évolution des technologies de fabrication des circuits intégrés et de nouveaux besoins méthodologiques de conception au niveau système. *Nous concluons que les nouveaux systèmes monopuces sont limités en fréquence maximale et en consommation minimale par leur complexité fonctionnelle.*

D'où la nécessité pour la communauté scientifique de rechercher de nouvelles théories, méthodologies et implémentations originales qui traitent le problème de l'accroissement des durées de transit des signaux numériques ainsi que le problème de la conception de circuits au niveau système et qui seront seules capables de préserver un « Time to market » acceptable par l'industrie des SoCs. De même, pour l'industrie de la CAO micro-électronique, il faut changer de référentiel : proposer de nouveaux outils de conception orientés communication plutôt que calcul qui permettent de traiter indépendamment et de façon complémentaire ces deux aspects du système [CAR02]. Cette séparation des problèmes à traiter est aussi nommée « orthogonalisation des sujets » (orthogonalization of concerns) par la communauté scientifique [KEU00]. C'est dans ce contexte dual d'optimisation de la fréquence et de conception au niveau système que nos travaux s'inscrivent.

7. Plate-forme PALMYRE, objectifs et contributions

Nous proposons dans ce mémoire d'apporter des solutions au problème des fréquences sub-optimales lorsque la cible technologique est un composant programmable de la classe des FPGA ou un ASIC. Les travaux ont eut lieu dans le cadre expérimental de la plate-forme de prototypage rapide PALMYRE [PAL04]. Cette plate-forme matérielle, logicielle et système a pour but de permettre la spécification de chaînes de radiocommunications numériques au niveau système et d'obtenir, par raffinements successifs, un prototype exécutable sur une architecture hétérogène en terme de composants de calcul et de moyens de communication.

Nos travaux recouvrent l'ensemble de la plate-forme. Au niveau plate-forme matérielle, il s'agit de l'étude des besoins applicatifs et de la sélection des composants matériels de calcul (FPGA et DSP) et de communication (liens série et bus) de puissances suffisantes. Au niveau plate-forme logicielle, il s'agit de la sélection et de la maîtrise des outils de développement (compilateurs, outils de synthèse matérielle, système d'exploitation temps-réel, bibliothèques de codes C et VHDL, …) relatifs aux composants matériels sélectionnés précédemment. Au niveau plate-forme système, il s'agit de l'introduction du raffinement d'algorithmes sous contrainte temporelle dans un flot de prototypage rapide. Pour cela nous avons exploité et étendu les fonctions de l'outil de synthèse de haut niveau GAUT [GAU04]. Les composants synchrones qu'il synthétise sont les éléments de base (les IP, ou VC) que nous encapsulons dans des wrappers originaux permettant une intégration plus aisée dans un système complet sans nuire aux fréquences maximales.

Notre contribution en terme de synthèse de haut niveau pour le prototypage rapide est double et se situe au niveau de la communication inter-composants (entrées/sorties externes) et intra-composant (mémoires internes) :

- La méthodologie des « systèmes insensibles à la latence» (LIS ou Latency Insensitive Systems) régule les transferts de données entre composants synchrones. Elle permet de composer des systèmes complexes à partir de composants synthétisés par GAUT à condition qu'ils respectent deux critères de réutilisabilité. Il faut qu'ils soient *suspensibles* et *encapsulés* dans un wrapper que nous qualifions de wrapper de synchronisation. Nos wrappers sont complémentaires aux wrappers de communication précédemment cités dans la mesure où nos composants sont strictement homogènes au niveau d'abstraction RTL et que leur communication est synchrone et sans protocole. Ils ont pour but de découpler les composants alors que les wrappers de communication ont pour but de les accoupler. Nous avons donc intégré la méthodologie LIS à GAUT à l'aide de processus/wrappers de synchronisation implantés sous la forme de machines d'états finis de type CFSMD micro-codées.

- Une architecture mémoire optimisée supportant les débits mémoires intra-composant est aussi requise lorsque les contraintes de temps calcul impliquent le pipelining de l'algorithme. Nous proposons alors une mémoire virtuelle multi-ports (implantée sous la forme de n bancs et p ports multiplexés) dont les générateurs d'adresses tiennent compte du parallélisme des exécutions d'un même algorithme pour accéder aux diverses instances d'une même variable.

Nous concluons ce chapitre en nous resituant par rapport aux besoins méthodologiques identifiés par l'ITRS. Nos contributions se situent donc dans la catégorie intitulée « besoins orientés conception ». Elles ont pour objectif d'améliorer deux facteurs clés qui sont 1) les performances et la réutilisabilité d'IP tierce partie par l'amélioration de l'encapsulation de composants synchrones en environnement insensible à la latence (LIS) et 2) l'extension de la synthèse de haut niveau d'IP de calcul par la génération automatique d'unités de mémorisation et de communication.

La suite de notre mémoire se décompose en 5 chapitres. Le chapitre 2 est un état de l'art sur les systèmes globalement asynchrones et localement synchrones (GALS) et les systèmes insensibles à la latence (LIS). Le chapitre 3 présentent notre contribution selon la taxonomie des plates-formes matérielle, logicielle, puis système. Au quatrième chapitre nous présentons l'introduction de l'outil de l'outil GAUT dans la plate-forme système. Le chapitre 5 présente les résultats applicatifs obtenus. Enfin, nous terminons par une conclusion et listons une série de perspectives que nos travaux suggèrent.

Chapitre 2

Etat de l'art

1. Introduction

Dans le chapitre précédent nous avons situé la problématique de mise en œuvre des composants synchrones dans le contexte des nouvelles technologies sub-microniques profondes et montré que les fréquences de fonctionnement maximales sont pénalisées par les chemins critiques créés par les longues lignes de communication inter-composants dans les SoCs. L'optimisation de la période minimale de fonctionnement dans les SoCs est un sujet abordé par différentes approches méthodologiques dont nous faisons un inventaire critique. Ensuite, le concept des systèmes globalement asynchrones et localement synchrones (GALS) est présenté afin d'en préciser la portée et d'introduire ensuite plus aisément la méthodologie de conception de circuits synchrones dite "des systèmes insensibles à la latence" (LIS). Cette dernière est décrite en détail ainsi que ses dernières optimisations et extensions. A la suite de cette présentation, nous terminons par un exposé des points à améliorer, des objectifs à atteindre et des travaux qui en découlent. Les chapitres suivants présentent notre contribution selon une approche orientée plate-forme qui est suivie par les résultats expérimentaux obtenus.

2. Limites et optimisations de la fréquence de fonctionnement des SoCs

La période minimale de fonctionnement des circuits synchrones est bornée par le chemin critique. Ce chemin critique est défini comme étant le plus long chemin, en terme de temps de propagation des signaux logiques, entre deux bascules. La Figure 2-1 illustre un chemin critique qui part du registre A et rejoint le registre E en traversant successivement un inverseur puis deux portes ET. Il suit la ligne en pointillés. La Figure 2-2 illustre son calcul.

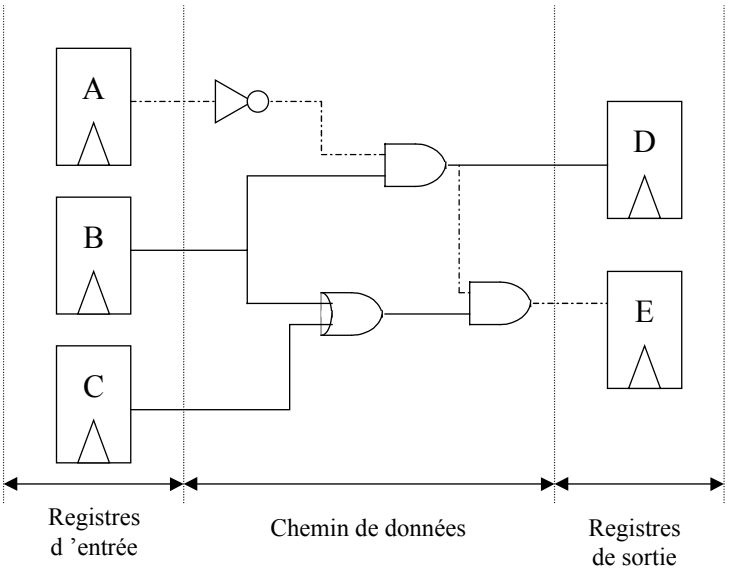


Figure 2-1 : Principe de réalisation des circuits synchrones

[FRI01] définit le chemin critique comme un chemin de données local dans la composition duquel interviennent :

- le temps de commutation des portes logiques : la somme des temps *tt1*, *tt2* et *tt3*
- le temps de propagation dans les interconnexions : la somme des temps *tp1*, *tp2*, *tp3* et *tp4*
- le temps de setup (temps de latch du premier niveau des bascules maître-esclave) des bascules d'arrivée : le temps *ts1*
- le décalage d'horloge dû aux différences de temps de propagation du signal sur les chemins distincts pour atteindre les bascules d'entrée et de sortie : le *skew*

$$chemin\ critique = \Sigma\, tt_i + \Sigma\, tp_i + ts + skew$$

La somme des temps de commutation des portes et de propagation dans les interconnexions est nommée temps de calcul combinatoire

$$temps\ de\ calcul\ combinatoire = \Sigma\, tt_i + \Sigma\, tp_i$$

Dans le cadre des technologies pré-VDSM, le temps de propagation dans les interconnexions était jusqu'à présent négligeable par rapport au temps de commutation des portes.

$$\Sigma\, tp_i \ll \Sigma\, tt_i$$
$$temps\ de\ calcul\ combinatoire \approx \Sigma\, tt_i$$

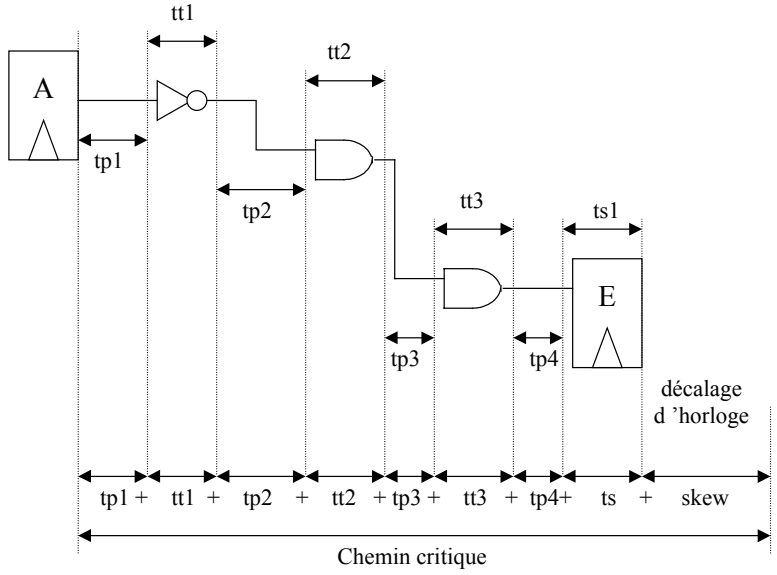


Figure 2-2 : Calcul du chemin critique

Or, dans le cadre des technologies VDSM, le temps de propagation dans les interconnexions n'est plus négligeable par rapport au temps de commutation des portes. Il peut dominer dans le calcul du temps de calcul combinatoire. De plus, le décalage d'horloge peut représenter de l'ordre de 10% du temps total [BAK90].

La structure physique des équipotentielles et de l'arbre d'horloge impactent donc fortement les performances des SoC actuels. Compte tenu de ce fait, il existe diverses approches pour optimiser la période minimale selon que l'on tente de minimiser les temps de commutation et de pré-positionnement (setup) des portes logiques, le décalage d'horloge ou le temps de propagation du signal sur les équipotentielles. Chacune de ces approches est mise en oeuvre dans des contextes méthodologiques différents qui sont maintenant présentés dans un ordre "bottom-up" en ce qui concerne leur niveau d'abstraction.

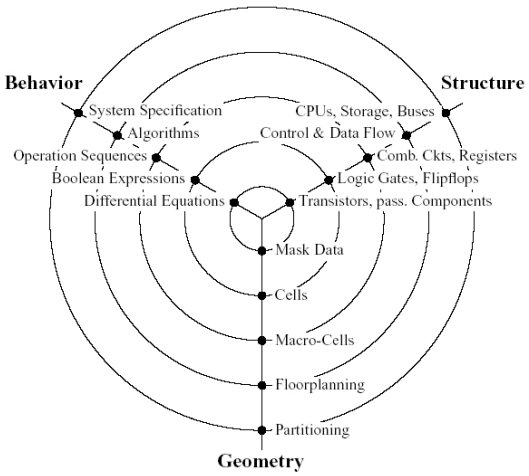


Figure 2-3 : Carte en Y de Gajski et Kuhn

Ces approches sont technologiques, topologiques, logiques, mixte physiques et logiques, architecturales ou systèmes et suivent la progression ascendante des niveaux d'abstraction de la carte en Y de [GAJ83].

2.1. Approches technologiques

Les approches technologiques concernent la conception des portes logiques élémentaires et des structures d'interconnexions (pistes et vias). Le chapitre "Problématique" a présenté la décroissance continue des temps de commutation en fonction de la réduction de surface des composants éléctroniques : les temps de commutation et de setup ne dominent plus dans le temps de calcul combinatoire et cela au profit du temps de propagation [ITR03]. La problématique se situe maintenant sur la maîtrise à grande échelle des temps de propagation des signaux logiques et des horloges sur les pistes métalliques véhiculant ces signaux entre les portes. Les technologues du silicium jouent sur le choix des matériaux conducteurs (métallisation aluminium ou cuivre) et l'accroissement de la section en profondeur pour la réduction de la résistance des pistes. L'approche technologique recouvre un ensemble d'optimisations locales dont les bénéfices sont immédiatement disponibles pour des circuits de très petites tailles dans lesquels le paradigme de la domination du temps de commutation est toujours vérifié car les pistes sont relativement "courtes". Les SoCs ne font pas partie de cette classe de circuits pouvant bénéficier naturellement des progrès technologiques, et c'est bien là le principal problème.

2.2. Approches topologiques

Les approches topologiques concernent la conception d'arbres d'horloges dont le but est d'annuler par construction, de compenser à posteriori, ou d'exploiter le décalage d'horloge. La terminologie "arbre" signifie que l'on considère la source du signal d'horloge comme le "tronc", alors que les multiples chemins de propagation du signal source sont les "branches" et les bascules sont les "feuilles".

L'annulation par construction du décalage d'horloge est obtenue par deux principales structures physiques d'arbres : ce sont les arbres bufferisés et les arbres en H ou en X. Les arbres bufferisés sont des arbres balancés et multi-niveaux. Ils sont découpés en strates de buffers dont le but est d'adapter l'impédance de l'arbre d'horloge à la forte charge capacitive représentée par les pistes et les entrées des bascules. Le nombre de buffers entre la source du signal d'horloge et n'importe laquelle des bascules est toujours le même. Ce sont les arbres à décalage d'horloge nul (ZSCT ou Zero Skew Clock Tree) de [TSA93]. Les arbres en H (H-tree) et les arbres en X (X-tree) sont des structures métalliques répétitives horizontales et verticales (H-tree) ou diagonales (X-tree). Elles se divisent régulièrement, à niveau de métal constant et avec une largeur de piste qui décroit, de façon à distribuer le signal d'horloge avec un temps de propagation constant et un décalage d'horloge nul dans tout le circuit. Les arbres bufferisés ont des caractéristiques opposées à celles des arbres en H et en X :

- ils réduisent le temps de propagation et le décalage provient essentiellement des buffers qui sont sensibles aux variations du procédé de fabrication (best, typique, worst) ainsi qu'aux conditions extérieures (voltage, température),
- ils autorisent une méthodologie de conception hiérarchique,
- et ils ne posent aucun problème de congestion du routage des signaux logiques.

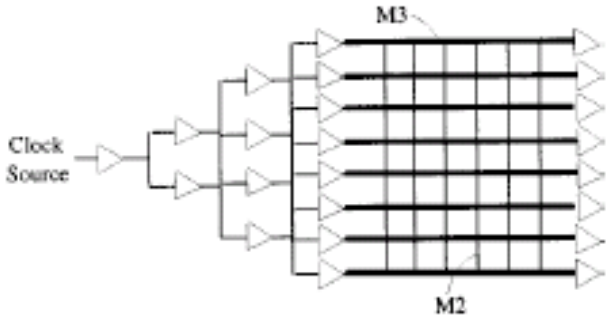


Figure 2-4 : Arbre d'horloge en peigne du microprocesseur DEC alpha

Les arbres bufferisés peuvent avoir des extensions en peigne de façon à diminuer la résistance du dernier niveau d'interconnexion et à favoriser le routage ultérieur des signaux logiques comme c'est le cas dans le processeur DEC-alpha [DOB02] (règles de routage préférentiel Métal-3 horizontal et Métal-2 vertical).

La synthèse et le routage des arbres d'horloges bufferisés sont optimisés de façon à réduire le nombre de buffers ainsi que la longueur des pistes et ils ne peuvent être parfaits à cause du placement des portes logiques qui réduit la routabilité. [CON97a] a prédit que le nombre de buffers d'un arbre d'horloge en technologies 0,050 μm serait de l'ordre de 800000 par circuit. La compensation a postériori du décalage repose sur le dimensionnement et le positionnement des nombreux buffers et l'ajustement de la largeur et de la longueur des pistes. La compensation réduit les perturbations des temps de propagation dues au routage des pistes en les alongeant artificiellement (pistes plus longues, technique dite du "snaking") ou en les raccourcissant avec des buffers plus puissants.

Enfin, l'exploitation du décalage d'horloge permet aussi d'améliorer les performances des circuits en exploitant le fait que l'on peut le déduire d'un étage de chemin de donnée pipeliné pour l'ajouter à ses voisins adjacents moins contraints que lui : le décalage d'horloge devient alors localement négatif. Les circuits dans lesquels cette technique est utilisée sont nommés circuits semi-synchrones et nécessitent un ordonnancement préalable du signal d'horloge (clock scheduling) [TAK97] avant routage de l'arbre. Le but de l'ordonnancement est de déterminer les décalages négatifs pour réduire la période d'horloge globale du circuit. Les circuits semi-synchrones sont synchrones car le signal d'horloge arrive périodiquement à toutes les bascules mais, contrairement aux circuits strictement synchrones, ces dernières n'ont pas besoin de recevoir le signal au même instant. Différentes terminologies sont employées pour désigner les techniques de synthèse et d'exploitation du décalage d'hologe négatif, ce sont la synchronisation double (double-clocking) [FIS90], les impulsions d'annulation de décalage (deskewing data pulses) [IBM85], le vol de cycle (cycle stealing) [LIN92], le décalage d'horloge utile (useful clock skew) [BAK90], et le décalage prescrit (prescribed skew) [CHA92].

2.3. Approches logiques

Les approches logiques sont de deux types: synchrone et asynchrone. L'étude des circuits asynchrones date des années 50, époque à laquelle il n'y avait pas de distinction entre circuits asynchrones et synchrones. Les circuits synchrones sont une restriction des circuits asynchrones à la classe des circuits dont la synchronisation est assurée par un unique signal d'horloge global. Dans les circuits asynchrones, tout autre moyen de synchronisation est accepté. On peut aussi les différencier en exprimant le fait que les circuits synchrones ont une synchronisation orientée « contrôle » alors que les circuits asynchrones ont une synchronisation orientée « flot de données » [SAL01].

- Approche synchrone

L'approche synchrone repose sur la combinaison de deux types de fonctions élémentaires : les fonctions dites combinatoires (ou sans mémoire) et les fonctions dites séquentielles (ou avec mémoire). Les fonctions combinatoires permettent d'implanter un chemin de donnée entre deux fonctions de mémorisation. Le principe est simple : un signal d'horloge global commande les opérations de mémorisation à intervalles réguliers tel que l'illustre la Figure 2-5. La capture de l'information peut être faite indifféremment sur un niveau avec des *latch* ou sur un front avec des *flip-flop*.

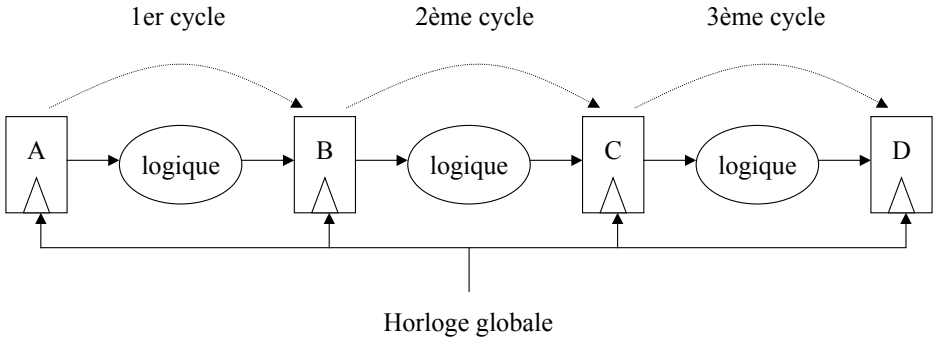


Figure 2-5 : Principe de fonctionnement d'un circuit synchrone

En conception logique synchrone, la méthode de conception au niveau « assemblage de portes » (gate level methodology) prédomine jusque dans les années 80, date à laquelle elle est remplacée par la méthodologie dite RTL. La méthodologie RTL (Register Transfer Level) repose sur les langages de description matérielle (HDL, ou Hardware description Language) tels que [IEE93, IEE98] VHDL (Européen) et [IEE95, IEE00] Verilog (USA) et a permis une augmentation de la productivité en conception. Ces langages permettent la description d'un circuit au niveau registres et opérateurs, la simulation fonctionnelle et la synthèse dite « logique ». Ils ciblent exclusivement les circuits synchrones. La totalité de ce qui est simulable n'est pas synthétisable et il est nécessaire de bien connaître le principe de fonctionnement « transfert de registres » pour exploiter correctement la spécification, la simulation et la synthèse au niveau RTL pour obtenir des circuits peu volumineux et rapides. La norme IEEE P1076.6 spécifie le sous-ensemble VHDL strictement synthétisable et permet ainsi aux concepteurs de rédiger des spécifications indépendantes des outils de synthèse et des technologies cibles.

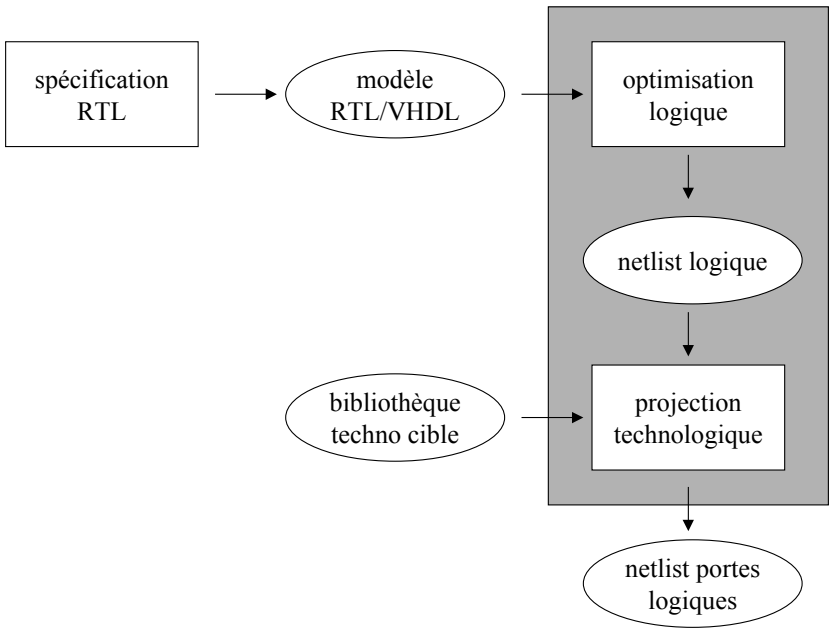


Figure 2-6 : Principe de la synthèse logique

Cet élévation du niveau d'abstraction repose sur deux facteurs clés qui sont l'optimisation logique et la projection technologique comme l'illustre Figure 2-6.

L'optimisation logique consiste à extraire du modèle RTL, puis à optimiser, les équations logiques qui lient les sorties des différents processus à leurs entrées. Ce sujet est toujours d'actualité car la complexité des équations logiques est telle que la recherche d'une solution optimale peut devenir impossible en temps raisonnable. Des heuristiques permettent alors d'obtenir des solutions sub-optimales en des temps plus réduits. La projection technologique (technology mapping) est la fonction qui consiste à répartir les équations logiques optimisées sur des portes logiques issues d'une bibliothèque pré-caractérisée en surface, vitesse et consommation. Cette bibliothèque peut être une bibliothèque de cellules standards (standard cells) ASIC fournies par un fondeur pour une technologie de fabrication donnée, mais aussi l'ensemble des blocs de base d'un circuit programmable de type FPGA. La projection technologique rend la méthodologie RTL indépendante de la technologie cible. Elle implante les expressions logiques sur des cellules et choisit ces dernières de façon à adapter leur puissance de sortie à la charge représentée par les pistes métalliques et les entrées des cellules auxquelles elle est connectée. Par exemple, pour un simple inverseur, il peut y avoir une dizaine de versions dont seule la puissance de sortie (et donc la surface) diffèrent. Ainsi, l'optimisation logique et la projection technologique permettent-elles d'améliorer la fréquence maximale de fonctionnement en exploitant les niveaux logiques (équations) et physiques (choix des cellules). [DEM94] présente une excellente introduction aux algorithmes d'optimisation logique. [REN02a] fait un point actualisé à ce sujet, plus particulièrement concentré sur la synthèse RTL. Cette approche, efficace localement, ne permet plus à elle seule de synthétiser avec succès un SoC.

- Approche asynchrone

Contrairement aux circuits synchrones ayant une synchronisation globale, les circuits asynchrones reposent sur une synchronisation locale comme l'illustre la Figure 2-7. Les opérateurs asynchrones *A*, *B*, *C* et *D* forment une chaîne de quatre opérateurs logiques dans laquelle les échanges d'informations se font indépendamment des autres opérateurs du circuit. Ce mécanisme garantit la synchronisation et la causalité des événements au niveau local et la correction fonctionnelle au niveau système [REN02b].

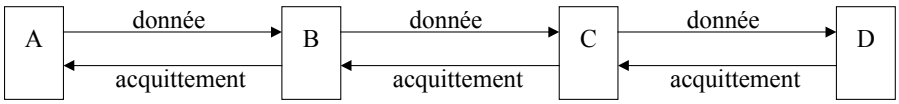


Figure 2-7 : Communication et synchronisation indépendantes du temps

Un opérateur asynchrone peut être considéré comme une cellule réalisant une certaine fonction en communiquant avec son environnement à travers des canaux de communication banalisés. L'opérateur peut alors être une porte logique élémentaire, une fonction plus complexe, voire un algorithme et peut contenir ou non des éléments de mémorisation.

Il est caractérisé par quatre paramètres fondamentaux : latence, temps de cycle, profondeur du pipeline et protocole de communication.

- La latence correspond au temps de propagation de la chaîne combinatoire la plus longue dans l'opérateur. Selon les données, toutes les chaînes combinatoires ne sont pas mises en œuvre et la latence est définie plus précisément par le triplet (latence minimale, latence moyenne, latence maximale).

- Le temps de cycle est le délai minimum entre deux entrées successives. Il caractérise le débit de l'opérateur et dépend du temps nécessaire pour échanger une donnée entre deux ressources internes de mémorisation.

- La profondeur du pipeline est le nombre maximal de données que l'opérateur peut mémoriser.

- Le protocole de communication permet, en entrée, de détecter la présence d'une information et de signaler qu'elle a été consommée et, en sortie, de signaler la présence d'une donnée et de détecter sa consommation par l'opérateur suivant.

Les protocoles sont de deux types en ce qui concerne l'acquittement : acquittement sur front à 2 phases (NRZ, Non Retour à Zero, ou mode Half-Handshake) et acquittement sur niveau à 4 phases (RZ, Retour à Zero, ou mode Full-Handshake). Le codage des données transmet à la fois la donnée ainsi que la signalisation de sa présence. Il s'agit de codages de type double rail (deux fils par bit) ou d'un signal de requête additionnel pour les protocoles de données groupées [SUT89]. Dans le cas d'un signal de requête, les données sont correctement transmises s'il existe une hypothèse temporelle qui garantit que le signal de requête succède toujours la stabilisation de la donnée.

Une des difficultés inhérente aux circuits asynchrone est la nécessité de concevoir des opérateurs sans aléas (ou glitches) car ils peuvent être interprétés comme des signaux de requête ou d'acquittement. Les aléas proviennent des portes logiques (aléas statiques et dynamiques), des différents temps de propagation des signaux (aléas combinatoires logiques) et de la conception elle-même (aléas fonctionnels et séquentiels). Cette difficulté n'existe pas dans les circuits synchrones car la période de l'horloge est dimensionnée de telle sorte que tous les signaux soient stabilisés (les aléas ont donc disparu) avant le début du cycle suivant. Les outils de synthèse logique conventionnelle ne peuvent donc pas être utilisés pour la synthèse de circuits asynchrones. Les circuits asynchrones peuvent être répartis en plusieurs classes qui dépendent de leurs modèles de délais.

Les circuits insensibles aux délais : ce sont les circuits asynchrones « purs », sans aucunes hypothèses temporelles. Ils nécessitent des bibliothèques de cellules beaucoup plus complexes que les portes logiques synchrones standard pour qu'il n'y ait pas d'aléas [HAU95].

Les circuits quasi insensibles aux délais ajoutent aux circuits insensibles aux délais la notion de fourche isochrone. On appelle fourche isochrone un fil qui connecte un émetteur unique à plusiseurs récepteurs et qui est telle que les temps de propagation sont identiques quelque soit la

branche choisie. On peut donc implanter un acquittement unique sur une seule des branches. [MAR90] montre que cette contrainte est facilement remplie par une conception soignée (placement & routage) et qu'elle permet la réutilisation de cellules standards.

Les circuits indépendants de la vitesse font l'hypothèse que les délais dans les équipotentielles sont négligeables par rapport aux temps de commutation des portes. Ils sont considérés comme équivalents aux précédents dans la mesure ou toutes les fourches sont considérées comme isochrones [HAU95].

Les circuits de Huffman font la même hypothèse temporelle que les circuits synchrones : tous les délais dans les cellules et les connexions sont bornés, voire connus. Ils permettent la conception de machine d'états finis dont les boucles combinatoires comportent des délais et réalisent la mémorisation des états. [NOW94] propose des machines d'états finis sensibles à des sous-ensembles des entrées-sorties : il s'agit d'une spécification appelée « burst-mode » (mode rafale).

Les micropipelines, introduits par [SUT89], combinent des portes de Muller (détection de rendez-vous) et des lignes à retard (anti aléas) en une structure de type FIFO qui est sensible à des transitions de signaux (protocole à deux phases de type données groupées). Cette structure peut être enrichie d'opérateurs de mémorisation et de traitement et devient alors un micropipeline ou « pileline élastique » dans la mesure ou le nombre de données présentes dans le circuit est variable.

Compte tenu de ces caractéristiques, l'approche asynchrone permet de faire fonctionner l'ensemble d'un circuit dans les conditions moyennes (et donc plus favorables) de latence des opérateurs plutôt que dans les conditions « pire cas » de l'approche synchrone. Il s'agit là de la principale contribution de l'approche asynchrone à la vitesse de fonctionnement. Il existe d'autres avantages [MAR89][HAU95] qui sont 1) l'aptitude à traiter naturellement les signaux asynchrones (interruptions pour un microprocesseur), 2) la composition plus aisée de blocs asynchrones (tel le circuit Amphin de [ROB97]), 3) l'élimination de la consommation et de la gigue de l'horloge et 4) un niveau de bruit plus faible.

Plusieurs démonstrateurs prouvent la viabilité de l'approche. Ce sont les microprocesseurs RISC 16 bits CAP [MAR89] et RISC 32 bits (architecture MIPS R3000) MiniMIPS [MAR97] de Caltech, les microprocesseurs RISC 32 bits (architecture de type ARM) Amulet1-2e-3i de l'Université de Manchester [GAR99], le microprocesseur RISC 32 bits (architecture de type MIPS R2000) TITAC-2 de l'Université de Tokio [TAK97b] et les microprocesseurs CISC 8 bits MICA [ABR01] et RISC 16 bits ASPRO [REN98] du TIMA. Malheureusement, un manque certain d'outils commerciaux de synthèse et de formation des concepteurs nuit grandement à son appropriation par les industriels. Les produits issus d'une implantation asynchrone sont donc extrêmement rares comparativement aux produits synchrones. Citons un ASIC mixte (215074 portes asynchrones, 244770 portes synchrones) chez MBDA [MBD02], un dérivé asynchrone du 80C51 dans le P87CL88 (un micro-contrôleur basse consommation dédié à la téléphonie) chez Philips [PHI02], et le DDMC (Data-Driven Media Processor) un circuit multi-processeurs basse consommation de traitement du signal et de l'image de Sharp [SHA97].

2.4. Approches mixtes logiques-physiques

Les approches mixtes physiques-logiques reposent sur des outils commerciaux pour lesquels il est ardu de faire une comparaison tant il est vrai que les documentations technico-marketing mises à la disposition du public sont partielles et partiales. Nous tentons néanmoins de les comparer entres elles objectivement. Le constat de toutes ces approches est que le modèle d'estimation statistique des charges (WLM, ou Wire Load Model) sur la(les) sortie(s) d'une porte n'est plus suffisant pour déterminer le chemin critique avant synthèse physique complète et extraction des caractéristiques RC des interconnexions. Dans ce contexte, le principal critère de différenciation entre outils est, à notre avis, le niveau d'abstraction d'où sont extraites les informations dites de rétro-annotation après placement et/ou routage partiel et dont le but est de procurer un nouveau modèle de charge adapté aux technologies VDSM. Ce sont, soit les « timings » traditionnels des cellules des bibliothèques standard-cells des fondeurs agrémentés des caractéristiques RC des interconnexions, soit des informations relatives au schémas électrique interne des cellules. En bref, certains outils « brisent » l'enveloppe des cellules, d'autres non : les sociétés concernées proposent des « méthodologies orientées boîtes noires » (black boxes) lorsque l'on ne connaît pas l'implantation des cellules, ou des « méthodologies orientées boîtes grises » (gray boxes) si l'on dispose de certaines informations fournies par le fondeur.

Ainsi BLAST de Magma [MAG04] et ZenTime de Zenasis [ZEN04] proposent d'exploiter le schémas électrique des cellules pour optimiser les chemins critiques. BLAST, à partir des schémas électriques extrait un modèle de charge des entrées et des sorties (modèle « Gain-Based » issu des travaux de [SUT99] dans lequel n'apparaît pas la résistance des interconnexions) et s'en sert pour créer une nouvelle bibliothèque de cellules caractérisées « gain-based » et génériques. La généricité des cellules repose sur le fait qu'elle ne correspondent pas à des cellules particulières mais à un sous-ensemble des cellules de la bibliothèques du fondeur. Les cellules « gain-based » ne sont pas dimensionnées. Leur dimensionnement dépendra des caractéristiques du routage final. Par exemple, il existera une seule cellule pour l'inverseur, même s'il y a plusiseurs versions du même inverseur dans la bibliothèque initiale. Enfin le routeur de BLAST peut aussi intervenir sur la largeur des pistes de façon à réduire leur résistance. Avec BLAST, on applique un dimensionnement des cellules tardif et un routage sur mesure (« full custom ») automatique. ZenTime, de son coté, considère que le problème vient de la bibliothèque de cellules du fondeur. ZenTime propose d'isoler le chemin combinatoire critique (schémas électriques des cellules compris) et de l'optimiser afin de créer de nouvelles cellules sur mesure. Ces nouvelles cellules sont caractérisées avec les outils classiques et remplacent les chemins critiques. La méthodologie ZenTime propose de remplacer tous les chemins critiques par des cellules optimisées : les ZenCells. BLAST et ZenTime, bien que ne l'affirmant pas, nécessitent certainement l'usage d'outils de re-routage partiel pour connecter les ZenCells de remplacement, et de compaction post-placement pour récupérer la place libérée par le sous-dimensionnement des cellules « gain-based ».

PKS de cadence [CAD04] et Physical Compiler de Synopsis [SYN04] ont une approche standard-cells classique avec exploitation des informations post-placement et/ou routage. Ces deux outils automatisent les rebouclages méthodologiques (modifications incrémentales, nommées ECO ou Engineering Change Order) des concepteurs de circuits. Ces outils

automatisent le dimensionnement de cellules, l'ajout de buffers, et la duplication de registres pour soutenir une charge de sortie trop élevée. Enfin, ils peuvent au niveau synthèse RTL revenir sur un choix de synthèse pour un chemin critique donné en fonction du placement. Les deux sociétés encouragent les méthodologies de conception hiérarchiques avec des stratégies de routage différentes selon que l'on se trouve au niveau bloc (de l'ordre de moins de quelques millions de cellules) ou au niveau circuits (quelques dizaines de blocs ayant des connexions de type bus).

Une étude très détaillée faite par Synopsis [SYN00], concerne quatre méthodologies différentes de conception de circuit. Elle montre que les gains les plus importants sont obtenus aux niveaux d'abstractions les plus bas. Le but de l'expérience est de mesurer à quel point divers types de synthèses prennent en compte les délais introduits par le placement des cellules et donc le routage des signaux. Le circuit cible est un bloc graphique de 115000 portes, fonctionnant à 125 MHz et fondu en 0,25 µm. Les valeurs obtenues pour chacune des quatre méthodologies sont des moyennes issues de plus de 100 tentatives correspondant à autant de placements différents.

Méthodologie	Dépassement de la période maximum(ns)	**Gain N/N-1**	**Gain cumulé**
1/ Synthèse + extraction RC + retiming	33		
2/ Synthèse + extraction RC + resizing	8.9	73 %	73 %
3/ Synthèse + extraction RC + re-synthèse logique	3.8	57 %	88 %
4/ Synthèse + extraction RC + re-synthèse RTL	2.6	31 %	92 %

Tableau 2-1 : Comparaison de quatre méthodologies de conception

Le Tableau 2-1 présente ces résultats. Le « retiming » [LEI83] consiste à déplacer les registres sur les chemins de données afin de répartir différemment les temps de propagation entre deux chemins de données adjacents. C'est la méthode d'optimisation qualifiée de méthode de base par Synopsis. Le « resizing » consiste à dimensionner les cellules (en général cela consiste à les agrandir) pour supporter les grandes charges en sortie. On constate que le « resizing » est la méthode d'optimisation la plus efficace. L'optimisation logique et la re-synthèse de niveau RTL apportent encore des gains importants, mais d'amplitude moindre, l'essentiel est déjà fait par les méthodes classiques et non strictement VDSM. Enfin, Precision Physical Synthesis de Mentor Graphics [MEN04] est dédié aux FPGA. Il illustre le fait que, bien que les FPGA soient en fait des ASIC programmables, la connaissance précise de leur topologie est absolument nécessaire pour obtenir la meilleure synthèse. De même que PKS et Physical Compiler, Precision Physical Synthesis exploite les techniques de « retiming », de duplication de registres et de « placement-aware » car la topologie de routage est figée et le nombre de degrés de liberté pour aller d'un bloc logique à un autre est relativement faible. Cette méthodologie est décrite par [SUA04].

2.5. Approches au niveau architectural

Il existe un grand nombre d'outils concernant la synthèse de haut niveau, dite aussi synthèse d'architecture. Ils se différencient par leur domaine d'application et les contraintes supportées : GAUT [MAR93], AMICAL [PAR93], CATHEDRAL, CADDY/DSL [CAM89], CALLAS [STO92] ou HERCULES/HEBE [DeM88] pour les travaux académiques et MONET [ELL00], A|RT Designer [ADE04] ou Behavioral Compiler [KNA96] (ainsi que leurs dérivés actuels orientés SystemC : Catapult C Synthesis et System C Synthesis) pour les produits industriels.

Les approches au niveau architectural intègrent à la synthèse de haut niveau, soit des modèles prédictifs du coût des interconnexions (en terme de temps de propagation), soit l'extraction post placement-routage des caractéristiques RC de ces dernières en vue d'une re-synthèse totale ou partielle du circuit, ou soit une combinaison de pré-placement et de synthèse simultanés.

Dans la classe des modèles prédictifs, les outils ont pour but de synthétiser en une seule passe un circuit dont le coût des interconnexions est minimum. [CAS01] et [JEG00] proposent un modèle de coût des interconnexions fondé sur 1) la localité des transferts de données entre opérations, 2) une estimation de la distance des interconnexions en fonction de la localité, et 3) une vitesse de propagation des signaux qui dépend de la technologie cible. La notion de localité est mise en oeuvre à l'aide d'une méthodologie de synthèse physique hiérarchique. Les transferts de données sont classés en trois types de localité qui dépendent du modèle d'architecture cible. Ce sont les transferts locaux (intra-opérateur), intra-composant (inter-opérateurs) et extra-composant (communication externe). Ils correspondent respectivement aux niveaux hiérarchiques « opérateur », « cluster » et « circuit » de la stratégie de placement-routage hiérarchique sous tendue. La stratégie de minimisation du coût des interconnexions de la synthèse repose sur le rapprochement géographique (les clusters) ou le fusionnement (opérateurs multi-fonctions) des opérateurs qui communiquent entre eux via des transferts intra-composant. Le fusionnement des opérateurs réduit alors le nombre d'opérateurs et les clusters sont représentés par des directives de placement-routage par région. Cette méthodologie permet de diminuer par un facteur 2 à 3 la longueur maximale des interconnexions d'un circuit.

Dans la classe des modèles post placement-routage, les outils de synthèse ont pour but de réduire le coût des interconnexions d'un circuit par re-synthèses d'architecture, logique et physique successives. [PAR99] propose une méthodologie dans laquelle une première synthèse complète permet d'obtenir un circuit constitué d'une unité de traitement et d'une unité de contrôle. Le coût des interconnexions n'étant pas estimé, il se peut que les temps de propagation dans l'unité de traitement après placement-routage soient tels que l'ordonnancement initial ne respecte pas la cadence imposée à cause d'une horloge qui ne peut être aussi rapide que prévue. La méthodologie consiste à réordonnancer les opérations sur le même matériel avec une période d'horloge différente (elle peut être plus lente ou plus rapide) telle que le délai total de calcul (nombre d'états x période) soit inférieur ou égal à la cadence imposée. Seule l'unité de contrôle est resynthétisée et la méthodologie repose sur le fait que le concepteur est capable de préserver le placement-routage de l'unité de traitement constant entre deux synthèses physiques. Les gains obtenus sont des réductions du temps calcul de l'ordre de dix à vingt pour cent avec des tests issus des benchmarks de HYPER [CHU89].

Les solutions sont beaucoup plus nombreuses dans la classe des méthodologies de placement-synthèse combinées. [DOU00] propose un modèle de donnée unifié pour le placement et la synthèse dans lequel la fusion (opérateurs multi-fonctions, fusion de registres, …) et l'éloignement de blocs sont dépendants d'un DFG annoté de forces (au sens mécanique du terme, mais en 2D seulement) d'attraction pour la fusion et de répulsion pour la duplication et la réduction de distance entre blocs. Ces forces pilotent un algorithme de placement quadratique [EIS98] qui recherche un état dont l'énergie potentielle est la plus faible. Les gains obtenus sont de l'ordre de +15 % en vitesse et de –50% en surface. [TAR98] propose de réunir géographiquement les unités de traitement, de mémorisation et de communication afin de réduire le chemin critique intra-composant. Enfin, [KIM01] et [JEO01] proposent d'encapsuler les composants par une barrière de registres qui découple les entrées-sorties externes du calcul interne. Les temps de propagation des entrées-sorties sur les liaisons physiques inter-composants peuvent alors être multi-cycles et n'impactent pas les performances locales de ces derniers. Les différentes solutions exploitent une hiérarchie dans laquelle les blocs fonctionnels synthétisés sont estimables en surface/délai.

On constate que, comme avec les approches mixtes logiques-physiques des principales sociétés de CAO micro-électronique, les approches architecturales reposent systématiquement sur la synthèse physique hiérarchique dont le but est soit de favoriser la localité des traitements, soit de préserver un sous-bloc optimisé entre deux placement-routages. L'importance de la hiérarchie s'accroît avec le niveau d'abstraction de l'approche comme nous le montrons maintenant avec les approches systèmes.

2.6. Approches systèmes

Dans ce paragraphe nous présentons tout d'abord les approches système de types synchrone et asynchrone, puis nous introduisons les systèmes GALS.

La spécification de systèmes synchrones repose sur les langages de programmation synchrones réactifs. Ce sont ESTEREL[BER00][EST03], LUSTRE[HAL91] et SIGNAL[BNV88]. Ils ont été proposés pour la spécification de systèmes embarqués temps-réels complexes et permettent de combiner simplicité du paradigme synchrone et concurrence dans une même spécification fonctionnelle. Alors que dans le monde asynchrone le découplage temporel est un concept de base, dans le monde synchrone, la notion de découplage entre processus synchrones existe aussi: il s'agit de la désynchronisation [BNV97][BNV99]. La désynchronisation est motivée par les implantations logicielles et a pour but de permettre à des processus localement synchrones (une tâche sur un processeur) de communiquer de façon asynchrone entre eux. Elle consiste à éliminer les barrières de synchronisation qui délimitent les réactions successives d'un système synchrone et repose sur les concepts d'endochronie et d'isochronie. Par définition, un processus est endochrone si, à chaque instant discret, la validité de tous ses signaux peut être déterminée par les valeurs portées par un sous-ensemble de ses signaux. Enfin, deux processus sont isochrones si tous leurs signaux sont synchrones pris deux à deux. Les implantations synchrones sont utilisées dans la conception de systèmes matériels et logiciels à cause de leurs propriétés qui facilitent la vérification et la synthèse. Elle reposent sur la notion de système synchrone réactif : les concept d'horloge globale, de diffusion instantanée de l'information, de parallélisme déterministe et de préemption sont au cœur de ce modèle.

Dans le domaine de la spécification système, la communauté asynchrone possède aussi ses langages et compilateurs tels que BALSA [BAR00], TANGRAM [PHI04] et TAST [REN99] pour les systèmes complexes, ou tels que PETRIFY [COR96] et MINIMALIST [FUH99] pour des systèmes de tailles plus modestes. Un point complet au sujet de la conception asynchrone dans l'industrie se trouve dans [EDW03]. Citons aussi le programme nord-américain DARPA « Clockless Logic » qui à vu le jour en 2003.

L'approche système, qu'elle soit synchrone ou asynchrone, repose sur la synthèse ou la réutilisation de composants et la synthèse de communication. C'est une approche de type hiérarchique. En quoi est-elle fondamentalement différente des autres ? La réponse est double.

Tout d'abord l'approche système prend le relais lorsque l'on a atteint le stade où il est plus économique de réutiliser (ou de synthétiser) et d'interfacer des composants que de les développer. C'est exclusivement une question de taille. Aujourd'hui, on peut considérer que les composants sont des circuits de type mémoires, processeurs logiciels et matériels, et des sous-systèmes de communication dont la taille n'excède pas quelques millions de portes alors que les systèmes complets contiennent plusieurs dizaines, voire centaines de millions de portes. Ensuite, et *contrairement à toutes les approches précédentes*, il y a une différence essentielle entre les méthodologies de conception au niveau système et les autres *: les transformations ne préservent plus la sémantique au cycle près*.

En effet, alors que les diverses méthodologies de conception hiérarchique permettent l'optimisation locale des différents composants, la connexion des entrées-sorties de ces blocs via des pistes très longues est telle qu'elle introduit des chemins critiques inter-composants.

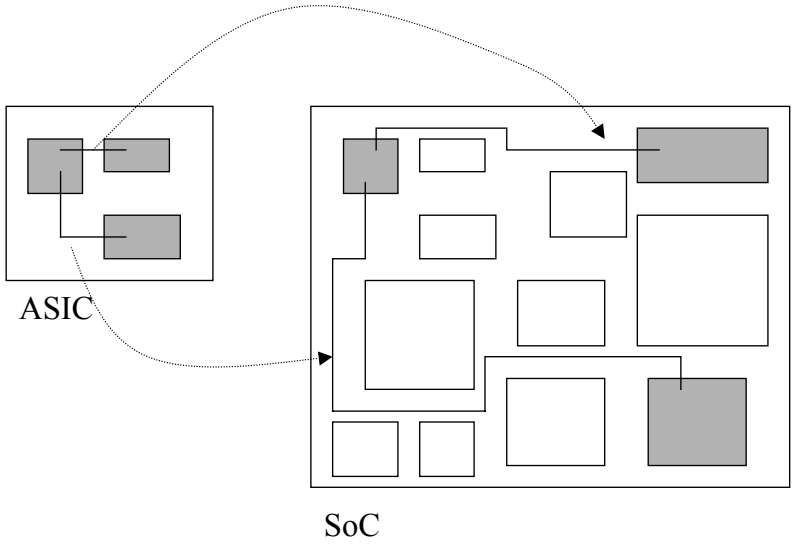


Figure 2-8 : Apparition des chemins critiques inter-composants dans les SoCs

Ces pistes inter-composants ont des dimensions supérieures à celles des composants qu'elles relient entre eux comme l'illustre la Figure 2-8. Elles réduisent à néant tous les efforts précédemment faits pour obtenir des composants fonctionnant à des fréquences élevées. Il n'y a plus de marge de manœuvre et la solution ne peut plus être de type optimisation locale. Les approches systèmes se distinguent donc, selon nous, par leur aptitude à fournir un cadre

méthodologique garantissant la sémantique à un niveau d'abstraction supérieur à celui du niveau cycle. Il ne s'agit plus de garantir que deux circuits sont équivalents au bit et au cycle près (équivalence CABA ou cycle accurate & bit accurate), mais de garantir l'équivalence au niveau des transferts de données sans se soucier d'une absolue précision temporelle.

Le niveau transactionnel, tel qu'il est défini par l'OSCI [OSC04], est divisé en trois niveaux qui sont le niveau message (Layer 3, Message Layer), le niveau transaction (Layer 2, Transaction Layer), et le niveau transfert (Layer 1, Transfer Layer). L'équivalence fonctionnelle au niveau bit et non temporisée des séquences des entrées et des sorties des approches systèmes correspond au niveau message TLM (Transaction Level Model). Ces approches découplent la communication inter-composants du calcul fait par ces derniers. Elles reposent sur l'intégration de l'asynchronisme au niveau de la communication inter-composants. *Ce sont les systèmes dits GALS*.

3. Systèmes GALS

3.1. Définition

L'acronyme GALS signifie Globalement Asynchrone et Localement Synchrone. Introduite par [CHA84], cette notion extrêmement riche concerne un modèle d'implantation de circuits numériques dans lesquels des blocs synchrones communiquent entre eux via un réseau de communication asynchrone. Les Systèmes GALS offrent les avantages des modes synchrones et asynchrones : réutiliser des composants synchrones conçus avec des méthodologies éprouvées et découpler temporellement les entrées-sorties du calcul.

Les systèmes GALS partagent les caractéristiques suivantes :

- 1) Les composants sont tous synchrones.
- 2) Les échanges entre blocs sont régulés par un réseau de communication assurant un contrôle de flux asynchrone tel que l'illustre la Figure 2-9.
- 3) Chaque bloc peut avoir sa propre horloge.
- 4) Tous les blocs synchrones sont encapsulés dans des wrappers qui réalisent l'interface entre le domaine d'horloge synchrone intra-composants et le domaine global asynchrone inter-composants tel que l'illustre la Figure 2-10.

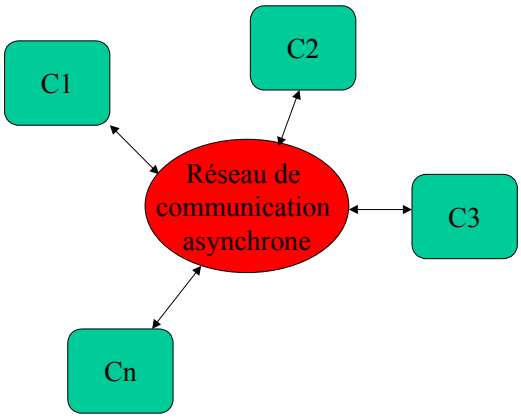


Figure 2-9 : Système GALS composé à partir de composants encapsulés

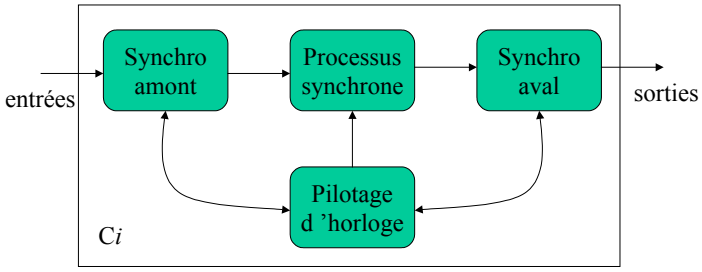


Figure 2-10 : Composant synchrone encapsulé dans un wrapper GALS

Grâce à ces caractéristiques, les systèmes GALS résolvent les deux problèmes technologiques majeurs des SoC : l'élimination des chemins critiques inter-composants grâce au réseau de communication asynchrone et le support de domaines d'horloge indépendants. Ils permettent la réutilisation intensive de composants synchrones préconçus ou synthétisés et répondent ainsi à deux besoins méthodologiques de conception recensés par l'ITRS qui sont la banalisation de la réutilisation de circuits tierce partie et la synthèse des interfaces.

La méthodologie GALS est aussi un des éléments qui introduit de l'hétérogénéité dans les SoCs. L'hétérogénéité des SoCs est un constat en terme d'implantation et de méthodologie de conception. En terme d'implantation, les SoCs contiennent des parties analogiques et numériques, des parties synchrones et asynchrones, des processeurs matériels et logiciels. En terme de méthodologie de conception, les SoCs requièrent des outils de spécification, raffinement et vérification qui exploitent divers modèles de calcul et divers niveaux d'abstraction [JER02a, JER02b].

Nous nous concentrons plus particulièrement sur deux aspects de la méthodologie d'implantation GALS appliquée aux SoCs. Ce sont la nature du pilotage de l'horloge et la structure des wrappers. Les travaux qui concernent les réseaux de communication excèdent la portée de notre étude, nous renvoyons le lecteur vers les publications de [BNI02a, BNI02b] qui concernent les NoCs. Le réseau de communication des GALS est constitué de canaux de communication asynchrones qui remplacent les longues lignes synchrones issues du routage des signaux inter-composant [QUA04]. Les NoCs développent la notion de micro-réseau de transports multi-couches (modèle de référence OSI à sept couches de l'ISO) par paquets. Ces micro-réseaux intègrent de l'accès physique, de la communication point à point, du contrôle de

flux, du routage de paquets et éventuellement de la détection d'erreur. Le réseau de communication GALS se situe à l'opposé des NoCs en terme de données (bit/paquets), en terme de position dans la pile OSI (niveau 1/niveaux 1 à 7) et en terme d'interface entre réseau et composant (protocole bit type poignée de main/interfaces normalisées de type Virtual Component Interface [VSI01] ou OpenCore Protocol [OCP01]). Nous illustrons ce fait en citant deux implantations de micro-réseaux ayant atteint le niveau transport : Spin [GUE00] et Aethereal [RAD02][RIP03] de Philips.

La méthodologie d'implantation de circuit GALS, bien que datant de 1984, est toujours d'actualité. Elle a été récemment proposée pour la réduction de la principale source de consommation du système : l'arbre d'horloge global. Elle part du principe que chaque bloc synchrone est piloté par une horloge locale qui est plus facile à synthétiser. Chacune des horloges locales peut être rythmée par une horloge globale moins complexe. L'horloge globale n'ayant plus à supporter les contraintes du routage elle peut être réduite en voltage, en fréquence et en distance parcourue pour atteindre chacun des blocs. Chaque bloc est responsable de générer sa source d'horloge locale à l'aide d'un oscillateur en anneau ou d'une boucle à vérouillage de phase (PLL). Les réductions de consommations obtenues sont alors de l'ordre de 30% [MEI98] pour un partitionnement du système en quelques dizaines de blocs.

Les systèmes GALS dont le réseau de communication est véritablement asynchrone posent le problème de la métastabilité aux frontières des domaines d'horloges des composants synchrones. La robustesse (c'est-à-dire la garantie de la non pollution des données transmises) du système nécessite de mettre en œuvre des mécanismes sûrs tels que des FIFO ayant une extrémité synchrone (coté composant) et une extrémité asynchrone (coté réseau) comme le font les FIFO faible latence pour systèmes à horloges multiples de [CHE01], ou encore la technique STARI de [CHA03].

Une autre réponse est possible : ce sont les *systèmes insensibles à la latence*. C'est le cœur de notre sujet et nous l'abordons immédiatement après deux concepts nécessaires à sa bonne compréhension. Ce sont le positionnement des GALS vis à vis de la synthèse de communication et les techniques de gel d'horloge disponibles pour implanter la notion de *suspensibilité* des composants.

3.2. GALS et synthèse de communication

Un système électronique peut être modélisé comme un ensemble de modules hiérarchiques représentant une machine abstraite. Les différents modules communiquent par des canaux de communication, via leurs interfaces. Le comportement des modules feuilles de la hiérarchie peut être une tâche élémentaire tout comme un processus complexe qui peut, à son tour, cacher des flots d'exécution parallèles et hiérarchiques. Enfin, les interfaces sont constituées de ports de communication et d'opérations sur ces port. Ces concepts de module, d'interface, de port et de canal peuvent être utilisés à plusieurs niveaux d'abstraction [NIC02].

La communication entre modules exploite les ports et les canaux pour transmettre les données et la synthèse de communication s'intéresse à l'automatisation de l'implantation de cette dernière. Les travaux en synthèse de communication se classent en trois domaines qui sont la synthèse d'interfaces, l'optimisation du réseau de communication, et la synthèse d'IP sous contraintes de communication selon que l'on se focalise respectivement sur l'interface composant/réseau, le réseau, ou le composant tel que l'illustre la Figure 2-11

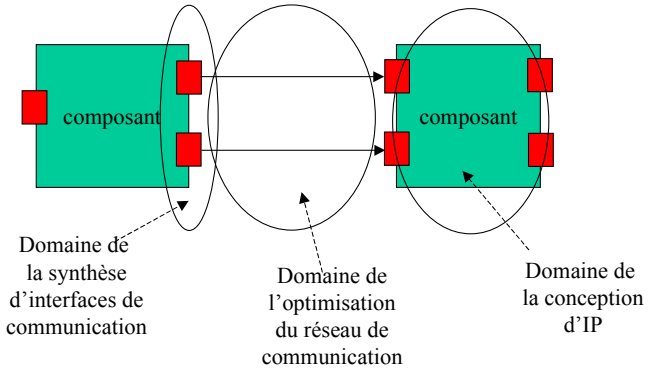


Figure 2-11 : Classification des domaines de la synthèse de communication

La *synthèse d'IPs sous contraintes de communication* consiste à synthétiser des IPs à partir d'une description de haut niveau indépendante de toute implantation. Il s'agit du domaine de la synthèse d'architecture (ou synthèse comportementale) dans laquelle ont été introduites des contraintes de communication [COU03]. Cette méthodologie a pour objectif l'optimisation globale du circuit : c'est l'adaptation de l'IP aux caractéristiques du réseau de communication. Elle vient en complément des méthodologies qui utilisent la synthèse de wrappers pour adapter les IPs au réseau (ou l'inverse selon le point de vue adopté). En effet, en raison des incompatibilités potentielles, la complexité des wrappers « classiques » peut annuler tous les gains en surface et en performance du réseau et des IPs qui ont été optimisés séparément.

Les premiers travaux qui concernent la *synthèse des interfaces* entre deux composants ont un niveau d'abstraction très faible et sont basés sur des chronogrammes. Ces approches se concentrent sur la synthèse des interfaces en utilisant une description détaillée des protocoles. Ainsi, à partir d'une description bas niveau des contraintes sur les signaux, elle génère un automate de contrôle. La description initiale peut être textuelle [NES86] ou graphique [BOR87], et peut faire référence à des communications de type *broadcast* comme dans Polis [CHI93].

[NAR95], [PAS98], et [LIN94] (méthode intégrée dans CoWare) réalisent l'interfaçage de composants utilisant des protocoles incompatibles et ont comme description initiale une spécification événementielle. [CHO95] et [MAD95] proposent une solution au problème spécifique de l'interconnexion de composants dans des architectures hétérogènes. Une architecture générique de wrapper est proposée par [YOO01], elle permet de supporter l'hétérogénéité des composants entre eux et avec le réseau de communication en terme de niveaux d'abstractions, de protocoles et d'implantations. Cette hétérogénéité a été détaillée dans le chapitre « Problématique ».

L'optimisation du réseau de communications affecte à chaque canal abstrait un protocole de communication de type synchrone ou asynchrone. Les approches existantes se distinguent par leur position dans le flot de conception, plus particulièrement par rapport à la phase de partitionnement. Ainsi lors du découpage matériel/logiciel, [HNK94] utilise des critères tels que le coût logiciel (taille du code), le coût matériel (surface) et la durée des transferts pour choisir le type et les modes de transfert dans une architecture composée d'un processeur, un coprocesseur et une mémoire partagée. [BND96] ajoute à ces métriques l'occupation des bus dans des architectures multiprocesseurs. Contrairement à [YEN96], il ne considère qu'un unique mode de communication DMA qui n'est pas adapté aux transferts de faible volume de données. La deuxième famille d'approches se situe après l'étape de partitionnement : le système est décrit à l'aide d'un graphe composé d'un ensemble de tâches (les nœuds) reliées par des canaux abstraits (les arcs). Dans ce contexte, la plupart des travaux tentent de minimiser les délais de synchronisation en réduisant les communications bloquantes. Dans [NAR94] les auteurs distribuent temporellement les communications liées à l'ordonnancement et fixées par le partitionnement. Ils maximisent ainsi l'utilisation du bus partagé sur lequel ils basent leur architecture système et ce pour une largeur minimale de bus. [GON96] étend les travaux précédants en ciblant quatre architectures basées sur l'utilisation de connexions point-à-point/ bus-partagé avec des mémoires de type locales/globales. La méthodologie proposée permet d'obtenir une partie contrôle gérant le protocole, de raffiner les primitives d'échanges de données (send, receive) et de définir un arbitre lors de l'utilisation d'un bus partagé. [FIL93] propose de modifier l'ordonnancement effectué durant le partitionnement afin de minimiser les communications. D'autres techniques de sélection de protocole de communication, similaires à celles précédemment décrites sont développées dans [FRE96] (Construction exhaustive de l'ensemble des solutions), [GOG98], [CUE01], ou [BND96], [DAV01] (Utilisation des taux de communication crêtes pour une synthèse d'interfaces par allocation). Enfin [KNU99], [DEY97], [YEN95], et [LAH01] proposent des méthodes d'analyse de performances au niveau conception système qui tiennent compte de la communication.

Les systèmes GALS se positionnent dans la catégorie « synthèse d'interface de communication » pour circuits synchrones. Ils concernent l'encapsulation de composants synchrones optimisés dans des wrappers de communication qui ne dégradent pas leur fréquence de fonctionnement maximale. Enfin, l'aptitude à synthétiser automatiquement ces wrappers est un élément essentiel de toute méthodologie de conception/prototypage rapide orientée plate-forme qui se veut réellement exploitable pour des applications de traitement du signal réalistes et actuelles.

3.3. Suspensibilité des composants GALS

Pour être réutilisable en environnement GALS ou LIS, un composant synchrone doit être *suspensible*. Par *suspensibilité*, on entend « gel » du composant dans un état stable pendant une durée variable en nombre de cycles d'horloge. Nous rappelons dans ce paragraphe que les horloges des circuits intégrés peuvent être d'un des quatre modèles suivants :

- Horloge globale : une seule horloge pilote tous les composants synchrones. Ce sont les systèmes dits mésochrones (déphasages fixes) ou multi-synchrones (déphasages arbitraires).

- Horloges à rapport de fréquence rationnel : une horloge de base permet de générer des horloges locales dont le rapport à la fréquence de base est un nombre rationnel. Ce sont des systèmes dits plésiochrones (fréquences identiques, rapport 1) ou hétérochrones (fréquences différentes).

- Horloges à fréquences proches : les horloges sont générées indépendamment et leurs fréquences sont quasiment identiques. Ce sont des systèmes dits hétérochrones.

- Horloges arbitraires : les horloges sont générées indépendamment et sans aucune contrainte. Ces systèmes font aussi partie de la famille des systèmes hétérochrones.

Quelque soit le modèle d'horloge d'un composant, son gel est indispensable pour rendre ce dernier insensible aux délais ou inactif afin de réduire sa consommation. On trouve dans la littérature trois principaux modes de gel. Ce sont :

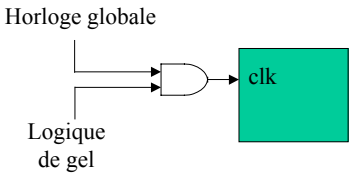


Figure 2-12 : Horloge globale combinatoire

- Les horloges globales combinatoires (clock gating) dont le principe repose sur un générateur d'horloge externe (quartz ou PLL) assisté par des portes logiques ET dont la fonction et de laisser passer ou non à volonté le signal d'horloge.

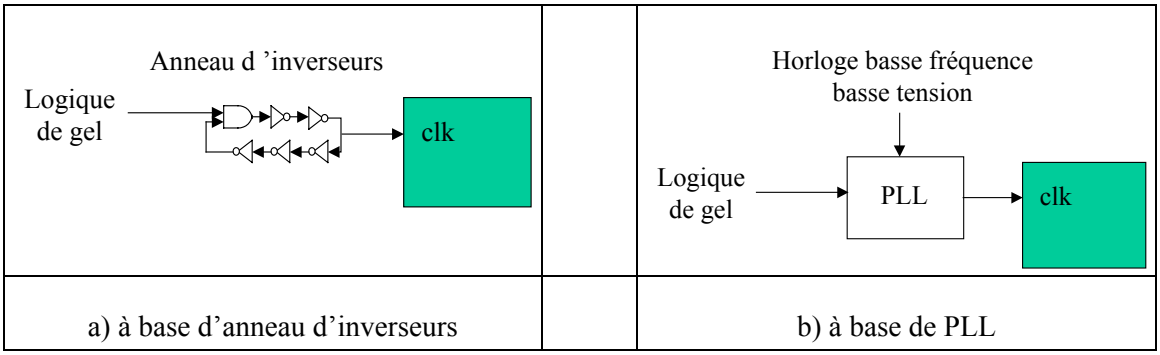
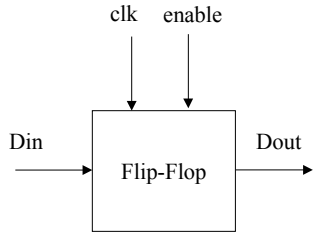


Figure 2-13 : Horloges locales suspensibles

- Les horloges locales « suspensibles » (clock pausing) [YUN96] qui reposent sur un générateur d'horloge local à base d'anneau d'inverseurs en nombre impair. Cette horloge locale est agrémentée d'une porte logique ET en série avec les inverseurs qui permet de stopper l'oscillation dans un état stable bas. L'intérêt (mais aussi l'inconvénient pour certains auteurs) de ces générateurs d'horloges est qu'ils évoluent avec les dérives du procédé de fabrication, de la température et du voltage. Ils permettent cependant d'obtenir des circuits fonctionnels, quelles que soient ces dérives [SJO00].



a) représentation schémas bloc

```
process (CLK)
    begin
        if (CLK'event and CLK='1') then
                if (ENABLE='1') then
                DOUT <= DIN;
            end if;
        end if;
end process;
```

b) modèle RTL

Figure 2-14 : Horloges avec signal de validité

- Les horloges accompagnées d'un signal de validité (clock enable). Ce sont des horloges locales ou globales qui oscillent en permanence, et ce jusqu'à l'entrée de chacune des bascules élémentaires du système. Les bascules sont pourvues d'une entrée « clock enable » qui permet de valider la transition d'horloge. La combinaison des signaux « load », « clock » et « clock enable » détermine si le stockage à lieu ou pas. Ce type de bascule est implanté dans les FPGA [XIL04] et des outils de conversion ASIC vers FPGA transforment automatiquement les horloges combinatoires en horloges validées [SYN03].

4. Systèmes insensibles à la latence (LIS)

Les systèmes insensibles à la latence sont des SoCs synchrones dont le réseau de communication est qualifié de « pseudo-asynchrone » par les auteurs par analogie avec les GALS [CAR99]. La notion de « pseudo-asynchronisme » signifie que le temps de traversée des signaux sur les longues lignes n'est plus quelconque mais est toujours un multiple du cycle d'horloge global. Dans ce contexte, et pour que le système complet soit équivalent au niveau transactionnel, les composants doivent être rendus résistants aux délais/retards des entrées et des sorties. Les systèmes insensibles à la latence sont un sujet qui couvre à la fois un formalisme, une théorie, une méthodologie et une méthode d'analyse et d'optimisation de performances pour des systèmes synchrones. Nous les présentons maintenant en détail.

4.1. Méta modèle LSV des signaux estampillés

Le formalisme LSV (Lee and Sangiovanni-Vincentelli) [LEE98] est un modèle mathématique dénotationnel (au sens de [STO77]) qui permet de définir formellement les systèmes synchrones et asynchrones sur la base de la notion d'événements tagués. Les tags représentent des relations

d'ordre (le temps s'il s'agit d'une horloge) entre événements sur un même signal et entre signaux. Ils permettent de définir deux classes de systèmes : les systèmes synchrones dans lesquels tous les signaux sont synchrones entre eux, et les systèmes asynchrones dans lesquels les signaux sont asynchrones entre eux. La définition de ces deux classes n'est pas telle que l'une soit le complément de l'autre : il existe une classe située « entre les deux » nommée classe des systèmes Globalement Synchrones et Localement Asynchrones (GALS). Les systèmes de la classe des GALS insensibles à la latence sont donc des systèmes partiellement asynchrones et partiellement synchrones selon la sémantique du modèle LSV. L'intérêt de ce modèle est qu'il permet de modéliser des circuits aussi bien que des systèmes distribués dans lesquels des « ilots » synchrones communiquent entre eux de façon asynchrone. Un système GALS est alors modélisé par un ensemble de processus synchrones communiquant au travers d'un processus asynchrone qui représente le media de communication.

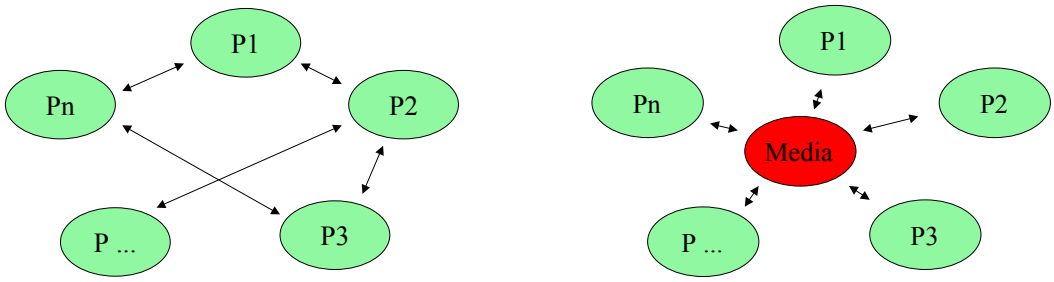


Figure 2-15 : Représentations classique et LSV d'un système multi-processus

Le média de communication est représenté par un processus central globalement asynchrone qui découple temporellement la communication entre les processus P*i* et demeure localement synchrone à chacune de ses interfaces avec les processus. Le processus média a pour fonction de remplacer le point de synchronisation de deux processus communiquants par deux points de synchronisation distincts : un pour l'émission et un pour la réception de la donnée. Ces nouveaux points de rendez-vous sont séparés par une latence qui dépend de l'implantation du média. L'exploitation du modèle LSV pour concevoir les GALS implique que l'on doive rendre insensibles à l'absence de données les processus synchrones P*i*. Enfin, le formalisme LSV a l'avantage de proposer un cadre unifié qui autorise la comparaison de différents modèles de calcul entre eux [LEE98]. Le méta-modèle LSV a servi à définir formellement la théorie, puis la méthodologie des systèmes insensibles à la latence.

4.2. Théorie et méthodologie LIS

La théorie des systèmes insensibles à la latence (LIS, ou Latency Insensitive Systems) [CAR01] est le fondement d'une méthodologie [CAR99a] correcte par construction de conception de systèmes monopuces à partir d'une spécification système synchrone. Elle permet d'utiliser les flots CAO actuels de conception de circuits synchrones. Son objectif est de réutiliser des composants synchrones (IP tierce partie ou IP synthétisable) et de préserver la fréquence de fonctionnement maximale du circuit grâce à 1) l'adjonction d'éléments mémorisants sur les chemins critiques des lignes équipotentielles les plus longues et

2) l'encapsulation automatique des composants dans une coquille (shell) assurant leur insensibilité à la latence introduite par les éléments mémorisants. Elle est exprimée à l'aide du formalisme LSV. Cette théorie est aussi présentée sous le nom de « protocoles insensibles à la latence » [CAR99b]. Elle a été testée avec succès sur la conception du processeur PDLX, un dérivé du DLX [HNN96].

Grâce au formalisme LSV, [CAR03] (théorème 3) prouve que les processus des LIS satisfont les propriétés d'endochronie et d'isochronie des processus « patients » de la désynchronisation de programmes synchrones réactifs. Il exploite alors un résultat important de la théorie des programmes synchrones réactifs [BNV00] : il s'agit du fait que « *si chaque processus d'un programme synchrone est endochrone et que tous les processus sont isochrones par paires, alors l'implantation du programme sur une architecture LIS est sémantiquement équivalente* ». Cette propriété des LIS permet d'affirmer que tout système distribué de type LIS est sémantiquement équivalent (en terme de séquences de valeurs pour chaque signal, abstraction faite des latences) au modèle synchrone de la spécification fonctionnelle initiale. Cette preuve formelle valide la transformation des processus du système synchrone en processus patients en vue de l'implantation LIS. Les processus (et donc les IPs) candidats à la réutilisation sont transformables en processus patients s'ils sont synchrones, fonctionnels (avoir des entrées et des sorties), causaux [LEE98] et suspensibles. La transformation consiste à encapsuler le processus synchrone par un wrapper (ou shell). Ce wrapper est constitué de processus qui, par composition avec le processus synchrone initial, dérivent un nouveau processus qui à la propriété d'être patient et sémantiquement équivalent en terme de latence. Ces processus d'encapsulation sont :

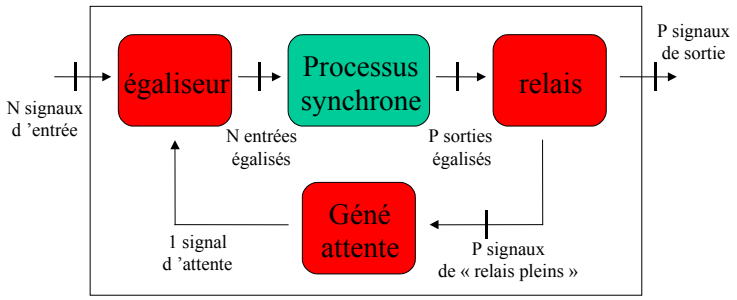


Figure 2-16 : Schéma bloc d'un processus patient

- L'égaliseur (equalizer) dont la fonction est de resynchroniser les entrées. Dans ce cadre, le terme resynchroniser signifie « insérer des états d'attente jusqu'à ce que tous les signaux d'entrée aient une valeur définie ».

- Les stations de relais étendues (extended relay stations) sont des éléments mémorisants qui préservent l'ordre, ont une capacité de deux places, et sont aptes à signaler s'ils sont remplis par un signal de rétroaction vers le générateur d'états d'attente. Il s'agit de FIFO de profondeur deux. Il peut y avoir de une à plusieurs stations de relais sur un chemin de données qui mène d'un composant à un autre. Leur nombre dépend de la longueur des chemins critiques qu'il faut briser pour préserver la fréquence maximale au niveau système.

- Le générateur d'états d'attente qui, à partir des signaux de remplissage des stations de relais étendues, génère un signal nommé τ (sémantique « attendre ») en direction de l'égaliseur

pour l'obliger à générer exclusivement des signaux d'attente jusqu'à ce que la condition de blocage des stations relais disparaisse.

Le processus patient ainsi obtenu a la propriété de préserver le nombre et l'ordre des événements sur chacun des ports d'entrée et de sortie. A partir de ces processus patients, la méthodologie consiste à utiliser les outils standards 1) de placement-routage des blocs durs des composants et 2) d'analyse statique de temps de propagation (static timing analysis). L'analyse statique extrait les chemins critiques qu'il faut briser et identifie quelles sont les équipotentielles inter-composants sur lesquelles il faut ajouter des stations de relais de façon à réduire le chemin critique du circuit complet. Il y a ensuite ajout de stations de relais pour briser ces longues lignes et re-synthèse physique pour optimiser la fréquence.

4.3. Implantation des processus patients

L'implantation des processus patients proposée par [CAR02] est illustrée par la Figure 2-3. Le composant synchrone d'origine, la « perle » (pearl module) est un circuit synchrone ayant trois entrées et deux sorties.

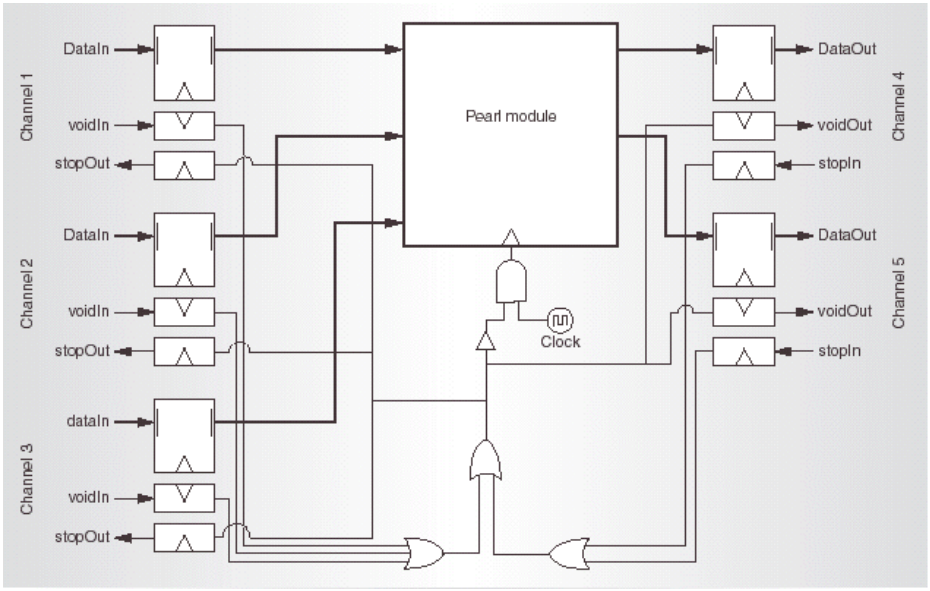


Figure 2-17 : Implantation d'un processus patient

Il est encapsulé dans une coquille (shell) constituée par :

- Les ports d'entrée et de sortie d'origine du circuit qui transportent les signaux de la spécification synchrone
- Pour la synchronisation interne :
  - A chaque entrée, un signal d'entrée « voidIn » qui qualifie le signal entrant de valide ou invalide. Les entrées qualifiées d'invalides propagent des états d'attente τ générés par le flot amont.

- A chaque sortie, un signal d'entrée « stopIn » qui représente la capacité du flot aval à consommer la donnée.

- Pour la synchonisation externe :

  - A chaque entrée, un signal de sortie « stopOut » qui bloque le flot amont si le circuit ne peut pas consommer les données entrantes. Ces signaux dépendent de deux phénomènes : 1) si l'une au moins des entrées n'est pas disponibles, alors toutes les entrées sont bloquées (processus d'égalisation), et 2) si le flot aval ne peut consommer les sorties qui seraient produites par le circuit alors toutes les entrées sont bloquées (processus de génération des états d'attente).

  - A chaque sortie, un signal de sortie « voidOut » qualifie la sortie de valide ou d'invalide selon les circonstances. Les sorties qualifiées d'invalides propagent des états d'attente τ générés vers le flot aval.

Enfin, l'horloge du circuit, ainsi que la matérialisation des signaux « stopOut », est câblée en séquence avec une porte ET conditionnée par la conjonction de la négation de toutes les conditions d'arrêt. Il s'agit d'une horloge combinatoire dite « sur portes » (gated clock ou clock gating). Cette technique d'horloge combinatoire, bien que très utilisée pour la réduction de consommation de l'ordre de 10 à 50% [SUE94][BEN95][MOT02][TEN03], pose quelques difficultés de réalisation aussi bien avec les FPGA qu'avec les ASIC. Les FPGA possédent un nombre limité d'arbres d'horloge. Ils ne permettent pas d'implanter cette stratégie à grande échelle, c'est à dire dans le contexte pour lequel la méthodologie a été créée. Par exemple, un Virtex dispose de quatre arbres d'horloge globaux et le VirtexII, bien que plus récent, est limité par ses huit DCM (Digital Clock Manager) [XIL04]. Enfin, malgré les efforts des vendeurs d'outils de CAO micro-électronique pour l'aide à la conversion d'ASICs contenant des horloges combinatoires en FPGA [SYN03], les fabriquants de processeurs et d'ASIC conseillent régulièrement à leurs concepteurs de circuits d'éviter autant que possible d'utiliser des horloges combinatoires car cela pose des problèmes d'insertion automatique de chaîne de test, d'augmentation du chemin critique et du décalage d'horloge [TI98]. Par exemple, Infinéon indique dans ses règles de « bon codage » VHDL [INF00] (règle n° 1.2.6.1) de veiller à n'utiliser des horloges combinatoires que lorsque *c'est absolument indispensable*.

La station de relais minimale est de profondeur deux pour pouvoir assurer le meilleur débit comme [CAR99] le prouve. Les deux registres « main » et « auxiliary » de la Figure 2-18 représentent ces deux places. La logique de contrôle pilote les signaux de lecture/écriture de ces deux registres et l'aiguillage des entrées/sorties. D'autres implantations des stations de relais ont été proposées. [CHE01] propose des FIFO à séquencement mixtes permettant de réaliser des interfaces (des stations de relais) entre des sous-systèmes synchrones de domaines d'horloge différents ou entre sous-systèmes synchrones et asynchrones.

L'implantation des stations de relais en tant que processus patients est la suivante :

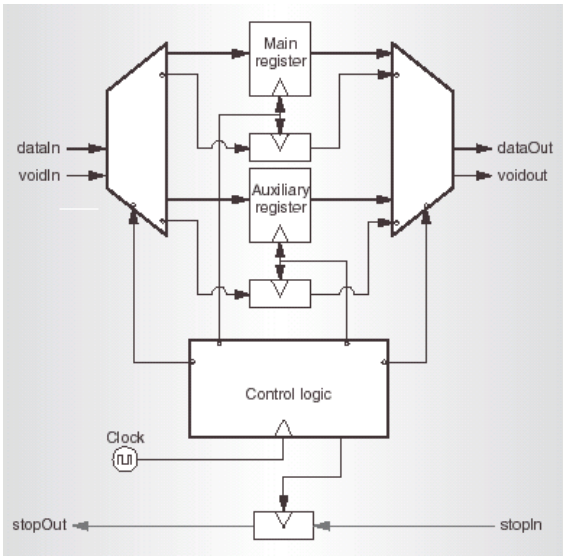


Figure 2-18 : Station de relais

A des fins de comparaison plus aisée, nous représentons à partir de maintenant le processus patient par le schéma de la Figure 2-19. Sa structure fait apparaître ce qui nous paraît être le plus important : 1) la nature exclusivement combinatoire de la logique de suspension du composant et 2) l'existence des signaux amont (*voidout* et *stopin*) et aval (*voidin* et *stopout*) typiques d'un protocole de synchronisation à quatre phases.

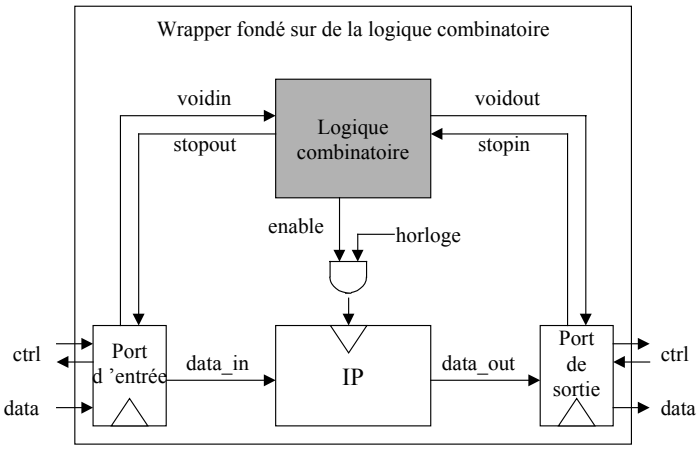


Figure 2-19 : Modèle de processus patient de Carloni

4.4. Analyse et optimisation des performances

Bien que la théorie et la méthodologie des LIS garantissent par construction l'équivalence sémantique du circuit avec son modèle synchrone, il n'en est pas de même en ce qui concerne son débit. La méthodologie garantit seulement de ne pas affecter la performance globale du système si le circuit ne contient pas de chemin de retour. Ainsi, seule une structure de réseau de communication arborescent est-elle susceptible de ne pas être pénalisée par l'ajout de latence

inter-composants. C'est un phénomène tout à fait identique à celui qui se rencontre dans les microprocesseurs RISC (MIPS, SPARC, ARM, …) qui optimisent le débit d'exécution d'une séquence d'instructions sans branchement au prix d'une latence [HNN94].

Afin de contrôler la performance du système [CAR00] propose une modélisation du phénomène à l'aide de graphes orientés et pondérés nommés « graphes des systèmes insensibles à la latence » (lis-graphs). La pondération des arcs représente la latence introduite par les stations de relais et permet, par une recherche du cycle (s'il y en a un) de poids maximal, d'extraire la contrainte de performance : à savoir la longueur en nombre de cycles d'horloge du plus grand cycle. Cette connaissance du cycle dit critique permet de calculer le temps de cycle du système et d'en déduire le débit du système qui est son inverse.

Grâce à cette modélisation, l'adjonction de stations de relais peut être caractérisée en terme de gain (ou de perte) de performances. On dispose ainsi d'un outil mathématique qui permet de contrôler a postériori, l'impact de l'insertion de stations de relais. La technique d'insertion de stations de relais a pour but de réduire le chemin critique, elle est aussi nommée recyclage (recycling) par les auteurs.

4.5. Restrictions des LIS

Malgré leurs nombreuses qualités, les LIS ont des contraintes qui réduisent leur application.

1) Ils sont mono-horloge car ils reposent sur une méthodologie de réutilisation de composants et d'un réseau de communication tous pilotés par la même horloge.

2) Les processus patients sont sensibles à la totalité de leurs entrées et sorties à chaque cycle. Ainsi les entrées et les sorties de tous les processus patients sont-elles traitées de la même façon : une valeur doit être présente sur chaque entrée et chaque sortie doit être capable de consommer la valeur produite pour que le composant soit activable et avance d'un pas de calcul synchrone. La non disponibilité d'une seule entrée ou d'une seule sortie entraîne la suspension du composant. Or, dans le cas général, un circuit synchrone peut n'exploiter qu'un sous-ensemble de ses entrées à chaque cycle. Ce mode de fonctionnement n'est malheureusement pas supporté par les LIS.

3) L'adjonction des stations de relais 1) impacte les circuits ayant des chemins de retour et 2) présuppose une réservation de place lors du placement initial des composants pour pouvoir être insérées sans nécessiter un re-placement complet du circuit. Il y a donc potentiellement une perte de surface en prévision de leur insertion.

5. LIS optimisés

Les LIS optimisés ont pour but de relaxer les contraintes des LIS « purs » selon quatre axes qui sont : 1) les systèmes multi-horloges, 2) les processus sensibles à un sous-ensemble pertinent des entrées-sorties, 3) d'autres structures du réseau de communication que les canaux point à point et 4) l'élimination par ordonnancement des signaux de synchronisation inter-composants.

5.1. Multi-horloges

L'introduction de domaines d'horloge différents pour les composants réintroduit le problème de méta-stabilite rencontré dans les GALS. Cette contrainte est traitée avec des FIFO et des horloges « suspensibles » dédiées aux circuits GALS, telles que [MOO02], et GASP [SUT01]. Ces systèmes sont aussi nommés :

- « systèmes réactifs hétérogènes » dans la théorie de [BNV03] qui permet de poser les conditions de validité et de valider la conception d'un circuit multi-horloges.

- « systèmes polychrones » dans la théorie de [GUE03], qui conclut que l'échange d'horloges de resynchronisation est nécessaire pour garantir l'équivalence sémantique d'une implantation multi-horloges.

5.2. Réactivité aux seules entrées-sorties significatives

[MON04] propose d'encapsuler les composants synchrones dans un wrapper plus évolué dont le comportement est du type « burst » au sens de [YUN99]. Le nouveau modèle de wrapper permet d'asservir le composant sur l'attente des seules entrées-sorties requises à chaque cycle.

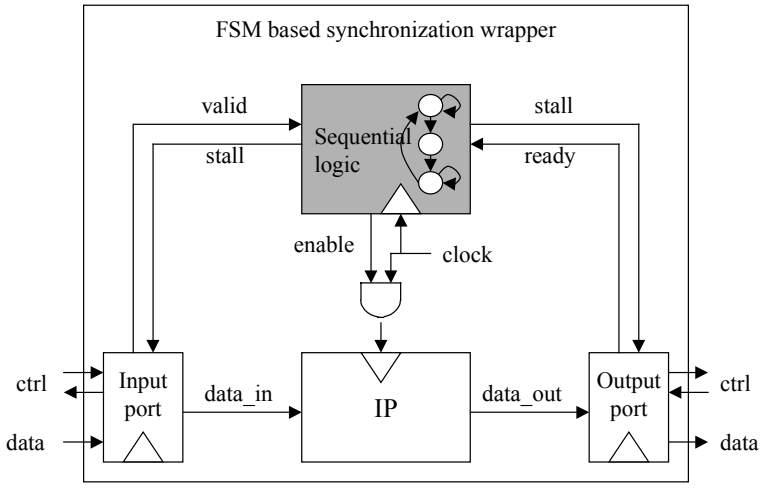


Figure 2-20 : Modèle de processus patient de Singh

Cette méthode se situe entre les NOC [BEN02a], qui sont plus complexes et orientés protocoles à paquets, et les systèmes insensibles à la latence. Elle repose sur une technique de gel de l'horloge réalisée strictement à base d'horloges combinatoires et de machines d'états finis. Elle

permet de construire des systèmes multi-composants à horloges locales de fréquences différentes.

L'implantation sous forme de machine d'états finis synchrone est similaire aux machines d'états asynchrones de [NOW94] (AFSM, Asynchronous Finite State Machine) qui ne sont sensibles qu'à un sous-ensemble des entrées-sorties. La Figure 2-20 illustre que les signaux amont et aval de synchronisation sont maintenant issus d'une FSM et peuvent être pilotés individuellement au lieu d'être la simple négation du signal d'horloge combinatoire de la Figure 2-17. La FSM de suspension du composant est une machine d'états finis « naïve » de type Mealy.

5.3. Autres modèles d'implantation du réseau de communication

Le réseau pseudo-asynchrone des LIS est orienté mot (n bits en parallèle sur un port), ne contient aucune fonction de routage (au sens de routage de paquets) et a des temps de traversée parfaitement connus. Malgré le cadre restrictif des LIS par rapport aux NoC (comme présenté dans le chapitre 2.6), il existe des topologies de réseau pseudo-asynchrones autres que les liaisons point à point. Ces topologies conservent toutes la propriété des processus patients : préserver le nombre et l'ordre des évènements sur chacun des canaux de communication, que ces derniers soient implantés par des longues lignes discrétisées temporellement par des stations de relais ou par tout autre moyen.

La méthodologie de [MON04] est assistée par un ensemble de composants orientés communication qui proposent des services de type duplication d'un flux d'entrée sur deux flux sorties (fork), éclatement d'un flux d'entrée sur plusieurs flux de sorties (split), et réunion de plusieurs flux d'entrée en un seul flux de sortie (merge). Ces composants sont fournis sous la forme de blocs synchrones lorsque le réseau de communication est implanté de façon synchrone (LIS) et sous la forme de blocs asynchrones (avec des signaux de synchronisation de type poignée de main) lorsque ce dernier est implanté de façon asynchrone (GALS). Ils permettent, par composition, de construire aisément des topologies physiques de réseaux de communication différentes des stricts canaux point à point. De même [VIL03] propose une topologie en double anneau bidirectionnel constitué de l'aboutement bidirectionnel de canaux point à point. Ces composants sont considérés comme des blocs réutilisables au même titre que les composants synchrones et illustrent la réutilisabilité de composants orientés communication.

5.4. LIS ordonnancés

Comme le fait remarquer à juste titre [CAR00], les cycles nuisent à la vitesse maximale de calcul du système en introduisant des latences dues aux stations de relais sur les longues lignes inter-composants. Nous illustrons ce fait par un exemple de cycle minimal entre deux processus A et B. La Figure 2-21 représente ce plus petit cycle. A chaque coup d'horloge A et B produisent une sortie qui dépend de l'entrée au coup précédent. Nous notons a_i et b_i les valeurs qui transitent à l'instant i sur les interconnexions entre les deux processus. Les processus sont représentés par des boîtes grises qui signifient que les processus sont actifs à chaque coup d'horloge (c'est à dire quel que soit i). Ainsi, les valeurs de a_i et b_i se succèdent au rythme d'une par coup d'horloge. Il s'agit du cas optimal.

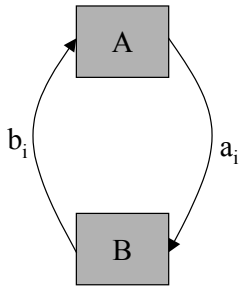


Figure 2-21 : Cycle minimal entre deux processus

Si les composants sont suffisamment éloignés l'un de l'autre sur le silicium ou dans le FPGA, il peut être nécessaire d'introduire une à plusieurs stations de relais le long des interconnexions. Pour les besoins de l'exposé, nous faisons l'hypothèse que l'une des interconnexions est moins bien routée que l'autre et qu'elle a besoin d'une station de relais alors que l'autre n'en a pas besoin. La Figure 2-22 illustre cette insertion. La valeur « τ » placée sur une interconnexion signifie que la valeur au pas *i* n'est pas valide et qu'il y a donc génération d'un état d'attente pour le composant destinataire au pas *i+1*.

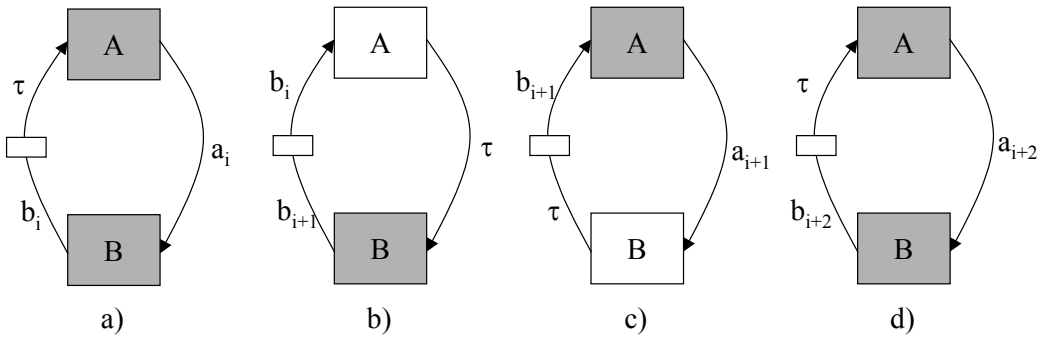


Figure 2-22 : Impact de l'insertion d'une station de relais dans un cycle

Nous décrivons le processus sur quatre cycles afin d'illustrer l'ajout de latence dans le cycle par la station de relais :

- A l'instant *i*, cas a), A et B sont actifs et produisent leurs sorties a_i et b_i. La station de relais, qui contient le signal τ (donnée invalide), produit le signal τ qui va bloquer le composant A au cycle *i+1*.

- A l'instant *i+1*, cas b), B est actif puisqu'il a reçu a_i , il produit donc b_{i+1} . La station de relais propage la valeur b_i en entrée de A qui sera actif au cycle *i+2*. A, ayant été bloqué par son entrée, produit lui aussi une valeur τ qui aura pour effet de bloquer B au cycle *i+2*.

- A l'instant *i+2*, cas c), B est inactif puisqu'il a reçu τ au cycle précédent , il produit donc τ. La station de relais propage la valeur b_{i+1} en entrée de A qui sera actif au cycle *i+3*. A, produit la valeur a_{i+1} qui aura pour effet de débloquer B au cycle *i+3*.

- A l'instant *i+3*, cas d), A et B sont actifs et produisent les valeurs a_{i+2} et b_{i+2}. La station de relais produit le signal τ.

A l'issue de ces quatre cycles, on remarquera que le cas d) est identique, à l'instant *i* près, au cas a) et que ce fonctionnement est cyclique. Le nombre de cycles pour atteindre le cas b) à partir du cas a) est de trois alors que dans le cas optimal il n'est que de deux.

[CAS04] exploite cette caractéristique cyclique et totalement prédictible du comportement du réseau pseudo-asynchrone de façon à minimiser la quantité de matériel nécessaire à la communication inter-composants. Il prouve que l'ordonnancement des phases d'activité et d'inactivité des composants (Functional Block Activation Scheduling) peut être déterminé statiquement pour l'ensemble des processus concurrents. La réduction de matériel repose sur le fait que si les instants d'activation des blocs sont tous prédictibles, alors le protocole de communication asynchrone n'a plus aucune raison d'être. L'optimisation repose sur 1) la suppression des signaux de synchronisation asynchrones des interfaces des wrappers de façon à limiter la congestion du routage et 2) l'implantation d'une stratégie de gel d'horloge de type horloge combinatoire pour laquelle la séquence des phases d'activité/inactivité est stockée dans un registre à décalage local au wrapper. La Figure 2-23 illustre cette architecture de wrapper.

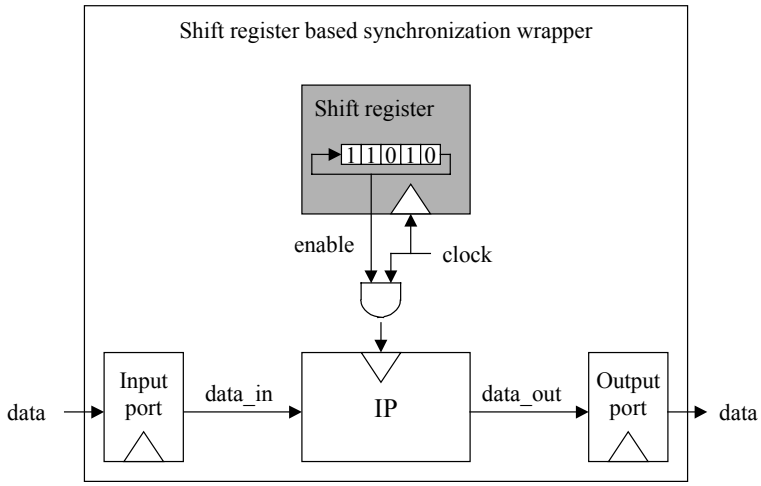


Figure 2-23 : Modèle de processus patient de Casu

Cette approche, bien qu'elle soit meilleure que l'approche classique LIS en terme de surface silicium, est très limitée en ce qui concerne les circuits réels car 1) en l'absence des signaux de synchronisation amont et aval elle ne tient pas compte des irrégularités de la communication avec l'environnement du circuit et 2) elle repose exclusivement sur une stratégie d'horloge combinatoire qui est peu adaptée à une synthèse pour FPGA. Elle n'est donc pas applicable à tous les circuits. Seule la technologie ASIC (nombre d'horloges combinatoires quelconque) pour des systèmes tels que l'environnement soit « toujours près » (entrées toujours disponibles à temps, sorties toujours libres à temps) permet de réaliser des circuits exploitant cette technique d'optimisation.

6. Conclusion

Après avoir constaté que l'accroissement de complexité des SoCs avec l'avènement des technologies VDSM était la principale cause de ralentissement des fréquences d'horloges, nous avons successivement 1) exploré l'ensemble des approches méthodologiques dont le but est d'optimiser la vitesse de fonctionnement du système, 2) présenté le concept des systèmes globalement asynchrones et localement synchrones (GALS) et 3) présenté en détail la méthodologie des systèmes insensibles à la latence (LIS) ainsi que 4) ses optimisations récentes.

La méthodologie des systèmes insensibles à la latence repose sur l'emploi de wrappers dont le rôle est de synchroniser un composant synchrone avec les irrégularités des flots de données amonts et avals. La complexité croissante des quantités d'entrées-sorties font que ces wrappers deviennent eux aussi des circuits synchrones de plus en plus complexes. Les meilleures implantations actuellement proposées reposent sur des machines d'états finis [MON04]. Nous savons par expérience que ces modèles d'implantation sont difficilement synthétisables physiquement lorsqu'il sont volumineux et qu'ils ont des fréquences maximales de fonctionnement inversement proportionnelles au nombre d'états. L'emploi de la méthodologie LIS dans des conditions réalistes est donc fortement compromis si les wrappers ne sont pas améliorés.

Notre contribution a pour but de proposer un modèle d'implantation LIS pour les circuits synchrones synthétisés par l'outil de synthèse comportementale GAUT. En particulier, les wrappers de synchronisation sont des machines d'états optimisées en surface et en fréquence et sont toujours synthétisables. Nous proposons d'implanter les machines d'état de synchronisation sous la forme de machines d'états finis micro-codées que nous nommons *processeurs de synchronisation*. Les circuits ainsi synthétisés seront alors mis en œuvre dans le cadre méthodologique d'une plate-forme de prototypage rapide pour applications de radiocommunications numériques de nouvelle génération. Notre méthodologie de conception au niveau système intègrant le modèle LIS sera alors validée par la conception, la simulation et le test de fonctions jugées actuellement très complexes par nos partenaires industriels et académiques lorsqu'elle sont développées au niveau RTL.

Chapitre 3

Conception de la plate-forme

1. Introduction

Nous avons présenté dans les deux chapitres précédents la problématique relative à la difficulté d'obtenir des circuits performants en vitesse lorsque leur complexité s'accroît et fait un état de l'art concernant les méthodologies d'optimisation de la fréquence aux différents niveaux d'abstraction possibles. Ces optimisations vont du niveau technologique de fabrication des semi-conducteurs jusqu'au niveau de conception système. Nous avons conclu que la voie la plus prometteuse se situe au niveau système. Il s'agit des méthodologies dites des systèmes globalement asynchrones et localement synchrones (GALS) et des systèmes insensibles à la latence (LIS). Dans le contexte LIS synchrone qui nous intéresse, les meilleures propositions actuelles d'implantation des mécanismes de synchronisation des composant synchrones ont néanmoins comme principal handicap de ne pas être toujours synthétisables, ni optimales en surface et en vitesse.

Dans ce chapitre nous présentons notre contribution en terme de plate-forme de prototypage rapide dans le cadre du projet PALMYRE. Nous étudions tout d'abord quelles sont les contraintes des applications de radiocommunications numériques actuelles afin de dimensionner les puissances de calcul et de communication requises. Puis nous étudions les différentes méthodologies de prototypage afin de déterminer la plus adéquate d'entre elles. Nous nous concentrons alors sur le prototypage semi-dédié et sur le prototypage/conception orienté plate-forme. Nous faisons une analyse critique des plates-formes de prototypage disponibles. Enfin nous proposons une architecture de plate-forme constituée des plates-formes matérielles, logicielles et systèmes. Nous décrivons ensuite l'approche méthodologique du LESTER au niveau système dans laquelle nos travaux s'inscrivent et détaillons plus particulièrement les plates-formes matérielles et logicielles que nous proposons. Plates-formes matérielles et logicielles sont des éléments critiques en terme de performances et, dans le chapitre suivant, nous présentons notre contribution au niveau système dans le cadre de l'introduction de la théorie des systèmes insensibles à la latence dans l'outil de synthèse de haut niveau GAUT.

2. Contraintes des applications de radiocommunications

Les applications de radiocommunications connaissent depuis quelques années une véritable explosion. La création de bouquets satellites (Canal Satellite, TPS,Via Digital, Telepiu, Sky Digital, Direct TV, PrimeStar, …), câble (Lyonnaise Câble, Medoaone, …), et terrestre (ONDigital en Angleterre) ainsi que la téléphonie portable (Bouygues, Orange, et SFR en France) en sont les retombées grand public les plus évidentes. De plus, l'évolution des produits dits « portables » fait que, du mainframe aux assistants personnels (pagers, PDA, téléphones) en passant par les PCs, la liaison radio de courte portée (de 1 mètre à 1000 mètres) est devenue une absolue nécessité pour interfacer simplement les nombreux équipements entre eux. Nous citons, à titre d'exemple, le modèle de référence Multisphere du Wireless World Research Forum [WWR04] qui fait intervenir 6 niveaux de communication radio qui vont de l'environnement personnel, proche et concret (niveau 1, PAN ou Personal Area Network) à un environnement interconnecté (à la « Internet »), lointain et abstrait (niveau 6, CyberWorld). Enfin, la

reconfigurabilité des systèmes de transmission, nommée « radio logicielle » (SDR, Software Defined Radio [SDR04]) doit permettre de produire des équipements suffisamment auto-adaptables (reconfigurables) à des coûts moindres et ainsi permettre le développement rapide du marché des équipements personnels communiquant via un canal radioélectrique.

Ce développement rapide est dû, d'une part aux nombreux avantages intrinsèques au format numérique (compression, qualité constante, interactivité) ainsi qu'à l'existence de standards et d'autre part au progrès des technologies de fabrication des circuits intégrés CMOS. Les différents media de transmission (câble, satellite, ou voie hertzienne) emploient les mêmes principes et donc les mêmes types de traitements. Rappelons, dans le cas général, le principe d'une chaîne de transmission numérique à l'aide de la Figure 3-1.

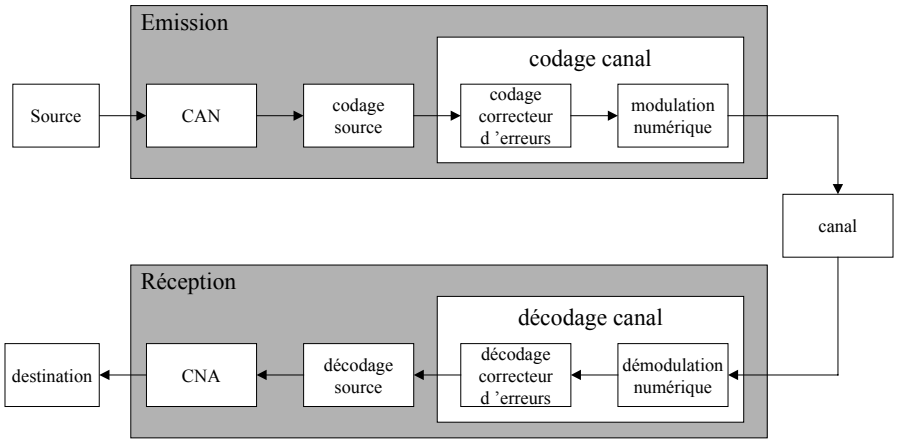


Figure 3-1 : Principe d'une chaîne de transmission numérique

Dans un premier temps, à l'émission, on effectue la capture des informations provenant d'une source analogique et on les *convertit en signaux numériques* (bloc CAN ou Conversion Analogique Numérique). Suit ensuite la phase de *codage source* dont la fonction est de représenter de façon efficace (ayant un minimum de redondance) les signaux numériques en une séquence binaire. Le signal ainsi obtenu doit être transmis par l'intermédiaire d'un medium de communication tel le câble, le satellite, la ligne téléphonique ou les ondes hertziennes. Ces canaux de transmissions ne sont pas parfaits et peuvent modifier le signal émis. Le signal reçu n'est donc pas identique au signal émis et, pour détecter et corriger ces erreurs (bruit du canal, interférences, échos, …), des bits de redondance sont ajoutés lors de la phase du *codage correcteur d'erreurs*. Ensuite le signal bande de base est transposé en un signal modulé à une fréquence porteuse dépendante du medium de communication. La *modulation* effectue cette tâche.

Dans un deuxième temps, à la réception, on reconstruit le signal émis par la source. Le signal reçu est tout d'abord transposé en bande de base et une estimation des symboles émis est effectuée par le *démodulateur*. Les erreurs de transmission sont détectées et corrigées dans la mesure du possible par le *décodage correcteur d'erreurs*. La reconstruction du signal d'origine est alors effectuée avec le *décodage source*. En dernière étape, une *conversion des signaux numériques en signaux analogiques* (bloc CNA pour Conversion Numérique Analogique) a lieu si la destination est analogique.

Les puissances de calcul nécessaires dépendent bien sûr des canaux de communication et des applications visées. Dans le domaine de la télévision numérique au standard DVB (Digital Video Broadcasting), les canaux de transmission ont des caractéristiques particulières. Le signal satellite est stable et faible (émetteur lointain) et le signal câble est puissant et soumis aux échos. Il existe donc plusieurs techniques de modulation pour utiliser au mieux les canaux de communication. Les modulations monoporteuses modulent une seule porteuse à haut débit (modulation en amplitude et phase QAM 16/64/256 pour le câble et modulation à déplacement de phase à quatre états QPSK pour le satellite). Les modulations multiporteuses (OFDM ou Orthogonal Frequency Division Multiplexing) modulent plusieurs porteuses à débits moindres. Enfin, à titre d'exemple dans le cadre de la télévision numérique via le satellite, la puissance de calcul du décodeur canal dépasse la dizaine de milliards d'opérations (10 Gops) par seconde à cause des hautes fréquences symboles utilisées. Par opération on entend opération arithmétique du type addition ou multiplication telles qu'elles apparaissent dans les filtres et les transformées de Fourier des différents blocs de la chaîne de transmission. [GAY99] présente à ce titre une analyse très détaillée dont la Figure 3-2 présente les conclusions.

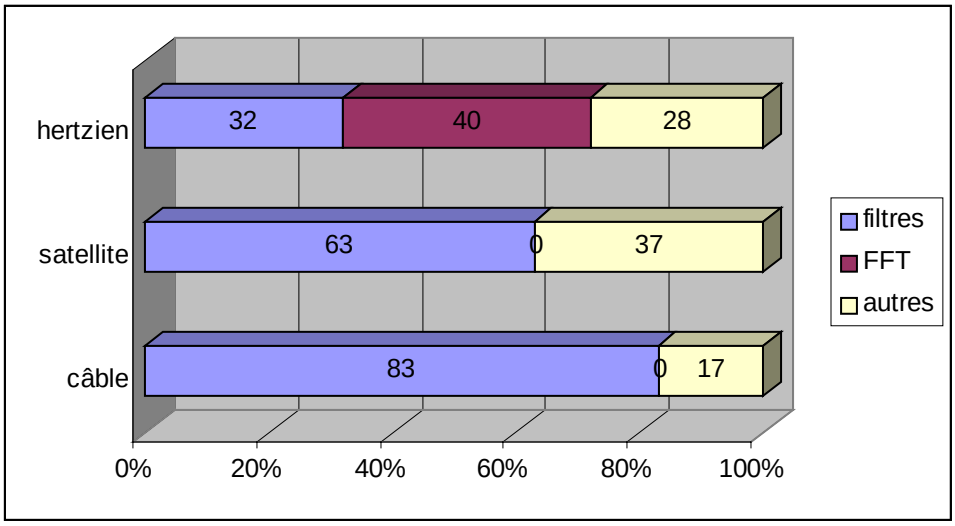


Figure 3-2 : Répartition de la puissance de calcul mise en œuvre par type d'algorithme

On observe sur la Figure 3-2 que les algorithmes de type filtres (filtrage anti-repliement, interpolateurs, Nyquist, égalisation, transposition en bande de base, et interpolation fréquentielle et temporelle) et la transformée de Fourier représentent la majeure partie des traitements. Toute architecture dédiée à ce type d'applications devra donc favoriser l'implantation efficace (en terme de débit) d'algorithmes orientés filtres et transformées de Fourier.

Dans le domaine des interfaces radio courte portée, [MAN02] estime que 1) les puissances de calcul requises par les différents types d'interfaces radios nécessitent de l'ordre de 10 Mops pour la 1ère génération à 13 Gops pour la 3ème génération et que 2) les débits sont de quelques Kbit/s pour les BAN (Body Transducers) à plus de 100 Mbit/s pour les WLAN (accès aux stations de base). Il fait aussi remarquer que les puissances de calcul liées aux applications, hors transmission des données, atteignent 5 Gops pour le codage/décodage MPEG 4 et sont de l'ordre de 100 Gops pour les traitements graphiques temps réels. La progression des besoins en puissance de calcul pour ces dernières excède très largement la loi de Moore et sort du cadre de notre étude qui se limite aux applications de transmissions de données.

Nous retiendrons que les applications de radiocommunications ont pour principales caractéristiques d'être :

- *Régulières* car orientées données plutôt que contrôle,
- Et *intensives* en calculs de types filtres et transformées de Fourier.

Nous concluons que les applications de radiocommunications que nous ciblons nécessitent des architectures à routage fixe « globalement de type pipeline avec peu de voies de retour» et à grain suffisamment gros pour exécuter sur un nœud un algorithme de type filtre ou FFT. Les puissances de calcul par nœud doivent être de l'ordre du Gops et les débits échantillons inter nœuds de l'ordre du Gbit/s.

Après avoir statué sur les contraintes des applications de radiocommunications, nous poursuivons par l'étude des méthodologies de prototypage qui proposent différentes approches en terme d'intégration du calcul et de la communication dans un même prototype.

3. Qu'est-ce que le prototypage ?

Facilité de développement et disponibilité anticipée d'un prototype sont deux besoins clés pour accélérer et réussir son plan de développement d'un SoC. Le prototypage repose donc sur une implantation exécutable particulière (le prototype) du système dont on veut valider la fonctionnalité. Cette technique de prototypage matériel retrouve la popularité qu'elle avait autrefois pour accélérer la phase de vérification sous le nom d'émulateur ou d'accélérateur. Son but actuel est d'obtenir rapidement un prototype du système qui soit capable de fonctionner dans des conditions aussi proches que possible de l'implantation finale avant de fabriquer cette dernière sous la forme d'un SoC. Nous présentons maintenant les compromis admissibles, les bénéfices attendus, les diverses classes de prototypage et la dépendance applicative du prototypage.

Afin d'obtenir facilement et rapidement le prototype, il se peut que des compromis soient consentis. Ce sont 1) l'approximation fonctionnelle, 2) le changement de technologie cible et 3) une perte de performances. Hormis le changement de cible technologique (cible FPGA au lieu d'un ASIC), l'approximation fonctionnelle et la perte de performances sont critiques et doivent être réduites au minimum pour que le prototype soit un authentique prototype temps-réel et non pas un simple accélérateur de simulation. L'approximation fonctionnelle est

probablement le point le plus critique car « approximer fonctionnellement » signifie que les données produites par le prototype ne sont pas strictement identique à celles du futur système. Par exemple, un modèle mathématique exprimé en virgule flottante donnera des résultats différents d'un modèles mathématique exprimé en virgule fixe. Alors que le premier est exécutable sur n'importe quelle machine ayant une unité de calcul flottant IEEE 754 [IEEE85] (ou une bibliothèque d'émulation de calcul flottant), le deuxième est plus délicat à obtenir et implique des compromis en terme de bruit de calcul qui impactent la surface du circuit final.

Les bénéfices du prototypage rapide sont nombreux : vérification en environnement réel (ICE ou In-Circuit Emulation), test d'intégration, preuve de concept lors d'une étude de faisabilité et développement anticipé du logiciel. Le terme « prototypage » couvre cependant une large variété d'activités et de moyens. Nous passons en revue les concepts de prototypage virtuel, de prototypage fondé sur l'émulation, de prototypage dédié, de prestations de fonderie rapide, de prototypage analogique, de prototypage semi-dédié et finalement de prototypage et de conception orientés plate-forme. Finalement, nous nous recentrons sur les prototypages semi-dédié et orienté plate-forme car ils nous ont fortement influencé dans le cours de notre étude.

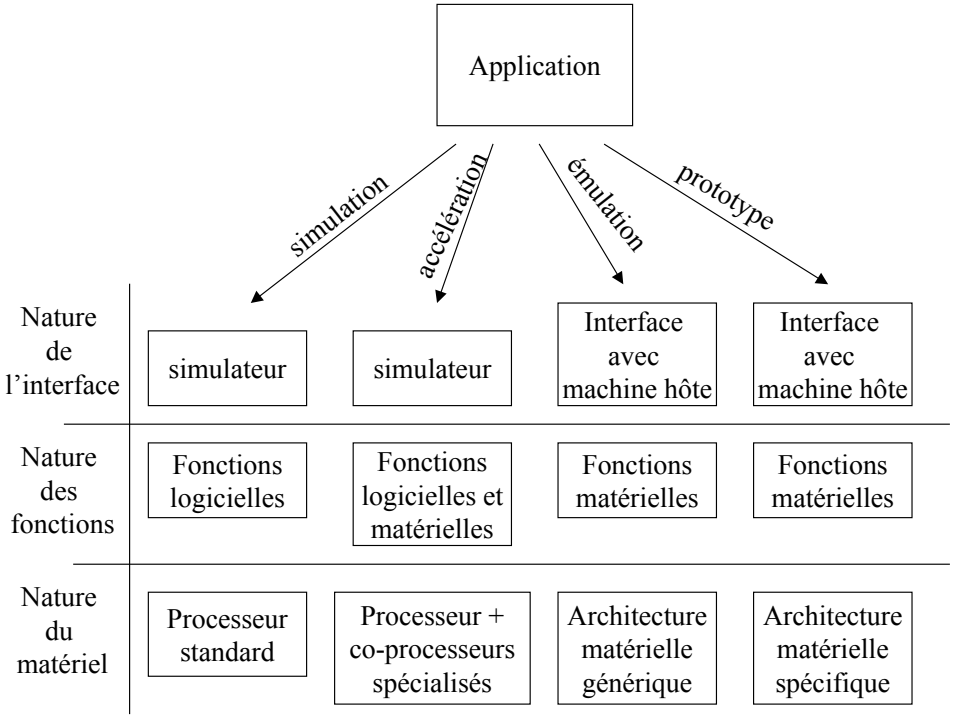


Figure 3-3 : Classification des méthodologies de prototypage par nature du matériel

La Figure 3-3 illustre tout d'abord une classification des méthodologies de prototypage qui repose principalement sur les techniques mises en œuvre et la rapidité d'exécution.

On distingue alors la simulation de l'accélération, l'émulation ou la conception du prototype.

- La simulation consiste à exécuter un modèle du système exprimé dans un langage de programmation quelconque sur un ou plusieurs ordinateurs standards.

- L'accélération est identique à la simulation, si ce n'est que des accélérateurs (co-processeurs) sont adjoints aux processeurs. La compilation du modèle fait alors intervenir des appels de procédures à des fonctions en bibliothèques ou de la génération de code en ligne.

- L'émulation est radicalement différente des précédentes dans la mesure ou il n'y a plus de simulateur logiciel. Le système est intégralement compilé pour une architecture générique offrant une puissance de calcul et de communication accrue.

- La conception du prototype consiste à assembler un calculateur spécifique et à compiler le système pour cette architecture particulière.

On constate que, de la simulation au prototype, l'apport du logiciel décroît et la spécificité du matériel croît. Les fonctionnalités de mise au point décroissent de même et le prototype, à cause de sa spécificité, est le moins « observable » de tous. Néanmoins actuellement, et à cause de la complexité croissante des systèmes conçus, la classification s'est enrichie et repose non plus seulement sur la rapidité mais sur les fonctionnalités offertes et la flexibilité de la plate-forme de prototypage [MED02]. Nous présentons en détail cette classification.

3.1. Méthodologies de prototypage

Le *prototypage virtuel* permet de générer rapidement un prototype mixte (matériel et logiciel), de synthétiser les interfaces, et de réutiliser des composants préconçus. Ce modèle mixte est exécutable (simulable) sur un ordinateur standard (PC, station de travail). Il est fonctionnellement correct et sert généralement de référence pour les raffinements matériels et logiciels ultérieurs. Ce modèle n'a pas pour but d'être précis au niveau cycle, ni au niveau portes. Ce sont les outils de co-conception/verification au niveau système (SLD ou System Level design) Cierto Virtual Component Co-design (VCC) [CAD04], N2C [COW04], CoCentric System Studio [SYN04], Seamless [MEN04], et SystemLab [CPU04]. A titre d'exemple, Cierto VCC de Cadence est un outil de co-conception qui permet de concevoir des bibliothèques d'IP, de spécifier un système et d'évaluer différentes alternatives d'implantation ainsi que l'impact en performances des communications à l'aide de la co-simulation. N2C de CoWare (une des sociétés fondatrices de SystemC), spécialisé dans la conception rapide de dérivés de circuits à base de cœurs de processeurs, permet de synthétiser automatiquement les interfaces matériel/matériel (avec des modèles de bus standards tels que AMBA ou CoreConnect) et matériel/logiciel (allocation des adresses des registres dans l'espace adressable, pilotes et routines d'interruptions). CoCentric System Studio propose une approche duale algorithme/architecture fondée sur le langages C et ses dérivés C++ et SystemC. Il possède une bibliothèque de près de deux mille modèles d'algorithmes. Seamless de Mentor Graphics est un environnement de co-verification doté d'une bibliothèque de plus de cent modèles (ISS ou Instruction Set Simulator) de processeurs actuels. Seamless encapsule les ISS et les simulateurs

logiques dans un environnement logiciel leur permettant de communiquer et de se synchroniser entre eux. Contrairement à N2C et CoCentric, System Studio n'est pas un outil de spécification/raffinement. SystemLab de CPU Tech, est un ensemble d'outils et de prestations de services dont le but est de développer des prototypes. Contrairement aux grandes sociétés de CAO, CPU Tech se positionne comme un prestataire de services ayant de l'expérience, pas comme un fournisseur de logiciels.

Le *prototypage fondé sur l'émulation* est une implantation matérielle d'un modèle système obtenue grâce à une phase de compilation vers une architecture spécifique autre qu'un ordinateur standard. Il offre des fonctions identiques à celles des simulateurs logiques telle l'observabilité interne des signaux et les points d'arrêt. Ces ordinateurs spécialisés se répartissent en deux catégories : les accélérateurs et les émulateurs. Les accélérateurs matériels sont fondés sur des architectures à base de processeurs standards que l'on a spécialisés par adjonction de matériel plus ou moins programmable, et les émulateurs sont des collections de FPGA qui communiquent entre eux au travers d'une architecture de communication propriétaire. Mercury, Cobalt, Radium, Palladium [CAD04], Celaro [MEN04], Vstation, Vsim [IKO04], Extreme [AXI04], Rave [SIM04] et System Explorer [APT04] se partagent le marché.

Le *prototypage dédié* repose sur des cartes spécifiques contenant des FPGA, des composants du commerce (COTS, commercial off the shelf) et un réseau de communication dédié. Des dizaines de sociétés proposent ce type de cartes dont l'attrait est dans les composants COTS qu'elles permettent de mettre en œuvre avec plus ou moins de flexibilité. ARDOISE [DEM99] est un exemple de prototype dédié à la reconfiguration dynamique qui inclut des FPGA d'Atmel [ATM04] et optionnellement un DSP. Chez Xtensa [XTE04] un réseau de processeurs est utilisé pour prototyper la parallélisation de turbo-décodeurs [GLI03].

Certaines fonderies de silicium proposent des *prestations rapides* à partir d'une netlist du système complet. Un prototype silicium peut être obtenu en 5 jours chez ChipExpress [CHP04] ainsi qu'une production de volume en 3 semaines avec leur technologie CX5000 de 0,18 µm de finesse.

Le *prototypage analogique* est récent et concerne des composants et des outils de CAO spécifiques. Les candidats actuels sont « in-system programmable » ispPACxx [LAT04], FPAA (Field Programmable Analogue Arrays) [ANA04], et EPAD (Electrical Programmable Analogue device) [ALD04].

Le *prototypage semi-dédié* est composé d'une à plusieurs cartes mères qui distribuent des interconnexions plus ou moins flexibles à des cartes filles au travers de câbles ou d'éléments de commutation dynamique de circuits. Il y a deux classes de prototypages semi-dédiés selon la stratégie de communication retenue : les centrées « bus » et les centrées « réseau ». Les sociétés Nallatech [NAL04], Sundance [SUN04], Transtech [TRN04], et Traquair [TRQ04] proposent des catalogues de produits contenant des cartes mères et des cartes filles contenant des FPGA de Xilinx [XIL04] et Altera [ALT04] et des processeurs DSP de Texas Instruments [TEX04]. BEE (Berkeley Emulation Engine) [KUU03] est une architecture monolithique centrée « réseau » (mono carte mère) sur laquelle 20 FPGA communiquent entre eux au travers d'un réseau de communication routé au travers de 4 FPGA supplémentaires pour supprimer tous les câbles.

La *conception et le prototypage orientés plate-forme* (platform based design) reposent sur le concept d'implantation d'une application sur une architecture mono-puce suffisamment flexible et générale pour pouvoir supporter une extension à coût quasiment nul vers d'autres applications ainsi que leurs futures évolutions. La section suivante décrit ce type de prototypage en détail.

Enfin, il faut aussi considérer que le prototypage rapide est, tout comme la conception d'un système, la combinaison d'éléments de calcul et de communication dont il est très difficile d'exploiter toute la performance pour toutes les applications. Ainsi, des domaines d'application spécifiques requièrent-ils des plates-formes de prototypage spécifiques. Notre étude des contraintes des applications de radiocommunications numériques puis des méthodologies de prototypage nous conduit à explorer plus en détail la conception et le prototypage orientés plate-forme afin de proposer une architecture de plate-forme qui dispose de la puissance de calcul et de communication ainsi que de la flexibilité dont nous avons besoin.

3.2. Prototypage/conception orienté plate-forme

Le terme prototypage/conception orienté plate-forme, idée récente, est l'application aux SoC d'un concept utilisé depuis les tubes à vide [SMI04] : il s'agit du « modèle de référence » (Reference Design) ou de la simple « note d'application » (Application Note) dont le but est de prouver la faisabilité d'un système nouveau par la mise en oeuvre de produits commerciaux disponibles (COTS). Dans les années 70 les micro-électroniciens se sont mis à exploiter ces modèles de référence pour les premières versions de leurs produits dits « embarqués », puis les ont spécialisés en fonction de l'évolution de leurs besoins. C'est à cette période que le terme de plate-forme apparaît pour désigner cette activité de réutilisation au niveau cartes. Les micro-processeurs intervenant de plus en plus dans les systèmes, la spécialisation du système peut se faire par programmation. Vient ensuite dans les années 80, la génération des « dérivés » de micro-processeurs : des processeurs aux jeux d'instructions extensibles et/ou aux périphériques et/ou coprocesseurs supplémentaires. Puis, dans les années 90, apparaissent les solutions à base de logique reprogrammable (FPGA). La spécialisation des modèles de références se fait alors de plus en plus par la programmation logicielle ou matérielle, évitant ainsi le recours systématique à un nouveau jeu de masques pour chaque nouveau système. L'intensification de la pratique de la réutilisation devient alors une méthodologie de conception.

Ainsi, RASSP (Rapid Prototyping of Application-Specific Signal Processors) [RAS96] introduit-il la notion de MYA (Model Year Architecture). MYA est une approche de conception reposant sur le renouvellement réguliers de composants matériels et logiciels conçus pour être réutilisés et mis à la disposition des intégrateurs pour la maintenance et le développement de systèmes. La conception orientée plate-forme pour SoCs est une mise en œuvre intensive du concept de réutilisation tel qu'identifié par l'ITRS. [KEU00] définit une plate-forme comme la réunion de trois parties essentielles qui sont : la plate-forme matérielle, la plate-forme logicielle et la plate-forme système.

La plate-forme matérielle est constituée d'une famille de micro-architectures telles que des cœurs de processeurs, des sous-systèmes d'entrées-sorties, des mémoires et de la logique programmable. La plate-forme matérielle est abstraite à un niveau auquel le logiciel ne voit plus qu'un matériel de « haut niveau » au travers d'une, ou plusieurs, APIs (Application Program

Interface). Ces APIs sont constituées de langages de programmation qui masquent les jeux d'instructions (ISA, Instruction Set Architecture) des cœurs de processeurs, d'un système d'exploitation (RTOS, Real Time Operating Système) qui implante la concurrence et la communication inter taches au niveau logiciel et assure la gestion de l'espace mémoire, de drivers qui masquent les périphériques, et de sous-systèmes de communication (piles de protocoles) qui masquent les interfaces réseau. Ces APIs représentent la plate-forme logicielle. La réunion des plate-formes matérielle et logicielle est la plate-forme système.

Dans ce contexte, de très nombreux travaux académiques et produits commerciaux se concentrent sur l'exploitation de machines parallèles multi-DSP pour l'implantation d'algorithmes, beaucoup moins sur des environnements mixtes matériel/logiciel. Plus récemment [YOO03] introduit le concept de HAL (Hardware Abstraction Layer) appliqué aux SoCs. Par définition, une HAL réunit tout le logiciel qui est directement dépendant du matériel sous-jacent. Il est similaire aux concepts de micro-noyau (nano-kernel), de pilotes de périphériques (device drivers), d'interface entre systèmes d'exploitation et pilotes (spécification DDI-DKI de SunOs/Solaris, HAL de VxWorks, HAL du network processor d'Intel, …). Cependant une HAL SoC couvre plus que l'aspect « interface » avec l'extérieur, elle contient aussi les services de démarrage (boot), de commutation de contexte (context switch), de configuration des registres des mémoires virtuelles et des caches, bref l'initialisation et le pilotage fin des contextes matériels des processeurs et de leurs espaces d'adressage. Le concept de HAL favorise le ciblage de solutions « tout processeur ». Il est néanmoins reconnu que de telles solutions coûtent de l'ordre de 10 à 50 fois plus en terme de surface et consommation [VIS03] que des solutions câblées de type composants synthétisables sous contraintes ou préconçus en vue d'une réutilisation.

Dernièrement [SAN04] raffine le concept de plate-forme en introduisant la notion de pile de plate-forme système (System Platform Stack) et décrit la méthodologie de conception orientée plate-forme comme un processus de « rencontre au milieu » (meet-in-the middle process). Dans ce processus de conception, les raffinements successifs des spécifications systèmes (approche top-down) rencontrent le niveau d'abstraction le plus élevé des implantations potentielles (approche bottom-up) comme l'illustre la Figure 3-4. Cette approche découple les deux activités de conception orientée plate-forme : le raffinement du modèle système et la conception de la/les micro-architecture(s) et APIs qui constituent la plate-forme système. Il fait, de plus, la distinction entre deux types de plates-formes : la plate-forme réseau (Network Platform) et la plate-forme analogique (Analog Platform). Ces deux dernières plates-formes systèmes diffèrent par leurs objectifs. Alors que la plate-forme réseau a pour but de modéliser le comportement fonctionnel de systèmes distribués dans lesquels la communication domine (cas des systèmes insensibles à la latence et des NoC), la plate-forme dite « analogique » est une plate-forme dans laquelle on dispose de modèles de performances et de modèles comportementaux (modélisant aussi les perturbations temporelles dues aux diverses implantations) pour tous les éléments de la plate-forme : nœuds de calcul et liens de communication.

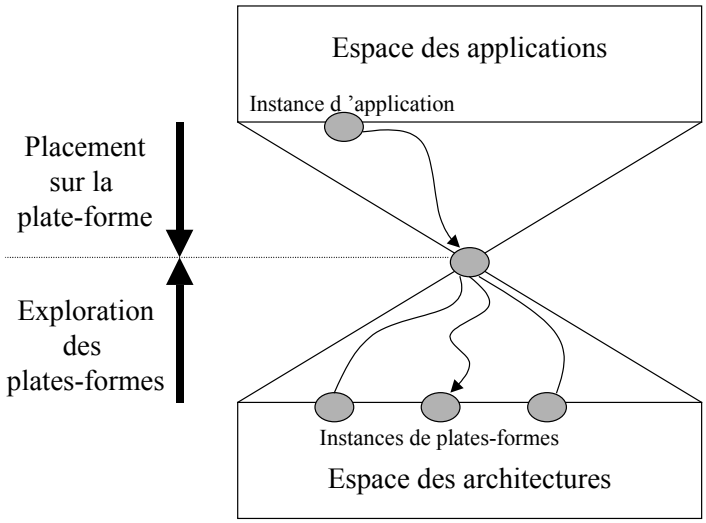


Figure 3-4 : Processus de conception de type « meet-in-the middle »

Bien que relativement peu d'expériences industrielles prouvent actuellement la viabilité du concept, (surtout en terme de performances du prototype obtenu [MDA03]), le marché des composants paramétrables se développe indéniablement. Sans prétendre être exhaustif, nous citons le C166S [INF04], le Flip8051 [DOL04], le PowerPc [IBM04] et un catalogue d'IP chez [D&R04]. Enfin, prototypage semi-dédié et orienté plate-forme sont très proches. En effet :

- Le prototypage semi-dédié repose sur des cartes mères et filles ainsi que sur une architecture de réseau de communication.

- Le prototypage orienté plate-forme est constitué par des puces sur lesquelles se trouvent plusieurs composants plus ou moins spécifiques et dont la configuration de la topologie du réseau de communication et la programmation permettent d'implanter un système particulier.

Nature	semi-dédié	orienté plate-forme
Composant de base	Circuit intégré	Bloc fonctionnel
Système	Assemblage de cartes	Puce
Communication	Câbles/bus/matrice de commutation	Matrice de commutation/bus

Tableau 3-1 : Similitudes entre protypage semi-dédié et orienté plate-forme

A un facteur d'échelle près, le paradigme est le même : disposer de composants que l'on peut programmer, assembler et faire communiquer au travers d'un réseau de communication flexible. Dans les deux cas, on retrouve les types d'architectures de réseaux de communication traditionnels : les bus et les liaisons point à point. Enfin, les liaisons point à

point peuvent être soit statiques (câbles, routage sur FPGA, …), soit dynamiques (matrice de commutation).

Nous retenons donc que le concept de conception/prototypage orienté plate-forme est l'élément méthodologique majeur qui nous permet de dissocier la conception de l'application au niveau système de la conception de l'architecture matérielle sur laquelle le prototype va s'exécuter.

Nous passons maintenant en revue les plates-formes de prototypage existantes afin de les caractériser et d'en connaître les qualités et les manques. Nous proposons et comparons ensuite notre plate-forme de prototypage rapide.

3.3. Plates-formes de protypage rapide

Il existe de très nombreuses plates-formes de prototypage. Dès les années 90 [GUC95] en recense plus de cent vingt à base de logique programmable et [MAR98][MAR99] présente les principes fondateurs de l'approche fondée sur la réutilisation intensive d'IP. Il insiste sur la programmabilité *logicielle ou matérielle* des composants qui est indispensable à l'adaptation rapide d'un composant à tout nouveau contexte. Sans prétendre être exhaustif, nous présentons celles que nous jugeons les plus significatives vis-à-vis de notre domaine d'application : le traitement du signal et de l'image. Le Tableau 3-2 cite les plates-formes les plus actuelles.

A la lecture de ce tableau, une première remarque s'impose : seuls les grands industriels des semi-conducteurs sont capables de proposer des plates-formes sur silicium. A cela, deux raisons principales : 1) c'est un excellent argument de vente d'un produit générique et 2) c'est leur avantage « métier » (ils sont les seuls à maîtriser la complexité de la chaîne de conception qui va de la spécification du système au silicium).

Les chercheurs, aux moyens financiers et humains en général moindres et aux objectifs différents, se concentrent sur la réutilisation de composants du commerce et donc sur le protypage semi-dédié.

Nous passons maintenant en revue ces plates-formes et décrivons leurs qualités. La description de chacune d'entre elles suit le même plan : 1) champ d'application, 2) architecture, 3) outillage et 4) exploitation.

Nom	Origine	Type / application	Calcul	Comm.	**Outils de conception**
BEE	Univ. Berkeley, USA	Semi-dédié / radiocoms	20 Virtex 2000E	Réseau routé et matrice de commutation	Simulink
Bournemouth	Univ. Bournemouth, UK	Semi-dédié / radiocoms	1 C62 et 1 Virtex 1000E	PCI	C, VxWorks, Handel-C
ECLIPSE	Univ. ESIEE, A^2SI, France	Semi-dédié / image	C4x, FPGA 4000	Com port C4x et PCI	SynDEx
NTT	Industriel (propriétaire) NTT, Japon	Semi-dédié / soft radio	4 C62 1 Power-PC	BUS VME	VxWorks
OMAP	Industriel (commercial) Texas Instruments, USA	Silicium / image et radiocoms	1 ARM11 1 C55x accélérateurs	Matrice de commutation	Nombreux RTOS et BSP
PLATON	Industriel (propriétaire) Eurecom, France	Semi-dédié / radiocoms	1 PC	Réseau Ethernet	Linux piles de protocoles
RICE	Univ. RICE, Texas, USA	Semi-dédié / radiocoms	4 Virtex2	FIFO pré-routée	Simulink
Nexperia, Viper	Industriel (propriétaire) Philips, Pays-Bas	Silicium / image	1 Trimedia 1 MIPS accélérateurs	Bus hiérarchique	Les RTOS pSOS et VxWorks
MATRICE	Projet européen www.ist_matric.org	MC-CDMA	C6x Virtex	Type « transputer »	DSP-BIOS

Tableau 3-2 : Liste des plates-formes actuelles

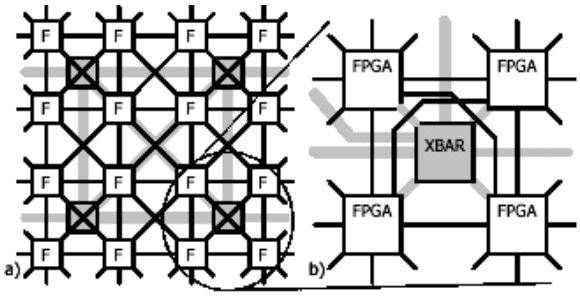


Figure 3-5 : Structure de la plate-forme BEE

- BEE (Berkeley Emulation Engine) est destinée au prototypage de systèmes de communication sans fil. La plate-forme est constituée de cartes sur lesquelles se trouve une grappe de vingt FPGA VirtexE 2000 dont seize sont destinés au calcul et quatre à la communication. Un réseau de communication est routé sur la carte et permet aux seize FPGA de calcul de communiquer directement avec leurs huit voisins et les quatre FPGA de la matrice de commutation. Une machine BEE est constituée de une à plusieurs grappes reliées entre-elles par des bus SCSI. La communication inter-FPGA peut atteindre 100 Méchantillons/s en direct et 70 Méchantillons/s via la matrice de commutation. Ces grappes sont accompagnées de convertisseurs analogiques-numériques. La conception système repose sur Simulink et le partitionnement du système sur les nombreux FPGA est manuel. Cette plate-forme est une des plus aboutie, il ne lui manque que l'hétérogénéité (des processeurs) pour prototyper efficacement des systèmes mixtes hardware/software. Une version BEE2 est en cours de développement : elle intègrera des Virtex2-Pro qui ont un cœur de Power-PC embarqué. BEE semble donc être déjà obsolète.

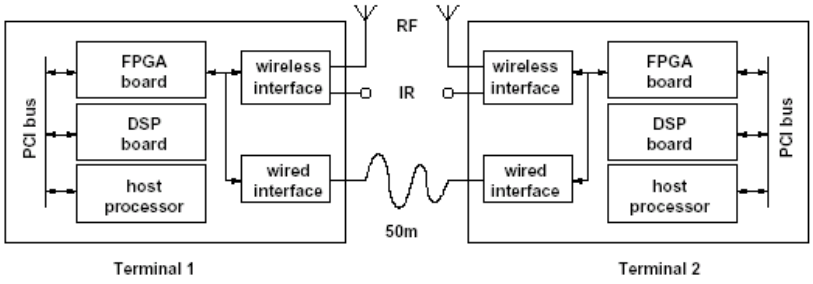


Figure 3-6 : Structure de la plate-forme de Bournemouth

- La plate-forme de Bournemouth [VAS01] est aussi dédiée aux communications sans fil. Elle est constituée par un émetteur et un récepteur ayant la même structure. Cette structure est composée d'un PC dans lequel sont intégrés (via le bus PCI) deux cartes : une pour un DSP C62, et une pour un FPGA Virtex 1000. Sa principale qualité nous semble être la possibilité de programmer le FPGA à l'aide du langage Handel-C : des non électroniciens peuvent ainsi avoir accès à la puissance de calcul du FPGA. Son principal défaut est sa non extensibilité en nombre de nœuds de calcul et sa limitation de la communication entre les nœuds de calcul par le bus PCI. Cette plate-forme ne semble pas être exploitée en dehors du laboratoire l'ayant conçue.

- La plate-forme ECLIPSE (projet RNTL pré-compétitif [NAK02]), est dédiée au traitement de l'image. Bien qu'intéressante car elle exploite les outils Scicos et SynDEx (INRIA) pour générer des exécutifs statiques (sans RTOS), elle est en retard sur les avancées technologiques des DSP et des FPGA. En effet ses composants sont des C4x et des FPGA 4000 et ses liens de communication (Comm Ports des C4x) sont limités à 20 Mo/s. Elle mériterait une mise a jour de ses composants, et probablement le développement des générateurs de codes pour ces nouvelles cibles à partir des macro-codes générés par SynDEx. La présentation en 2002 des résultats du projet affirme que l'environnement logiciel est libre et utilisable industriellement et que la plate-forme est un excellent cahier des charges pour une *future plate-forme française ouverte et multi-métiers de conception de systèmes temps réels*.

- La plate-forme NTT [SHI03] est destinée à l'expérimentation de la reconfiguration dynamique dans le cadre de la radio logicielle. Il s'agit de tester la reconfigurabilité « over-the-air » (OTA). Les temps de reconfiguration sont de l'ordre de 11s. L'architecture est constituée de plusieurs nœuds de calcul assemblés autour d'un bus VME64. Les nœuds de calcul sont un Power-PC sous VxWorks, et une grappe de quatre C62 reliés à une paire de FPGA (modèle inconnu) qui supportent la reconfiguration dynamique. La topologie de type bus entre les nœuds de calcul, et de type point à point entre les C62 et les FPGA limitent l'extensibilité et la réutilisabilité de l'architecture pour d'autres applications. Cette plate-forme, n'est pas un bon candidat pour les traitements intensifs requis par notre domaine. Cette plate-forme ne nous semble pas avoir d'autre rôle que d'être un démonstrateur lors d'une étude de faisabilité. Son exploitation par des tierces parties n'est d'ailleurs pas prévue puisqu'il ne s'agit pas d'un produit commercial.

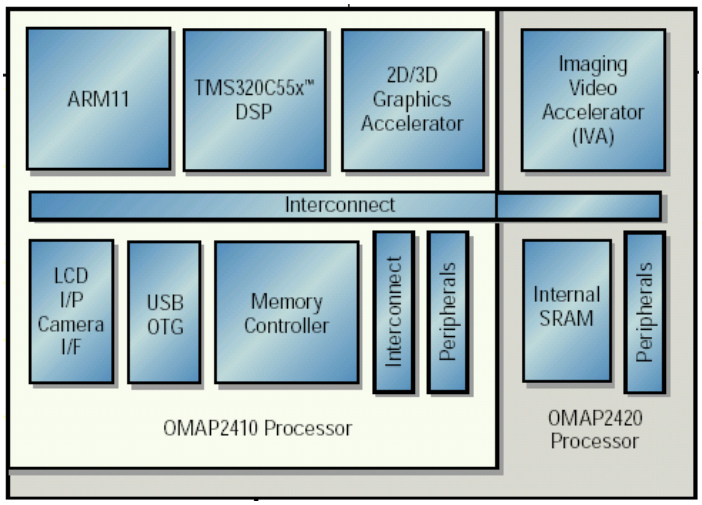


Figure 3-7 : Structure de la plate-forme OMAP

- La plate-forme OMAP (All-in-One Entertainement Architecture) de Texas Instruments est destinée aux applications de traitement d'image en environnement de téléphonie mobile avancée (c'est-à-dire avec intégration de la vidéo). La plate-forme est proposée en deux versions qui diffèrent par le nombre et le type d'accélérateurs graphiques. Elle est dotée de deux processeurs logiciels : un ARM11 pour le contrôle et un DSP C55 pour les traitements. Une matrice de commutation reconfigurable permet de mettre en place la topologie du

réseau de communication nécessaire au système. Cette plate-forme est adaptable par programmation des cœurs et des accélérateurs graphiques et par reconfiguration du réseau de communication. Enfin, elle ne contient aucune logique programmable qui permettrait d'intégrer en matériel un nouvel accélérateur. Cette plate-forme est une sorte de super micro-controleur 32 bits dont les utilisateurs utilisent à leur gré tout ou partie des fonctionnalités présentes. Cette plate-forme est exploitée commercialement : c'est un produit au catalogue de Texas Instruments dont on peut obtenir des exemplaires.

- Platon [PLA04] est une plate-forme ouverte pour les nouvelles générations de communications mobiles. Elle offre un environnement d'expérimentation d'applications, de services et d'éléments technologiques reposant sur une interface radio de type UMTS. Elle cible les communications mobiles dans la bande 1900-1920 MHz en temps réel. La puissance de l'émission peut atteindre 1 Watt. La plate-forme actuelle comprend essentiellement : une antenne d'émission/réception, une carte radio fréquence d'émission/réception dans la bande 1900-1920 MHz, une carte d'acquisition des échantillons numériques et un PC pentium avec un OS Linux/RTLinux. Cette plate-forme repose sur des nœuds de calcul de type PC : c'est à dire des systèmes très répandus et bien outillés pour le développement. L'extensibilité du nombre de nœuds de calcul de la plate-forme via les protocoles réseaux standards (TCP/IP) et l'usage de réseaux non déterministes (Ethernet) nous fait douter de son aptitude à offrir des performances suffisantes de communication en configuration multi-nœuds. Le projet RNRT PLATON a débuté en mars 2000 et il associe les équipes d'Eurecom, de Philips Semiconductors, de France Télécom R&D et 6Wind. Eurecom, chef de file du projet, exploite les résultats du projet.

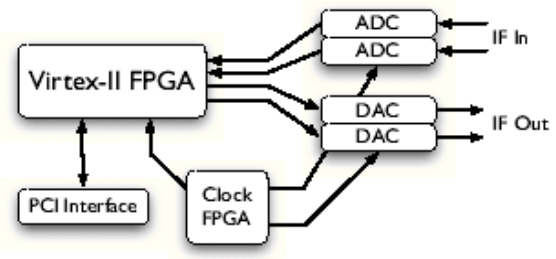


Figure 3-8 : Structure de la plate-forme RICE

- La plate-forme de l'Université de RICE [MUR03], est destinée à l'expérimentation de systèmes MIMO (modélisés avec Simulink) afin de comparer leurs performances théoriques avec celles de prototypes réels. Elle repose sur des FPGA de la gamme Virtex2 connecté à des convertisseurs analogiques-numériques. Grâce à un bus dédié de type FIFO (architecture DIME de la société Nallatech), les communications entre FPGA peuvent être de l'ordre de 10 Mo/s. Cette plate-forme, fortement orientée traitement flot de données, n'est pas infiniment extensible en nombre de FPGA (4 maximum sur une carte mère) et ne permet pas de bâtir aisément des topologies « haut-débit » autres que le strict pipeline. Son exploitation semble être réduite au CMC [CMC04] (Center for Multimedia Communications).

- La plate-forme Viper [DUT03] de Philips est destinée aux applications de type set-top-boxes (décompression/décodage/décryptage de divers formats numériques audio et vidéo) et de télévision numérique. Son architecture repose sur une solution bi-processeurs (Trimédia pour les traitements de données intensifs + MIPS pour le contrôle) accompagnée d'accélérateurs matériels. La communication interne est fondée sur une architecture de bus hiérarchique à deux niveaux : un bus haut débit 64 bits pour l'accès direct à la mémoire et un bus moyen débit 32 bits (PI bus) segmenté permettant le parallélisme des transferts sur chacun des segments. Les produits issus de cette plate-forme sont les dérivés PNX-85xx. son exploitation est claire : Philips met actuellement sur le marché sa prestation Nexperia de plate-forme système extensible (hardware PNX-8550 et software associé) pour television numérique.

- La plate-forme IST-MATRICE, issue du projet européen de même nom, réunit des partenaires académiques (LETI Grenoble, Institut des Télécommunications de Polo de Aveiro/Portugal, Université Polytechnique de Madrid/Espagne, Université de Surrey/Royaume-Uni, INSA Rennes) et des industriels (France telecom R&D, Mitsubishi, STMicroelectronics, et Nokia). Elle est fondée sur la mise en oeuvre de cartes de la société Hunt qui contiennent des processeurs DSP C6x et des FPGA de Xilinx. Actuellement seuls des nœuds de calcul de type FPFA sont utilisés par le démonstrateur. La communication est de type « transputer » : l'architecture de communication est fondée sur des grappes de quatre noeuds de calcul ayant quatre liens de communication chacun. Chaque nœud est relié aux trois autres par l'un de ses liens et à une autre grappe par son quatrième. MATRICE est destinée à la définition et à la validation des concepts fondés sur la technologie MC-CDMA nécessaires aux composants des futurs réseaux de téléphonie cellulaire.

3.4. Intérêt technico-économique du prototypage rapide

Remarquons que, contrairement à la conception orientée plate-forme de SoC, notre objectif n'est pas de produire en grande série une implantation monopuce d'un système mais d'obtenir rapidement un prototype fonctionnant à une fréquence comparable à celle du système final. La principale conséquence est que nous n'avons pas l'obligation de réduire au minimum le matériel nécessaire à la fabrication du prototype. Nous pouvons nous permettre de sur-allouer les ressources matérielles de façon à simplifier le placement du système sur l'architecture matérielle et accélérer ainsi la production du prototype au détriment de son coût unitaire. La notion de coût n'est pas la même entre un SoC et un prototype, précisons la.

Tout d'abord, le coût unitaire de production d'un SoC est extrêmement important car il sera fabriqué en grande série. Le coût unitaire d'un prototype, par contre, est amorti par tous les prototypes successifs qui seront fabriqués par réutilisation des mêmes éléments. Ensuite, le coût de production d'un SoC intervient dans les calculs de prix de vente et donc de marge, alors que le prix du prototype intervient dans le NRE (Non Recurrent Engineering). La Figure 3-9 illustre (à l'aide de la courbe idéalisée en tirés qui représente l'évolution du bilan financier d'un projet) que la marge de vente des puces amortit graduellement le coût de conception, puis génère des profits (ROI ou Return On Investment [MAR04]) au delà du point de profitabilité (break even).

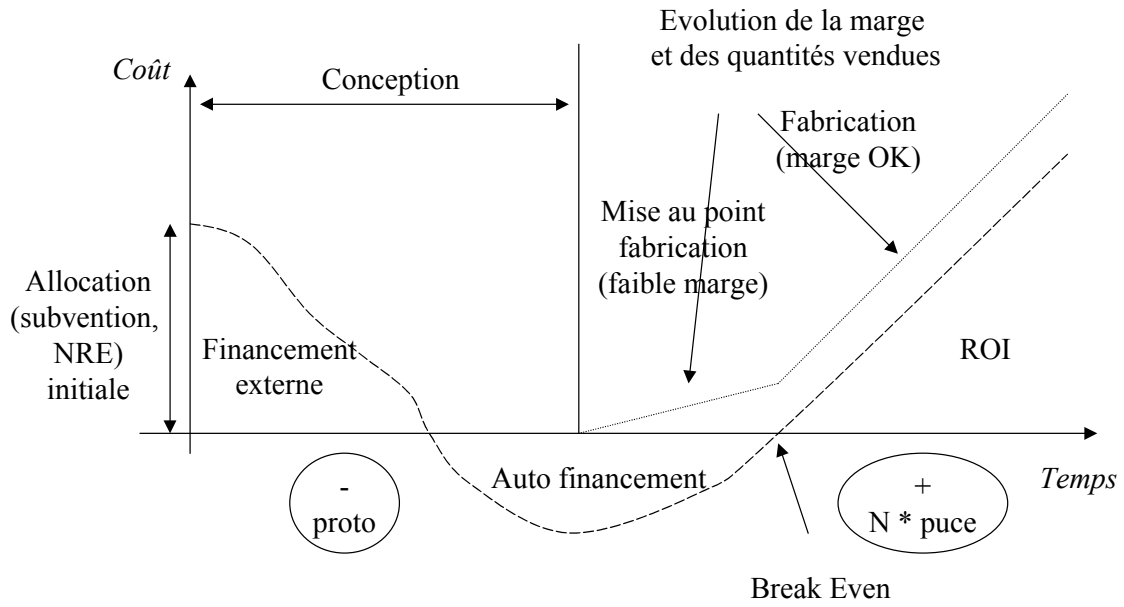


Figure 3-9 : Evolution typique du coût d'un projet

La classification des projets compétiteurs pour obtenir des ressources au sein d'une même entreprise repose sur des modèles économiques nommés « figures de mérite » dans lesquels interviennent les estimations planifiées des coûts et des recettes pour tout nouveau produit. Globalement, les figures de mérite expriment toujours qu'un bon projet 1) a un temps de conception court, 2) une part d'autofinancement faible et 3) un ROI important. *Dans ce cadre, le partage du coût d'une plate-forme de prototypage par plusieurs projets leur permet de réduire leurs frais de conception*.

L'intérêt technico-économique pour le prototypage étant posé, le LESTER a proposé d'intégrer l'outil de synthèse de haut niveau GAUT dans la plate-forme PALMYRE pour 1) accélérer le raffinement matériel des composants critiques et réduire le temps de conception du prototype et 2) vérifier par prototypage la validité de notre approche LIS. Nous disposons ainsi d'une plate-forme opérationnelle qui permet de passer d'un modèle système à un prototype de façon semi-automatique.

Les paragraphes suivants présentent successivement 1) la méthodologie de conception/prototypage d'IP virtuels au niveau algorithmique proposée par le LESTER dans le cadre de laquelle la plate-forme a été conçue, 2) la plate-forme matérielle, 3) la plate-forme logicielle et 4) notre approche de la spécification d'un modèle système exécutable ainsi que du prototypage rapide sur cette plate-forme. Les améliorations apportées à l'outil GAUT seront présentées dans le chapitre suivant.

4. Plate-forme de prototypage rapide PALMYRE

Notre activité de recherche s'est déroulée dans le cadre du projet PALMYRE. Son but est de concevoir une plate-forme de prototypage rapide destinée à l'expérimentation sur site de nouveaux algorithmes numériques pour applications en radiocommunications. Nous avons pu mettre à profit les moyens de ce projet pour valider nos travaux dans un contexte matériel réel (autrement que par simulation) et avec de véritables algorithmes de traitement du signal.

Le projet PALMYRE [PAL04] est un projet de la région Bretagne, inscrit au contrat de plan Etat/Région (CPER) visant la conception d'une plate-forme de développement et d'évaluation de systèmes radioélectriques. Les partenaires du projet sont le laboratoire TAMCIC de l'Ecole Nationale Supérieure des Télécommunications de Bretagne (ENSTB) [ENS04], l'Institut d'Electronique et de Télécommunications de Rennes (IETR) [IET04] qui regroupe des chercheurs de l'Institut National des Sciences Appliquées de Rennes (INSA) [INS04b], l'Université de Rennes 1 et SUPELEC [URE04] et le Laboratoire Electronique des Systèmes Temps Réels (LESTER) [LES04] de l'Université de Bretagne Sud (UBS) [UBS04]. Dans le cadre de ce projet, une des tâches du LESTER est de concevoir la plate-forme système numérique (plates-formes logicielle et matérielle) et d'en assurer la mise à disposition ainsi que le support auprès des partenaires.

4.1. Statégie de conception d'IP du LESTER

Dans une approche stratégique globale, nous distinguons les rôles des concepteurs de composants virtuels de celui des intégrateurs. De même l'orthogonalisation de la spécification algorithmique (quelle fonction de traitement du signal) et de l'intégration système (avec qui et comment communique le composant) est un élément clé de nos projets.

Le processus de synthèse classique, où la conception d'un circuit démarre en fait au niveau structurel par une synthèse dite RTL, ne permet plus actuellement de répondre efficacement aux multiples critères à satisfaire (performances, fiabilité, *time to market*, …) en raison de la complexité des applications à intégrer. L'accélération du processus de conception consiste à élever le niveau d'abstraction choisi comme point de départ de la conception et développer en parallèle des outils de conception correspondants. En travaillant au niveau architectural : le comportement est décrit sous une forme très proche du niveau algorithmique et des processus (semi-)automatisés conduisent à l'obtention, sous contrainte de coût (temps, surface, consommation, etc…), d'une description structurelle du circuit basée sur un modèle d'architecture défini a priori. On parle alors de synthèse d'architectures ou synthèse de haut niveau. Le manque de maturité des outils commerciaux actuels de synthèse de haut niveau nous a conduit à développer un cadre formel pour spécifier et utiliser des composants virtuels de ce niveau d'abstraction. Dans ce cadre, nous développons notre propre méthodologie et notre propre outil de synthèse de haut niveau (GAUT). Cet outil, dont le cœur est la synthèse de l'unité de traitement sous contrainte temps réel, présente la particularité par rapport aux outils commerciaux, de réaliser également la synthèse de l'unité de mémorisation et de l'unité de communication. En plus de travailler sur l'aspect méthodologique, ce projet a pour vocation le transfert de nos compétences et notre méthodologie de conception haut niveau à nos partenaires.

Ces derniers sont, en effet, fortement demandeurs d'assistance dans le domaine de la conception de SoCs à l'aide de composants virtuels réutilisables. Les applications ciblées sont des chaînes de réception obéissant à la norme de diffusion par satellite DVB-S pour des services de type DSNG (Digital Satellite News Gathering). La définition d'une plate-forme système, en vue de l'intégration des IPs développés dans un environnement système, en relation avec les outils développés est en cours de réalisation.

Une collaboration avec l'équipe MCSE de Polytech'Nantes dans le cadre d'une thèse "Integration des contraintes d'interface dans la conception plate-forme : application à la radio-communication" vise à utiliser le support logiciel *CoFluent Studio* [COF04] de la méthodologie de co-conception de systèmes électroniques MCSE pour extraire des contraintes de communication entre blocs fonctionnels et/ou matériels en tenant compte, selon le niveau de description, du support de communication physique. A partir de ces informations, il sera possible d'affiner le comportement aux entrées/sorties d'une description comportementale, en vue de sa synthèse. La prise en compte aussi tôt que possible dans le flot de conception de cette contrainte de communication évite de limiter les solutions architecturales envisageables ainsi que leurs performances.

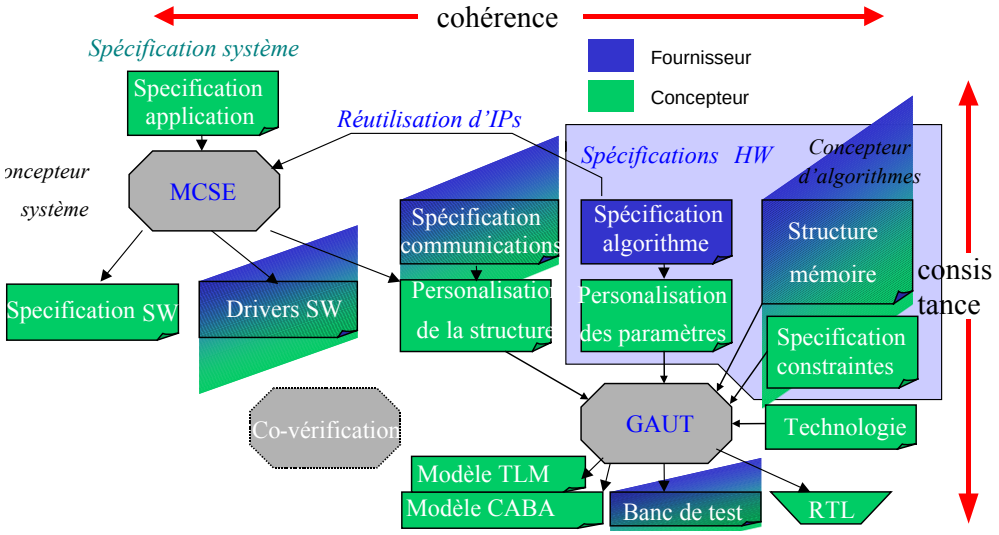


Figure 3-10 : Stratégie globale de conception et d'intégration d'IP virtuels algorithmiques

La Figure 3-10, extraite du rapport d'activité annuel du LESTER 2004, illustre les rapports entre MCSE/Cofluent Studio et GAUT dans notre stratégie de conception d'IP au niveau système. Le rôle de MCSE consiste à 1) permettre la spécification du modèle système et sa vérification par simulation, 2) la spécification de l'architecture cible et 3) l'aide au placement du modèle système sur cette dernière. Les contraintes d'entrées-sorties identifiées par MCSE sont alors transmises à l'outil GAUT qui les prend en compte pour synthétiser à son tour les IPs virtuels en vue de leur exécution sur l'architecture cible. De plus, cette plate-forme est accessible via Internet pour favoriser son utilisation à distance et rendre plus facile le partage des ressources de prototypage. Une approche fondée sur une modélisation Matlab et une génération de code par Simulink en amont de GAUT est aussi en cours d'étude.

La plate-forme de prototypage rapide doit donc permettre 1) de construire des architectures cibles ayant les performances requises par le domaine d'application des radiocommunications et 2) de générer semi-automatiquement les codes logiciels (pour processeurs DSP) et matériels (bitstreams de configuration de FPGA) nécessaires à l'exécution du prototype sur la plate-forme matérielle. Nous décrivons maintenant la plate-forme que nous proposons.

4.2. Plate-forme matérielle

La plate-forme matérielle est la réunion de composants matériels réutilisables et composables de façon à fabriquer à volonté l'architecture exécutive du prototype du système. Cette conception a volontairement été fondée sur la réutilisation systématique de composants du commerce. Pour déterminer la (ou les) société(s) fournisseur(s) des composants que nous recherchions, plusieurs études ont été menées afin de connaître le marché des équipements de prototypage, de sélectionner les candidats pertinents, puis de les comparer entre eux afin de retenir le meilleur d'entre eux.

4.2.1. Cahier des charges

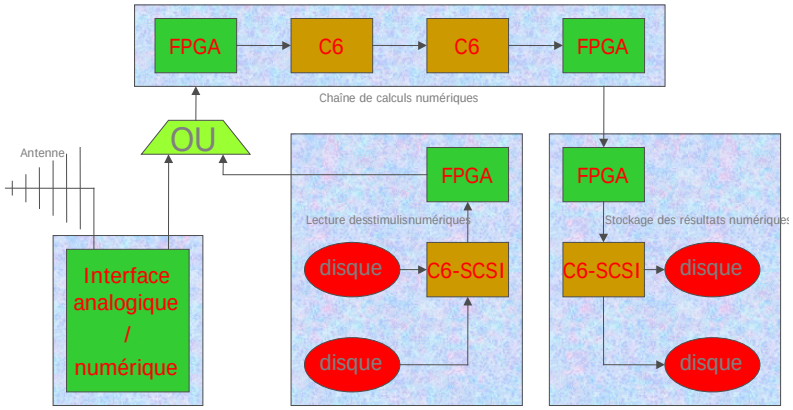


Figure 3-11 : Exemple d'architecture exécutive

Notre cahier des charges est : *pouvoir assembler facilement des architectures de topologies diverses incluant des DSP de la famille C6x de Texas Instruments [TEX04] et des FPGA communiquant au travers de liaisons point à point pour prototyper des systèmes entrant dans la catégorie des applications en radiocommunications que nous avons précédemment étudiées ».* Le tout, bien sûr, pour un budget acceptable !

Cette formulation rassemble l'ensemble des contraintes technologiques des partenaires du projet et laisse suffisamment de liberté pour avoir des chances de trouver plusieurs candidats potentiels. La Figure 3-11 illustre une architecture exécutive possible pour un prototype.

Les systèmes que nous voulons prototyper sont orientés traitement ce qui signifie que les calculs et les communications sont intensifs et réguliers. Les nombres d'opérations et les débits attendus sont prévisibles et ne dépendent pas des données. Ils sont de l'ordre de 1 Gops pour les calculs et 1 Gbit/s pour les communications.

4.2.2. Etude du marché et sélection du meilleur candidat

Le marché des équipements de prototypage peut être segmenté en cinq catégories commerciales. Cette classification est complémentaire à celle des méthodologies de prototypage de la section 3.1 et repose sur le degré d'intégration du produit. On trouve donc, du circuit intégré à l'ordinateur spécialisé :

- Les vendeurs de composants programmables (FPGA) qui intègrent (ou pas) peu à peu des cœurs de processeurs, des mémoires et des unités de calcul spécialisées autour de leur logique programmable, tels Actel [ACT04], Altera [ALT], Gatefield (racheté par Actel), Lucent [LUC04], Quicklogic [QUI04] et Xilinx [XIL04].

- Les vendeurs de processeurs qui font la démarche inverse en intégrant autour de leurs cœurs de processeurs de la logique programmable tels Atmel [ATM04] et Triscend [TRI04].

- Les vendeurs de cartes plus ou moins complexes et plus ou moins extensibles sont extrêmement nombreux et se distinguent par le type et le nombre de composants et d'interfaces qu'ils intègrent sur leurs cartes.

- Les vendeurs d'architectures modulaires à base de cartes qui proposent des catalogues de sous-systèmes (cartes filles et cartes mères) composables entre eux pour obtenir des topologies différentes tels Nallatech [NAL04], Sundance [SUN04], Transtech [TRN04] et Traquair/Hunt [TRQ04]/[HUN04].

- Enfin, les vendeurs de systèmes « clés en main » ou d'émulateurs tels Aptix [APT04], Axis [AXI04], Ikos [IKO04], Cadence/Quickturn [CAD04], Mentor Graphics/MetaSystems [MEN04], Simutech [SIM04] et Synopsys [SYN04].

Les solutions de types composants ont été rejetées car nous recherchons une solution de type prototypage rapide semi-dédié et les solutions de type émulateurs l'ont été aussi pour des raisons de coût. Seules les solutions à base de cartes satisfont alors nos besoins. Parmi les fournisseurs de cartes, seuls les quatre fournisseurs d'architectures modulaires du genre « mécano électronique » proposent la flexibilité dont nous avons besoin.

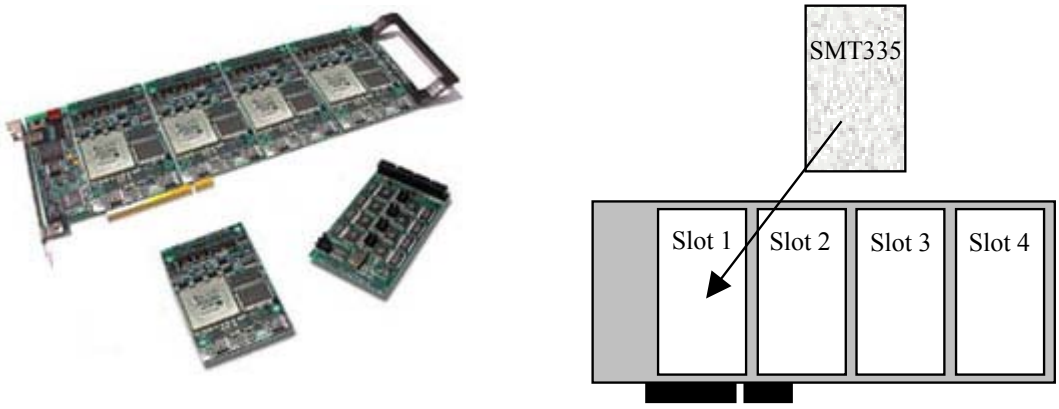
Le choix s'est finalement arrêté sur la société Sundance [SUN04] dont les produits satisfont au mieux nos critères de sélection. Les produits de Sundance sont à base de processeurs de Texas Instruments C6x, sont en ligne avec les derniers produits de Xilinx (leader du marché) et supportent une architecture originale à base de cartes mères et de câbles qui permet de construire (par câblage à la main ou dynamiquement à l'aide de matrices de commutation) des topologies point à point et bus partagé selon les besoins. De plus, toutes les cartes filles sont compatibles avec le format TIM-40 [TEX04] ce qui permet de les réutiliser dans un tout autre contexte si nécessaire. L'ensemble des partenaires du projet s'est équipé de matériels Sundance et nous pouvons plus facilement échanger des expériences, des résultats et même des personnes formées afin de poursuivre de nouvelles recherches en commun.

4.2.3. Architecture de la plate-forme

La plate-forme matérielle est constituée d'une baie contenant un PC industriel ayant 12 slots PCI, des cartes mères et des cartes filles achetées au fur et à mesure des besoins.

Les cartes mères ont quatre emplacements au format TIM-40 (slot TIM) qui peuvent chacun recevoir une carte fille. Les cartes mères sont des cartes longues au format PCI. Elles permettent au système de communiquer via le bus PCI avec le PC. Les cartes mères retenues sont des SMT310Q. Les cartes filles sont de divers types selon les besoins. Elles vont de la carte mono-DSP C62 (200 MHz), C64 (600 MHz), ou C67 (166 MHz), mono-FPGA VirtexE ou VirtexII (50 MHz) à la carte combinée DSP+FPGA. La Figure 3-12 illustre ce montage.

La communication inter cartes filles se fait par des liaisons point à point ou par bus partagé. Le bus partagé est prérouté sur les cartes mères, il est donc local à une carte. Un bus global partagé peut être implanté si l'on sacrifie quelques connecteurs des interfaces point à point afin de relier les cartes mère entre elles. La communication par liaisons point à point peut être faite à l'aide de deux interfaces différentes : les CP (comm-ports) et les liens SDB (Sundance Digital Bus).



a) présentation commerciale b) schéma de principe

Figure 3-12 : Carte mère STM310Q et carte fille en slot 1

Les CP sont des liaisons parallèles 8 bits, asynchrones, point à point, spécifiées par Texas Instruments avec le format TIM-40. Ce sont des liaisons ayant un débit maximum de 20 Mo/s (80 Mbits/s). Les SDB sont des liaisons parallèles, 16 bits, synchrones (50 MHz ou 100 MHz), spécifiées par Sundance. Elles ont un débit maximum de 100 Mo/s (800 Mbits/s) à 50 MHz et de 200 Mo/s (1.6 Gbits/s) à 100 MHz. Le nombre de CP et de SDB est variable selon les cartes filles. Les configurations disponibles sont de 6 CP et 2 SDB pour les cartes filles mono-processeurs et de 4 CP et 4 SDB pour les cartes filles mono-FPGA. Les cartes mixtes ont 6 CP et 4 SDB. Un prototype est potentiellement multi-cartes mères pour réunir tous les nœuds de calcul requis. Notre plate-forme matérielle est actuellement dimensionnée pour accueillir jusqu'à 48 nœuds de calcul. Au-delà, un autre fond de panier PCI doit être ajouté et la communication inter-PC faite au choix avec des CP ou des SDB.

Critère	Contrainte applicative	PALMYRE	Meilleur « compétiteur »	Performance du « compétiteur »
Nombre de FPGA	Puissance de calcul de l'ordre du Gops	Nombre illimité Virtex, VirtexE, Virtex2, Virtex2 Pro F = 50 ou 100 MHz	BEE	N x 20 Virtex, Virtex2 (BEE2) F = 100 MHz
Nombre de DSP	Puissance de calcul de l'ordre du Gops	Nombre illimité C62 à 200 MHz C67 à 177 MHz C64 à 600 MHz	NTT	4 C62 à 200 MHz et 1 Power-PC 400 MHz
Vitesse de comm max	Débit max de l'ordre du Gbit/s	comm-port 20 Mo/s SDB 200 Mo/s HSDB 400 Mo/s	BEE	100 Mmot/s
Nombre de canaux de comm/noeud	Maximum possible	4 à 6 comm-port + 2 à 4 (H)SDB	BEE	8 liens
Bande passante cumulée par nœud	Maximum possible	480 Mo/s min 1720 Mo/s max	BEE	800 Mmot/s

Tableau 3-3 : Position de la plate-forme PALMYRE face à la « compétition »

Grâce à ses caractéristiques techniques, notre plate-forme se situe actuellement dans le haut de gamme des plates-formes de prototypage rapide semi-dédié. Elle combine l'ensemble des qualités requises sans avoir aucun des inconvénients des autres. La comparaison du Tableau 3-3 illustre ce fait. Les caractéristiques les plus évidentes sont le nombre et la puissance des différents éléments constitutifs de la plate-forme : 1) le nombre, le type et la fréquence des FPGA, 2) le nombre, le type, et la puissance en Mops des processeurs logiciels et 3) le nombre, le type, et la vitesse des liens de communication des FPGA et des processeurs.

Cette comparaison prouve que notre plate-forme dispose des ressources les plus grandes en terme de puissance de calcul et de communication. Enfin, lorsque l'on sait que les besoins en terme de puissance de calcul et de vitesse communication unitaire sont respectivement de l'ordre du Gops et du Gbit/s, on constate que la plate-forme possède les ressources matérielles nécessaires au prototypage d'applications de radiocommunications comme l'illustre le Tableau 3-4. En ce qui concerne la vitesse de communication, seuls les liens SDB à 100 MHz et les liens HSDB supportent la contrainte de 1 Gbit/s. Néanmoins, les SDB à 50 MHz et les CP peuvent être utilement employés pour les liaisons moins exigeantes en débit.

Critère	Contrainte	PALMYRE
Puissance de calcul DSP	1 Gops	C62 200 MHz – 1.60 Mops C67 166 MHz – 1.00 Mflops C64 600 MHz – 4.80 Mops
Puissance de calcul FPGA	1 Gops	à 50 MHz 20 mult. => 1 Gops à 100 MHz 10 mult => 1 Gops
Vitesse de transmission	1 Gbit/s	CP => 0.16 Gbit/s SDB 50 MHz => 0.80 Gbit/s SDB 100 MHz => 1.60 Gbit/s HSDB => 3.20 Gbit/s

Tableau 3-4 : Position de la plate-forme PALMYRE par rapport aux contraintes

Malgré ces résultats favorables, la simple liste des performances matérielles crêtes ne suffit pas à prouver qu'une plate-forme permet de prototyper un système. Elle permet seulement d'affirmer que les ressources nécessaires sont là, mais ne garantit nullement une aptitude à les utiliser efficacement. En effet, la plate-forme logicielle (API) peut ne pas être aussi performante que la plate-forme matérielle et peut donc ruiner tous les avantages de cette dernière. La véritable comparaison entre plates-formes doit donc se situer au niveau de la fonctionnalité globale (puissance cumulée de calcul et de communication simultanés pour des topologies données). Le manque total d'informations quantitatives relatives aux performances des plates-formes logicielles « concurrentes » ne nous permet pas de nous comparer. Conscient de ce problème nous proposons néanmoins une méthodologie de mesure que nous appliquons à notre plate-forme et espérons que d'autres candidats s'en inspireront afin de caractériser objectivement les performances de leurs plates-formes.

4.3. Plate-forme logicielle

La plate-forme matérielle « native » est agrémentée d'un outil de compilation C/C++ pour DSP (Code Composer Studio de TI), d'un RTOS (DSP-Bios de TI) et de l'outil de synthèse RTL pour FPGA (ISE de Xilinx). En terme d'API, seuls quelques exemples de codes C « technico-marketing » favorables au discours des vendeurs sont fournis avec le matériel pour communiquer entre deux DSP ainsi que les interfaces VHDL RTL des CP et SDB sous accord de non divulgation. Ils ne sont aucunement qualifiables d'API et ne sont ni génériques, ni réutilisables, ni exempts de bugs …

La plate-forme logicielle doit néanmoins comporter une API de communication permettant de faire communiquer naturellement entre elles des implantations matérielles et logicielles d'IPs virtuels. L'API que nous proposons, et que nous avons développée, a pour but de supporter la sémantique du modèle du réseau de processus de Kahn. Elle implante une communication point à point entre processus. De plus, comme toute implantation est forcément bornée, l'API doit

permettre de régler le découplage (la taille des buffers/FIFO intermédiaires) afin que l'implantation ait un comportement aussi proche que possible d'un véritable PN (process network). La détermination automatique de la taille optimale des FIFOs sort du cadre de cette étude et nécessite 1) de caractériser le comportement statistique des flux de données et 2) de tenir compte des contraintes applicatives en terme de rupture de flux. Par exemple, un lecteur de CDs audio qui doit résister à une rupture du flux de lecture de 30 secondes doit pouvoir stocker en permanence en mémoire une séquence de données qui représente 30 secondes d'écoute.

4.3.1. Spécification et méthodologie d'évaluation de l'API

La spécification de notre API est hiérarchique et repose sur une décomposition C++ en classes avec notion d'héritage pour les diverses implantations possibles (CP, (H)SDB).

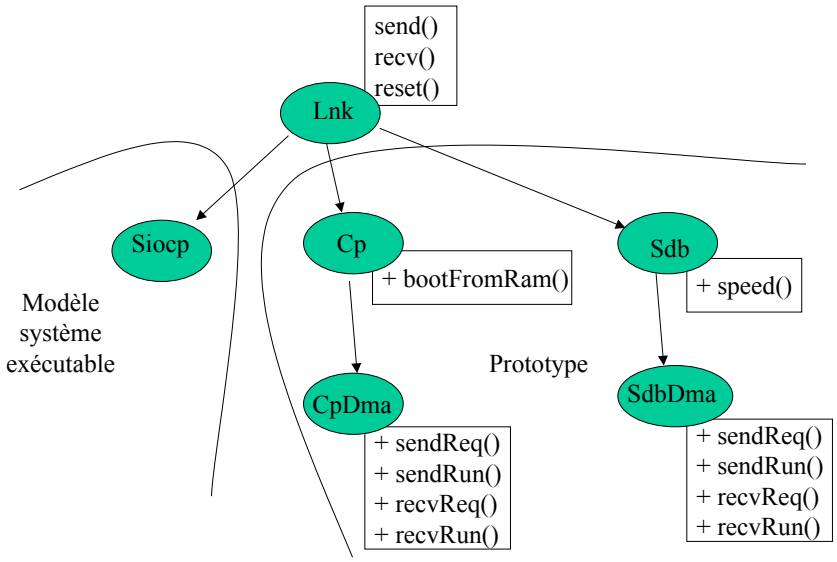


Figure 3-13 : API de la plate-forme logicielle

La hiérarchie des classes débute par la classe *LnK* qui contient les trois méthodes communes à toutes les sous-classes et donc à toutes les implantations possibles. La classe *Lnk* est porteuse de la sémantique commune : les sous-classes ne sont que des extensions de cette dernière. L'API minimale est constituée par une méthode *send()* d'envoi de données, une méthode *recv()* de réception de données, et une méthode *reset()* de remise à zéro (un équivalent du reset matériel). Les sous-classes de la classe *Lnk* se répartissent en deux catégories : le modèle système exécutable, et le prototype.

La catégorie « modèle système exécutable » ne contient qu'une seule classe : la classe *Siocp* qui implante la sémantique de communication au dessus des IOSTREAM de DSP-BIOS. Cette classe permet de spécifier le système comme un ensemble de processus C++ qui communiquent entre eux au travers de notre API. Ce modèle système est exécutable sur un C6x et simulable sur un PC avec le simulateur de C6x. On peut donc, grâce à cette API, exécuter la spécification du système et disposer d'une référence fonctionnelle. Cette catégorie sera étendue s'il est nécessaire d'utiliser un autre RTOS que DSP-BIOS.

La catégorie « prototype » contient quatre classes : *Cp*, *CpDma*, *Sdb* et *SdbDma*. Les classes *Cp* et *CpDma* se distinguent par le fait qu'il s'agit d'une implantation qui utilise les services des CP (comm-ports) et que la première utilise le DSP pour transférer les données (*Cp*) alors que la deuxième utilise un DMA (*CpDma*). Ces classes possèdent une méthode supplémentaire *bootFromRam()* qui sert à configurer les FPGA à partir d'un bitstream stocké en mémoire. *CpDma*, grâce à l'usage du DMA, permet d'écrire des codes qui exploitent le temps de communication pour faire du calcul à l'aide des méthodes *sendReq(), sendRecv(), recvReq*, et *recvRun()*. On peut donc écrire des applications qui calculent et communiquent en temps masqué (en paralléle). Les classes *Sdb* et *SdbDma* sont l'équivalent pour les liens SDB (Sundance Digital Bus). Elle possèdent la méthode *speed()* qui permet de spécifier à quelle vitesse on veut fonctionner (50 MHz ou 100 MHz) et n'ont pas la méthode *bootFromRam()* qui n'est applicable qu'aux CP. Enfin, comme il se doit en C++, chacune de ces classes est dotée d'un constructeur spécifique. Le raffinement d'une classe en une autre se limite à la surcharge de l'appel du constructeur avec les paramètres pertinents.

Une API mal conçue peut ruiner les qualités de la plate-forme matérielle. Afin de tirer le meilleur parti des composants matériels il est tout d'abord nécessaire de caractériser leurs comportements en fonction de la communication pour déterminer quelles sont les conditions favorables à l'obtention des meilleures performances globales. Nous proposons la méthodologie de caractérisation et de test suivante :

- Phase 1 *: Caractérisation des conditions optimales de fonctionnement*. Cela consiste en un test systématique de tous les liens de communication disponibles entre tous les types de nœuds de calcul. Ce test consiste à envoyer des paquets de données de taille croissantes et à mesurer la vitesse et la consommation CPU atteintes. Pour une configuration donnée émetteur-lien-recepteur, on obtient la courbe vitesse/taille. Cette courbe permet de savoir à partir de quelle taille de paquet la vitesse maximale est atteinte. Elle permet de dimensionner la taille minimale des buffers sur les nœuds de calcul pour obtenir la vitesse de communication maximale.

- Phase 2 : *Mise en œuvre d'un test de référence*. Nous proposons une chaîne de traitement simple qui est constituée d'un producteur d'échantillons (DSP), d'un FIR (FPGA) et d'un consommateur de résultats (DSP). Il s'agit d'obtenir un système trivial, simple à mettre en œuvre avec l'API synchrone. Ce système exhibe les défauts attendus d'une API qui serait, soit de mauvaise qualité, soit mal utilisée. Il s'agit d'une mesure en nombre d'exécutions par seconde de l'algorithme synthétisé sur le FPGA par GAUT. On obtient alors un nombre d'exécutions par seconde de l'algorithme « pire cas ». Quelle que soit la durée de calcul du FIR sur le FPGA, nous voulons mesurer la surcharge de communication due à l'exploitation simple, mais non optimale de l'API.

- Phase 3 *: Mise en œuvre du regroupement des données en paquets pour exploiter les conditions matérielles optimales*. Afin d'augmenter le nombre d'exécutions par seconde de l'algorithme, il est tout d'abord nécessaire de se mettre dans les conditions matérielles optimales précédemment identifiées. Il s'agit de grouper les données avant de les transmettre. Les mesures de performances indiquent quelle est la taille minimale de buffer à

remplir avec N échantillons avant de les transmettre. On obtient alors un deuxième nombre d'exécutions par seconde de l'algorithme. Il s'agit de l'optimum matériel.

- Phase 4 *: Mise en œuvre de la communication en temps masqué pour exploiter les conditions logicielle optimales*. Afin d'atteindre le véritable optimum, il est nécessaire d'exploiter le mode « asynchrone » de l'API qui permet (par utilisation du DMA en parallèle avec la CPU) de communiquer en temps masqué avec le calcul. Ainsi, lors de la transmission par le DMA des N échantillons stockés dans un buffer, on calcule avec le DSP les valeurs des N suivants que l'on stocke dans un autre buffer. Il s'agit d'utiliser les primitives asynchrones de l'API à la place des synchrones. On obtient alors le nombre maximal d'exécutions par seconde.

Cette méthodologie est applicable à toute plate-forme logicielle et permet de mesurer l'optimum atteignable avec un test simple mais représentatif de la mixité d'un système complet. Nous appliquons cette méthodologie à la plate-forme PALMYRE et donnons les résultats obtenus.

4.3.2. Phase 1 : Caractérisation

Tout programme qui s'exécute sur un processeur dépend du compilateur et des ressources matérielles mises en œuvre. Ainsi le compilateur CCS possède plusieurs niveaux d'optimisation du code généré qui permettent de tester différents compromis vitesse/taille du code. D'autre part, la performance de tout programme dépend aussi du placement des données. Les lectures et écritures de données sont toujours plus rapides avec une mémoire interne (64K octets dans les C6x) qu'avec une mémoire externe (plusieurs Moctets de SRAM). Enfin la disponibilité d'accélérateurs (tel un DMA pour la communication) permet d'effectuer certaines opérations (les transferts de données) de façon beaucoup plus efficace que le processeur lui-même.

Nous avons donc conçu une API accompagnée de programmes de tests qui permettent :

- De faire varier le degré d'optimisation du code lors de la compilation : nous nous sommes limités aux modes « non optimisé » et « optimisé au maximum ».

- De faire varier le placement des données : mémoire interne et mémoire externe.

- D'exploiter le processeur ou un accélérateur DMA pour le transfert des données.

Ces programmes de test de vitesse et de consommation CPU en fonction de la taille des données envoyées donnent les résultats suivants pour les liens CP et SDB.

4.3.2.1 Caractérisation des CP

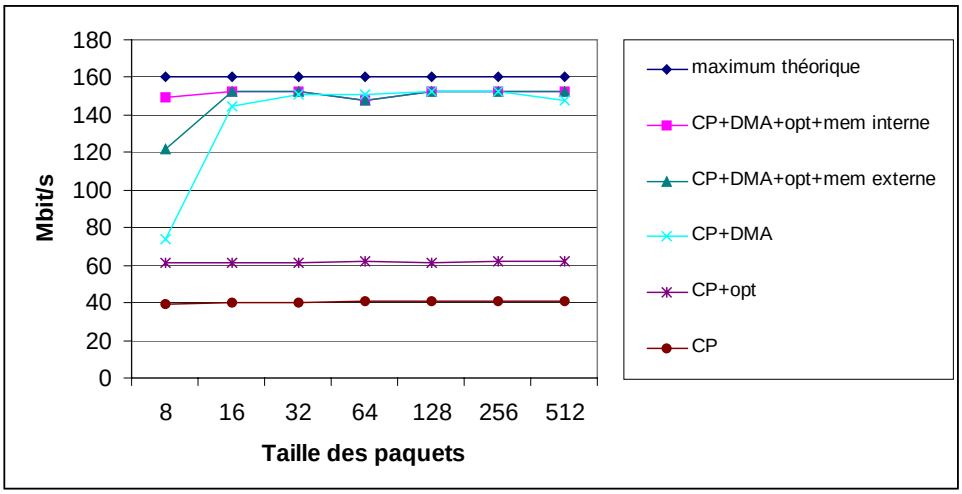


Figure 3-14 : Vitesse des CP en fonction de la taille des paquets pour différentes configurations

Le débit crête accessible théoriquement d'après la spécification du protocole CP est de 160 Mbit/s. Les résultats illustrés par la Figure 3-14 prouvent que les performances maximales du lien de communication CP sont effectivement accessibles au niveau API si l'on utilise le DMA pour transmettre des paquets de données d'au moins 16 à 32 mots. Dans tous les autres cas, la limite en vitesse est imposée par le coût non négligeable d'un appel de procédure par rapport au temps de transfert (cas des petits paquets) et par la lenteur relative du DSP par rapport au DMA lorsqu'il doit effectuer lui-même les transferts de données.

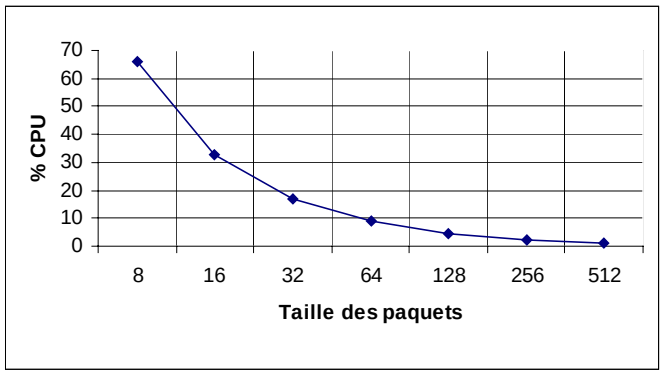


Figure 3-15 : Consommation CPU des CPDMA en fonction de la taille des paquets

Les résultats illustrés par la Figure 3-15 prouvent que la puissance CPU consommée par l'appel aux services de l'API avec la classe *CpDma* est négligeable (inférieure ou égale à 5 %) à partir de tailles de paquets de 128 mots. En effet, lorsque le DMA effectue le transfert des données, le DSP peut être employé à toute autre tâche. La classe *Cp*, quand à elle, consomme 100% de la CPU car elle n'utilise pas les services du DMA et le DSP est alors responsable du mouvement des données.

4.3.2.2 Caractérisation des SDB

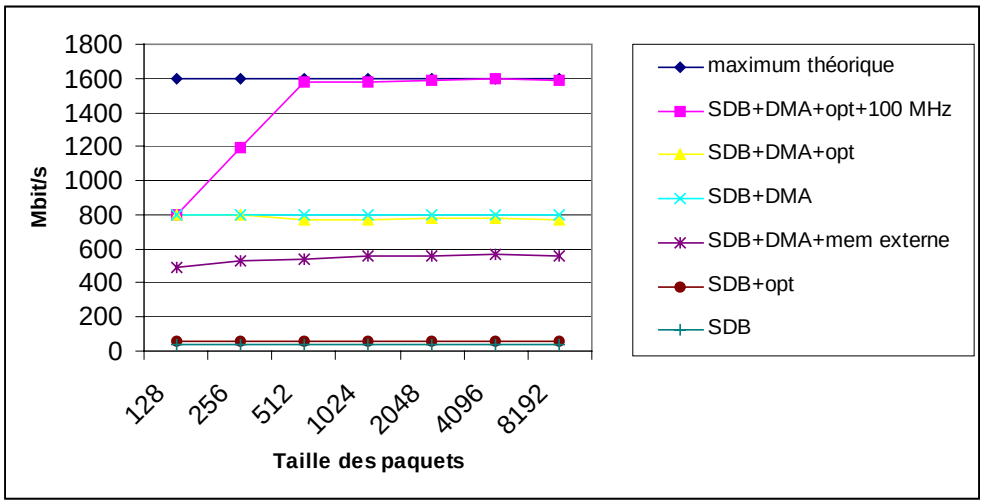


Figure 3-16 : Vitesse des SDB en fonction de la taille des paquets pour différentes configurations

Le débit crête accessible théoriquement d'après la spécification du protocole SDB est de 1600 Mbit/s. Les résultats illustrés par la Figure 3-16 prouvent que les performances maximales du lien de communication SDB sont accessibles au niveau API si l'on utilise des données placées en mémoire interne et le DMA pour les transmettre. A 100 MHz la taille des paquets intervient dans les performances et il faut qu'elle soit supérieure à 512 mots pour que la vitesse maximale soit atteinte. Dans tous les autres cas, la limite de vitesse est imposée soit par la bande passante de la mémoire externe, soit par le coût des appels de procédures et par la lenteur relative du DSP par rapport au DMA lorsqu'il doit effectuer lui-même les transferts de données.

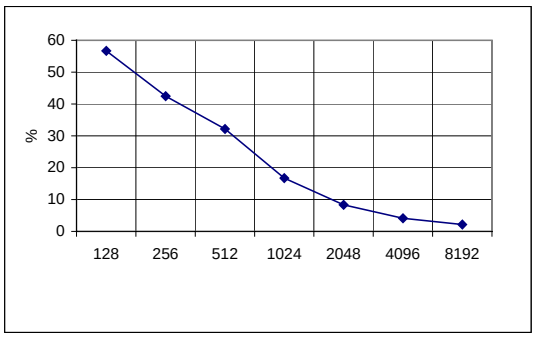


Figure 3-17 : Consommation CPU des SDBDMA en fonction de la taille des paquets

De même que pour les CPs, la puissance CPU consommée par l'appel aux services de l'API avec la classe *SdbDma* est négligeable (inférieure ou égale à 5 %) à partir de tailles de paquets de 4 Kmots.

4.3.2.3 Conclusion

Les conditions optimales de fonctionnement dépendent de la nature des besoins en calculs. S'il n'est pas nécessaire de confier au DSP des calculs importants lors de la communication alors des buffers mémoires de tailles minimales suffisent. Le Tableau 3-5 les résume.

Type de lien	Taille minimum (mots)	Débit atteint (Mbit/s)
CP + DMA	16	160
SDB + DMA	512	1600

Tableau 3-5 : Taille minimale des buffers pour atteindre l'optimum en débit

Par contre, s'il est nécessaire d'utiliser le mode asynchrone de l'API pour confier des calculs au DSP, pendant que le DMA effectue le transfert des données, alors des buffers de tailles sensiblement plus grandes sont nécessaires. Le Tableau 3-6 les résume.

Type de lien	Taille minimum (mots)	% puissance CPU libre
CP + DMA	128	> 95 %
SDB + DMA	4096	> 95 %

Tableau 3-6 : Taille minimale des buffers pour atteindre l'optimum en débit/CPU

4.3.3. Phase 2 : Référence « pire cas »

Le test de référence est constitué d'un simple filtre FIR de façon à avoir relativement peu de calculs et ainsi observer l'impact de la communication sur la puissance du prototype. La puissance du prototype est exprimée en nombre d'exécution du filtre par secondes.

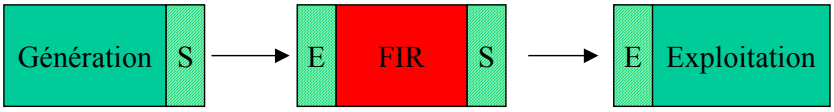


Figure 3-18 : Modèle du test de référence

Ce test a pour but de mesurer l'impact d'une programmation logicielle ne tenant pas compte des contraintes mémoires identifiées par la caractérisation des liens de communication. Ainsi les données communiquées (un échantillon et un résultat par exécution de l'algorithme) le sont par paquets de 1 échantillon.

- Puissance maximale de calcul

La synthèse non contrainte en temps du filtre, pour un VirtexE-1000 cadencé à 50 MHz, nous a fourni un circuit non pipeliné dont la latence de calcul et la cadence d'échantillonnage sont de 24 cycles, soit 480 ns. La puissance maximale est donc de 2,1 millions de calculs par seconde. Le débit nécessaire pour soutenir cette puissance de calcul est de 2,1 Méchantillons/s soit 67,2 Mbit/s.

- Mesure

Le fait de transmettre les échantillons par paquets de 1 mot place le prototype dans des conditions défavorables qui sont essentiellement dues à une mauvaise exploitation de l'API. La mesure confirme ce fait : la vitesse maximale obtenue est de 4,45 Mbit/s soit une puissance de seulement 130 000 calculs par seconde.

4.3.4. Phase 3 : Regroupement des données en paquets et API en mode synchrone

Ce test consiste à factoriser la transmission des échantillons et des résultats et à observer le comportement en puissance du prototype en fonction de la taille des buffers de communication.

Le fait de faire varier la taille des paquets d'échantillons nous permet de mesurer les vitesses de transmission avec les mêmes programmes que ceux qui ont servis à la caractérisation de l'API, et donc de déduire la puissance de calcul du prototype. Les résultats obtenus sont illustrés par la Figure 3-19. Ils sont obtenus avec un CP avec DMA.

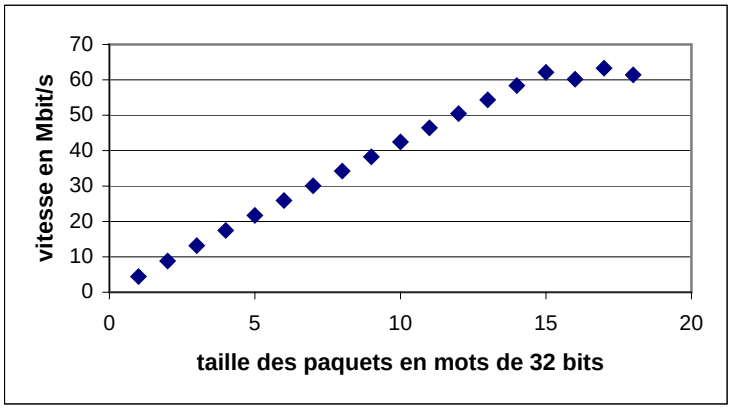


Figure 3-19 : Mesure du débit en fonction de la taille des paquets

Cette expérience, très simple, montre que le bon dimensionnement des buffers est indispensable pour pouvoir exploiter la puissance de calcul d'un circuit synthétisé par GAUT. On observe l'accroissement de la vitesse en fonction de la taille des paquets qui est quasiment linéaire jusqu'au point ou le calcul domine et ou la vitesse est imposée par la vitesse de calcul du filtre. Cette vitesse limite est, aux incertitudes de mesure près, de l'ordre de 60 Mbit/s.

4.3.5. Phase 4 : Communication en temps masqué et API en mode asynchrone

Ce test consiste à exploiter le mode asynchrone de l'API de façon à pouvoir calculer avec le DSP lorsque le DMA effectue le transfert des données. La Figure 3-20 illustre les résultats obtenus avec le même test de communication mais en mode asynchrone.

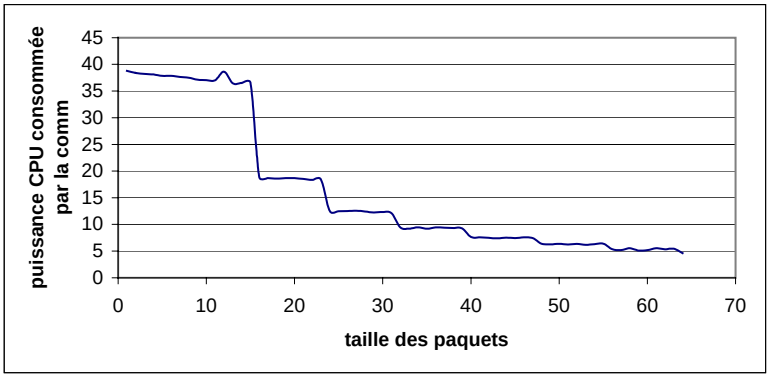


Figure 3-20 : Mesure de la puissance CPU consommée par la communication en fonction de la taille des paquets

Les tailles de paquets vont de 1 à 64 mots, taille à partir de laquelle la consommation chute en dessous de 5 %. Les paliers que l'ont observe sur la courbe illustrent les performances de l'interface CP qui repose sur un fonctionnement particulier du DMA. Le DMA transmet exclusivement des paquets de tailles multiples de la demie FIFO qui le découple du protocole de l'interface physique. Lorsque les données ne sont pas en nombre suffisant pour obtenir un multiple de 8 mots, il y a ajout de mots de « bourrage ». Cette FIFO contient 16 mots et son remplissage (vidage) est fait lorsqu'elle est à demie vide (pleine), c'est-à-dire tous les 8 mots. On constate alors que la puissance CPU consommée dépend essentiellement de la quantité de données effectivement transmises (des multiples de 8 mots) : ce sont les tailles 16, 24, 32, 40, 48 et 56 mots.

A la suite de l'application de cette méthodologie d'évaluation de l'API dans un contexte particulier, nous concluons que la taille des paquets qui permet de faire fonctionner le filtre dans les meilleures conditions est d'au moins 15 à 16 mots. Dans ces conditions la consommation de puissance CPU du DSP est de l'ordre de 35 à 40 %. Il est alors possible de réduire la consommation CPU pour faire d'autres calculs en temps masqué en exploitant le mode asynchrone de l'API et en acceptant d'allouer des buffers d'au moins 60 mots pour transmettre les paquets. Dans ces conditions, la consommation de puissance CPU approche les 5 % et peut être considérée comme négligeable.

Après avoir sélectionné la plate-forme matérielle puis développé et caractérisé l'API de la plate-forme logicielle, nous présentons la plate-forme système qui réunit ces deux dernières et intègre la synthèse de haut niveau par GAUT d'unités de traitement, de mémorisation et de communication.

4.4. Plate-forme système

Notre plate-forme système actuelle est constituée de la réunion des plates-formes logicielle et matérielle, et des outils de conception (GAUT), validation (Modelsim, remote debugger) et raffinement (CCS et ISE). Les travaux concernant MCSE n'étant pas terminé, nous décrivons un processus automatisable qui est, à l'heure actuelle, dirigé par le concepteur.

Notre méthodologie de prototypage rapide, comme l'illustre la Figure 3-21, est articulée en trois phases : spécification système, partitionnement matériel/logiciel et raffinement du prototype. Nous les passons maintenant en revue et présentons les outils employés ainsi que leurs relations.

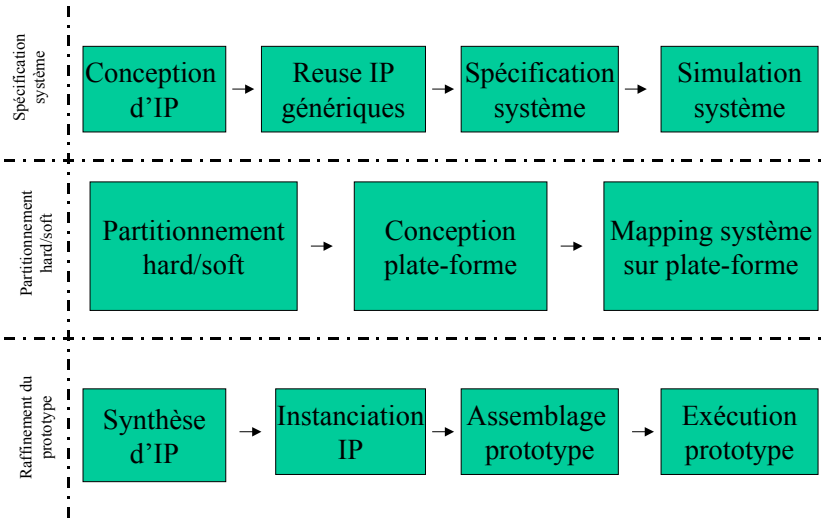


Figure 3-21 : Synoptique de la méthodologie de protypage rapide

4.4.1. Spécification d'un modèle système exécutable

Cette spécification consiste à écrire des programmes C++ utilisant l'API définie et/ou à réutiliser des programmes déjà écrits au dessus de cette API, puis à les assembler à l'aide des IOSTREAM dans un environnement concurrent DSP-BIOS. L'écriture des codes repose sur l'API spécialisée de la classe *SioCp*. Ensuite, par simulation avec un ISS (Instruction Set Simulator) ou par exécution sur une plate-forme matérielle limitée à un seul nœud C6x, on vérifie la validité fonctionnelle du système muti-tâches décrit. Ce modèle système est un graphe de taches communiquantes au travers de liaisons point à point. Les liaisons point à point sont associées à des STREAMS DSP-BIOS. Les STREAMS sont des liaisons point à point « logicielles » qui assurent un flot de données entre deux tâches, elles ont la même sémantique (du point de vue des séquences d'entrées-sorties) que les liaisons point à point système. Ce sont des objets du RTOS dont la création est dynamique et qui portent chacun un nom unique permettant de les différencier entre eux. La spécification des deux extrémités d'une même liaison se fait en employant le nom du STREAM dans les constructeurs qui instancient les objets de la classe *SioCp* aux extrémités de la liaison.

Ainsi, pour un système « simpliste » (qui correspond à notre test de référence Figure 3-18) du type pipeline, le modèle système et l'architecture exécutive sur lequel il s'exécute sont les suivants :

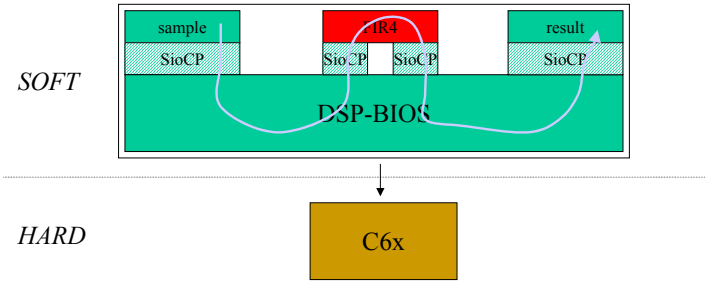


Figure 3-22 : Exécution du modèle système sur un C6x

On constate qu'un seul processeur DSP exécute la totalité du système et que le RTOS DSP-BIOS implante en temps partagé la fonctionnalité multi-tâches et la communication inter-tâches nécessaires. Cette implantation est, au nombre de processeurs près et à la vitesse près, équivalente à une solution multi-DSP, plus coûteuse. Avec l'ISS le contexte de l'exécution est le suivant :

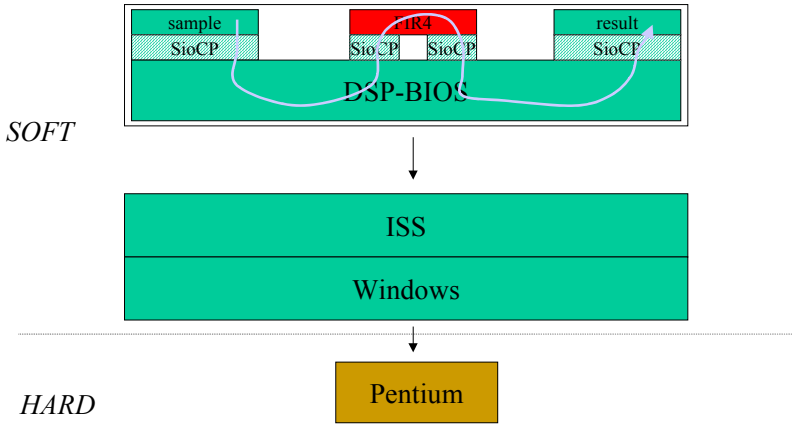


Figure 3-23 : Exécution du modèle système sur un PC avec un ISS

On remarque que ce n'est plus un processeur de la plate-forme matérielle qui exécute le système, mais un processeur standard. On peut donc simuler le système, dans des conditions très proches (excepté la vitesse) d'une plate-forme matérielle multi-DSP, sur son PC de bureau.

4.4.2. Partitionnement et placement matériel/logiciel

Le partitionnement et le placement sont actuellement manuels car les travaux concernant l'intégration de GAUT avec MCSE ne sont pas terminés à l'heure de la rédaction. Nous poursuivons l'exposé avec un exemple de placement (décidé par le concepteur) du calcul du filtre FIR sur un FPGA, et des autres tâches sur des DSP séparés. Nous obtenons un système à trois nœuds de calcul et deux liaisons point à point.

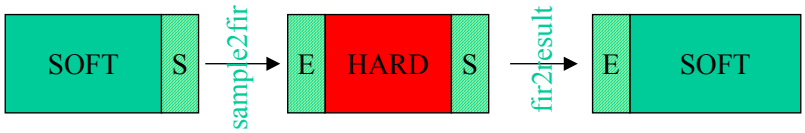


Figure 3-24 : Partitionnement matériel/logiciel

La phase de conception de la plate-forme matérielle consiste à assembler une architecture exécutive sur laquelle on place ensuite les tâches du système. La meilleure architecture (d'un point de vue partitionnement/placement rapide manuel) est celle qui est la plus proche de la topologie du système. On peut aussi concevoir une architecture dans laquelle les deux tâches logicielles s'exécutent sur un seul DSP. Dans ce cas il est alors nécessaire d'inclure le RTOS DSP-BIOS avec ces tâches. Notre philosophie est de sur-allouer en ressources de calcul et de communication pour accélérer le prototypage plutôt que de chercher à optimiser à tout prix (réduire le nombre de noeuds de calcul) et à complexifier le raffinement du prototype. La solution à trois nœuds de calcul nous garantit que la totalité des ressources des DSP et du FPGA est consacrée aux seules tâches qui ont de l'intérêt pour nous. Avec DSP BIOS, le partage des ressources est moins prévisible et il pourrait pénaliser la fréquence maximale d'exécution du FIR à cause du coût des changements de contextes entre les deux tâches.

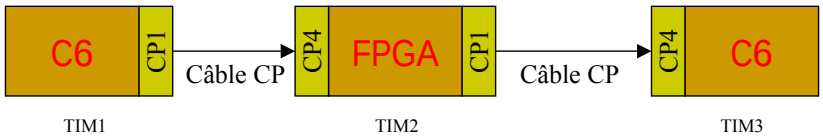


Figure 3-25 : Assemblage de la plate-forme matérielle

L'architecture matérielle étant maintenant isomorphe à l'architecture système, le placement est immédiat et ne nécessite aucun mécanisme de partage de ressources (ni RTOS pour le multi-tâches, ni multiplexage pour la communication).

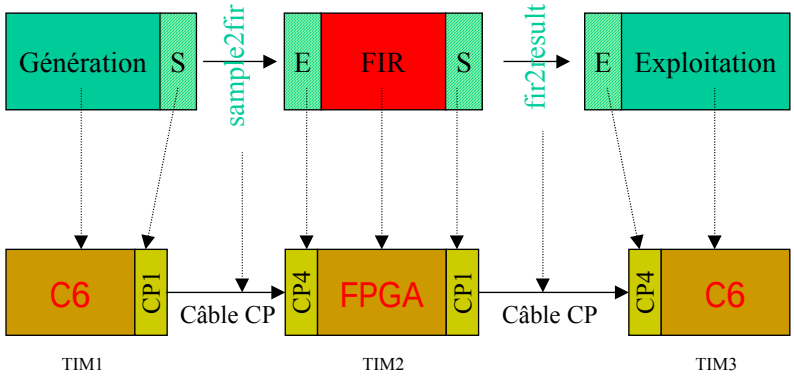


Figure 3-26 : Placement du système sur la plate-forme matérielle

La Figure 3-26 illustre ce placement : lors du raffinement du système vers le prototype, on placera les tâches logicielles sur les DSP, la tâche FIR sur le FPGA, et les canaux logiques sur des canaux physiques de type CP.

4.4.3. Raffinement du prototype

Le raffinement du prototype consiste à raffiner les parties logicielles et matérielles.

Les parties logicielles sont raffinées par simple remplacement du constructeur SioCp (le constructeur des STREAM) par le constructeur de la classe *Cp*, *CpDma*, *Sdb*, *SdbDma* qui correspond au matériel alloué pour l'implantation de la liaison point à point. Le placement du canal logique (identifié par un nom de STREAM) sur le canal physique (un numéro de CP) consiste en la spécification du paramètre qui identifie la liaison point à point physique du *CP* ou du *SDB*. La description à base de nom de STREAM est donc remplacée par une description physique des extrémités après placement du modèle sur l'architecture exécutive. La modification est mineure car elle ne concerne que l'appel du constructeur, le reste du code des applications reste inchangé.

```
SioCp canal(n)          Cp          canal (n, in_out) ;
                        CpDma       canal (n, in_out) ;
                        Sdb         canal (n, in_out) ;
                        SdbDma      canal (n, in_out) ;
```

Figure 3-27 : Exemple de raffinement logiciel par spécialisation des classes

La Figure 3-27 illustre le raffinement d'un canal SioCp en un canal de communication physique de type Cp, CpDma, Sdb ou SdbDma. Le paramètre *n* du constructeur SioCp qui représente le nom du STREAM est remplacé par deux paramètres *n* et *in_out* qui sont respectivement le numéro physique du CP ou SDB utilisé et un booléen qui indique s'il s'agit d'un émetteur ou d'un récepteur au reset.

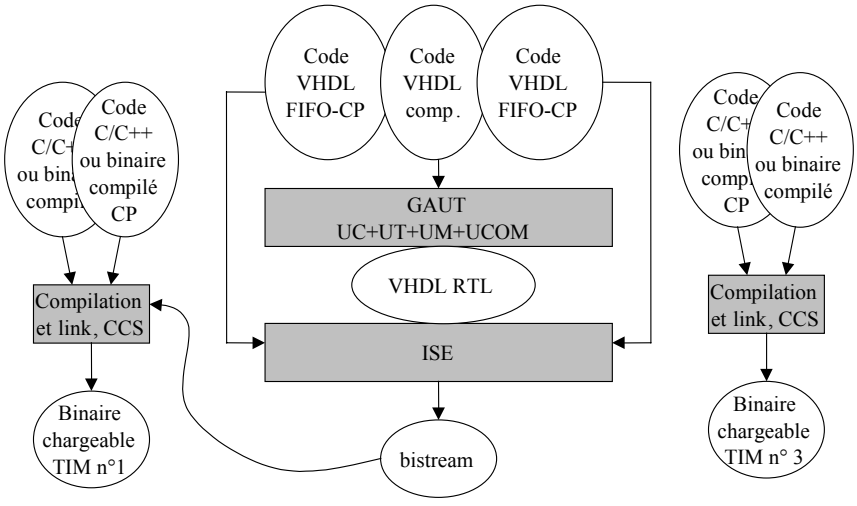


Figure 3-28 : Raffinement du système en prototype

Les parties matérielles sont raffinées à l'aide de l'outil GAUT. Actuellement GAUT accepte en entrée un algorithme écrit en VHDL comportemental. La traduction C/VHDL doit être faite manuellement car le projet RNTL SystemCmantic, dont le but est d'apporter à GAUT une interface de spécification en SystemC, vient juste de commencer. En attendant la disponibilité de cette nouvelle interface de spécification, il est important de vérifier par simulation la

correction fonctionnelle du modèle comportemental VHDL obtenu. L'outil GAUT est ensuite utilisé pour synthétiser les unités de calcul, de contrôle, de mémorisation, de duplication et de communication permettant à l'algorithme d'être exécuté sur un FPGA relié au reste des nœuds de calcul via des CP ou des SDB.

4.4.4. Chargement, exécution, mise au point

Le chargement de la plate-forme matérielle avec les codes compilés des tâches se fait avec l'outil CCS qui permet de charger individuellement chacun des processeurs avec son code via la chaîne JTAG. Le chargement des FPGA est fait au démarrage des DSP qui embarquent dans leurs données une image binaire du bistream de configuration. La méthode *bootFromRam()* permet d'indiquer explicitement quelle tâche est responsable du boot du FPGA. Il existe aussi une technique de chargement instantané de tous les processeurs que nous n'avons pas utilisée car, dans une phase de développement et de mise au point, nous préférons observer individuellement tous les processeurs. La mise au point des codes est faite à l'aide d'un debugger croisé symbolique au niveau C++ ou à l'aide de sondes dans les FPGA. La Figure 3-29 illustre ce mode de fonctionnement. Contrairement à toutes les figures précédentes, l'ordre des couches « contrôle », « plate-forme matérielle » et « prototype » est volontairement inversé pour indiquer au lecteur qu'il faut successivement traverser d'abord le « contrôle » de la plate-forme, puis la plate-forme matérielle elle-même avant d'atteindre les fonctionnalités logicielles et matérielles du prototype qui seules nous intéressent réellement.

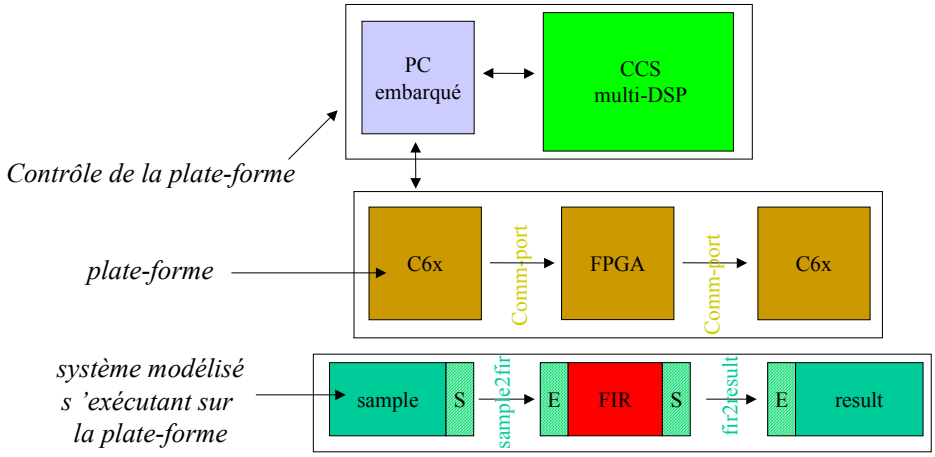


Figure 3-29 : Prototype complet

L'outillage de mise au point de plate-forme de prototypage permet ainsi d'observer les tâches du système individuellement :

- De façon intrusive pour les tâches logicielles avec le debugger : les points d'arrêt et les traces peuvent modifier les conditions temps réel d'exécution du système et masquer les conditions de détection des problèmes fonctionnels.

- De façon non intrusive avec des sondes dans les FPGA. Les FPGA ayant plusieurs ports de communication (CP et SDB), quelques dizaines d'entrées-sorties sont généralement disponibles, car non utilisées pour la communication. Le routage des signaux que l'on souhaite observer vers les broches des connecteurs libres permet de les observer avec un analyseur logique. Cette instrumentation n'est pas automatisée actuellement et il serait souhaitable qu'elle soit prochainement intégrée à la synthèse de haut niveau de GAUT.

- De façon intrusive en surface FPGA, mais pas en temps : on peut utiliser l'IP d'analyse logique ChipScopeILA de Xilinx qui charge l'équivalent d'un analyseur logique à l'intérieur du FPGA. L'exploitation des signaux échantillonnés par ChipScopeILA ne peut alors se faire que de façon « post-mortem », c'est-à-dire après interruption volontaire de l'exécution.

5. Conclusion

Dans ce chapitre nous avons présenté notre approche orientée plate-forme. Nous faisons tout d'abord une étude des contraintes des applications de radiocommunications numériques, explorons les méthodologies de prototypage rapide, analysons les architectures, domaines d'application et outillages des plates-formes actuelles et justifions l'intérêt technico-économique du prototypage rapide. Nous proposons alors une plate-forme complète de prototypage rapide dans le cadre du projet PALMYRE. Nous montrons que le dimensionnement de cette plate-forme répond aux contraintes identifiées en termes de puissance de calcul, de débit de communication et de nombre de nœuds de calcul. Nous démontrons aussi que notre plate-forme fait partie des plates-formes les plus puissantes et les plus flexibles. Nous proposons, développons et caractérisons une API logicielle qui permet de communiquer naturellement entre IPs virtuels implantés en logiciel ou en matériel. Nous identifions les conditions optimales qui permettent d'exploiter les performances crêtes. Ces paramètres « d'optimalité » locale deviennent alors des contraintes d'allocation mémoires pour la synthèse d'IPs. Nous proposons enfin une plate-forme système permettant la (semi-)automatisation du flot, la fiabilisation de la conception et la réduction de son délai de mise en œuvre.

La conception de prototypes mixtes logiciel/matériel introduit inévitablement des asynchronismes néfastes au bon fonctionnement des circuits synchrones haute performances. Nous décrivons dans le chapitre suivant notre contribution en terme d'introduction de l'outil GAUT dans la plate-forme système et en particulier la conception de nouvelles unités de communication et de mémorisation qui exploitent la théorie des systèmes insensibles à la latence afin d'optimiser la surface et de préserver les fréquences de fonctionnement des IPs synthétisés.

Chapitre 4

Introduction de GAUT dans la plate-forme système

1. Introduction

Ce chapitre est le deuxième qui décrit nos travaux. Il suit la conception des composantes matérielles et logicielles de la plate-forme de prototypage rapide PALMYRE. Afin de permettre la synthèse de haut niveau d'IPs comportementaux, nous intégrons maintenant l'outil GAUT dans la plate-forme système. Pour cela, deux contributions essentielles à cet outil sont alors proposées et développées. Ce sont :

1) l'introduction de la théorie des systèmes insensibles à la latence dans l'outil GAUT pour préserver les fréquences de fonctionnement optimales des composants synthétisés et

2) l'adaptation de l'unité de mémorisation aux nouvelles configurations des lectures-écritures en mémoire et des entrées-sorties qui résultent de la mise en œuvre et de la validation par prototypage de circuits pipelinés par GAUT.

Ce chapitre est organisé en trois sections qui présentent successivement la synthèse de haut niveau avec l'outil GAUT, l'introduction de la théorie des LIS et enfin la nouvelle unité de mémorisation.

2. GAUT

GAUT (acronyme de Générateur Automatique d'Unité de Traitement) est un environnement de synthèse d'architecture matérielle, dédié aux algorithmes de Traitement du Signal et de l'Image (TDSI) sous contrainte de cadence d'échantillonnage. A partir de la spécification d'un algorithme de TDSI en VHDL, il synthétise une description structurelle VHDL de niveau RTL optimisée en surface et destinée aux outils de synthèse logique du marché tels que ISE/Foundation de Xilinx, Quartus d'Altera ou Design Compiler de Synopsys. Cet outil universitaire résulte de travaux de recherche commencés au LASTI dans les années 1990 et poursuivis au LESTER depuis 1994 ([MAR93] et [PHI95]). Le développement informatique représente actuellement un volume de l'ordre de 100000 lignes de code C et JAVA.

Le modèle cible des architectures synthétisées par GAUT est un cœur générique de processeur dédié au traitement du signal DSP. Ce modèle est composé de trois unités fonctionnelles travaillant en parallèle : unité de traitement (UT), unité de mémorisation (UM), et unité de communication (UCOM). Une horloge commune assure le fonctionnement synchrone entre les différentes unités qui fonctionnent en mode pipeline et s'échangent les données au travers d'un réseau multi-bus parallèles.

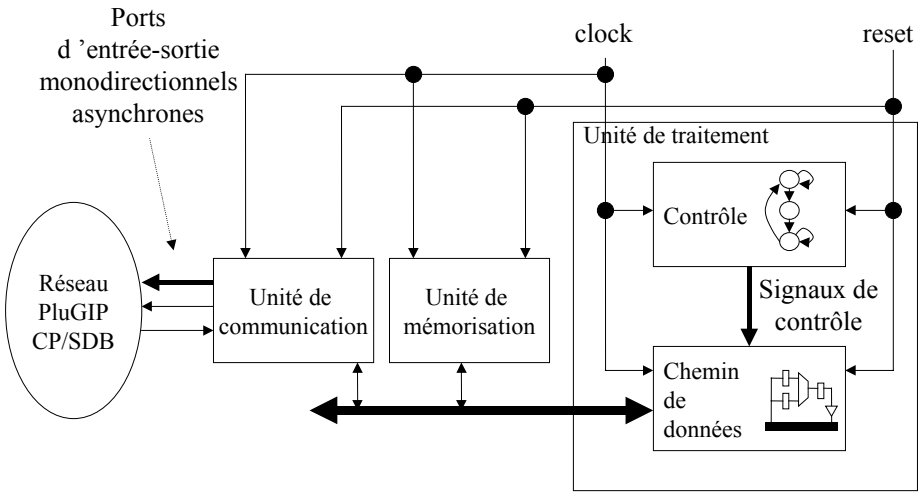


Figure 4-1 : Modèle structurel d'un circuit synthétisé par GAUT

- **Unité de Traitement**

L'unité de traitement (UT) est en charge de la partie calcul de l'algorithme. Le modèle sur lequel elle est basée s'articule autour de cellules (arithmétiques ou logiques) élémentaires. Chaque cellule est composée d'un opérateur, d'un ensemble de registres, de multiplexeurs, de démultiplexeurs et de buffers trois états. Les registres permettent le stockage temporaire des données et la synchronisation des transferts de données au sein de l'UT. Les multiplexeurs, démultiplexeurs et buffers trois états ont en charge l'aiguillage des données. Leur présence optionnelle dans une cellule résulte du partage des registres et des opérateurs, réalisé durant la phase d'optimisation du matériel. Les cellules communiquent au travers d'un réseau de bus parallèles. Les bus dédiés assurent les échanges entre les cellules alors que les bus généraux permettent les accès aux mémoires contenues dans l'unité de mémorisation. Une machine d'états finis, qui résulte de l'ordonnancement des opérations, et un décodeur d'instructions se chargent de générer les signaux de contrôle à destination des registres et buffers trois états.

- **Unité de mémorisation**

L'unité de mémorisation (UM) répond au problème de stockage des données. Elle est constituée de bancs mémoires (RAM / ROM), de registres et de générateurs d'adresses. Un générateur est une machine d'états finis dont chaque état correspond à la lecture de la valeur d'une constante ou à la lecture/écriture de la valeur d'une variable en RAM. Une thèse est actuellement en cours au LESTER [COR03A, COR03B] sur la synthèse comportementale sous contraintes de ressources mémoires.

- **Unité de Communication**

L'unité de communication UCOM est composée d'une partie opérative (Chemin de données) et d'un contrôleur [BAG98]. Le chemin de données inclut des modules de stockage des données d'entrées-sorties (FIFO, LIFO, registres), des bus (internes et externes) et des éléments d'interconnexion (multiplexeurs, démultiplexeurs et barrières trois états). Les bus externes représentent les liens physiques entre le cœur d'IP et les composants du système. Les bus

internes représentent les liens physiques avec l'unité de traitement. Le nombre est défini par la synthèse et peut être supérieur ou égal au nombre maximum d'échanges simultanés de données entre l'UCOM et l'UT. Le contrôleur réalise d'une part le protocole de communication système et d'autre part synchronise l'unité de communication et l'unité de traitement.

Dans [BAG97] l'auteur présente le flot de conception de l'UCOM à partir des contraintes d'une unité de traitement, synthétisée sous contrainte de cadence, et des contraintes systèmes dans un flot de conception matériel / logiciel. L'auteur démontre que, dans certains cas, les contraintes imposées par l'unité de traitement et le système sont trop éloignées pour être adaptées par une interface rendant ainsi la synthèse de l'UCOM impossible. Actuellement, la stratégie de synthèse accepte en entrée une spécification des contraintes temporelles sur les entrées-sorties [COU03].

Le flot de conception mis en œuvre dans GAUT est représenté par la Figure 4-2. La description algorithmique de la fonction à réaliser est décrite à l'aide d'un sous-ensemble du langage VHDL par un unique processus (couple entité / architecture). La description initiale est accompagnée d'une contrainte temporelle. Cette valeur, appelée « cadence », correspond à la plage temporelle sur laquelle s'étale l'arrivée de l'ensemble des données nécessaires pour une itération de l'algorithme. La phase de compilation effectue une analyse syntaxique, un contrôle sémantique et une parallélisation du code. Cette dernière étape réalise 1) la suppression des dépendances de contrôle (déroulage des boucles, mise en ligne des appels de procédures, parallélisation des branchements conditionnels) et 2) la suppression des fausses dépendances de données. En effet une distinction entre les vraies dépendances (producteur-consommateur) et les fausses dépendances que sont l'anti-dépendance (consommateur-producteur) et la dépendance de sortie (producteur-producteur) est faite dans l'outil GAUT. En raison de l'assignation unique la parallélisation du code se termine par un renommage des variables.

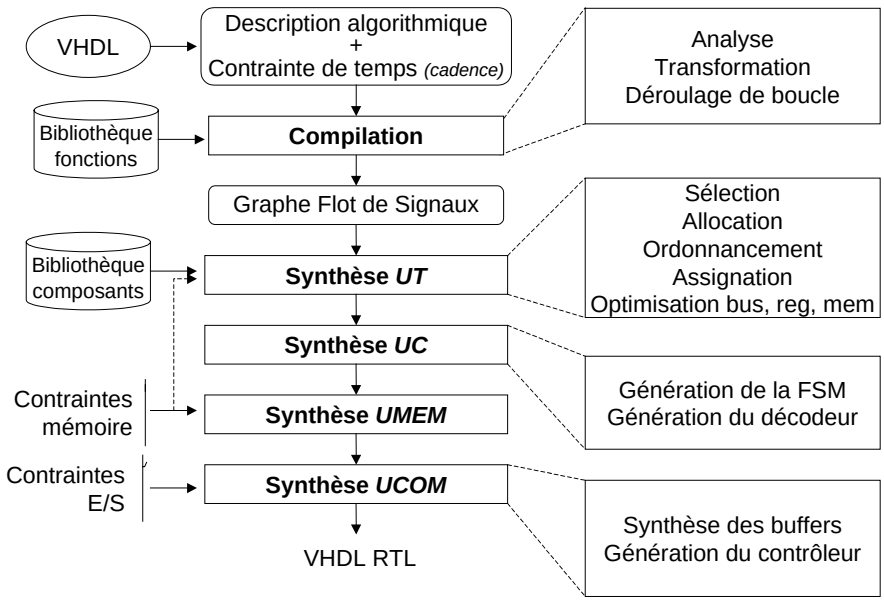


Figure 4-2: Flot de conception de l'environnement *GAUT*

La compilation produit une représentation interne de l'algorithme sous la forme d'un graphe flot de signaux (SFG) spécifié dans le format GC [INR93]. Le format GC est une représentation hiérarchisée de style flot des données ou bloc-diagramme d'un programme synchrone. Il est utilisé par les outils de programmation synchrone. Il permet l'interfaçage du cœur de synthèse de GAUT avec ces derniers, bien qu'aucune tentative n'ait été menée à ce jour. Cette modélisation permet l'expression du parallélisme de l'algorithme. Après l'étape de synthèse de l'unité de traitement, les unités de contrôle, de mémorisation et de communication sont générées. Afin de garantir la compatibilité avec le plus grand nombre d'outils de synthèse RTL du commerce, le VHDL RTL produit par GAUT est strictement conforme à la norme IEEE P1076 (Standard for VHDL Register Transfer Level Syntesis).

3. Le modèle LIS dans GAUT

Nous présentons dans cette section comment le modèle LIS s'intègre aux architectures spécifiques de circuits numériques synthétisés par GAUT. Nous présentons successivement la nature des interfaces entre IP et réseau de communication pseudo-asynchrone, la structure de nos wrappers/processeurs de synchronisation, l'implantation des qualités nécessaires de suspensibilité et d'insensibilité à la latence qui rendent nos composants équivalents à des processus patients et enfin décrivons en détail l'implantation des CFSMD micro-codées ainsi qu'une mesure des résultats. Nous justifions alors les gains en surface et en vitesse obtenus.

3.1. Spécification des interfaces

Nous présentons maintenant la nature de l'interface entre IP et réseau de communication.

3.1.1. Modèle d'interface des IPs

Les composants synchrones que nous envisageons de réutiliser sont des UT mono-horloge et suspensibles car elles doivent respecter les conditions nécessaires et suffisantes pour être intégrables dans un système LIS insensible à la latence. Leur modèle d'interface est illustré par la Figure 4-3. Deux types de modèles existent, le modèle a) originel de GAUT, et le modèle b) que nous affirmons être le seul acceptable pour des cibles de type FPGA. La justification est la suivante : le modèle b) permet d'implanter la stratégie de gel d'horloge de son choix, c'est à dire une horloge avec signal de validité pour les FPGA, ou bien une horloge combinatoire, suspensible ou avec signal de validité pour les ASIC. Le modèle d'interface que nous proposons ne dépend pas de l'implantation du gel d'horloge retenue.

Les modèles ont deux paramètres : les nombres p et q des bus respectivement d'entrée et de sortie. Une instance du modèle a) ou b) sera obtenue lors de chaque synthèse selon que l'on génére des composants synchrones suspensibles ou pas. Nous poursuivons à partir de maintenant notre exposé exclusivement avec le modèle b). Le nombre total de bus de tout circuit synthétisé par GAUT dépend du nombre de ports d'entrées-sorties, de la contrainte de cadence et de l'ordonnancement des lectures-écritures des variables et constantes internes à l'algorithme. Les bus portant des entrées et des sorties sont un sous-ensemble de ces bus.

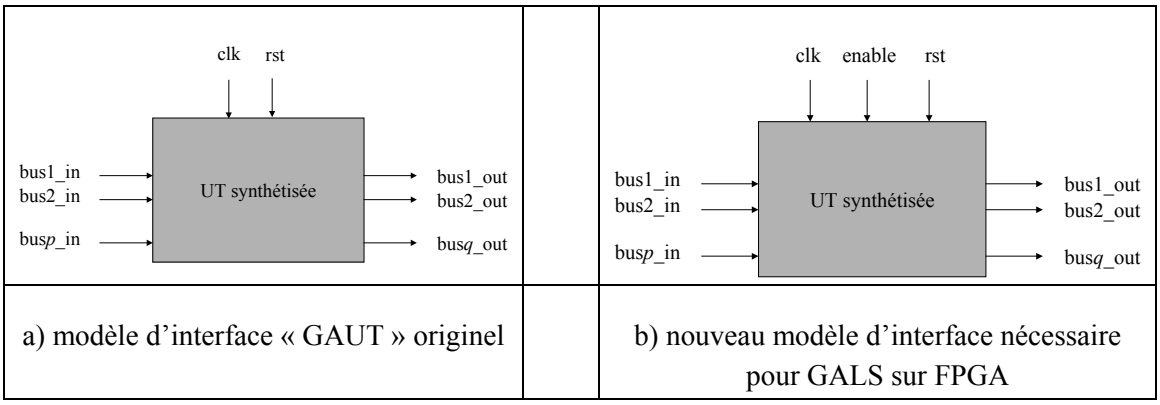


Figure 4-3 : Modèle d'interface de l'UT de GAUT

Les signaux *bus** représentent les bus par lesquels transitent les données. Lors de chaque exécution de l'algorithme que le circuit implante, le chronogramme de transit des données sur chacun de ces bus est identique. En effet, le modèle de spécification des entrées-sorties utilisé par GAUT pour décrire le comportement temporel des entrées-sorties lors de l'exécution de l'algorithme est *statique* et implicitement *répétitif*. Par *statique* nous entendons qu'il ne dépend ni du contrôle lors de l'exécution, ni des données lors de la communication mais seulement du graphe flot de signaux issu de la compilation de l'algorithme. Par *répétitif* nous entendons qu'il est implicitement convenu que les entrées-sorties sont identiques pour toutes les exécutions de l'algorithme et que le calcul est permanent : c'est à dire que les entrées sont toujours présentes lorsque le circuit consomme des données et que les sorties sont toujours disponibles lorsque le circuit produit des données.

Nous savons donc, à partir d'informations générées par GAUT (les fichiers .mem), quelles sont les configurations des entrées-sorties sensibles à chaque pas de calcul. Nous insistons sur le fait que seules les entrées/sorties, et non pas le transit des variables stockées en mémoire, nous intéressent pour déterminer la FSM de synchronisation avec le réseau de communication LIS. En effet, les bus étant partagés entre les entrées/sorties et les lectures/écritures de variables internes, ils sont réutilisés pour le transit de donnée interne lorsqu'ils sont inactifs en communication.

Les signaux synchrones *clk* et *enable* ont pour fonction d'implanter une horloge suspensible de type « clock enable ». Le signal asynchrone *rst* (actif bas) assure le reset du circuit complet.

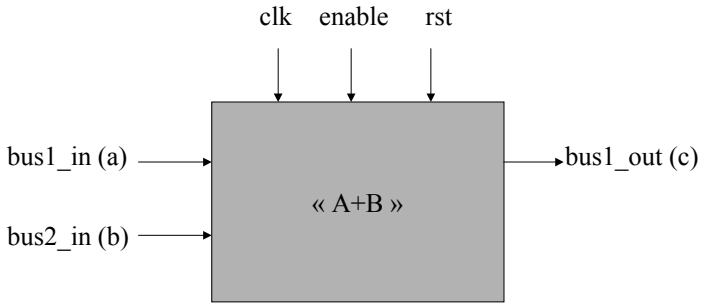
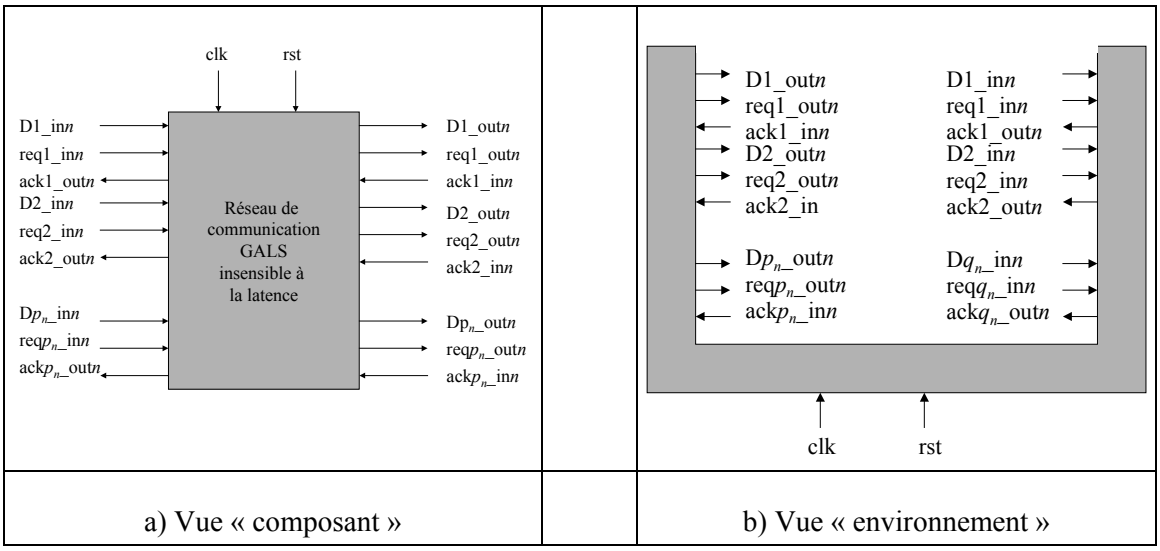


Figure 4-4 : Interface de l'UT « A+B »

A titre d'exemple, nous illustrons à la Figure 4-4 l'interface d'une UT dont le but est de calculer la somme de deux entrées (a et b) et dont les entrées-sorties sont toutes sur des bus différents : *a* sur le bus 1 en entrée, *b* sur le bus 2 en entrée et *c = (a+b)* sur le bus 1 en sortie.

3.1.2. Modèle d'interface du réseau

Le modèle d'interface du réseau de communication limité à un seul composant est illustré sur la Figure 4-5. Il est strictement conforme à la spécification des réseaux de communication pseudo-asynchrones des systèmes insensibles à la latence [CAR01]. Chaque port de communication est accompagné de signaux de synchronisation req/ack. Notons la différence de vocabulaire, nous appelons *bus* les bus synchrones du circuit synthétisé, et nous nommons *port* les ports asynchrones du réseau de communication.



a) Vue « composant » b) Vue « environnement »

Figure 4-5 : Modèle d'interface du réseau de communication

La représentation, orientée composant, est avantageusement remplacée par une représentation orientée « environnement » comme l'illustrent les vues a) « composant » et b) « environnement ». Le modèle multi-composants est défini par un nombre de composants *n*, et pour chaque composant *i* par les nombres p_i et q_i de ses ports de communication asynchrones.

Les signaux *D** représentent les ports d'entrées/sorties du réseau auxquels sont associés les paires de signaux *req** et *ack** du protocole de poignée de main asynchrone. Les signaux *clk* et *rst* implantent le reset asynchrone et l'horloge du réseau. Ces signaux sont nécessaires car le réseau de communication est en réalité synchrone, seule la latence des transits des signaux au sein du réseau est variable et qualifiée d'asynchrone car c'est un multiple de la période d'horloge. Ainsi, la latence de transit des données dans le réseau pseudo-asynchrone introduit des retards et le circuit synchrone doit attendre des données valides et des sorties libres pour pouvoir reprendre le cours de son calcul.

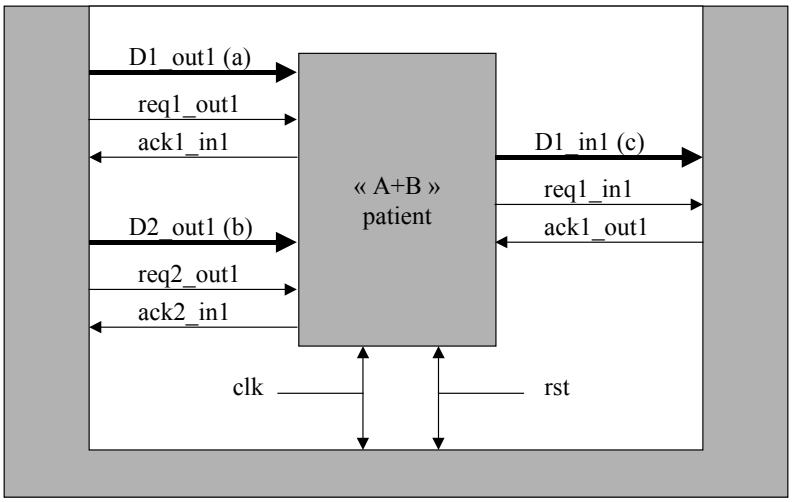


Figure 4-6 : Interface du réseau de communication avec deux circuits « A+B »

A titre d'exemple, nous illustrons à la Figure 4-6 l'interface d'un réseau pseudo-asynchrone dont le but est de véhiculer les données du composant « A+B » rendu patient. Remarquons que le composant et le réseau partagent à l'interface la même horloge. Cette horloge peut ne pas être la même pour tous les composants du système : le réseau est alors responsable d'assurer la fiabilité de la communication entre les divers domaines d'horloge.

Après avoir décrit les interfaces des composants synchrones suspensibles et du réseau de communication pseudo-asynchrone qui communique avec les processus rendus patients, il est nécessaire de présenter maintenant les wrappers LIS dont le but est de rendre les processus synchrones suspensibles « patients » et de faire ainsi la jonction en terme d'interface logique entre le composant purement synchrone et le réseau de communication.

3.2. Wrapper LIS

Le rôle du wrapper est d'interfacer le composant synchrone avec le réseau de communication et de suspendre l'exécution du composant lors de l'attente d'entrées ou de sorties. Le wrapper que nous proposons est une implantation particulière des wrappers LIS de [CAR01]. Il garantit par construction l'équivalence sémantique de l'implantation LIS avec le modèle synchrone du système complet.

3.2.1. Modèle d'interface

Le wrapper se situe entre le réseau et le composant comme l'illustre la Figure 4-7. Son modèle d'interface est la réunion des modèles d'interface du composant et du réseau limité au composant. Pour des raisons de simplicité, les signaux d'horloge et de reset sont maintenant omis. Le signal de sortie *enable* est produit par le wrapper et a pour but d'autoriser ou de suspendre l'exécution du composant synchrone. Ce signal s'applique directement au signal d'entrée *enable* du composant.

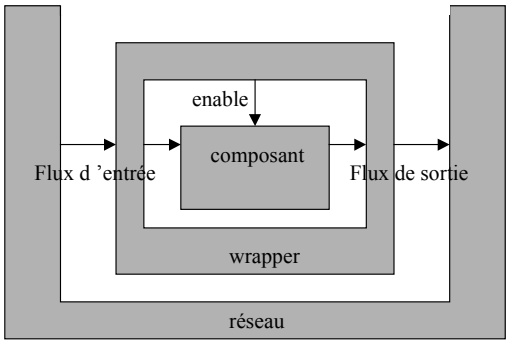


Figure 4-7 : Position du wrapper

Toujours à titre d'exemple, la Figure 4-8 illustre le composant « A+B » encapsulé et intégré au réseau de communication.

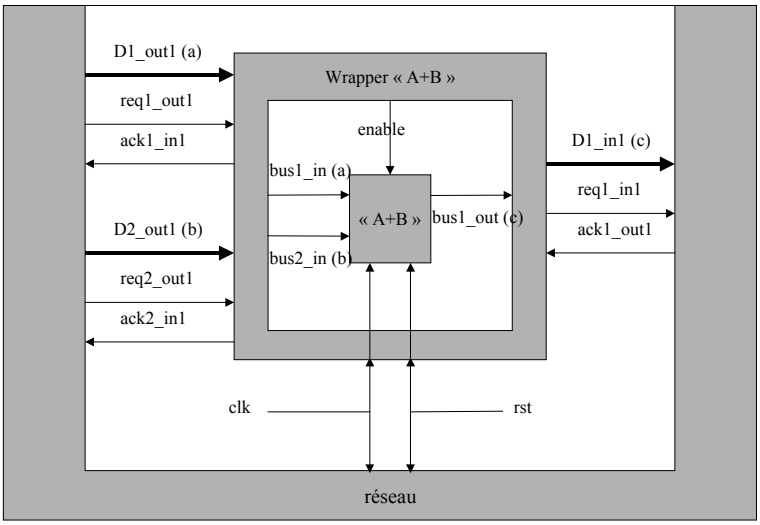


Figure 4-8 : Encapsulation LIS complète de « A+B »

3.2.2. Modèle structurel

La structure du wrapper est illustrée par la Figure 4-9. Il assure trois fonctions: adaptation de protocole, découplage et synchronisation.

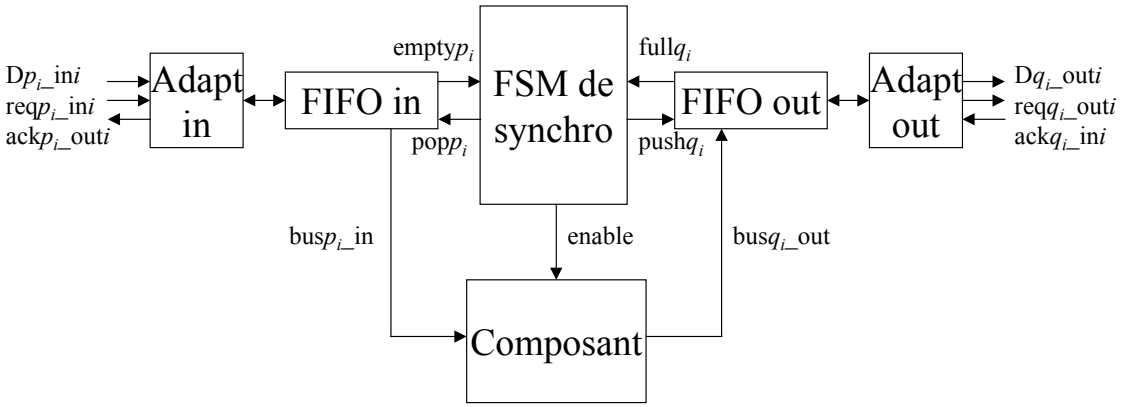


Figure 4-9 : Modèle structurel du wrapper

Il est constitué d'une FSM de synchronisation, de FIFOs de découplage des entrée et des sorties et d'adaptateurs de protocole aux extrémités. La correspondance entre le formalisme LIS et notre implantation est la suivante :

1) Notre *composant* est le processus suspensible « perle » (pearl) encapsulé dans un wrapper. Cette encapsulation lui confère alors la propriété d'être un processus patient composable avec d'autres dans un même système LIS.

2) La FSM de synchronisation représente les fonctions égaliseur et générateur d'états d'attente. Les états d'attente sont représentés par l'état bas du signal *enable* qui pilote le composant synchrone. Lorsque *enable* est à l'état haut, le circuit et la FSM avancent d'un état à chaque coup d'horloge jusqu'à ce que la FSM passe dans un nouvel état d'attente d'entrées présentes et/ou de sorties libres. Lorsque *enable* est à l'état bas, le composant est suspendu dans le dernier état de calcul.

3) Les signaux d'entrée sont les signaux de contrôle des FIFO d'entrée et les signaux d'adaptation au protocole asynchrone req/ack de notre réseau pseudo-asynchrone. Ce sont :

   a) busp_i_in et Dp_i_in$_i$ sont équivalents à DataIn

   b) emptyp_i est équivalent à voidIn, et reqp_i_in$_i$ à (non voidIn)

   c) popp_i et ackp_i_out$_i$ sont équivalents à (non stopOut)

4) Les signaux de sortie sont les signaux de contrôle des FIFO de sortie et les signaux du réseau. Ce sont :

   a) busq_i_out et Dp_i_out$_i$ sont équivalents à DataOut

   b) reqp_i_out$_i$ est équivalent à (non voidOut)

   c) fullq_i est équivalent à stopIn et ackp_i_in$_i$ à (non stopIn)

Nous concluons que le couple (wrapper, composant), au renommage près des signaux d'interface, est strictement équivalent à un processus patient.

Remarquons de plus que l'adjonction de FIFOs dans le wrapper est fonctionnellement équivalente à l'insertion des stations de relais (relay stations) sur les grandes lignes du modèle d'implantation LIS. Les stations de relais représentent un pipeline de stockage de données dont la profondeur est déterminée par le nombre de registres insérés dans le chemin et ne sont pas réglables en profondeur. Nos FIFOs de profondeur deux sont strictement équivalentes aux stations de relais à deux places et ont les mêmes performances : le débit maximal est supporté car une entrée et une sortie peuvent avoir lieu simultanément à chaque cycle. Les FIFOs ont, de plus, l'avantage de permettre une plus grande extensibilité : dans notre contribution la profondeur des FIFOs est un paramètre réglable lors de la synthèse du wrapper.

3.2.3. Modèle de machine d'états finis

Le modèle fonctionnel du wrapper concerne la FSM de synchronisation. Il s'agit d'une CFSMD (Concurrent Finite State Machine with Data path). Notre CFSMD est la combinaison d'une FSMD (Finite State Machine with Dapapath) et d'une HCFSM (Hierarchical and Concurrent Finite State Machine) dont nous n'exploitons pas l'aspect hiérarchique. Afin de définir notre modèle de CFSMD nous présentons successivement les FSMD et HCFSM, ainsi que notre CFSMD.

Les machines d'états finis, ou FSM, font partie de la classe des modèles de calcul. Nous les nommons aussi modèle d'exécution car ils ont pour but de spécifier un comportement. Ces modèles se différencient par leurs granularités (processus, objet), leurs mécanismes de communication inter tâches (passage de message, appel de procédure distante RPC, variables partagées), leurs mécanismes de synchronisation (FIFO, poignée de main), leurs orientations (vérification formelle, synthèse, simulation), leurs domaines d'application, mais aussi leurs supports de la hiérarchie et leurs notions de temps. Toutes ces propriétés sont exprimées de façon implicite ou explicite et reflètent les caractéristiques du système modélisé. Il y a d'ailleurs diverses façons de classifier les langages de spécification système selon que l'on s'intéresse à leur syntaxe [GAJ94] ou aux modèles de calcul sous-jacents [JER97b]. Dans ce cadre, les FSMs sont des automates dont les sorties dépendent des entrées et d'un état courant. On les classe en deux catégories dites de Mealy ou de Moore selon que leurs sorties dépendent des entrées ou non. Elles sont composées de deux éléments : des nœuds et des arcs. Les nœuds représentent les états du système et les arcs les transitions. Ces derniers sont étiquetés par des conditions de transition appelées *gardes* et par des *actions* qui représentent la génération des sorties. Le nombre d'états croit de façon exponentielle proportionnellement à la taille du système à modéliser et leur nature synchrone n'autorise pas la description aisée d'un système dans lequel les composants évoluent en parallèle puisque à tout instant le système est représenté par un unique état. Les FSM ont enfin l'avantage d'être très proches d'une implantation matérielle, [STA97] les qualifie d'ailleurs de « formes intermédiaires orientées architectures ».

Les FSMDs, inroduites par [GAJ92] sont des machines d'états finis étendues à l'aide d'opérations sur des données (des variables internes). La FSMD est une FSM à laquelle on a ajouté un chemin de donnée (variables + opérations).

Les HCFSMs sont une extension des FSMs selon deux axes qui sont soit structurel, soit relatif à la communication. L'axe structurel permet l'adjonction de hiérarchie : un état d'une FSM peut représenter une FSM de niveau inférieur. L'axe communication permet à plusieurs FSMs de communiquer et de se synchroniser entre-elles. Ainsi, l'état global du système est-il la combinaison des états simultanés des différentes FSMs concurrentes et permet d'exprimer (pour spécifier un comportement tout comme pour décrire une implantation) un système très complexe en nombre total d'état (le produit cartésien des ensembles d'états de chacune des FSMs) par un ensemble de FSMs de moindre complexité. Les HCFSMs ont été introduites par le formalisme Statecharts [HAR87] et de nombreuses variations existent [BEE94] ainsi que le langage Argos [MRN92]. Les HCFSMs sont très bien adaptées aux calculs orientés contrôle.

Enfin, notre FSM est concurrente avec la FSM de contrôle du composant encapsulé d'où le « C » de CFSMD. Elle synchronise aussi le transit des données dans le chemin de donnée constitué par les FIFOs et le composant d'où le « D » de CFSMD.

3.2.4. Conception du contrôle

Le contrôle que doit assurer le wrapper dépend de la nature du scénario des entrées-sorties attendues par le circuit synthétisé. Nous présentons comment passer du chronogramme des entrées-sorties produit par GAUT à une spécification exprimée sous la forme d'une séquence d'opérations. Cette séquence est assimilable à la séquence d'instructions d'un processeur qui exécute un programme perpétuellement répétitif. C'est pour cette raison que nous nommons notre wrapper un *processeur de synchronisation*.

- **Chronogramme des entrées-sorties**

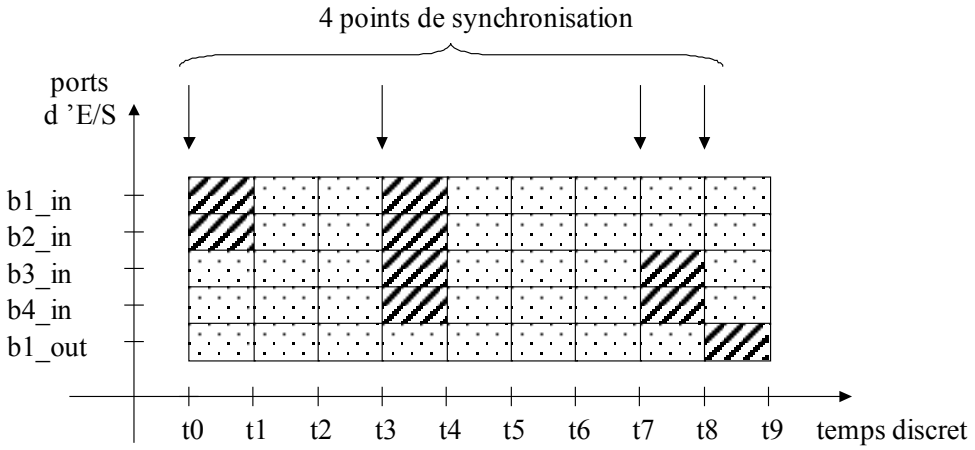


Figure 4-10 : Exemple de chronogramme avec points de synchronisation

La Figure 4-10 illustre le chronogramme d'une UT pour laquelle la cadence est de neuf cycles. Ces neuf neufs cycles sont de deux catégories : les cycles de synchronisation-et-de-calcul (notés « cycles de synchronisation » par la suite) et les cycles de calcul seuls. Alors que les cycles de calcul n'ont besoin ni d'entrées ni de sorties pour être actifs, les cycles de synchronisation ont besoin d'entrées disponibles et/ou de sortie libres pour pouvoir être activés. Nous nommons « point de synchronisation » l'instant qui se situe entre un cycle de calcul et un cycle de synchronisation. Cet instant est totalement caractérisé par 1) sa position dans le chronogramme et 2) la liste des ports d'entrée et/ou de sortie nécessaires pour avancer d'un état. Notre exemple est un circuit synchrone ayant quatre bus d'entrés (bus1_in à bus4_in) et un bus de sortie (bus1_out). Chacun des bus est agrémenté d'un signal qui permet de savoir si le port d'entrée associé est porteur d'une entrée et si le port de sortie est près à recevoir une sortie. Le nombre de points de synchronisation est de quatre.

- **Dérivation de la FSM de synchronisation**

La FSM associée à ce chronogramme est automatiquement dérivée sous la forme d'un graphe états/transitions à l'aide des règles suivantes :

1) Si le cycle est un cycle de calcul, créer un état à partir duquel il n'y a qu'une seule transition avec le signal *enable* actif. Relier la transition de l'état précédent à celui ci.

2) Si le cycle est un cycle de synchronisation, créer un état pour lequel il y a deux transitions possibles :

   a) Une transition d'attente (rebouclage sur l'état) dont la condition est la disjonction des négations des signaux de validité des ports et pour laquelle le signal *enable* est inactif

   b) Une transition vers l'état suivant dont la condition est la négation de la précédente et pour laquelle le signal *enable* est actif.

3) Enfin connecter la transition issue du dernier état au premier.

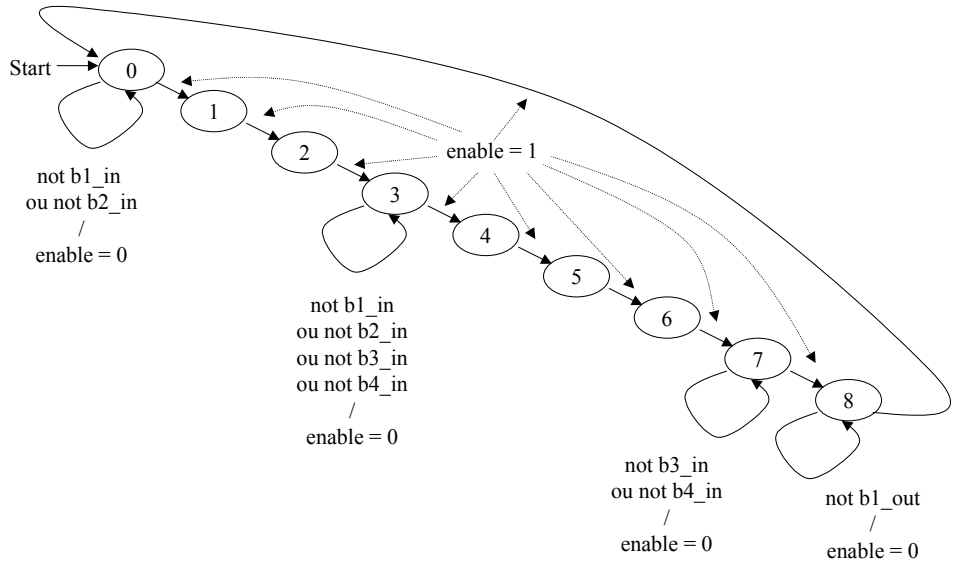


Figure 4-11 : CFSM[D] avec synchronisation

La Figure 4-11 décrit cette FSM dans le cadre de l'exemple de la Figure 4-10. Remarquons que son nombre d'états est égal au nombre de cycles de la cadence du circuit synchrone encapsulé. Enfin, cette FSM est concurrente (signal *enable*) car sont exécution se fait en parallèle avec celle de l'UT, mais pas encore de CFSMD puisqu'elle n'a pas de variables internes, ni ne pilote les FIFOS du wrapper. Lorsque la sortie *enable* est active, la FSM de synchronisation et celle du composant (concurrence) avancent de façon synchrone avec leur horloge commune, sinon le composant est suspendu dans son état courant. Dans la mesure où nos FSMs sont linéaires (exception faite des états d'attente), la description sous la forme d'un graphe peut-être remplacée par une description textuelle de type liste. Nous proposons la représentation suivante de la FSM.

- **Représentation textuelle équivalente de la FSM de synchronisation**

Ce formalisme textuel à l'avantage de faire apparaître simplement les informations dont nous avons besoin pour le chemin de données de notre CFSMD. Il consiste en une liste d'états qui représentent soit des cycles de synchronisation, soit des cycles de calcul. Un cycle de synchronisation est un état potentiel d'attente, un cycle de calcul est un cycle non dépendant des entrées-sorties, donc un cycle qui n'a aucune raison d'attendre quoi que ce soit. On associe à chaque état la spécification de la liste des ports/FIFOS d'entrées qui doivent être non vides et la liste des ports/FIFO de sorties qui doivent être non pleines. Les cycles de synchronisation ont des listes d'entrées-sorties non toutes vides, alors que les états de calcul ont des listes d'entrées-sorties toutes vides. Notre FSM est totalement spécifiée par une liste de couples *(Ei, Si)* où *Ei* représente un sous-ensemble des ports/FIFO d'entrée et *Si* représente un sous-ensemble des ports/FIFO de sortie à l'instant *i*.

Dans le cas de notre circuit hypothétique, nous obtenons la liste suivante :

```
L = ( ( {b1_in, b2_in}, Ø ),
      ( Ø, Ø),
      ( Ø, Ø),
      ( {b1_in, b2_in, b3_in, b4_in}, Ø),
      ( Ø, Ø),
      ( Ø, Ø),
      ( Ø, Ø),
      ( {b3_in, b4_in}, Ø),
      ( Ø, {b1_out})
    )
```

Figure 4-12 : Exemple d'une liste de spécification de la synchronisation

La liste L contient neuf états, c'est à dire autant d'états que de cycles de cadence. Chaque état est représenté par un couple constitué de la spécification des ports d'entrées et de la spécification des ports de sortie. Les états de synchronisation (premier, quatrième, huitième et neuvième couples) ont des listes non toutes vides. Les états de calcul (tous les autres couples) ont leurs listes de ports vides. Enfin, le signal *enable* est implicitement inactif lorsque la FSM est en phase d'attente sur un état de synchronisation, et actif dans tous les autres cas. C'est à partir de ce modèle textuel que notre CFSMD sera définie.

3.2.5. Indépendance vis à vis de l'implantation du « gel d'horloge »

Notre modèle de composant synchrone permet de masquer (à l'aide du signal *enable*) la méthode d'implantation du gel d'horloge nécessaire pour suspendre le composant synchrone. Ce point architectural est important car il permet d'implanter un SoC LIS sur tout type de FPGA, alors que l'approche « horloge combinatoire » ne le permet pas.

Le Tableau 4-1 liste le nombre d'arbres d'horloge pré-routés dans les FPGA les plus communs des sociétés Xilinx (le leader du marché) et Altera (son principal concurrent). Il prouve que l'intégration d'un grand nombre de composants synchrones ayant chacun leur horloge est

impossible sur FPGA si l'on requiert une horloge combinatoire pour chacun d'entre eux comme le préconise la théorie des LIS.

Fabriquant	Modèle	Nombre maximum de d'arbres d'horloge
Xilinx	Virtex II Pro X, Virtex II Pro, Virtex II	16
Xilinx	Virtex-E extended memory, Virtex E, Virtex	4
Xilinx	Spartan 3	8
Xilinx	Spartan-IIE, Spartan-II, Spartan/XL	4
Altera	Stratix II, Stratix GX, Stratix	48
Altera	Cyclone	4
Altera	ACEX	1

Tableau 4-1 : Nombre de domaines d'horloges des FPGA

Il est essentiel que le modèle de wrapper de synchronisation ne dépende pas de la technique employée pour le « gel d'horloge » afin d'être indépendant de la cible technologique. Pour cela, le masquage du gel d'horloge, nous permet de modéliser des systèmes à base de composants synchrones qui ciblent aussi bien une technologie FPGA qu'une technologie ASIC comme l'illustre la Figure 4-13.

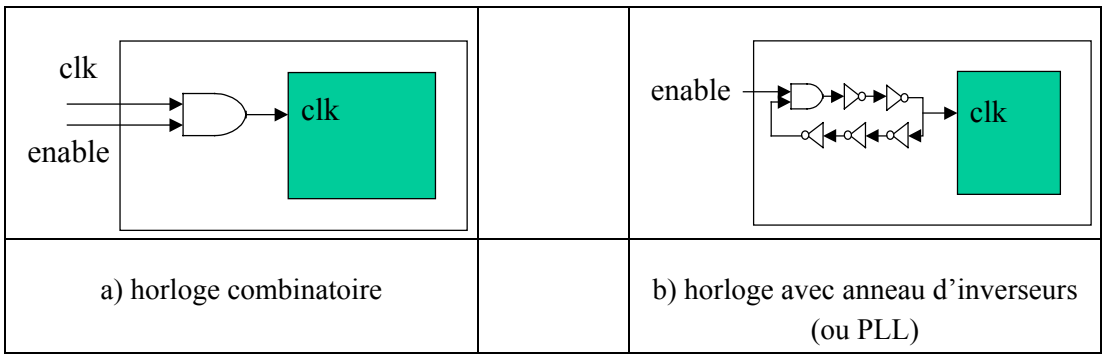


Figure 4-13 : Réalisation d'un composant synchrone avec validation d'horloge à partir d'autres modèles de gel d'horloge

3.3. Construction des processus patients

L'introduction de la théorie des LIS dans GAUT implique 1) rendre les composants suspensibles par implantation d'un mécanisme d'horloge validé (clock enable), 2) synthèse d'un wrapper d'interface avec le réseau de communication pseudo-asynchrone et 3) génération des « protocoles d'extrémités ».

3.3.1. Suspensibilité

Le modèle structurel originel d'un circuit synthétisé par GAUT contenant un chemin de données et son contrôleur est illustré par la Figure 4-14. Contrôle et chemin de données sont synchrones et ont en entrée les signaux *clock* et *reset*. L'interface du composant est constituée par les bus d'entrée et de sortie. Les signaux inter contrôle et chemin sont les signaux de contrôle des registres, multiplexeurs, démultiplexeurs et buffers trois états qui se trouvent dans l'UT pour implanter le chemin de donnée. Le sujet n'étant pas la synthèse d'architecture, nous ne rentrerons pas dans les détails de l'UT. Les bus d'entrée et de sortie sont en réalité fusionnés en un unique ensemble de bus bidirectionnels qui réduit d'autant le nombre de bus d'interface.

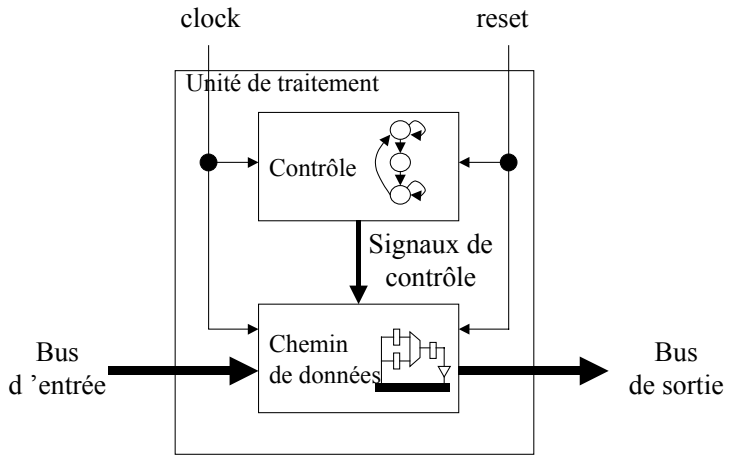


Figure 4-14 : Modèle structurel originel d'un circuit GAUT (UC+UT)

Cette représentation fusionnée ne provoque pas de conflits de données sur les bus car l'ordonnancement des entrées/sorties du circuit en tient compte et garantit qu'à chaque cycle un sous-ensemble (éventuellement vide) des bus est en entrée et un autre sous-ensemble disjoint (éventuellement vide lui aussi) est en sortie. La communication sur les bus est du type TDM (Time Division Multiplex) statique : l'allocation des « time-slots » est faite une fois pour toute et il n'y a jamais plus d'une donnée en entrée ou en sortie sur un bus quelconque.

Rentrons un peu plus dans les détails. Le circuit complet synthétisé par GAUT contient une unité mémoire (UM) et une unité de communication (UCOM). Sa structure est illustrée par la Figure 4-1. L'UM assure le transit des données internes à l'algorithme : constantes, variables, temporaires. Elle gère 1) le vieillissement des fenêtres d'échantillons « à la DSP » grâce à des buffers circulaires, 2) la duplication de variables de même nom dont les durées de vie se recouvrent et 3) la gestion dynamique des adresses en cas de pipelining de l'architecture. L'architecture de l'UM est un ensemble de bancs mémoires reliés aux ports de l'UT via des

multiplexeurs. Chacun des bancs mémoire dispose de sa FSM qui ordonne les accès mémoire (lecture/écriture, adresse) en fonction des time slots auxquels sont affectées les variables en mémoire. L'UCOM est en charge de 1) l'interfaçage de l'UT, circuit synchrone, avec le monde extérieur à l'aide de divers protocoles spécifiques (PlugIP, CP, SDB), 2) la fonction d'aiguillage des entrées-sorties physiques vers les bus synchrones de l'UT et 3) la gestion du pipeline en cas d'architecture pipelinée. L'implantation actuelle de l'UCOM ne contient pas le multiplexage des entrées-sorties sur les canaux physiques, les travaux sont en cours dans le cadre d'une thèse CIFRE avec la société STMicroelectronics. PlugIP est un produit de la société TNI-Valiosys dont le rôle est de synthétiser une interface de communication entre un bus (AMBA, ST-BUS) et un circuit synchrone. Les protocoles CP et SDB sont ceux mis en œuvre par la plate-forme de prototypage rapide PALMYRE.

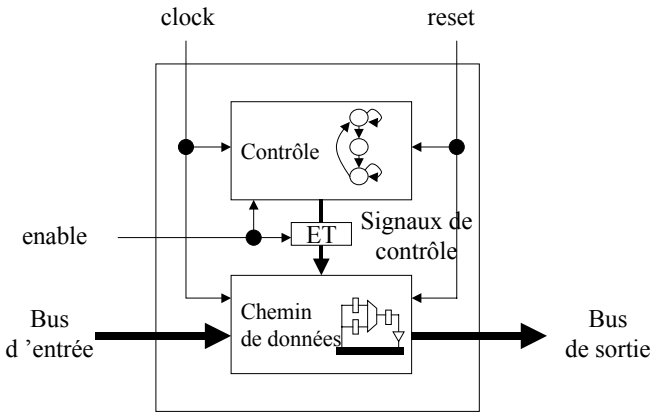


Figure 4-15 : Modèle structurel d'un circuit GAUT suspensible

Le modèle structurel du circuit que nous proposons pour implanter la suspensibilité est illustré par la Figure 4-15. Ce modèle possède un signal d'entrée supplémentaire comme requis par le modèle d'interface des composants suspensibles : le signal *enable*. Ce signal *enable* est distribué au contrôle de l'UT pour conditionner l'avancement d'un état de la machine d'états finis seulement s'il est actif et utilisé en conjonction avec chacun des signaux de contrôle de chargement des registres et d'autorisation de passage pour les buffers trois états.

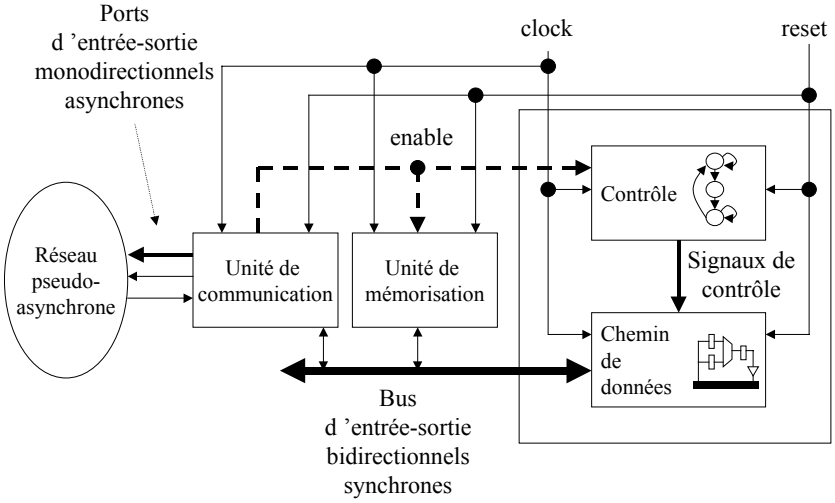


Figure 4-16 : Modèle structurel d'un circuit GAUT complet suspensible

La représentation VHDL du contrôle est une machine d'états finis dans laquelle le « case » des états est conditionné par le signal *enable*. La structure du circuit complet est illustrée par la Figure 4-16. L'impact de l'ajout de la suspensibilité se traduit par le signal de retour *enable* de l'UCOM vers l'UM et l'UT (en tirés gras sur la figure).

3.3.2. Insensibilité à la latence

Le modèle d'interface du réseau LIS et notre contribution en terme de wrapper à base de processeur de synchronisation nous ont amené à créer un nouveau modèle de composant généré par GAUT. La Figure 4-17 illustre le modèle structurel de ce nouveau composant. Il contient deux composants : le circuit suspensible (CS) et le wrapper qui se substitue à l'UCOM originelle. Le wrapper/UCOM assure l'interface entre les ports asynchrones d'entrée et de sortie différenciés (comme le requiert le modèle LIS du réseau de communication) et les bus bidirectionnels du CS. Le wrapper est responsable de la génération du signal *enable* de suspension du CS. Le CS contient l'UT et l'UM qui sont tous deux suspensibles avec le signal *enable*.

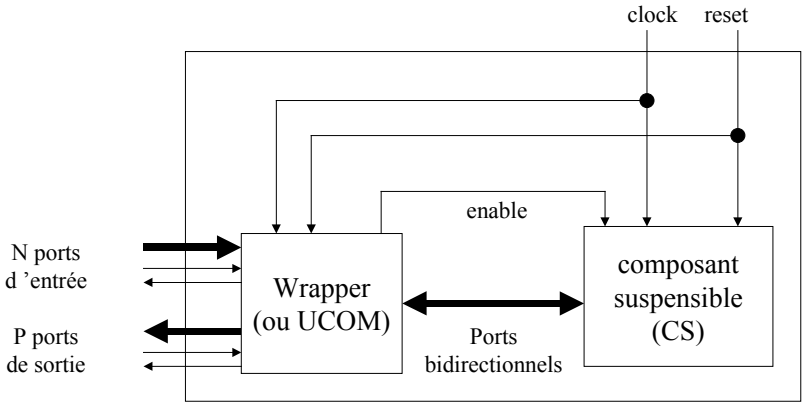


Figure 4-17 : Modèle structurel d'un circuit GAUT insensible à la latence

3.3.3. Synthèse d'interfaces d'extrémités

Notre système, qu'il soit composé d'un ou de plusieurs composants insensibles à la latence reliés par un réseau de communication pseudo-asynchrone, doit avoir un certain nombre de ses ports d'entrée et de sortie connectés à l'environnement. Il est donc nécessaire d'adapter le protocole asynchrone du réseau à celui des interfaces physiques externes.

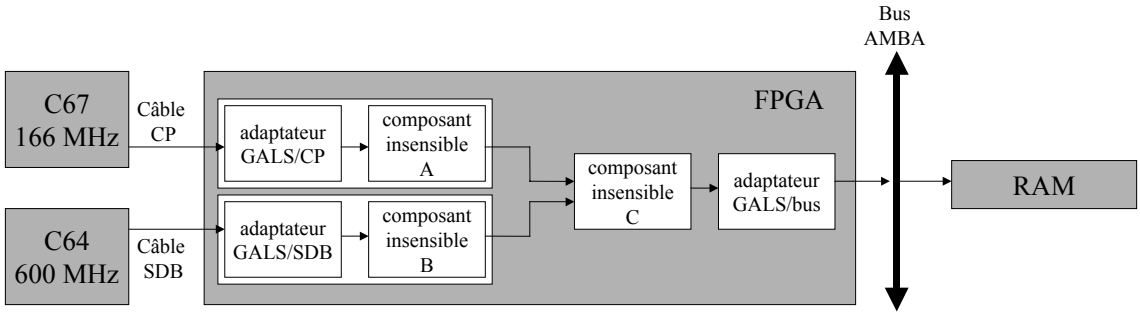


Figure 4-18 : Système avec adaptateurs de protocoles

Dans le cadre de notre plate-forme de prototypage rapide nous avons retenu trois protocoles qui sont : Plug-IP un outil de la société TNI-Valiosys [TNI04] qui permet d'interfacer un cicuit (via un bus AMBA ou ST-bus) avec une RAM externe et les liaisons point à point bidirectionnelles CP et les SDB de la société Sundance. La Figure 4-18 illustre un tel système et la Figure 4-19 donne un exemple d'architecture physique, utilisant tous les types d'interfaces externes, sur laquelle ce système est exécuté. Le système est constitué de trois composants insensibles à la latence (A, B, et C) embarqués dans un FPGA. Ces composants communiquent naturellement entre eux, via le réseau de communication pseudo-asynchrone, avec deux DSP via les protocoles CP et SDB et avec une RAM via un bus AMBA.

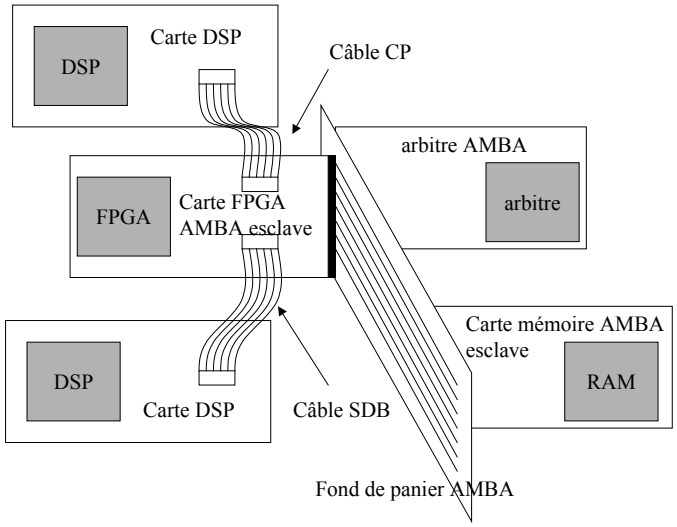


Figure 4-19 : Architecture physique du prototype

Les adaptateurs de protocole sont, soit simplement aboutés aux ports d'entrée ou de sortie dont ils assurent la conversion (cas du bus AMBA), soit directement intégrés au wrapper du composant suspensible afin de réduire le nombre de protocoles (cas des CP et SDB). Pour cela nous avons développé deux modèles d'adaptateurs de protocoles (CP et SDB) et utilisé un outil de génération automatique d'interface avec un bus (AMBA ou ST-bus).

- **Le générateur d'interface avec un bus**

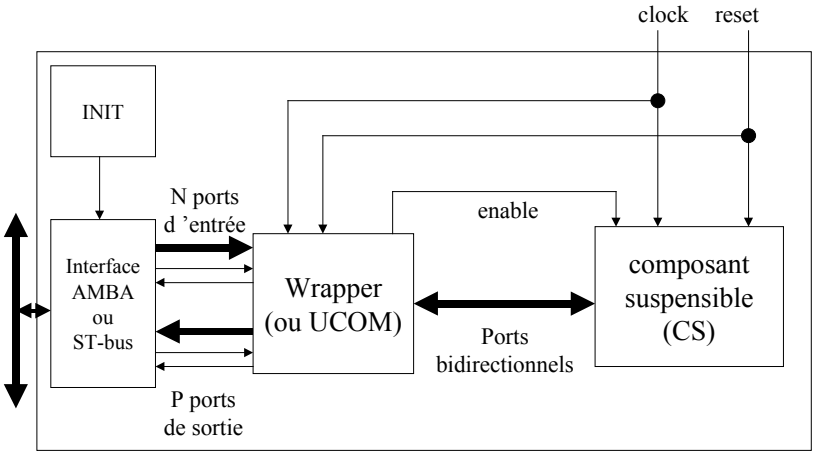


Figure 4-20 : Adaptation de protocole avec PlugIP

L'outil PlugIP permet de spécifier, de générer, puis de vérifier par (co)simulation tout système constitué de deux composants qui communiquent via un bus AMBA ou un bus ST-bus. Nous avons retenu cette solution dans le cadre du projet ALIPTA et avons spécifié avec la société TNI la nature de l'interface logique entre nos composants suspensibles et les blocs synthétisés par PlugIP. Après de nombreuses (quatre itérations en six mois) présentations, comparaisons et discussions, l'interface du réseau LIS « pseudo-asynchrone » (req/ack + données) a été finalement retenue pour des raisons de simplicité (peu de signaux) et de généricité (nombre de bits des données). Dans ces conditions, ni la structure interne, ni le protocole du réseau n'ont du être modifié dans nos composants. Nous avons seulement adjoint à GAUT la synthèse automatique du code RTL d'initialisation du bloc « interface AMBA ou ST-bus » nécessité par l'outil PlugIP au démarrage du système.

- **Les modèles d'adaptateurs**

Les modèles d'adaptateurs de protocoles sont des variations du protocole standard du réseau pseudo-asynchrone. Ils concernent actuellement les protocoles CP et SDB qui sont tous deux orientés FIFO avec signaux de contrôle de flux. D'autres protocoles, en fonction de l'évolution future des besoins, pourront être insérés de la même façon.

Nous avons donc ajouté à la génération de l'UCOM de GAUT (le wrapper LIS) la possibilité de spécifier port par port le type du protocole de communication. Il est ainsi possible d'affecter à un port quelconque le protocole CP ou SDB en remplacement du protocole req/ack du réseau pseudo-asynchrone. Ainsi le concepteur de circuits « spécifie » les contraintes sur les protocoles des seules extrémités en contact avec l'environnement du FPGA, et laisse à GAUT les détails de synthèse RTL de tous les protocoles supportés. La Figure 4-21 illustre une configuration de wrapper multi-protocoles pour lequel les protocoles CP, SDB et pseudo-asynchrone sont mis en œuvre dans un même circuit.

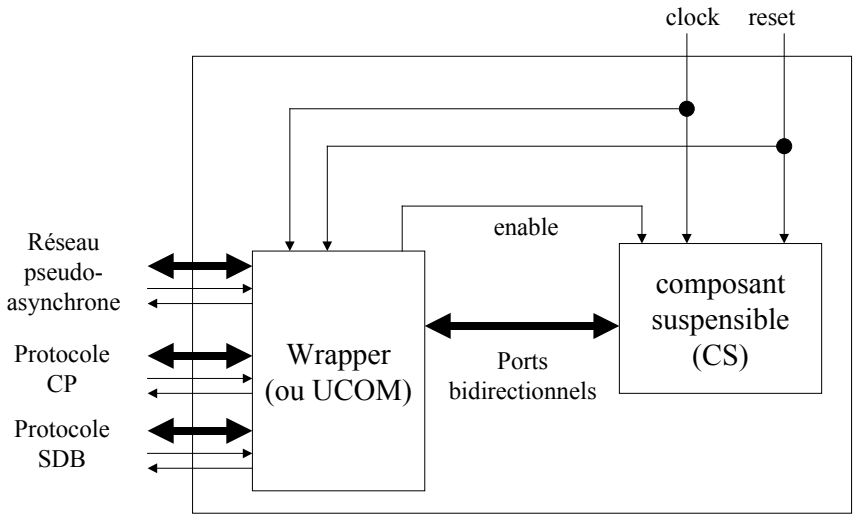


Figure 4-21 : Interface du wrapper multi-protocoles

Nous présentons maintenant les protocoles CP et SDB en détail.

Le protocole CP [SUN99a] est asynchrone de type poignée de main (quatre phases par donnée transmise) et de largeur 8 bits. La vitesse maximale est de 20 Mo/s. Le bus CP est constitué de douze signaux : NCACK et NCREQ pour le retournement de la liaison (émetteur et récepteur permutent), NCSTRB et NCRDY représentent les signaux du protocole de poignée de main, et les huit bits restant sont la donnée transmise. Le protocole à quatre phase req/ack est supporté par les signaux NCSTRB actif bas et NCRDY actif bas. La Figure 4-22 illustre la transmission de deux mots de trente-deux bits (0x00000001 et 0x00000002) octets de poids faibles en premiers.

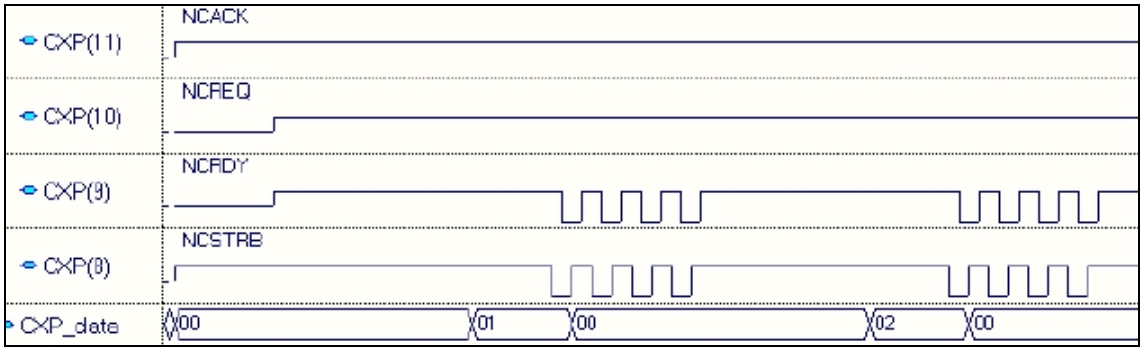


Figure 4-22 : Protocole CP

Le protocole SDB [SUN99b] est synchrone avec signaux de contrôle de flux. L'émetteur pilote l'horloge ainsi qu'un signal de validité des données. Le récepteur peut suspendre ou autoriser l'émission par assertion ou négation d'un signal d'arrêt de la transmission. Le bus SDB est constitué de dix-neuf signaux : seize signaux de donnée, CLK_IO le signal d'horloge de l'émetteur, ENABLE le signal de validité des données, REQ et ACK les signaux de retournement de la liaison (émetteur et récepteur permutent), enfin ACK employé seul par le récepteur est utilisé pour le contrôle de flux et stoppe l'émetteur lorsqu'il est à l'état haut.

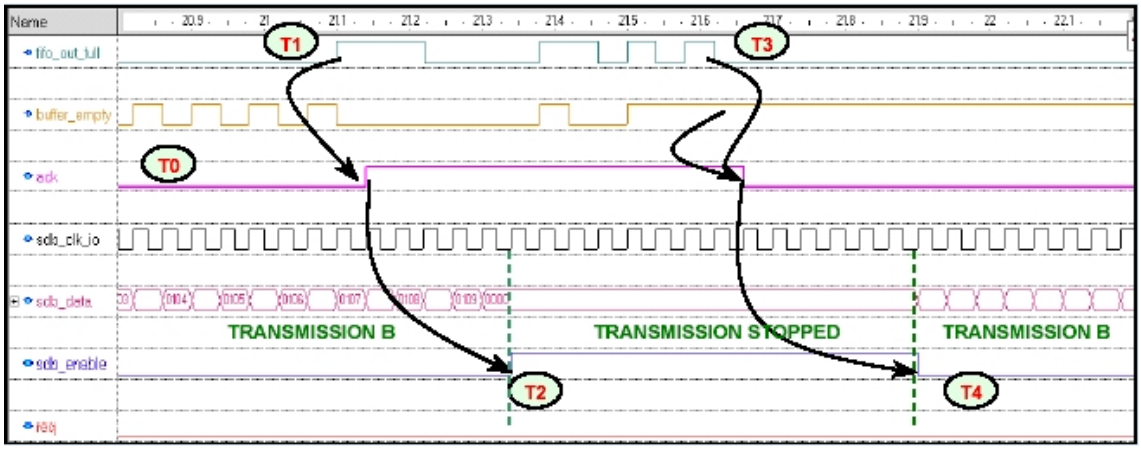


Figure 4-23 : Protocole SDB

Les modèles d'adaptateurs de protocoles pour les CP et les SDB ont été developpés conjointement par l'auteur et la société Sundance. Ils sont composés de deux parties : l'interface de communication externe (Sundance) et la FSM d'adaptation qui assure le transit des données entre la FIFO et l'interface comme l'illustre la Figure 4-24.

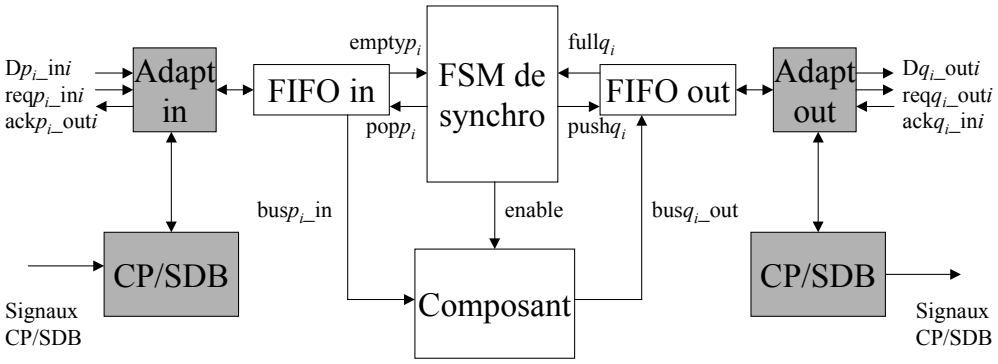


Figure 4-24 : Structure du wrapper multi-protocoles

3.4. Implantation du processeur de synchronisation

Dans cette section nous présentons notre modèle d'implantation CFSMD de la FSM de synchronisation des wrappers multi-protocoles. Nous nommons, par analogie avec un processeur qui exécute un programme, ce dernier un *processeur de synchronisation*.

3.4.1. Justification du besoin

```
process FSM(clk, rst)
  if rst = '1' then
    state := S0 ;
    enable <= '0' ;
  elsif clk'event and clk = '1' then
    case state

    when <a synchronization state> =>
      if  <check_data> and <check_room> then
        state := <nextstate> ;
        enable <= '1' ;
        <pop_push_fifos> <= '1' ;
      else
        enable <= '0' ;
        <pop_push_fifos> <= '0' ;
      endif ;

    when <a computing state>
      state := <nextstate> ;
      enable <= '1' ;
      <pop_push_fifos> <= '0' ;
    end case ;

  endif ;
end process ;
```

Figure 4-25 : Implantation naïve de la FSM en pseudo-VHDL

Nous rappelons que nous disposons d'un modèle d'implantation de notre wrapper (voir section 3.2.4) sous la forme d'une FSM spécifiée par une liste d'états de synchronisation et de calcul et d'une sémantique d'attente associée. Sa génération automatique est triviale. Elle peut être représentée par une spécification générique en VHDL comme l'illustre la Figure 4-25. La sémantique des différents éléments de cette implantation VHDL est la suivante : *state* est la variable d'état qui contient l'identification de l'état courant, *check_data* représente la conjonction des signaux d'état « non vides » des FIFO d'entrées, *check_room* représente la conjonction des signaux d'état « non pleines » des FIFO de sorties, *pop_push_fifos* représente l'activation des signaux de « pop » des FIFO d'entrées et de « push » des FIFO de sorties.

> *Nous affirmons que ce modèle HDL (VHDL ou verilog, peu importe) de FSM, bien que très simple, n'est pas synthétisable pour le domaine de spécification servant à implanter des FSMs dans les SoCs*.

La justification est la suivante : ces FSMs sont d'autant plus volumineuses (nombre d'état X nombre de signaux de synchronisation) que le nombre de ports, d'états de synchronisation/calcul et de calcul sont nombreux. Au-delà d'un certaine taille, l'outil de synthèse ne supportent plus cette complexité ou bien la surface nécessaire pour implanter la FSM croît trop. Les limites expérimentales atteintes (pour une FFT) avec l'outil ISE de Xilinx (leader mondial du marché des FPGA) sont les suivantes :

Paramètre	**Limite de non synthétisabilité**
Nombre de ports	64
Nombre d'états de synchronisation	64
Nombre moyen d'états de calcul entre deux états de synchronisation	32

Tableau 4-2 : Limite de synthétisabilité avec ISE

Bien que cette limite dépende de l'outil (et de sa version) nous considérons que le modèle « naif » de FSM n'est pas acceptable pour une méthodologie basée sur la synthèse automatique et la réutilisation de composants à fort taux de parallélisme en communication. Il faut donc proposer un autre modèle d'implantation pour ces FSMs : ce sont les CFSMDs auxquelles nous parvenons à la fin de cette section.

Tout d'abord, nous devons définir ce que nous considérons comme *la complexité* des FSMs « naïves ». Du point de vue de la communication à l'interface du composant, la complexité est synonyme du nombre des entrées-sorties et de leur répartition dans le temps. Nous avons mis en place une campagne d'expériences avec l'outil GAUT afin de mesurer la complexité de quelques algorithmes classiques qui exhibent un fort taux de parallélisme potentiel des entrées-

sorties : ce sont la FFT et la DCT. La cible technologique de synthèse est un Virtex à la fréquence d'horloge de 100 MHz. Les résultats sont recensés dans le Tableau 4-3 qui présente les valeurs obtenues lors des synthèses. La colonne "taille" indique le nombre de points de la FTT/DCT. La colonne "cadence" indique la contrainte de cadence d'exécution de l'algorithme spécifiée lors de la synthèse. Plus cette cadence diminue, plus le parallélisme augmente. La colonne "nombre de ports" indique combien de bus bidirectionnels (dédiés à la communication) ont été synthétisés avec optimisation des bus pour respecter la cadence. Le "nombre d'états de synchro" représente le nombre de cycles pendant lesquels ont lieu des entrées et/ou des sorties. Le "nombre moyen d'état de calcul" est l'intervalle moyen en cycle d'horloge entre deux états de synchonisation, il est égal à la cadence divisée par le nombre d'états de synchronisation. La colonne "surface de communication" est le produit du nombre de bus par le nombre de points de synchronisation, elle représente le nombre total de time slots disponibles pour faire transiter des entrées-sorties. La colonne "ratio ES/surface" est le rapport entre le nombre total des entrées-sorties et la surface de communication. Le ratio ES/surface est un pourcentage qui indique le taux de remplissage des time slots de la "surface de communication".

Fonction	Taille	Cadence (µs)	Nombre de ports	Nombre d'états de synchro	Nombre moyen d'états de calcul	Surface de comm.	Ratio ES/surface (%)
FFT	64	10,2	3	192	53	576	44
	64	4,7	9	127	37	1143	22
	128	8,4	16	197	43	3152	16
	128	5,1	25	156	33	3900	13
	256	38,4	6	515	85	3090	33
	256	19,2	9	381	50	3429	30
	512	87,1	6	1027	85	6162	33
	512	43,8	9	765	57	6885	30
DCT	4	1,0	4	39	26	156	21
	4	0,5	8	19	26	152	21
	8	3,4	9	91	37	819	16
	8	2,0	16	69	29	1104	12

Tableau 4-3 : Scénarios de communication

Nous remarquons, qu'à taille constante pour un algorithme donné, le nombre de bus croît et le nombre de points de synchronisation/calcul décroit lorsque la cadence diminue. La décroissante de la cadence a donc pour conséquence de concentrer les entrées-sorties et d'augmenter le nombre de bus. Le ratio ES/surface décroit lui aussi lorsque la cadence diminue et illustre que GAUT synthétise des circuits optimisés en surface pour lesquels le lissage de la communication n'est pas prioritaire. Le nombre de bus dépend donc du pire cas (l'état de synchronisation pour lequel le nombre d'entrées-sorties est maximal) et non pas de la valeur moyenne (nombre d'entrées-sorties divisée par nombre de points de synchronisation). Une amélioration possible de GAUT serait d'intégrer le lissage des entrées-sorties afin de réduire la surface silicium des unités de mémorisation et de communication qui en sont directement dépendantes. Ce travail est actuellement en cours d'étude dans le cadre d'une thèse.

Sur la base de ces observations, nous avons mis en place une deuxième campagne d'expériences de synthèse de FSM de tailles très diverses. Cette campagne d'expérience s'est voulue aussi simple et exhaustive que possible et a été menée avec l'outil ISE 6.2i de Xilinx, sur un PC bi-processeurs ayant 2 Giga-octets de mémoire.

Notre méthodologie de test à consisté à générer automatiquement et aléatoirement des FSM « naïves » ayant la même complexité que les FFT/DCT précédentes. Ce sont des FSM qui ont des nombres de ports d'entrée-sortie et des nombres d'états de synchronisation et de calcul variables. Nous avons ensuite tenté de les synthétiser pour une cible VirtexE-1000, et avons enfin observé les résultats en terme de surface et de vitesse pour les comparer entre elles. Les variables retenues pour la génération des FSM sont : le nombre de ports d'entrée et de sortie, le nombre de points de synchronisation et enfin le nombre moyen d'état de calcul entre deux points de synchronisation. Nous avons ainsi exploré automatiquement une partie de l'espace des FSM naïves. La Figure 4-26 décrit notre flot de génération de FSM « naïves » mais aussi le flot de génération des processeurs de synchronisation (nommés PS) qui leur seront substitués par la suite.

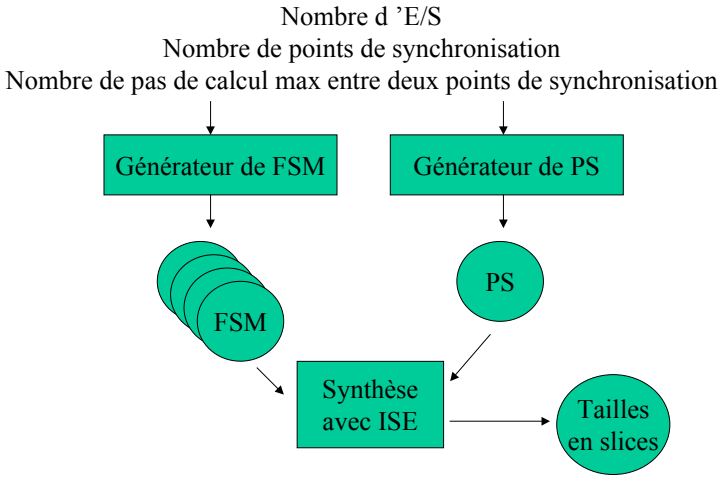


Figure 4-26 : Méthodologie de test de la synthètisabilité

Ainsi par exemple, pour une configuration à deux ports d'entrée et de sortie, deux états de synchronisation et deux états de calcul (une FSM 2/2/2), nous avons généré la FSM illustrée par la Figure 4-27.

Les résultats obtenus prouvent que la solution de type FSM « naïve » n'est pas viable pour deux raisons :

1) La taille des FSMs en « slices » (les slices sont l'unité de surface allouée par l'outil XST et représentent deux CLBs, les CLBs sont les blocs de base configurables du Virtex et contiennent une LUT et un registre de un bit) est loin d'être négligeable et pénalise la fréquence maximale de fonctionnement. Cette réduction de la fréquence est directement due à l'accroissement du nombre de CLBs de la FSM. En effet, plus les circuits sont volumineux, plus la fréquence maximale de fonctionnement décroît à cause du routage inter CLBs.

2) La taille est telle que l'outil n'est plus capable de synthétiser au-delà d'un certain seuil (au moins à partir des FSMs 64/64/32). Remarquons que ce seuil n'est pas irréaliste en terme de complexité si on le compare avec les comportements des FFT et DCT.

```
entity fsm is
      port( rst : in std_logic;
            clk : in std_logic;
            data_ready : in std_logic_vector(2-1 downto 0);
            free_space : in std_logic_vector(2-1 downto 0);
            enable : out std_logic
      );
end;
architecture fsm_arch of fsm is
begin
      process(rst, clk)
            type STATE is (S0, S1, S2, S3);
            variable s : STATE;
      begin
            if rst = '1' then
                  enable <= '0';
                  s  := S0;
            elsif clk = '1' and clk'event then
                  case s is
                  when S0 =>
                        if data_ready = "11" and free_space = "10" then
                              enable <= '1';
                              s := S1;
                        end if;
                  when S1 =>
                        s := S2;
                        enable <= '0';
                  when S2 =>
                        if data_ready = "10" and free_space = "01" then
                              enable <= '1';
                              s := S3;
                        end if;
                  when S3 =>
                        s := S0;
                        enable <= '0';
                  end case;
            end if;
      end process;
end;
```

Figure 4-27 : FSM 2/2/2 générée automatiquement

Le Tableau 4-4 montre un court extrait de notre campagne d'expériences. Nous concluons qu'il faut un autre modèle d'implantation des FSM : des CFSMD.

Nombre de ports	Nombre d'états de synchronisation	Nombre moyen d'états de calcul	Nombre de slices	**Fréquence maximale (MHz)**
4	4	2 (4)	15	162
8	8	4 (8)	36	148
16	16	8 (16)	110	123
32	32	16 (32)	429	105
64	64	32 (64)	impossible	Impossible
128	128	64 (128)	impossible	Impossible

Tableau 4-4 : Campagne de test de synthèse de FSM

3.4.2. Le processeur de synchronisation

Le processeur de synchronisation que nous proposons est une FSM optimisée pour la synthèse physique. Il s'agit d'une CFSMD car : 1) elle est concurrente avec la FSM du composant synchrone et se synchronise avec ce dernier par le signal *enable* et 2) son chemin de données contient une mémoire dans laquelle sont stockées les opérations qu'elle exécute en séquence. La Figure 4-28 illustre la différence entre une FSM classique et notre CFSMD. Une FSM classique est constituée d'un registre d'état et de logique combinatoire qui calcule les valeurs des sorties ainsi que l'état suivant.

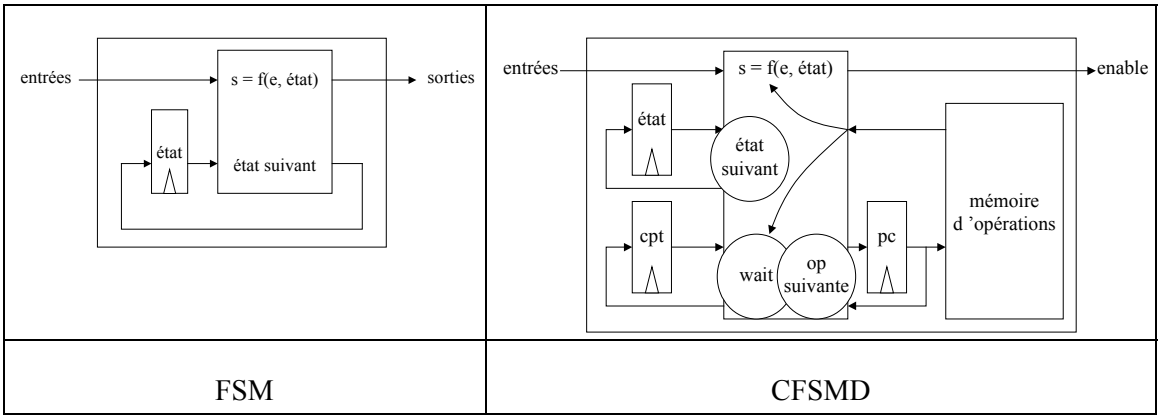


Figure 4-28 : Comparaison FSM/CFSMD

Notre CFSMD est constituée d'un chemin de donnée qui contient en plus du registre d'état : un compteur d'états de calcul sans synchronisation (cpt), un compteur d'opération (pc) et une ROM stockant le code des opérations. La partie combinatoire calcule la valeur de la sortie *enable* en fonction des entrées et de la valeur du chemin de données. Bien que cette description ressemble à celle d'un processeur logiciel, ce n'en est pas un. Notre jeu d'opérations n'est pas tel que nous puissions prétendre avoir défini une ISA. En effet, nous n'avons aucune instruction de test ni de branchement. De plus nous n'avons pas de compilateur et encore moins de langage de programmation pour coder le contenu de la mémoire d'instructions. Il s'agit donc d'un processeur matériel dont le modèle d'implantation est une CFSMD. L'intérêt d'une solution à base de FSM microcodée est que :

1) La taille du processeur ne dépend plus de la complexité des entrées-sorties, elle est désormais constante pour un nombre d'entrées-sorties données.

2) La taille du processeur étant constante, sa vitesse l'est aussi.

3) La partie du processeur qui dépend du scénario de synchronisation est rangée dans une mémoire que l'on ne fait que lire, jamais écrire. On profite donc de la densité supérieure des mémoires (ROM sur ASIC, blocs RAM spécialisés sur FPGA) pour implanter plus efficacement la partie sensible à la complexité de la FSM.

3.4.3. Conception du jeu d'opérations du processeur

La conception du jeu d'opérations consiste à définir successivement la sémantique, le format et le codage des opérations que le processeur va interpréter.

Le jeu d'opérations est issu de l'analyse de la spécification du comportement du composant sous la forme d'une liste de couples d'ensembles de ports d'entrées et de sorties significatifs à chaque cycle et sur la distinction entre état de synchronisation et état de calcul. Ce format textuel a été précédemment défini à la section 3.2.4.

Ainsi, si l'on reprend l'exemple de la Figure 4-12 et que l'on factorise les états de calcul de façon à faire apparaître des séquences de calcul, on obtient une nouvelle liste strictement équivalente de couples (*type*, <*attributs*>).

```
L = ( ( Sync, ({b1_in, b2_in}, ∅ )),
      ( Calc, 2),
      ( Sync, ({b1_in, b2_in, b3_in, b4_in}, ∅)),
      ( Calc, 3),
      ( Sync, ({b3_in, b4_in}, ∅)),
      ( Sync, (∅, {b1_out}))
    )
```

Figure 4-29 : Factorisation des points de calcul

La nouvelle liste est à nouveau une liste de couples. Chacun de ces couples contient le type de l'état et un attribut. La valeur *type* est « Sync » ou « Calc » selon qu'il s'agit d'un état de

synchronisation, ou d'une séquence d'états de calcul. La valeur *attribut* est un ensemble d'informations dépendantes du type. Dans le cas d'un *Sync*, l'*attribut* est le couple *(Ei, Si)*, et dans le cas d'un *Calc*, l'*attribut* est le nombre d'états de calcul qu'il représente.

Une dernière simplification peut être apportée à cette liste. Sachant qu'un *Sync* peut être suivi facultativement d'un *Calc*, on peut réduire la liste à une liste de triplets *(Ei, Si, n)* où *Ei* et *Si* ont la même sémantique et *n* représente le nombre d'états de calcul qui suivent (0 s'il n'y en a pas).

```
L = ( ( {b1_in, b2_in},                Ø,          2),
      ( {b1_in, b2_in, b3_in, b4_in}, Ø,          3),
      ( {b3_in, b4_in},                Ø,          0),
      ( Ø,                             {b1_out}, 0)
    )
```


Figure 4-30 : Extraction de la liste d'opérations

C'est à partir de cette liste de synchronisation factorisée et simplifiée que le jeu d'opérations est conçu. La sémantique des opérations est la suivante : chacune des opérations représente un état de synchronisation, spécifié par ses ports d'entrée et de sortie et une suite (éventuellement vide) d'états de calcul. Le codage des instructions est une implantation directe de cette liste en mémoire. Le format des instructions dépend du nombre de ports d'entrée et de sorties et du nombre maximum de points de calcul successifs. Le format des opérations est le suivant :

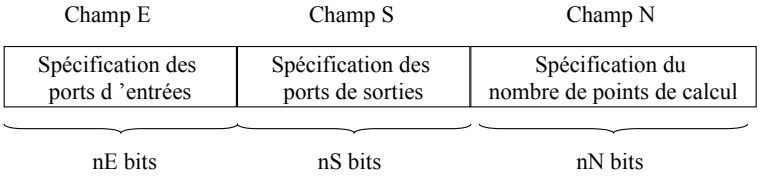


Figure 4-31 : Format des opérations

Les nombres *nE*, *nS*, et *nN* représentent respectivement les nombres de ports d'entrée et de sortie du composant, et le nombre maximum de bits qu'il faut pour coder en binaire non signé le nombre maximum de points de calcul successifs. Dans notre exemple nE = 4, nS = 1, et nN = 3 (n maximum = 4) et le format des opérations est :

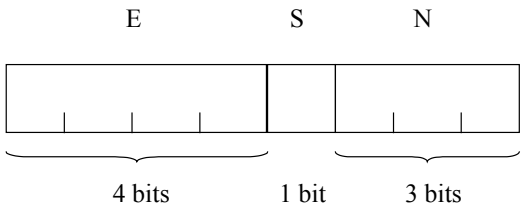


Figure 4-32 : Exemple de format des opérations

La mémoire d'opération est une mémoire ROM (ASIC) ou RAM (FPGA) à lecture asynchrone ayant les dimensions :

1) longueur *nL* égale à la longueur de la liste factorisée et simplifiée des points de synchronisation et de calcul

2) largeur égale à *nE* + *nS* + *nN*

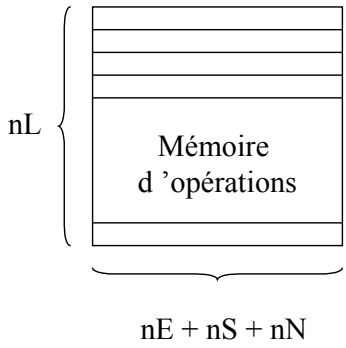


Figure 4-33 : Spécification de la taille de la mémoire d'opérations

Le codage de la spécification des ports d'entrée et de sortie repose sur le même principe : un bit est associé à chaque port du composant et la valeur binaire '1' signifie que ce port est une entrée (ou une sortie) nécessaire pour quitter l'état de synchronisation (arrêter d'attendre) , '0' autrement. Le codage du nombre de points de calcul est une représentation binaire non signée. Dans notre exemple nL = 4, le contenu de la mémoire est :

1 1 0 0	0	0 1 0
1 1 1 1	0	1 0 0
0 0 1 1	0	0 0 0
0 0 0 0	1	0 0 0

Figure 4-34 : Exemple de contenu d'une mémoire d'opérations

3.4.4. Modèle de contrôle

Il s'agit d'une CFSMD à trois état. L'état *reset* représente l'initialisation de la machine d'états finis lors d'un reset matériel. Les états *wait* et *run* représentent l'interprétation des opérations.

La notation des transitions *x/y* signifie que *x* est la garde et que *y* est l'action. Les actions sont de deux types, elles concernent le chemin de données et la synchronisation avec la FSM concurrente du composant encapsulé.

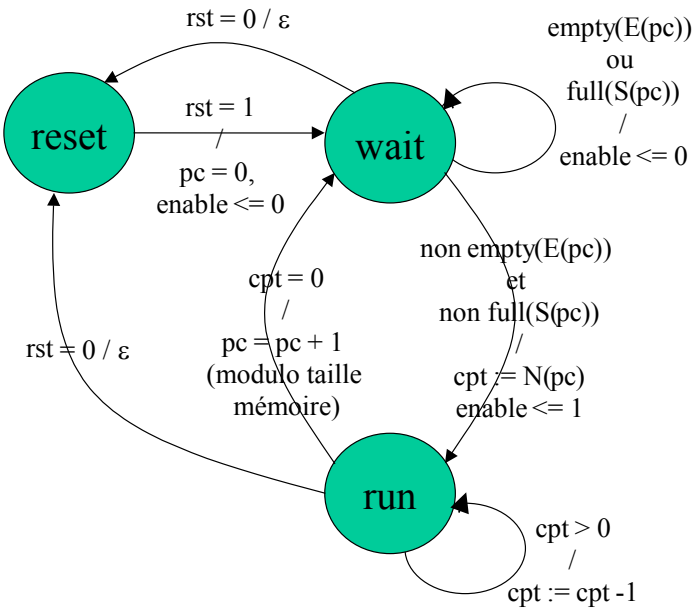


Figure 4-35 : FSMD de contrôle du processeur de synchronisation

Une action notée ε signifie qu'il n'y a aucune modification du chemin de donnée ni aucune synchronisation particulière avec la FSM concurrente. Ce cas ne se présente que lors d'un reset. Dans tous les autres cas la sémantique de *y* est la suivante :

1) Lorsque l'on utilise l'opérateur d'affectation `:=` il s'agit d'une opération sur le chemin de données.

2) Lorsque l'on utilise l'opérateur d'affectation `<=` il s'agit du pilotage du signal *enable* de synchronisation de la FSM concurrente.

Le chemin de donnée est constitué des éléments suivants :

1) *pc* est un compteur programme : c'est l'adresse de la prochaine opération à lire et exécuter. La notation « *pc* := … » signifie que l'on place une nouvelle adresse en entrée de la mémoire. A l'aide d'une mémoire à lecture asynchrone, la lecture est immédiate. Le compteur *pc* est incrémenté modulo la taille de la mémoire et implante simplement un parcours séquentiel rebouclé de la mémoire d'opérations.

2) *E(pc)* représente le décodage du champ qui représente le masque des ports d'entrée qu'il faut tester et valider pour pouvoir avancer d'un cycle

3) *S(pc)* représente le décodage du champ qui représente le masque des ports de sortie qu'il faut tester et valider pour pouvoir avancer d'un cycle.

4) *N(pc)* est le décodage du champ « nombre de cycles de calcul » nécessaire pour initialiser la phase purement synchrone de rebouclage sur l'état « *run* ».

5) *cpt* est un compteur décrémenté qui compte le nombre de cycles dans la phase synchrone de l'état « *run* ».

3.4.5. Implantation VHDL et résultats

- L'implantation VHDL de la FSM du processeur de synchronisation est triviale. Les résultats de sa synthèse dans les mêmes conditions que les FSM naïves sont :

1) Surface 16 slices équivalente à celle d'une FSM naïve de 4x4x4

2) Fréquence maximale 124 MHz équivalente à celle d'une FSM naïve de 16x16x16

La synthèse d'un grand nombre de FSM de diverses tailles donne la courbe de la Figure 4-36. On remarque qu'il est nécessaire que la taille des FSMs reste en deçà de 300 slices pour que des fréquences de fonctionnement supérieures à 100 MHz soient garanties. Enfin, il est tout à fait envisageable d'obtenir des FSMs de tailles très supérieures à 500 slices dont la fréquence n'atteint pas les 100 MHz.

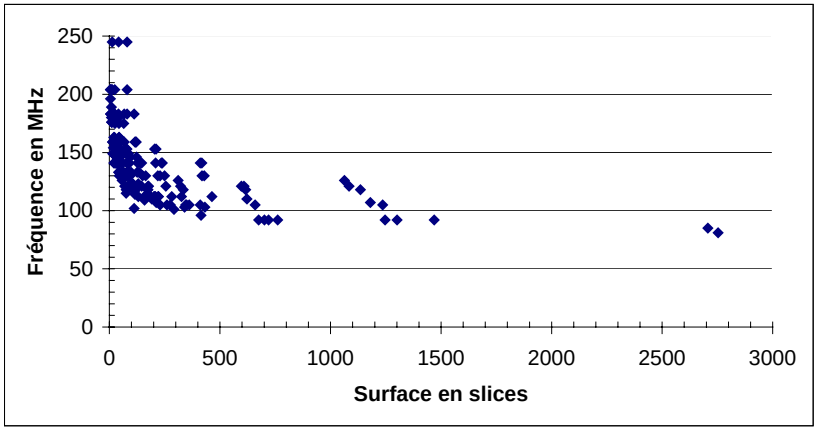


Figure 4-36 : Répartition fréquence-surface des synthèses de FSM (XST)

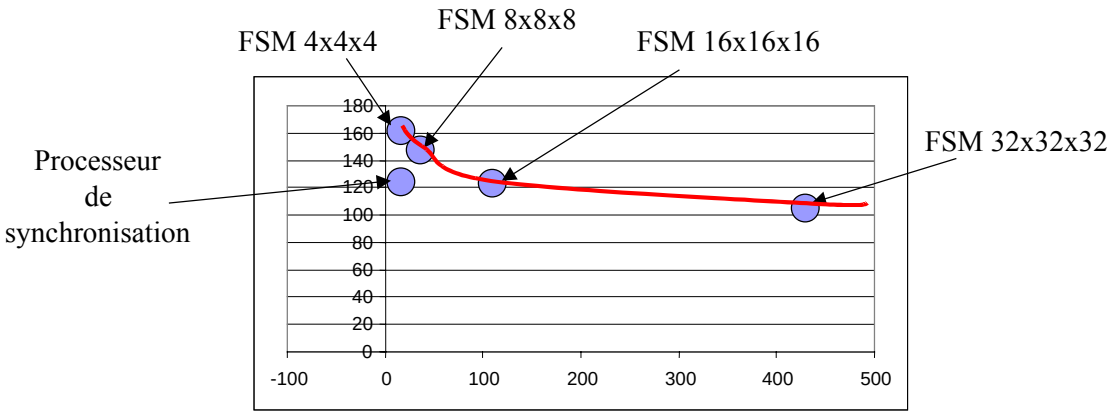


Figure 4-37 : Position du processeur de synchronisation

La Figure 4-37 présente la position du processeur de synchronisation par rapport à la courbe surface-fréquences des FSM naïves limitée à moins de 500 slices. Nous avons conservé quelques points significatifs des complexités mise en œuvre dans les FSMs naïves. La « bonne » position se situe en haut (fréquence élevée) et à gauche (surface faible). On constate que le processeur de synchronisation se situe légèrement en deçà (75%) des meilleures performances

(mais des FSM aussi simples sont-elles vraiment représentatives de systèmes réels ?) et se place très en tête de quasiment toutes les alternatives en terme de surface. On remarque aussi qu'une FSM 32x32x32 (32 entrées-sorties, 32 points de synchronisation et une moyenne de 32 états de calcul entre deux points de synchronisation) se situe déjà dans une zone défavorable en vitesse et en surface alors qu'elle n'est pas aussi complexe que les nombreux scénarios de communication attendus pour des FFT et des DCT.

Afin de ne pas faire reposer notre argumentaire sur un seul outil de synthèse (ISE de Xilinx), nous avons fait les mêmes expériences avec les outils :

- QUARTUS (Altera) pour la cible FPGA EPXA (Excalibur)
- DC (Design Compiler, Synopsys) pour :
  - la technologie ASIC CSX 0,35 µm du fondeur AustriaMicroSystems (AMS)
  - la technologie ASIC HCMOS7 0,25 µm du fondeur STMicroelectronics
  - la technologie ASIC 0,18 µm du fondeur STMicroelectronics
  - la technologie ASIC 0,15 µm du fondeur TSMC

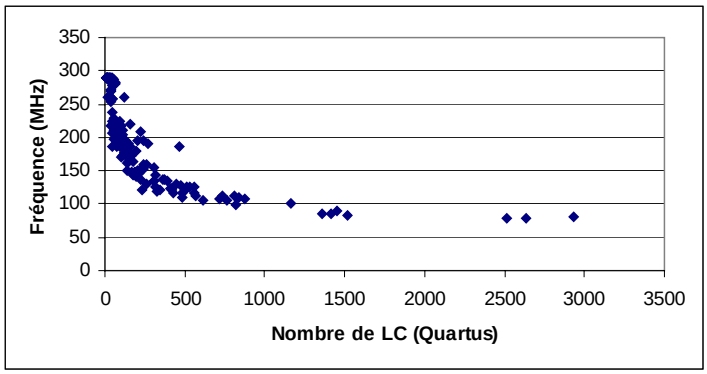


Figure 4-38 : Cible FPGA EPXA (Quartus)

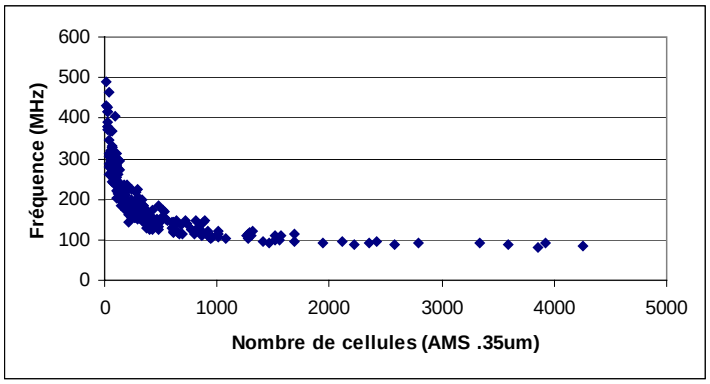


Figure 4-39 : Cible AMS 0,35 µm (DC)

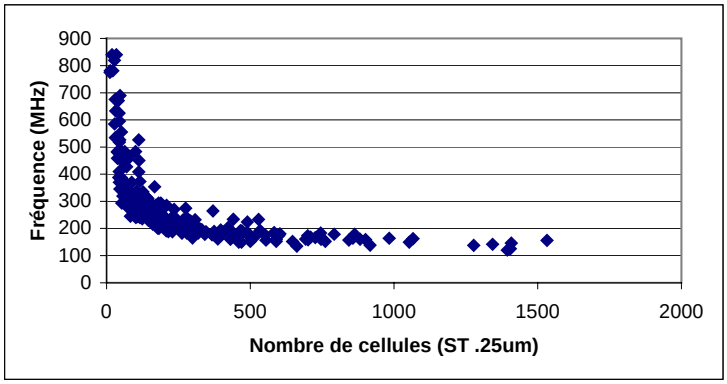


Figure 4-40 : Cible ST 0,25 µm (DC)

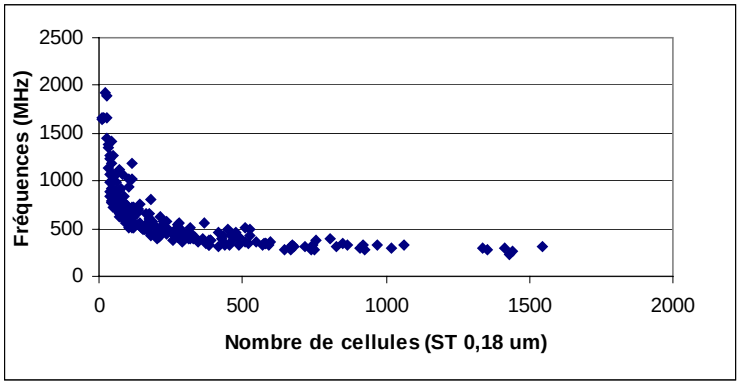


Figure 4-41 : Cible ST 0,18 µm (DC)

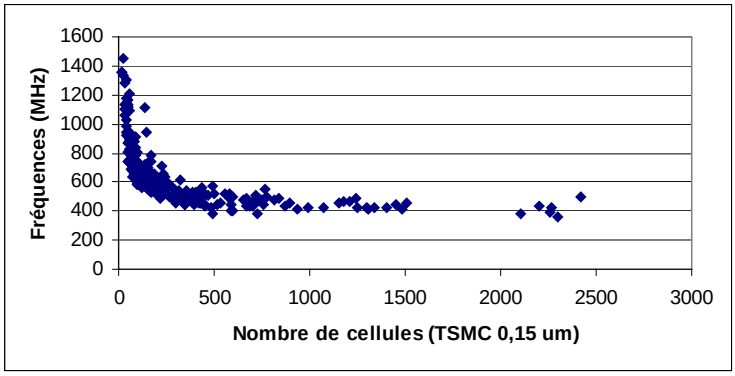


Figure 4-42 : Cible TSMC 0,15 µm (DC)

Les comportements observés sont absolument les mêmes : les fréquences les plus élevées sont atteignables seulement pour des FSM de tailles réduites. Dans les cas de figures qui nous concernent, les fréquences se stabilisent rapidement à des valeurs moyennes qui sont de l'ordre de 100 MHz pour Xilinx, Altera et AMS 0,35 µm, de 150 MHz pour ST 0,25 µm, 250-300 MHz pour ST 0,18 µm et 350-400 MHz pour TSMC 0,15 µm.

Enfin, l'ensemble de ces résultats concerne exclusivement la synthèse de la FSM du processeur de synchronisation. L'économie en surface réalisée est importante mais est faite au détriment de la mémoire d'opérations dont la largeur (masques des E/S et nombre de pas de calcul sans synchronisation) et la longueur (nombre d'opérations) sont représentatifs de la complexité du scénario des entrées-sorties. Alors que la largeur de la mémoire peut être réduite par codage des masques des E/S, la longueur ne peut l'être. Le principal attrait des mémoires est leur plus grande densité (nombre de point mémoire par unité de surface) sur les FPGA mais aussi dans les ASICs lorsque ce sont des ROMs.

Nous avons donc mesuré le coût mémoire dans le contexte des FPGA de Xilinx. Il faut tout d'abord savoir que les FPGA des familles Virtex, VirtexE et VirtexE « étendus » contiennent tous des blocs mémoires nommés SelectRam. Ce sont des blocs durs contenant 4 Kbits. Ils permettent de construire des mémoires de facteurs de forme allant de 4096 lignes d'un bit à 256 lignes de 16 bits. La Figure 4-43 représente la consommation en bits (pire cas, sans codage des masques) de nos processeurs de synchronisation. On trouve en abscisse le nombre de slices des FSM naïves équivalentes et en ordonnée le nombre de bits requis pour stocker en mémoire la totalité des opérations. On remarque que, à quelques exceptions près, la majorité des mémoires d'opérations requièrent un seul bloc SelectRam, et que le pire cas n'en consomme que trois. Sachant que les FPGA de Xilinx contiennent de 8 à 280 blocs SelectRam, nos processeurs de synchronisation peuvent être logés dans n'importe lequel d'entre eux. Enfin, dans le cadre de la plate-forme PALMYRE, les FPGA retenus contiennent de 96 à 160 blocs SelectRam. PALMYRE nous permet donc de concevoir des systèmes mono-FPGA combinant plusieurs composants encapsulé dans des processeurs de synchronisation.

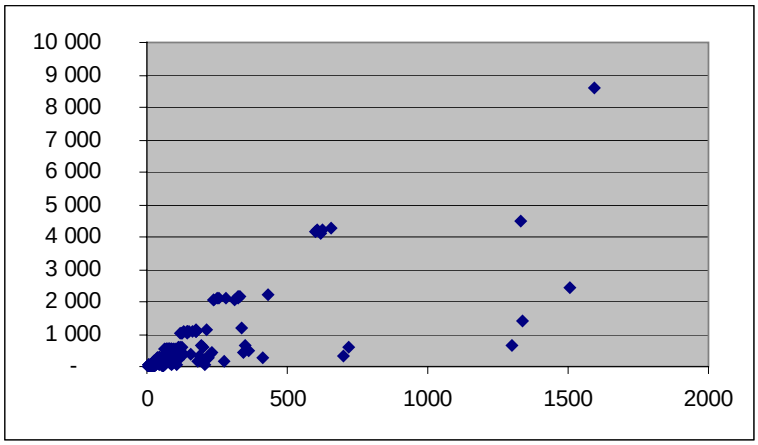


Figure 4-43 : Surface mémoire requise par les processeurs de synchronisation

Nous concluons donc que notre processeur de synchronisation est un moyen efficace de préserver des fréquences de fonctionnement élevées, pour un coût en surface raisonnable, et qu'il permet réellement de mettre en œuvre une méthodologie de conception de type LIS.

4. Unité de mémorisation

Dans la conception des SoC, tout comme pour les systèmes embarqués, les mémoires jouent un rôle important et impactent significativement les performances, la consommation, et le coût d'une implantation particulière [PAN01]. Le fait que la vitesse de fonctionnement et l'énergie consommée pour une application donnée soient des objectifs distincts qui requièrent des stratégies d'optimisation différentes est établi depuis 1994 par [WUY94]. Nous nous sommes concentrés, dans le cadre de ces travaux, sur l'optimisation des performances. L'ITRS [ITR03] prévoit à court terme une évolution des fréquences des processeurs de 3 à 12 GHz alors que, dans le même temps, les fréquences des bus périphériques hautes vitesses n'évolueront que de 2 à 7 GHz. L'écart s'accroît donc de plus en plus entre les fréquences d'exécution des instructions des processeurs et les fréquences d'accès aux contenus des mémoires. L'accès aux mémoires, s'il est mal géré, peut donc devenir le goulot d'étranglement qui contraint alors le système tout entier. Cela est particulièrement vrai avec les processeur spécialisés générés par GAUT car ils sont potentiellement parallèles et leur fonctionnement optimal dépend de l'aptitude du système à 1) leur fournir à temps les données à traiter et à 2) consommer à temps les résultats produits.

L'optimisation des accès mémoires a suscité de nombreux travaux que l'on peut classer en deux catégories : les transformations de code indépendantes de toute implantation, et les optimisations liées à la connaissance de l'architecture cible et du comportement « flot de données » du système. Les transformations de code sont mises en œuvre au niveau langage de programmation par des compilateurs et concernent principalement la transformation de boucles. Elles ont pour objectif de minimiser le nombre d'accès (utilisation de registres), d'accroître la localité temporelle et spatiale des accès de façon à diminuer la probabilité de défaut de cache et d'augmenter le parallélisme de tâches ou de données. Bien qu'indépendantes de toute architecture, ces optimisations reposent sur une hiérarchisation à priori de la mémoire (registres, caches, mémoires, mémoires de masse) et placent au plus près des unités de calcul les données les plus utilisées. ATOMIUM [ATO04] de l'IMEC, est un exemple d'outil d'analyse des accès mémoire, de définition d'une architecture mémoire et d'optimisation des générateurs d'adresses récemment passé dans le domaine commercial [POW04], de même, au niveau algorithmique, [MAR04] identifie un placement mémoire des structures de données qui réduit l'activité sur les bus et permet de diminuer la consommation du système. Dans le domaine de la synthèse de haut niveau, les optimisations mémoires concernent l'allocation des ressources physiques (registres, bancs de registres, mémoires multi-ports) et l'assignation des mémoires logiques (les données de l'algorithme) aux mémoires physiques. Elle nécessite la modélisation temporelle des accès aux mémoires physiques pour pouvoir allouer les ressources mémoires minimales et ordonnancer les accès à ces dernières de façon à respecter une contrainte de temps. Les modèles d'accès aux mémoires permettent d'optimiser aussi bien l'accès à des mémoires simples comme des registres que l'accès à des mémoires aux protocoles complexes comme les DRAM (Les DRAM peuvent aussi faire intervenir de l'accès rapide en mode page et des séquences de « refresh » pendant lesquelles les accès sont interdits).

Avec l'outil GAUT, le nombre, la taille et le type des mémoires physiques (registres ou DPRAM) sont totalement définis à l'issue de la synthèse. La spécificité de l'implantation de ces mémoires physiques provient du fait que le circuit généré par GAUT peut être pipeliné. Cette notion de pipelining d'algorithme est inspirée des techniques employées par les compilateurs

ILP (Instruction Level Parallelism) pour processeurs VLIW et sera présentée en détail dans la suite de ce document (section 4.2). L'unité de mémorisation (UM) est un composant dont le rôle est d'assurer le stockage et le transit des variables internes des algorithmes synthétisés par GAUT. Selon les contraintes de synthèse et les durées de vie et/ou d'inactivité des variables, la stratégie de minimisation du nombre de registres consiste à renvoyer en mémoire externe les variables insuffisamment utilisées [MAR92]. L'ordonnancement des calculs et des lectures/écritures sous contraintes de placement des variables et des constantes en mémoire [GUE04] est prévu pour exploiter une UM contenant plusieurs bancs mémoires. Les mémoires actuellement disponibles sur les FPGA sont des DPRAM (Dual Port Random Access Memory). Elles nous permettent de lire et d'écrire deux variables différentes au même cycle. Alors que les techniques traditionnelles de pipelining sont appliquées aux niveau opérateur RTL (opérateurs pipelinés), au niveau architecture (processeurs pipelinés) ou au niveau langage (pipelining logiciel), GAUT exploite le pipelining au niveau de l'algorithme. Le seul acteur commercial actif sur le sujet est la société get2chip [GET04] dont l'outil Pipeline Master serait capable d'introduire du pipelining au niveau algorithme. Depuis son rachat par Cadence pour ses outils de synthèse combinée logique/physique nano-métrique en avril 2003, plus aucune mention de Pipeline Master n'est faite. A l'heure de la rédaction, il semble que GAUT soit le seul outil de synthèse exploitant cette méthode : le pipelining d'algorithme de GAUT et notre structure d'unité de mémorisation associée seraient donc inédits.

Nous présentons maintenant la taxonomie des variables d'un algorithme, puis la nature du problème posé par le pipelining de ce dernier et ensuite décrivons notre contribution en terme d'unité de mémorisation pour GAUT.

4.1. Taxonomie des variables d'un algorithme

L'unité de mémorisation doit supporter les lectures/écritures de toutes les données pour lesquelles il y a du transit. Ces données représentent les valeurs des variables et des constantes utilisées par l'algorithme lors de son exécution. Afin de spécifier le problème posé par le pipelining d'algorithme, il faut tout d'abord caractériser la nature des constantes et des variables d'un algorithme. Tout langage de programmation/spécification possède les notions explicites de constantes (mot clé « const » en C++, mot clé « constant » en VHLD …) et de variables. Les constantes, comme leur nom l'indique, ont une valeur constante et ne sont jamais modifiées. Les variables, comme leur nom l'indique aussi, peuvent changer de valeur en cours d'exécution.

Dans le contexte de GAUT, la classification est plus riche : les données peuvent être de quatre types : *constantes*, *variables*, *variables rebouclées*, et *variables vieillissantes.* La spécification des algorithmes étant actuellement faite en VHDL comportemental, nous définissons l'interprétation de tels codes.

- *Constantes*. Ce sont les constantes VHDL spécifiées avec le mot clé « constant ». Leur valeur est connue lors de la synthèse. Elles permettent des optimisations de code du type :

    1) propagation des constantes

    2) remplacement des multiplications et divisions avec des coefficients multiples de 2 par des décalages arithmétiques.

Ce sont aussi les variables VHDL lues et jamais écrites. La valeur de ces constantes, dont on ne connaît pas explicitement la valeur dans l'algorithme (par exemples les coefficients d'un filtre), est fixée lors de la spécification des contraintes de placement en mémoire. Ces « fausses variables » ont l'intérêt de permettre une modification post-synthèse d'un filtre par simple remplacement d'une ROM par une autre. Il est donc essentiel de préserver leur lecture « externe » afin de garantir une plus grande flexibilité des circuits générés.

- *Variables*. Ce sont les variables VHDL spécifiées avec le mot clé « variable ». Dans le cadre de l'exécution implicitement répétitive d'un algorithme, les variables se divisent en deux catégories : les variables rebouclées ou non. Nous nommons *variable* exclusivement les variables non rebouclées.

- *Variables rebouclées*. Ce sont les variables qui sont lues avant d'être modifiées. Cela signifie que la valeur initiale d'une variable lors de l'exécution *N* d'un algorithme dépend de l'exécution *N-1* du même algorithme. Par opposition, les *variables* « simples » sont écrites avant d'être lues.

- *Variables vieillissantes*. Il s'agit d'un cas particulier de variables rebouclées permettant une réduction du nombre d'accès à la mémoire. Ce sont les fenêtres d'échantillons qui sont fréquemment utilisées dans les algorithmes de traitement du signal. Une fenêtre est définie par un tableau dans lequel, lors de chaque exécution de l'algorithme, un échantillon entre et un échantillon sort. Hormis cela, aucune modification n'est faite sur les éléments du tableau. L'optimisation mise en œuvre consiste à utiliser des buffers circulaires pour stocker les éléments des tableaux et à utiliser un adressage basé-indexé pour y accéder. La base est alors mise à jour à la fin de l'exécution de l'algorithme et évite ainsi le déplacement de tous les éléments du tableau en mémoire. L'économie réalisée en nombre d'accès mémoire est proportionnelle à la longueur des tableaux.

Nous résumons cette taxonomie ainsi :

1) Les constantes ne sont que lues.

2) Les variables sont affectées, puis lues.

3) Les variables rebouclées sont lues puis affectées.

4) Les variables vieillissantes sont un cas particulier de variables rebouclées.

4.2. Le pipelining d'algorithme

L'analyse d'un algorithme produit une représentation intermédiaire de type flot de signaux (SFG, ou Signal Flow Graph). Ce graphe représente statiquement les dépendances de données qui existent au sein de l'algorithme, toutes boucles déroulées.

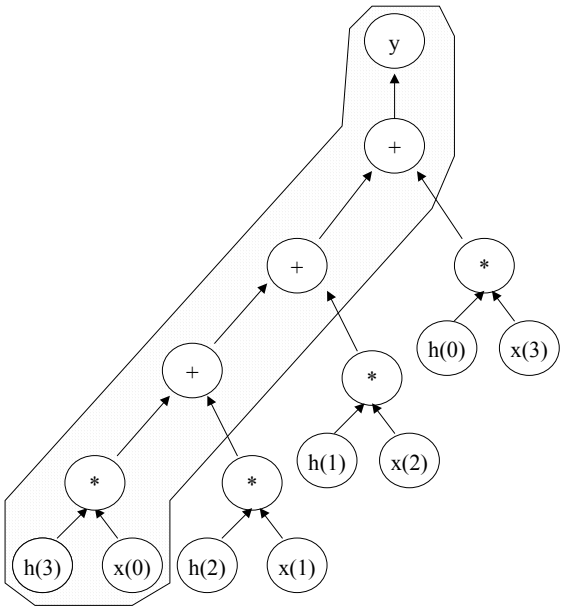


Figure 4-44 : Chemin critique dans un FIR4 « séquentiel »

Le *chemin critique* est la plus longue succession d'opérations qui mène d'une entrée à une sortie de l'algorithme. Quelques soient les opérateurs alloués et le placement des opérations sur ces opérateurs, le calcul spécifié par le SFG est fait dans un ordre précis et dure un temps donné qui dépend exclusivement de l'implantation des opérateurs sur la cible technologique sélectionnée pour la synthèse. La Figure 4-44 illustre un SFG pour un filtre FIR de dimension quatre, spécifié de façon séquentielle. Le chemin critique va du couple échantillon/coefficient *(h(3), x(0))* jusqu'à la sortie *y* et contient une multiplication et trois additions. GAUT génère un circuit pipeliné lorsque le chemin critique de calcul d'au moins une des sorties dépasse la contrainte de cadence. Par la mise en parallèle du calcul de plusieurs algorithmes, la contrainte de cadence est alors tenue au prix d'une latence supérieure à la cadence

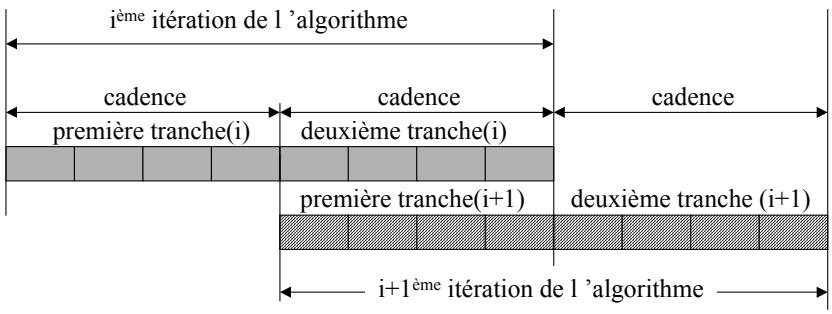


Figure 4-45 : Pipelining si cadence < chemin critique

Les FSMs générées alors par GAUT ont une durée multiple de cadence et le nombre de tranches est égal à la latence divisée par la cadence. On peut donc obtenir des circuits dans lesquels existent *N* tranches correspondant à *N* exécutions en parallèle d'un même algorithme. La Figure 4-45 illustre cela pour deux tranches de pipeline et une cadence de quatre cycles d'horloge.

Néanmoins le pipelining n'est pas toujours possible car il dépend des boucles critiques.

Une *boucle critique* est un chemin de calcul dont la sortie est une modification de variable rebouclée. La Figure 4-46 montre une boucle critique, lors de l'itération *i*, sur la variable *x* dans l'exécution d'un algorithme pipeliné en deux tranches.

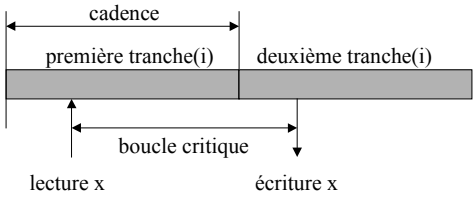


Figure 4-46 : Chronogramme avec boucle critique

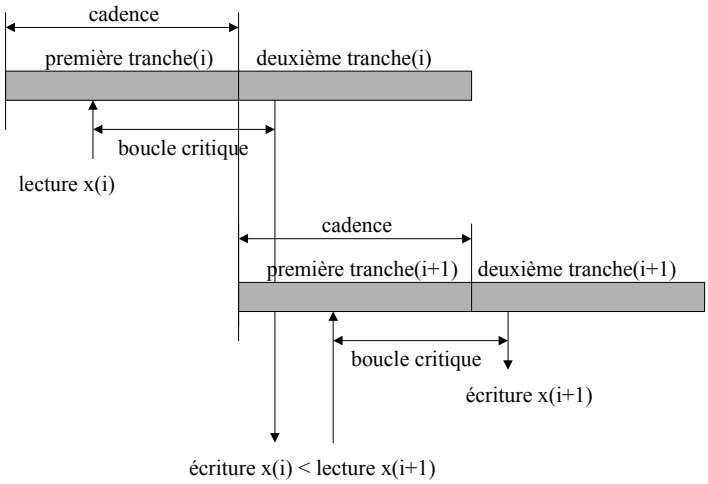


Figure 4-47 : Bonnes conditions de pipelining

La mise en parallèle des tranches de pipeline superpose, modulo la cadence, les lectures et les écritures de la variable *x*. Deux situations peuvent apparaître : de bonnes conditions de « pipelinabilité » (Figure 4-47) ou de mauvaises (Figure 4-48).

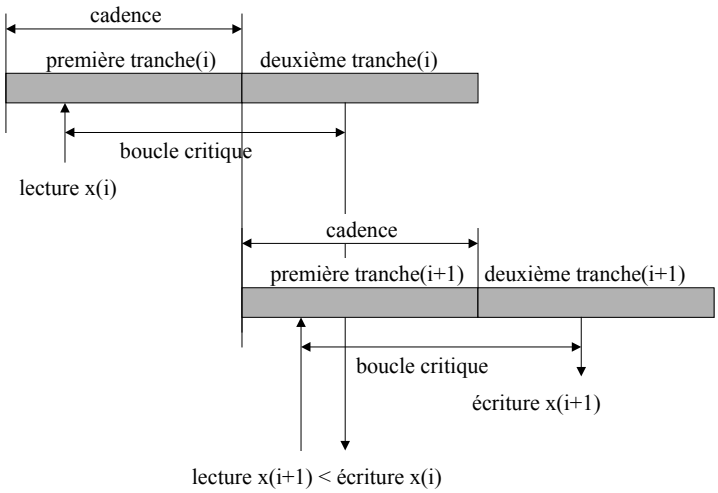


Figure 4-48 : Mauvaises conditions de pipelining

La notation *x(i)* signifie variable *x* lors de l'itération *i*. Le pipelining, pour être fonctionnellement correct, doit garantir que, quelque soit *i*, la date d'écriture de *x(i)* est faite avant la date de lecture de *x(i+1)*. Soit :

$$\forall x,\ \forall i,\ Ecriture(x(i)) \bmod cadence < Lecture(x(i+1)) \bmod cadence$$

Or, quelque soit la variable *x* et quelques soient les indices d'itération *i* et *j* :

$$\forall x,\ \forall i,\ \forall j,\ Ecriture(x(i)) \bmod cadence = Ecriture(x(i+j)) \bmod cadence$$
$$Lecture(x(i)) \bmod cadence = Lecture(x(i+j)) \bmod cadence$$

Ces équations permettent d'énoncer la condition de « pipelinabilité » :

$$\forall x,\ \forall i,\ Ecriture(x(i)) \bmod cadence < Lecture(x(i)) \bmod cadence$$

Résumons les conditions de « pipeline-abilité » d'un algorithme :

1) Si le chemin critique de calcul des sorties est plus court que la cadence, le circuit peut être synthétisé sans pipelining. Il y aura plus ou moins de parallélisme interne pour respecter la contrainte de cadence.

2) Sinon, si modulo la cadence l'écriture des variables rebouclées précède toujours leur lecture, alors le circuit peut être synthétisé avec pipelining.

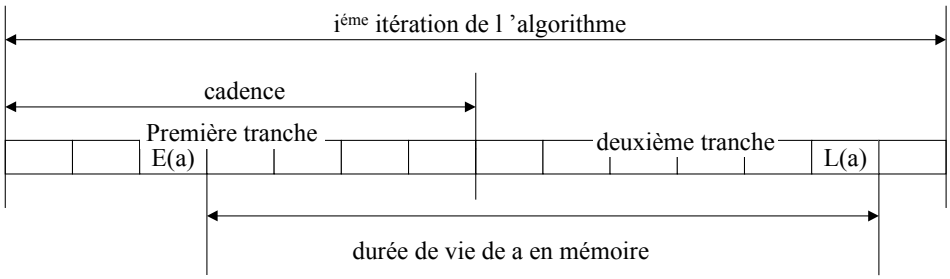
3) Sinon il ne peut pas être synthétisé à la cadence requise et l'algorithme doit être transformé par le concepteur de façon à raccourcir la boucle critique. GAUT indique alors quelle est la variable en cause.

4.3. Conséquences du pipelining sur l'unité de mémorisation

Le pipelining fait apparaître trois phénomènes distincts : 1) la duplication d'une variable, 2) la duplication d'une variable rebouclée et 3) l'allongement d'une séquence de variables vieillissantes. Nous spécifions maintenant ces trois phénomènes et leur apportons une solution en terme de générateur d'adresse dans l'unité de mémorisation.

4.3.1. Duplication de variables

La Figure 4-49 illustre un chronogramme en 14 cycles des lectures/écritures d'une variable *a* pour un algorithme hypothétique qui ne tient pas la contrainte de cadence et qui va être pipeliné. Chaque cycle d'horloge est représenté par un rectangle. On distingue la différence entre latence d'exécution de l'algorithme et cadence d'arrivée des échantillons. La latence est découpée en tranches d'une durée égale à la cadence. La variable *a* a une durée de vie qui recouvre les deux tranches de pipeline et va de la fin du cycle 3 à la fin du cycle 13.



L(a) = lecture de a
E(a) = écriture de a

Figure 4-49 : variable candidate à la duplication

Si l'on superpose deux exécutions de l'algorithme, on obtient la situation illustrée par la Figure 4-50. Les deux tranches s'exécutent en parallèle et correspondent chacune à une partie de l'exécution de l'algorithme. On observe qu'il y a une période de temps pendant laquelle plusieurs occurrences de la variable *a* existent en même temps en mémoire. La première occurrence (itération i+1) va du cycle 4 au cycle 7, la deuxième (itération i) va du cycle 1 au cycle 6. La zone de recouvrement des deux durées de vies va donc du cycle 4 au cycle 6.

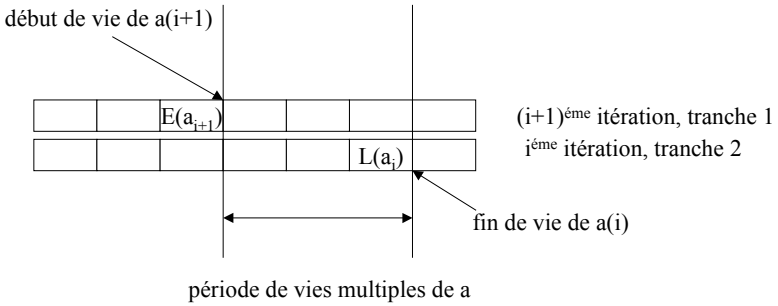


Figure 4-50 : Impact du pipeline sur la duplication de variables

Remarquons que l'exemple ne montre que deux occurrences de la variable *a* car il n'y a que deux tranches. Le cas général est un nombre potentiel d'occurrences inférieur ou égal au nombre total de tranches. Plus généralement, pour qu'il n'y ait pas de collision des durées de vie d'une même variable lors du repliement du pipeline, il faut que les dates de lecture (modulo cadence) précèdent les dates d'écriture (modulo cadence) de cette variable. Si cette condition n'est pas vérifiée, alors il y a nécessairement duplication de la variable en mémoire.

Le modèle d'implantation de l'unité de mémorisation est donc adapté comme suit :

1) *Au niveau structurel*. Allouer autant de cases mémoires qu'il y a d'occurrences de *a*. La variable *a* est affectée à un banc particulier par le concepteur. Le générateur d'UM doit alors allouer les autres occurrences dans les bancs disponibles (y compris celui du *a* originel) ou en créer de nouveaux si nécessaire. Deux raisons justifient la création de nouveaux bancs : 1) les lectures/écritures des nouvelles occurrences ne peuvent pas être placées dans un banc dans lequel il y a déjà une lecture/écriture (de n'importe quelle variable ou constante) au même moment, 2) il se peut que le concepteur n'ait pas défini autant de bancs qu'il y a d'occurrences. Le résultat de cette allocation est un ensemble d'adresses (n° de banc, adresse dans le banc) pour la variable *a*. Ces valeurs sont stockées dans la *table des adresses de a* et un index calculé dynamiquement permet de retrouver la bonne adresse quelque soit l'itération, et quelque soit la tranche en cours d'exécution.

2) *Au niveau fonctionnel*. Le générateur d'adresses tient compte du numéro de la tranche de pipeline et de l'index de l'exécution pour déterminer l'adresse de *a* comme le spécifie la Figure 4-51. A la fin de chaque cadence, l'indice d'exécution est incrémenté modulo N. Le numéro de tranche est spécifié statiquement par le chronogramme donc la valeur *N-tranche* est aussi une constante. Une adresse de variable est constituée par un numéro de banc et une adresse dans ce banc.

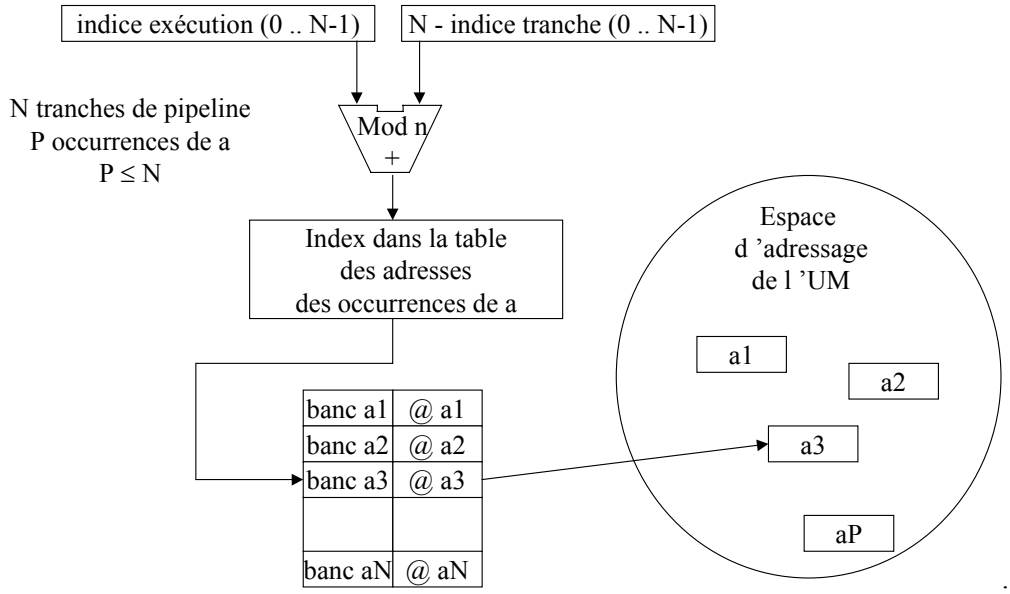


Figure 4-51 : Générateur d'adresses de variables à occurrences multiples

4.3.2. Duplication de variables rebouclées

La figure Figure 4-52 illustre le chronogramme des lectures/écritures d'une variable rebouclée *a* dans un algorithme hypothétique. « *E(a)* » signifie dans ce cas écriture de la valeur de rebouclage et non pas mise à jour de la variable *a*. Cette écriture est ordonnancée au plus tôt pour permettre à la première tranche du pipeline de l'exécution suivante d'en disposer rapidement. A partir de l'instant d'écriture de la valeur de rebouclage, les lectures suivantes doivent toujours obtenir la valeur précédente, et cela quelque soit le nombre de lectures. A partir d'une écriture de valeur de rebouclage il y a donc deux occurrences de la variable si des lectures la suivent dans le chronogramme. De plus, le repliement du pipeline fait apparaître le même phénomène que pour les variables non rebouclée.

L'allocation et la génération des adresses des occurrences des variables rebouclée est identique à celle des variables non rebouclées et l'adresse d'écriture de la valeur de rebouclage fait intervenir l'indice d'exécution suivant au lieu du courant.

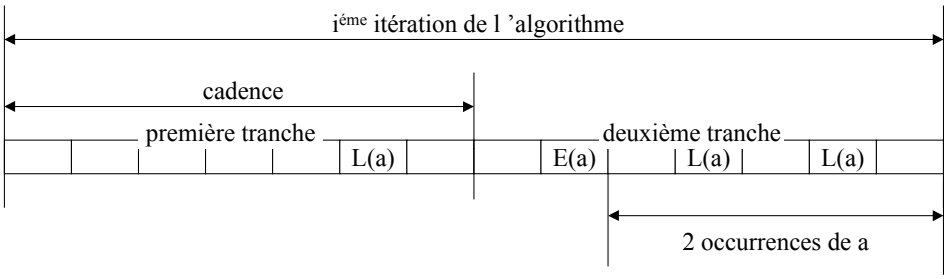


Figure 4-52 : Variable rebouclée candidate à la duplication

4.3.3. Impact du pipeline sur les variables vieillissantes

Nous rappelons que la gestion des vecteurs vieillissants est faite à l'aide de buffers circulaires avec ajout en tête. Ainsi, chaque itération de l'algorithme lit les échantillons stockés dans un buffer et y dépose de nouvelles valeurs. Le déplacement d'une base (modulo la taille du vecteur) implante un vieillissement de toutes les variables en un seul cycle et évite les nombreux transferts de données qui seraient nécessaires sinon.

L'impact du pipeline est qu'il y a autant de « têtes » (bases) de séquences actives qu'il y a de tranches. Chaque tranche dispose alors de sa base de calcul (pointeur sur un buffer circulaire) pour l'adressage des échantillons. Il faut donc que la séquence soit augmentée d'autant de cases mémoires qu'il y a de bases supplémentaires (moins une). La Figure 4-53 illustre une séquence de vieillissement de cinq échantillons qui est étendue de 3 cases pour supporter un pipeline de 4 tranches. Elle illustre que quatre itérations du même algorithme ont bien des visions simultanées et différentes des variables vieillissantes *X(0)* à *X(4)* stockées dans une zone mémoire totale de 8 cases.

Pour un seul vecteur vieillissant : le nombre total de cases est égal à la longueur du vecteur vieillissant augmenté du nombre de tranches en cours d'exécution et le nombre de bases est égal au nombre de tranches.

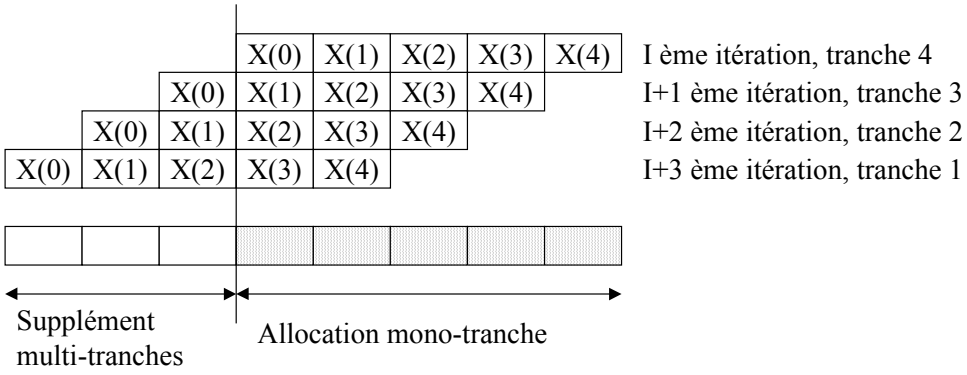


Figure 4-53 : Multiples occurrences d'une séquence de vieillissement

Les quatre bases nécessaires sont représentées par la Figure 4-54. Les positions respectives des bases restent constantes, on peut donc implanter un calcul d'adresse basé indexé à partir d'une seule base de référence (celle de l'itération la plus ancienne encore en cours de calcul) et d'un décalage proportionnel à la différence des numéros de tranches.

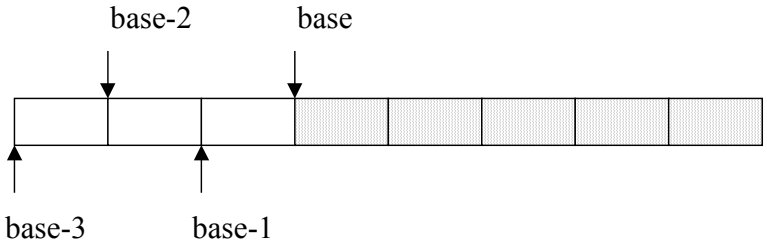


Figure 4-54 : Calcul des bases à partir de la base de référence

Le modèle d'implantation de l'unité de mémorisation est donc adapté comme suit :

1) *Au niveau structurel*. Allocation d'autant de cases mémoires en amont de la séquence de variables vieillissantes qu'il y a de tranches supplémentaires. La même technique d'allocation que pour les duplications de variables rebouclées ou non s'applique.

2) *Au niveau fonctionnel*. Génération d'adresses telle que spécifiée par la Figure 4-55. Ce générateur de base d'accès à une séquence de vieillissement a en commun avec le générateur d'adresses d'occurrences multiples la partie de calcul du décalage tranche/exécution. Ce décalage est fonction de l'indice de l'exécution en cours et de l'indice de la tranche concernée. Il y a une base par séquence d'échantillons.

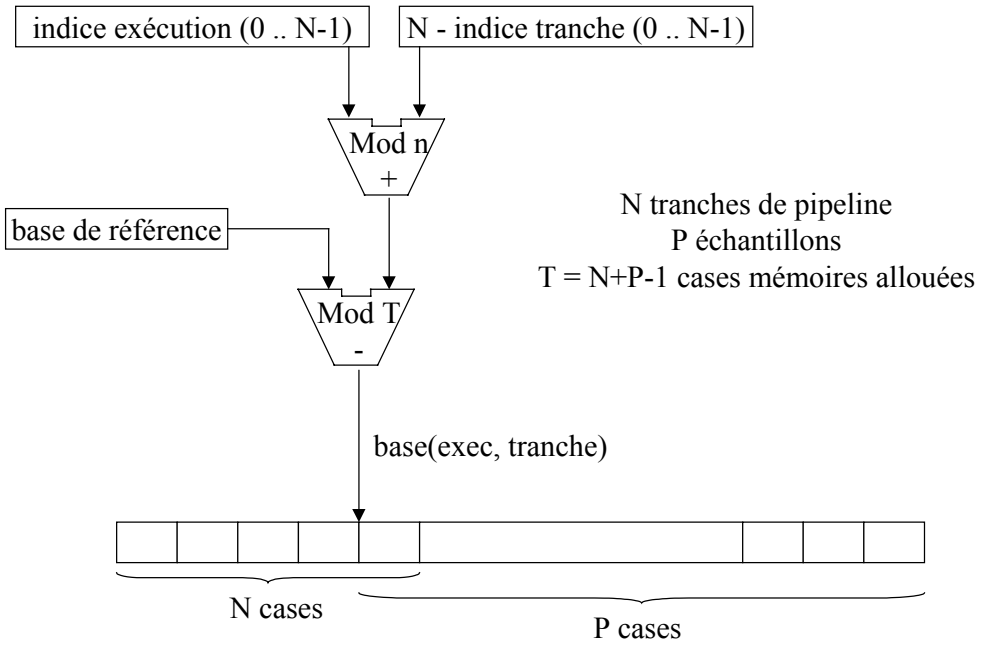


Figure 4-55 : Générateur de base d'accès a une séquence vieillissante

4.4. Conception de l'unité de mémorisation

4.4.1. Modèle structurel du circuit insensible à la latence avec UM

Le support d'architectures pipelinées nécessite une unité de mémorisation adaptée telle que la spécifie notre contribution. Le nouveau modèle de composant suspensible avec unité de mémorisation est spécifié par la Figure 4-56.

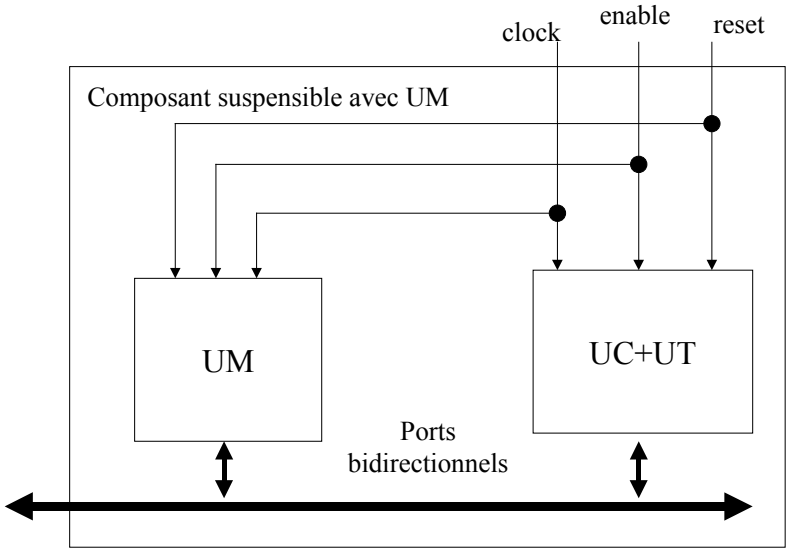


Figure 4-56 : Modèle structurel du composant suspensible avec UM

4.4.2. Modèle structurel de l'UM

Le modèle d'architecture de l'unité mémoire est décrit par la Figure 4-57. L'UM est un assemblage de trois types de composants de base : le banc mémoire, le port de communication synchrone avec les bus de l'UT et le générateur d'adresses. Il y a autant de bancs que de bancs mémoires ont été spécifiés à l'aide de l'interface graphique de saisie des contraintes mémoires. Il y a autant de ports qu'il y a de ports bidirectionnels dans le circuit suspensible. Il y a un seul générateur d'adresses.

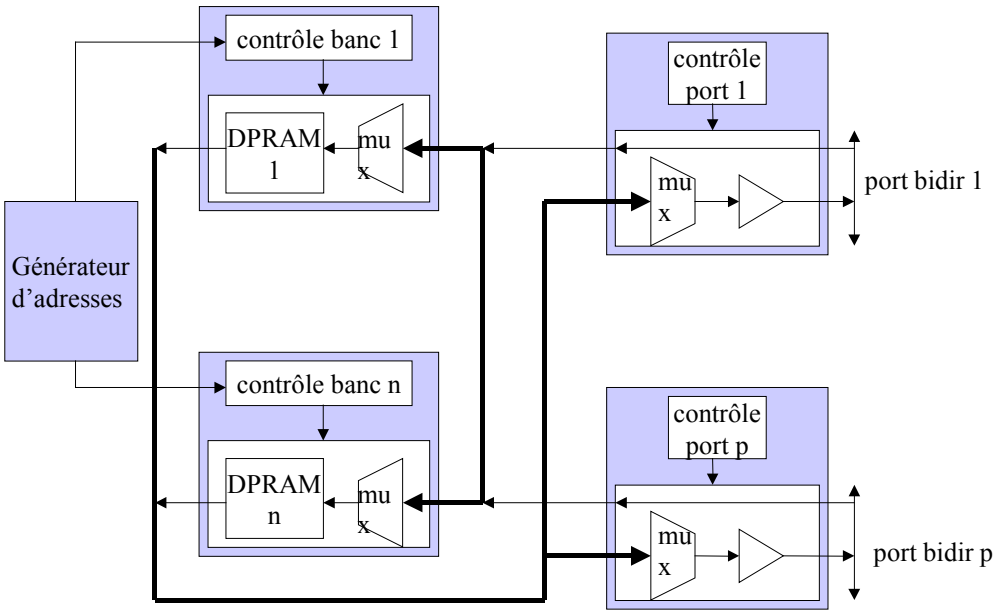


Figure 4-57 : Modèle d'architecture de l'unité de mémorisation

Chaque banc a en entrée (pour les écritures en mémoire) les sorties de tous les ports. Ainsi les données produites par l'UT sur un ou plusieurs bus peuvent-elles être écrites simultanément dans un ou plusieurs bancs mémoires.

Chaque port a en entrée (pour les lectures en mémoire) les sorties de tous les bancs. Ainsi les données consommées par l'UT sur un ou plusieurs bus peuvent-elles être lues à partir d'un ou plusieurs bancs simultanément.

Le générateur d'adresses est relié à chaque banc et lui indique à chaque cycle s'il doit faire une écriture et/ou s'il doit faire une lecture ainsi que les adresses d'écriture et de lecture.

Les bancs et les ports ont une architecture similaire. Ils sont constitués en entrée d'un multiplexeur qui sélectionne la source pertinente à chaque cycle et d'une partie aval constituée d'une DPRAM dans les bancs et de buffers trois états dans les ports.

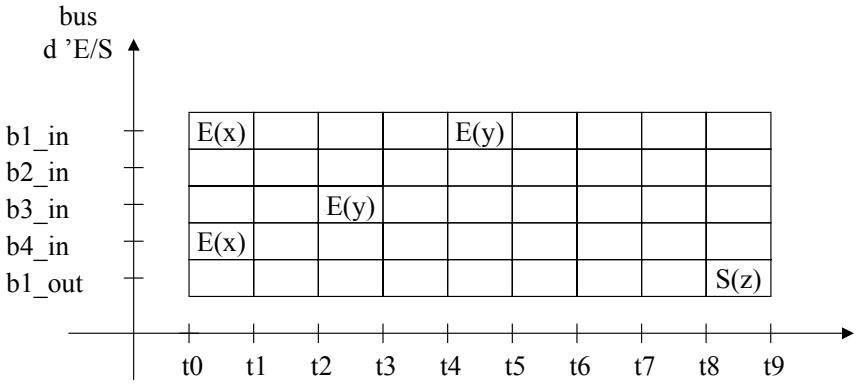
On peut construire, à l'aide de ces trois types de blocs, toute UM constituée d'un nombre quelconque de bancs mémoires et interfacée avec un nombre quelconque de ports. L'adaptation à l'ordonnancement des entrées/sorties s'obtient grâce à la FSM du générateur d'adresses.

La machine d'états finis des ports contient les signaux d'autorisation des buffers trois états ainsi que les signaux de pilotage du multiplexeur d'entrée. Elle permet de rediriger en parallèle les sorties des bancs mémoire vers tous les ports d'entrée.

La machine d'états finis des bancs contient la génération des signaux de contrôle et des adresses de la DPRAM, ainsi que les signaux de pilotage du multiplexeur d'entrée.

4.5. Extension de l'unité de mémorisation à la duplication des entrées

Les circuits générés par GAUT peuvent avoir besoin de plusieurs occurrences d'une même entrée alors que cette dernière est spécifiée en exemplaire unique dans l'interface fonctionnelle de l'algorithme. La Figure 4-58 illustre ce phénomène pour un algorithme qui s'exécute en neuf cycles d'horloge et pour lequel l'entrée *x* doit être présentées au même moment sur deux bus différents, et l'entrée *y* doit être présentée à deux instants distincts.



E(x) = entrée de x
S(z) = sortie de z

Figure 4-58 : Chronogramme avec entrées multiples

Ce type de chronogramme correspond à l'activité, en terme d'entrées-sorties, que l'algorithme suivant pourrait avoir après synthèse :

```
A := x + 1 ;
B := x * 3 ;
…
C := y * 2 ;
…
D := (2 * c) + y ;
…
Z := A + (B * C) + D ;
…
```

Figure 4-59 :Exemple d'algorithme ayant des entrées-sorties x, y et z

Le rôle de l'unité de duplication (UD) est de dupliquer localement les entrées dont le circuit nécessite plusieurs occurrences. La duplication peut prendre deux formes : 1) une entrée doit

être présentée sur plusieurs bus d'entrée au même moment et 2) une entrée doit être présentée plus tard sur un bus d'entrée. Le premier cas est résolu par l'ajout de buffers trois états qui vont d'un port vers l'autre et laissent passer à volonté la valeur en entrée du premier vers le second. Le deuxième cas est résolu par allocation d'un registre (ou d'une case mémoire supplémentaire) qui mémorise la valeur d'entrée entre les accès. Le modèle structurel du circuit suspensible avec UT, UM et UD est spécifié par la Figure 4-60. Le modèle d'architecture de l'unité de duplication est spécifié par la Figure 4-61.

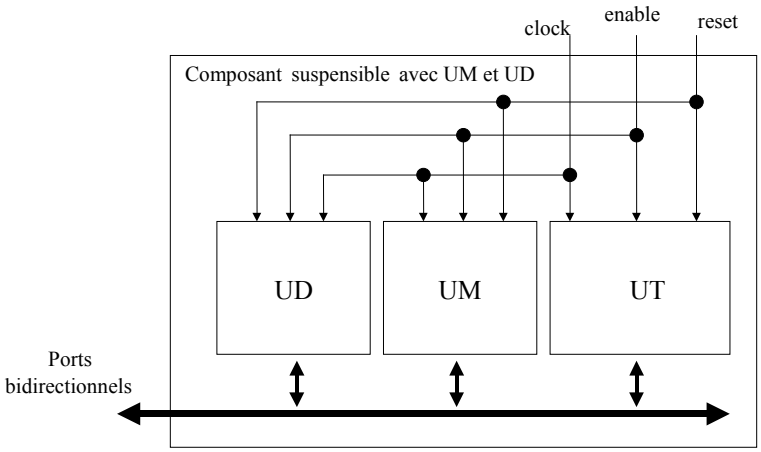


Figure 4-60 : Modèle structurel du composant suspensible avec UM et UD

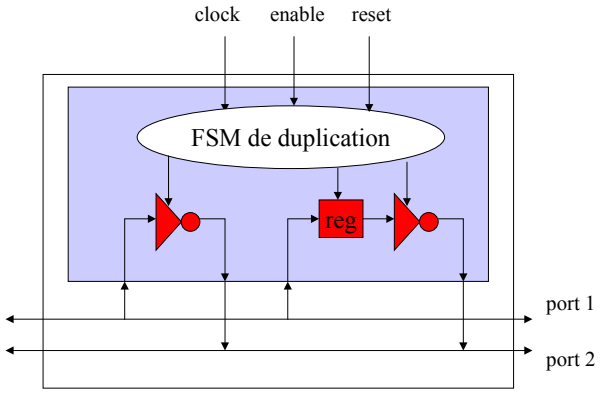


Figure 4-61 : Modèle d'architecture de l'unité de duplication

Le modèle d'architecture de l'UD est tel qu'il permet de propager les entrées du port 1 vers le port 2 et de mémoriser les entrées du port 1 et de les présenter plus tard au port 2. Ce modèle est étendu à toutes les entrées nécessitant duplication en réutilisant le matériel de façon à ne pas dupliquer des chemins déjà établis. La FSM de duplication pilote les signaux des registres et des buffers trois états en fonction du chronogramme des entrées/sorties et des duplications associées. On constate que, hormis la propagation instantanée des données d'un bus vers un autre, la création de registres « de stockage intermédiaire » est équivalente à l'ajout de cases mémoires dans des bancs mémoires de l'unité de mémorisation. En conséquence, l'implantation finale de l'unité de duplication ne contient que les buffers trois états, et les registres (les cases mémoires) sont alloués dans les bancs de l'UM de la même façon que pour les variables à occurrences multiples. Le traitement de ce cas particulier d'entrées-sorties entre dans le cas général de la duplication de variables.

5. Conclusion

Dans ce chapitre nous présentons nos contributions en terme d'introduction de la synthèse de haut niveau dans la composante système de la plate-forme de prototypage rapide PALMYRE. Grâce à la synthèse matérielle automatique, des prototypes de systèmes composés de tâches spécifiées au niveau algorithmique peuvent être automatiquement raffinés vers une implantation matérielle de type FPGA. Pour cela, nous avons introduit la théorie des LIS dans l'outil GAUT et avons proposé une nouvelle architecture d'unité de mémorisation.

Dans une première partie, nous décrivons comment nous construisons les processus patients de cette théorie à partir des composants synthétisés et comment notre architecture à base de *processeurs de synchronisation* permet d'implanter efficacement la suspensibilité ainsi que l'insensibilité à la latence requises. Nous prouvons par l'expérience et à l'aide de mesures comparatives avec les FSM de Singh [SIN04], que des gains importants en surface et en vitesse sont obtenus grâce à notre modèle d'implantation de type CFSMD micro-codées. Ces *processeurs de synchronisation* ont de plus l'aptitude à s'interfacer naturellement avec tout environnement qui serait par nature asynchrone. Nous exploitons ce fait pour la synthèse des interfaces d'extrémités. Ces dernières nous permettent de produire des circuits qui communiquent avec les canaux de communications de la plate-forme matérielle.

Dans une deuxième partie, nous présentons quelles adaptations sont nécessaires dans l'unité de mémorisation pour supporter le pipelining d'algorithme que peut introduire GAUT lorsque le système devient si complexes en communication et en calcul que la contrainte de cadence applicative ne peut plus être tenue par la simple mis en œuvre de plus de parallélisme au niveau matériel. La duplication de variables sous diverses formes (duplication des variables rebouclées ou non, allongement des vecteurs vieillissants et duplication des entrées) devient alors nécessaire pour supporter la charge de calcul du circuit pipeliné.

Nous concluons que l'introduction de la théorie des LIS et la nouvelle unité de mémorisation permettent de synthétiser des circuits numériques efficaces en vitesse qui profitent intégralement des possibilités offertes par la synthèse de haut niveau et qui sont exécutables sur notre plate-forme de prototypage rapide.

Le prochain chapitre présente des expériences de mise en œuvre de la synthèse des *processeurs de communication* dans le cadre des projets RNRT ALIPTA et de contrat de plan état/région PALMYRE.

Chapitre 5

Application à la conception d'un modem DVB-DSNG

1. Introduction

GAUT est un logiciel dont la création remonte à 1990, il a maintenant « 14 ans d'age ». Une quarantaine d'utilisateurs, répartis dans 11 entreprises et 29 Universités et Grandes Ecoles de 11 pays différents (Afrique du Sud, Allemagne, Belgique, Canada, Chine, France, Grèce, Inde, Italie, Taiwan, USA) sont recensés. Les ajouts de fonctionnalités et les corrections sont permanents et produisent de une à quatre nouvelles versions de GAUT par an. Enfin, en plus des publications et des contrats, le salon annuel européen DATE est un de nos vecteurs de communication vers les communautés scientifiques et industrielles. Parmi nos contrats, l'un d'entre eux exploite le processeur de synchronisation que nous proposons : le projet ALIPTA.

Nous avons précédemment justifié dans le chapitre 4 l'intérêt de l'introduction de la théorie des systèmes insensibles à la latence dans les circuits numériques synthétisés par GAUT et mesuré les gains de performances en surface et en vitesse obtenus. Nous ne reviendrons donc pas sur ces points que nous considérons comme acquis. Nous appliquons maintenant l'outil GAUT, agrémenté de la synthèse des nouvelles unités de communication et de mémorisation, à la synthèse d'IPs dans le cadre du projet RNRT ALIPTA. Le projet ALIPTA concerne la conception, l'intégration et la validation des blocs de traitement numérique d'une chaîne de réception DVB-DSNG. Six partenaires géographiquement distants ont participé à ce projet. Cette expérience démontre l'intérêt et l'applicabilité de nos travaux dans un cadre préindustriel qui comporte des rapports de type client-fournisseur où le client est un intégrateur système et les fournisseurs sont, soit des concepteurs d'outils de CAO électronique, soit des concepteurs d'IPs.

2. Projet ALIPTA

Le projet ALIPTA a pour but d'améliorer la méthodologie de conception des systèmes sur puce dédiés à des applications de télécommunication du type modem. Il vise à formaliser le concept de composant virtuel, ou IP, au niveau comportemental. Il est en ligne avec la tendance actuelle qui consiste à encourager la réutilisation de blocs matériels entre projets et l'élévation du niveau d'abstraction de la conception. En tant que projet RNRT, il rassemble des partenaires académiques et industriels.

2.1. Présentation des partenaires

Les partenaires du projet sont les sociétés Arexsys, Thales, Sacet et Turboconcept, ainsi que l'ENST-Bretagne et le laboratoire LESTER.

- Arexsys, société Grenobloise, est une PME de quelques dizaines de personnes qui est spécialisée dans l'édition de logiciels dédiés à la co-simulation (CosiMate) et le co-design (ArchiMate). Sa récente fusion avec la société Valiosys la dote de compétences en vérification formelle. Arexsys est en charge de l'outil PlugIP de synthèse d'interface de communication entre IPs matériels synthétisés par GAUT et un bus de type AMBA.

- Thales est l'un des plus grands groupes mondiaux d'électronique qui sert trois grands marchés : l'aéronautique, la défense et les technologies de l'information. La division Thales

Communications emploie 9000 personnes dans 14 pays. C'est un des premiers fournisseurs mondiaux de systèmes de communications pour les marchés militaires et civils. Son expertise concernant les traitements modem et les communications par satellite constitue une ressource importante en terme de spécification, d'intégration et de validation de systèmes embarqués à fort degré d'intégration. Thales Communication est chargée d'intégrer à son modèle de chaîne de reception DVB-DSNG les IPs de décodage Viterbi et de décodage Reed-Solomon.

- Sacet, société Rennaise, est une PME spécialisée dans les technologies pour applications industrielles de télécommunications sans fil. Elle valide pour les industriels des implantations C ou VHDL issus de travaux académiques. Elle participe activement à un consortium IEEE dans le but de définir une nouvelle norme radio pour boucle locale à 3.5 gHz, principalement sur la couche physique IEEE 802.16.3. Sacet est en charge de la conception du décodeur de Viterbi à 64 états.

- Turboconcept, société Brestoise, est une PME créée sous le parrainage de l'ENST Bretagne et des inventeurs des turbo codes. Son activité est le développement et la commercialisation auprès des fabricants et opérateurs des télécommunications d'IPs de codage correcteur d'erreurs, notamment « turbo codes ». Ses produits pour FPGA sont actuellement développés au niveau RTL générique. Turboconcept est en charge de la conception du décodeur Reed-Solomon.

- L'ENST Bretagne, département électronique, met en œuvre ses moyens modernes de conception de composants électroniques à des fins d'enseignement et de recherche. Ses activités de recherche sont orientées vers le traitement du signal, les communications numériques et le codage de canal ainsi que la conception de circuits numériques et analogiques. C'est, en particulier, l'expertise en codage canal qui est mise conjointement en œuvre avec la société Turboconcept. L'ENSTB, de par son expertise en synthèse de haut niveau avec l'outil GAUT, est en charge de l'assistance à la conception du décodeur Reed-Solomon avec Sacet.

- Le LESTER, équipe Adéquation Algorithme Architecture (AAA), est spécialisée en conception de circuits numériques pour applications de TDSI fondée sur les méthodologies de réutilisation d'IPs synthétisés à l'aide d'outils de synthèse de haut niveau. GAUT est un des outils qui met en oeuvre ce type de méthodologie. Il permet de valider expérimentalement les nouveaux concepts proposés. Le LESTER est en charge de l'outil GAUT et assiste les différents partenaires à la synthèse d'IPs algorithmiques.

La collaboration de ces six partenaires a débuté début 2002. Elle a été initialement prévue sur une durée de deux années et a été prolongée de six mois pour se terminer en fin d'année 2004. Le projet est constitué de trois sous-projets qui sont, pour le premier, l'étude de la spécification et de la synthèse d'IPs avec GAUT, pour le second, l'application de la synthèse d'IPs aux blocs spécifiques d'un modem, et pour le troisième, l'intégration des IPs précédemment développés sur une plate-forme monopuce de type FPGA.

2.2. Les contraintes commerciales du DVB-DSNG

L'application choisie pour le projet ALIPTA est un modem basé sur le DVB-DSNG. La norme EN 301 210 [ETS98] définit un système souple dit « DVB-DSNG » qui s'appuie essentiellement sur le système de radiodiffusion par satellite (DVB-S) qui offre toute une gamme de qualités d'images à divers débits.

Les caractéristiques techniques de ce système DVB sont en grande partie dictées par le marché. La définition des services de type SNG (Satellite News Gathering), contenue dans la recommandation SNG.770-1 de l'UIT-R, est la suivante : ce sont des « *Transmissions temporaires et occasionnelles, à court préavis, de télévision ou de son pour la radiodiffusion, à l'aide de stations terriennes des trajets montants extrêmement mobiles ou transportables fonctionnant dans le cadre du service fixe par satellite* ». Les terminaux DSNG doivent être portables (y compris par avion) ou transportables dans des cars de reportage. Une grande rapidité d'intervention et une relative simplicité de réglage sont nécessaires. Les débits binaires sont d'environ 8 Mbit/s (profil MP@ML du MPEG-2) ou de 8 Mbit/s à 34 Mbit/s (profil 422P@ML du MPEG-2) lorsqu'une meilleure qualité d'image ou des fonctions de montage sont requises. Les temps d'exécution des algorithmes de codage sophistiqués d'aujourd'hui, qui admettent des rapports élevés de compression du débit, doivent être très courts pour autoriser le dialogue des journalistes lors de transmissions en direct.

Enfin, le succès des normes DVB est également dû à l'adoption, sur tous les supports (satellite, câble, réseaux MMDS), de solutions communes pour le codage vidéo/audio et le multiplexage numérique qui rend possible la production en série des circuits intégrés pour les récepteurs/décodeurs domestiques.

Rapidité des calculs et faible coût de production des circuits intégrés sont des contraintes fortes pour les applications DVB-DSNG. Le projet ALIPTA est donc une mise en œuvre commercialement pertinente d'une méthodologie de réutilisation intensive d'IPs synthétisés.

2.3. Caractéristiques techniques générales du DVB-DSNG

Les codages vidéo et audio ne faisant pas partie du modem, nous ne les présentons pas en détail. Il s'agit du format de codage MPEG-2 [MPE04] auquel nous renvoyons le lecteur. En ce qui concerne le multiplexage des services vidéos dans le flux de transport, le système DVB-S a adopté une structure commune de verrouillage de trame, basée sur le multiplex de transport du MPEG-2, avec des paquets de longueur fixe (188 octets) comprenant un octet de synchronisation, trois octets d'en-tête et 184 octets utiles. Cette structure permet un interfonctionnement facile entre canaux de radiodiffusion et réseaux de télécommunications fondés sur des protocoles ATM [ATM04b].

La principale caractéristique du système DVB-DSNG est sa flexibilité. Il permet de choisir, au cas par cas, le système de modulation, le débit symboles et le taux de codage afin d'optimiser le fonctionnement de la liaison par satellite [ETS97]. Il s'agit de réduire le degré d'occupation du spectre du répéteur satellite ainsi que la puissance requise. Pour cela, la modulation MDPQ ainsi que la concaténation des codes convolutionnels et de Reed-Solomon ont été adoptées.

Pour illustrer cette flexibilité nous présentons les différents taux de codage Viterbi permis par la norme dans le Tableau 2-1. On remarque que le taux de codage du codeur/décodeur Viterbi peut aller de 1/2 à 7/8. Pour chacune de ces valeurs [UER98], la valeur minimale du rapport signal sur bruit (bruit gaussien additif) qui permet d'obtenir le niveau de qualité garanti par la norme est donné. Ce niveau de qualité (QEF) est spécifié par un taux d'erreur bit résiduel (TEB) minimum de 2. 10^{-4}avant décodage de Reed-Solomon. On peut donc dynamiquement adapter le taux de codage Viterbi aux conditions de transmission.

Taux de codage Viterbi	RSB requis pour obtenir un TEB de 2 10^{-4} avant RS (dB)
1/2	4,5
2/3	5,0
3/4	5,5
5/6	6,0
7/8	6,4

Tableau 5-1 : Résistance au bruit des différents taux de codage Viterbi

Pour conclure ce bref exposé, nous reprenons une figure extraite de [UER98] qui illustre la meilleure résistance au bruit d'un signal DSNG par rapport à un signal PAL analogique. Cette meilleure résistance est obtenue avec un signal DSNG à 17 Mbit/s et un taux de codage Viterbi de 3/4. Dans ces conditions, un RSB d'environ 3 dB est suffisant par rapport au 12-13 dB qui sont nécessaires en PAL analogique. On peut donc transmettre un signal de même qualité avec moins d'énergie en DVB-DSNG qu'en analogique. Etant donné ces caractéristiques, le numérique peut prétendre mieux acheminer la qualité de son et d'image que les transmissions analogiques.

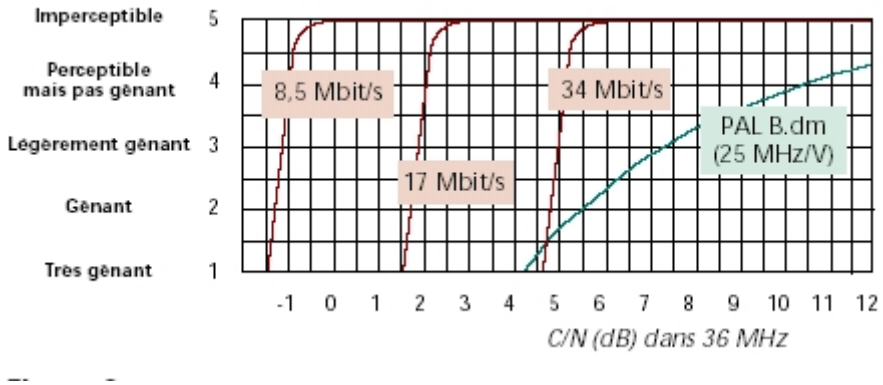


Figure 5-1 : Comparaison des performances entre DVB-DSNG et PAL analogique

2.4. Architecture du modem

La Figure 2-1 décrit la chaîne de traitement qui compose un modem DVB-DSNG. La société responsable de l'intégration (Thales Communications) dispose d'un modèle complet de ce modem. Ce modèle lui sert de référence afin de valider les résultats fournis par les nouvelles implantations matérielles proposées par les concepteurs d'IPs.

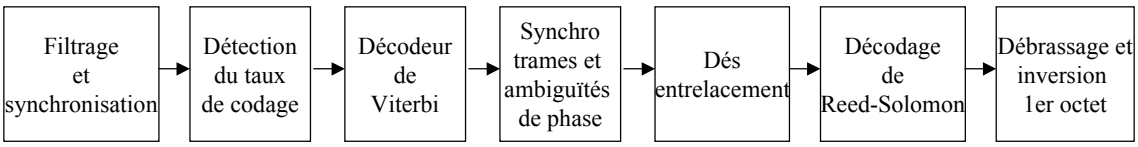


Figure 5-2 : Modem de réception DVB-DSNG : système, partitionnement et placement

La chaîne de traitement est composée de sept fonctions en cascade qui sont :

1) Le filtrage et la synchronisation dont le but est de reconstituer la porteuse et de démoduler le signal entrant.

2) La détection du taux de codage dont le but est de déterminer quel taux de codage a été employé par le codeur de Viterbi émetteur et ainsi adapter dynamiquement le décodeur récepteur.

3) Le décodeur de Viterbi à 64 états (dont la complexité croit exponentiellement avec le nombre d'états) qui assure un TEB de $2.\ 10^{-4}$.

4) La « synchronisation trames » dont le but est de repérer dans le flux octets les débuts des trames de 204 octets.

5) Le désentrelacement des octets de la trame, dont le but est de reconstituer leur ordre initial. Cette étape est nécessaire pour rendre plus résistant aux erreurx de transmission le décodage Reed-Solomon. En effet, les erreurs étant regroupées en salves à la sortie du décodeur de Viterbi, le désentrelacement les disperse sur 12 cellules de 17 octets. Des TEB de l'ordre de $2.\ 10^{-10}$ sont alors obtenus après décodage RS.

6) Le décodage de Reed-Solomon dont le but est de détecter et de corriger les erreurs de transmission. La capacité de correction maximale par trame est de 8 octets faux et la capacité de détection est de 16 octets.

7) Le débrassage dont le but est le désembrouillage des données encapsulées dans les 184 octets utiles des trames MPEG-2.

Les sociétés fournisseuses d'IPs sont Thales Communications pour le « filtrage et la synchronisation », la Sacet pour le décodeur de Viterbi et Turboconcept/ENSTB pour le décodeur de Reed-Solomon. Thales Communications valide l'ensemble de la nouvelle chaine chaîne avant de l'implanter sur un circuit programmable.

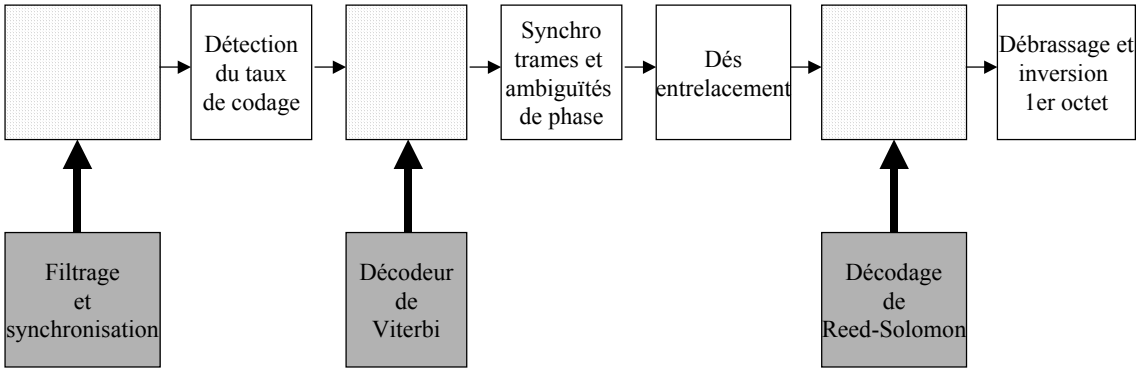


Figure 5-3 : Intégration des nouveaux IPs dans la chaîne de reception DVB-DSNG

2.5. Stratégie de développement des IPs

Les fournisseurs d'IP conçoivent et synthétisent avec GAUT leurs IPs pour la cible technologique retenue et fournissent les algorithmes à l'intégrateur qui peut à son tour les synthétiser et les intégrer dans sa chaîne. La cible technologique retenue pour l'implantation monopuce est le circuit « Excalibur » de la société Altera. La Figure 5-4, extraite de la plaquette commerciale de ce produit, le représente.

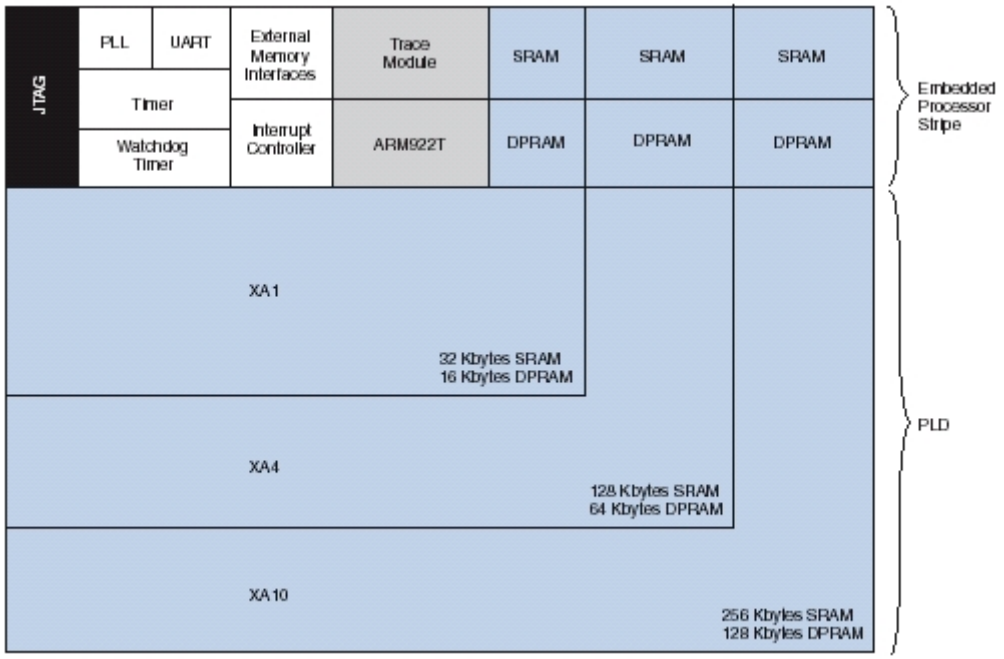


Figure 5-4 : Composition du circuit Excalibur

Les circuits EPXA sont des circuits programmables qui contiennent un APEX 20K, un cœur de processeur ARM 922T (RISC 32 bits à 200 MHz), des mémoires simple port et double ports de type SRAM, des périphériques standards tels une liaison série, un timer, un watchdog (alarme), un module de trace pour l'aide au debug logiciel et un bus AMBA qui interface le processeur avec l'APEX. L'architecture de ce circuit permet d'implanter une chaîne de traitement partiellement en logiciel grâce au processeur ARM et partiellement en matériel grâce à l'APEX. Cette implantation doit fonctionner à une fréquence de l'ordre de 100 MHz. L'implantation finale envisagée de la chaîne est la suivante :

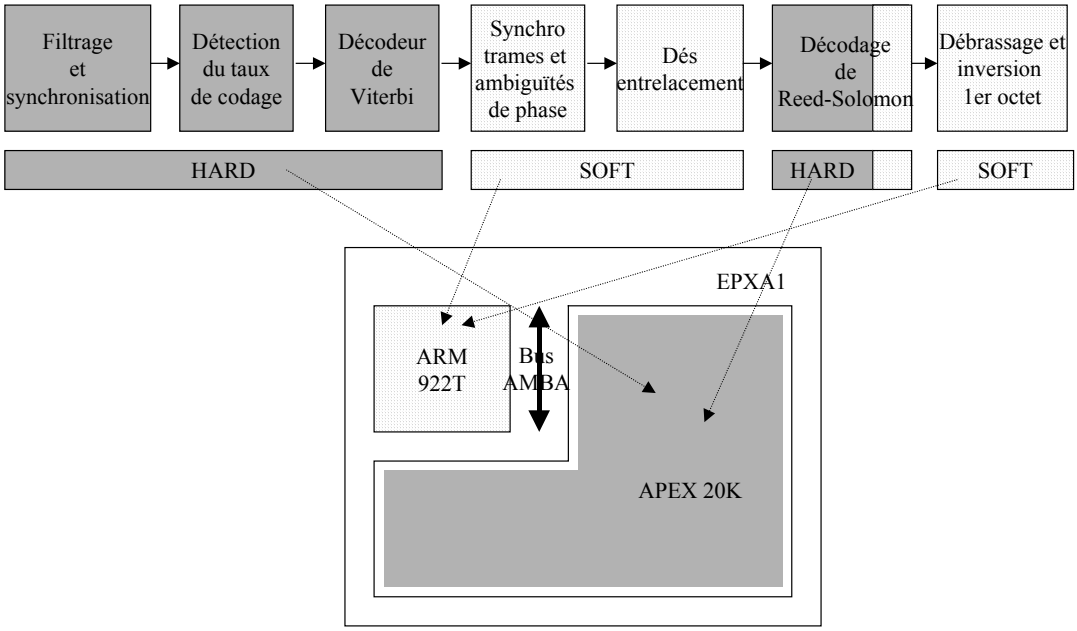


Figure 5-5 : Implantation monopuce envisagée

2.5.1. Filtrage et synchronisation (Thales Communications)

La synthèse de l'algorithme de filtrage et de synchronisation est strictement confidentielle. Nous ne décrirons absolument rien de technique à son sujet. Par contre, les résultats sont extrêmement intéressants, dans la mesure où seule Thales Communications a été en mesure de synthétiser avec GAUT, puis de synthétiser physiquement pour l'EPXA, un circuit numérique fonctionnant à 70 MHz alors que la cible de synthèse était de 100 MHz. Les autres sociétés ont validé par simulation leurs résultats mais, à l'heure de la rédaction, la synthèse physique n'est pas terminée, ni d'ailleurs réclamée pour terminer le projet. L'écart entre les 100 MHz attendus et les 70 MHz obtenus s'explique par le fait que la caractérisation des opérateurs arithmétiques employés par GAUT est un peu trop optimiste. Elle sera refaite dans des conditions plus conformes à l'architecture des cellules de bases des circuits générés. En particulier les temps de traversée des multiplexeurs et des démultiplexeurs seront ajoutés. Néanmoins, cette fréquence de 70 MHz est très largement supérieure aux 40 MHz que les ingénieurs de Thales ont obtenus en développant le circuit de référence en RTL « à la main ».

La conception de ce bloc illustre que, grâce à l'emploi du processeur de synchronisation, la fréquence du bloc n'est pas limitée par les machines d'états nécessaires à sa synchronisation avec les flux entrants et sortants.

2.5.2. Reed-Solomon (Turboconcept/ENSTB)

Le décodeur de Reed-Solomon utilisé dans la chaîne décode le code (204, 188, 8), version raccourcie du code (255, 239, 8) dont les symboles sont des octets. Il est découpé fonctionnellement en cinq sous-blocs : le calcul des syndromes, la détermination des polynômes localisateur et évaluateur d'erreurs, le calcul des racines du polynôme localisateur d'erreurs, le calcul de la valeur des erreurs et la correction des erreurs, comme l'illustre la Figure 5-6 extraite de [JEG03]. Le développement s'est fait en trois phases : écriture d'un modèle de référence en C, écriture et synthèse avec GAUT des algorithmes de chacun des blocs en VHDL comportemental et validation par simulation des codes RTL générés.

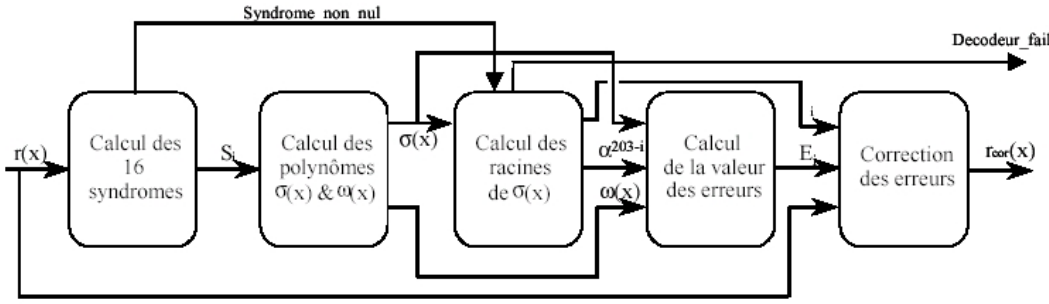


Figure 5-6 : Architecture fonctionnelle du décodeur de Reed-Solomon

L'implantation initiale envisagée est matérielle pour les quatre premiers blocs et logicielle pour le dernier. La raison principale de ce découpage en cinq blocs repose sur le fait que l'algorithme complet est beaucoup trop complexe et génère des architectures trop volumineuses et trop lentes. L'optimisation des fréquences et de la surface nécessite une conception par sous-blocs de tailles plus petites, pour lesquels on maîtrise mieux les performances lors de la synthèse logique et physique.

Ce phénomène s'inscrit dans la démarche de conception des systèmes insensibles à la latence dont le but est de réaliser des systèmes par composition de blocs localement plus performants. L'introduction de la méthodologie LIS dans GAUT permet donc d'envisager le découpage d'un algorithme en plusieurs séquences plus performantes et leur aboutement ultérieur sans pertes de performances dues à la synchronisation inter-blocs.

Nous illustrons maintenant la complexité en terme de communication pour chacun de ces blocs.

- **Bloc de calcul des syndromes**

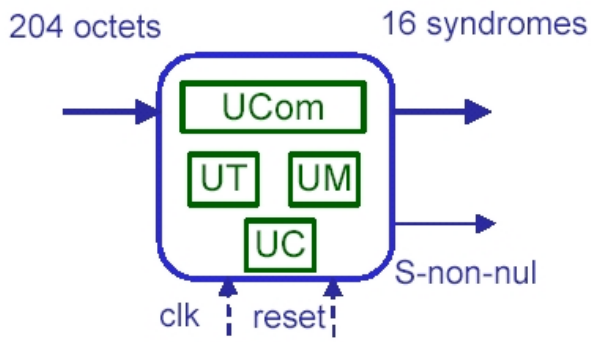


Figure 5-7 : Interface du bloc de calcul des syndromes

Les résultats de synthèse sont les suivants :

Latence (ns)	32700	16730
Nombre d'états de la FSM = nEtats	1671	3269
Nombre de bus = nBus	41	5
Nombre d'entrées-sorties = nES	221	221
Débit atteint (Mbit/s)	50	97,6
FSM équivalente (nSync/nCalc/nSig)	6/279/41	44/74/41
Surface/Vitesse FSM équivalente (slices/MHz)	623/92	1124/92
Surface/Vitesse du processeur équivalent (slices/MHz)	42/103	45/100
Gains en %	-93/+12	-96/+9

Tableau 5-2 : Performances du bloc de calcul des syndromes

On constate que des débits, qui vont quasiment du simple au double, sont atteignables et que le nombre d'états de calcul va de 1671 à 3269 états. La communication avec la chaîne fait apparaître un total de 221 données à recevoir ou à transmettre par itération de l'algorithme sur un nombre de bus qui va d'un minimum de 5 jusqu'à un maximum de 41. Ce bloc est synthétisé avec une unité de communication à base de *processeur de synchronisation* dont nous estimons le gain en surface et en vitesse. Pour cela nous déterminons les caractéristiques de la FSM la plus petite qui assure la synchronisation des 221 entrées-sorties du bloc. Ainsi, en faisant l'hypothèse que les entrées-sorties transitent en un nombre minimum d'états de synchronisation on obtient les règles de calcul suivantes :

$$nSync = nES / nBus \qquad nCalc = nEtats / nSync \qquad nSig = nES$$

- **Bloc de calcul des polynomes**

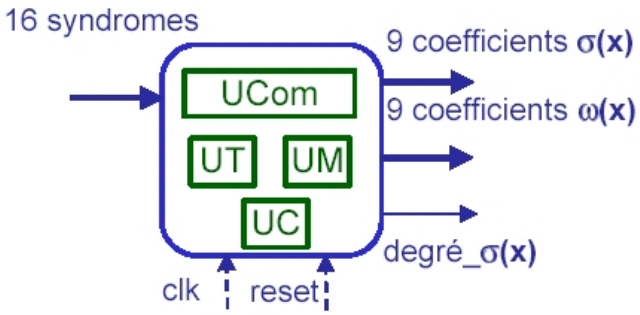


Figure 5-8 : Interface du bloc de calcul des polynomes

De même, pour le bloc de calcul des polynomes, les résultats sont les suivants :

Latence (ns)	3440	2170
Nombre d'états de la FSM = nEtats	343	217
Nombre de bus = nBus	15	6
Nombre d'entrées-sorties = nES	35	35
Débit atteint (Mbit/s)	470	752
FSM équivalente (nSync/nCalc/nSig)	3/114/35	6/36/35
Surface/Vitesse FSM équivalente (slices/MHz)	74/133	80/130
Surface/Vitesse du processeur équivalent (slices/MHz)	34/104	34/104
Gains en %	-54/-22	-58/-20

Tableau 5-3 : Performances du bloc de calcul des polynomes

On constate que, dans ce cas précis, la fréquence du processeur de synchronisation est plus faible que celle de la FSM. Cela est du au fait que la FSM n'est pas suffisamment complexe (3 points de synchronisation seulement) pour que sa vitesse chute en dessous des 100 MHz visés.

- **Bloc de calcul des racines**

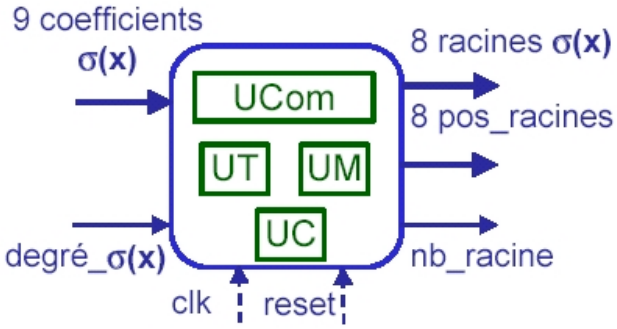


Figure 5-9 : Interface du bloc de calcul des racines

De même, pour le bloc de calcul des racines, les résultats sont les suivants :

Latence (ns)	37890
Nombre d'états de la FSM = nEtats	3789
Nombre de bus = nBus	4
Nombre d'entrées-sorties = nES	27
Débit atteint (Mbit/s)	43
FSM équivalente (nSync/nCalc/nSig)	7/541/27
Surface/Vitesse FSM équivalente (slices/MHz)	1569/87
Surface/Vitesse du processeur équivalent (slices/MHz)	37/100
Gains en %	-98/+15

Tableau 5-4 : Performances du bloc de calcul des racines

On constate une réduction de la surface et un accroissement de la fréquence du wrapper grâce à l'emploi du processeur de synchronisation..

- **Bloc de calcul de la valeur des erreurs**

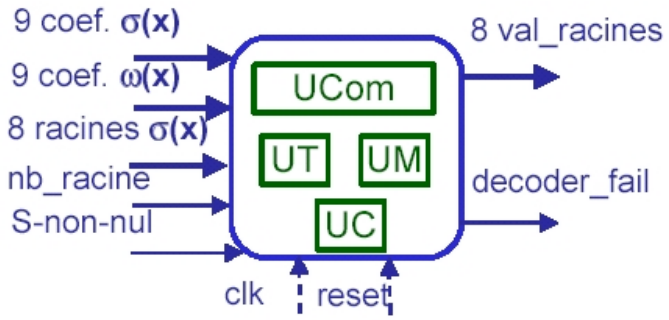


Figure 5-10 : Interface du bloc de calcul de la valeur des erreurs

De même, pour le bloc de calcul de la valeur des erreurs, les résultats sont les suivants :

Latence (ns)	1060
Nombre d'états de la FSM = nEtats	105
Nombre de bus = nBus	18
Nombre d'entrées-sorties = nES	37
Débit atteint (Mbit/s)	1539
FSM équivalente (nSync/nCalc/nSig)	3/35/37
Surface/Vitesse FSM équivalente (slices/MHz)	62/133
Surface/Vitesse du processeur équivalent (slices/MHz)	36/104
Gains en %	-42/-22

Tableau 5-5 : Performances du bloc de calcul de la valeur des erreurs

De même que pour le bloc de calcul des polynomes, la fréquence du processeur de synchronisation est inférieure à celle de la FSM car cette dernière n'est pas assez complexe.

- **Conclusion**

Le Tableau 5-6 résume les résultats obtenus pour le décodeur de Reed-Solomon complet. On constate que, si le *processeur de synchronisation* n'est pas utilisé, alors le wrapper de synchronisation consomme de l'ordre de 2328 à 2835 slices en surface et fonctionne à une fréquence maximale de seulement 92 MHz à cause du bloc de calcul des syndromes qui est le plus lent. Par contre, l'emploi du *processeur de synchronisation* consomme de 149 à 152 slices et fonctionne à une fréquence de 100 MHz, malgré les deux « contre-performances » locales pour les blocs de calcul des polynomes et de calcul de la valeur de l'erreur.

Gains (%)	Syndromes		Polynomes		Racines	Valeur des erreurs
	Cad. min	Cad. max	Cad. Min	Cad. Max		
	1124/92	623/92	80/130	74/133	1569/87	62/133
	45/100	42/103	34/104	34/104	37/100	36/104
Surface	-96	-93	-58	-54	-97	-42
Fréquence	+9	+12	-20	-22	+15	-22

Tableau 5-6 : Résumé des gains obtenus pour le décodeur Reed-Solomon

Le processeur de synchronisation réduit donc la surface du wrapper de synchronisation de deux ordres de grandeur et améliore légèrement la fréquence maximale de fonctionnement. La cible de synthèse étant 100 MHz, nous concluons que le processeur de synchronisation ne pénalise par la synthèse des unités de traitement et de mémorisation et remplit ainsi son rôle.

2.5.3. Viterbi (Sacet)

Le décodeur de Viterbi employé par la norme DVB-DSNG est un décodeur à 64 états supportant les taux de codage de 1/2, 2/3, 3/4, 5/6, 6/7 et 7/8. Il peut fonctionner dans des conditions de densités spectrales différentes et permet d'adapter la puissance de calcul nécessaire à mettre en œuvre (et donc l'aptitude à corriger des erreurs) à la « mauvaise » qualité du canal de transmission satellite. Cette flexibilité est requise car le canal de transmission peut être bruité par des conditions atmosphériques particulières dont les plus contraignantes sont la pluie.

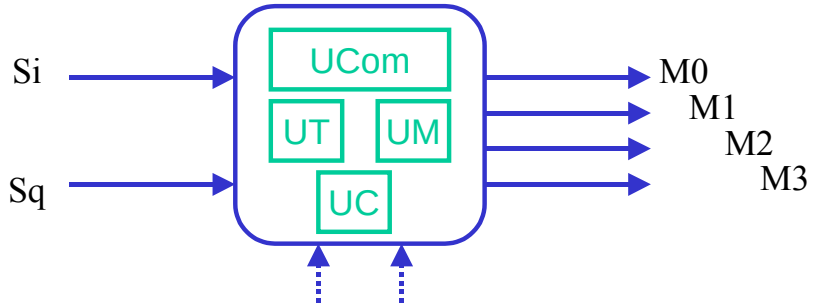


Figure 5-11 : Interface du décodeur de Viterbi

Le décodeur de Viterbi de la Sacet est synthétisable en un seul bloc. Ses entrées sorties sont peu nombreuses : 2 entrées et 4 sorties à chaque exécution. Les résultats sont les suivants :

Latence (ns)	**10000**
Nombre d'états de la FSM = nEtats	699
Nombre de bus = nBus	6
Nombre d'entrées-sorties = nES	6
FSM équivalente (nSync/nCalc/nSig)	3/2/3495
Surface/Vitesse FSM équivalente (slices/MHz)	3002/81
Surface/Vitesse du processeur équivalent (slices/MHz)	25/113
Gains en %	-99/+40

Tableau 5-7 : Performances du décodeur de Viterbi

Le décodeur de Viterbi est un exemple de circuit synchrone spécifié par un seul algorithme. Dans ces conditions, les résultats obtenus prouve que le processeur de synchronisation joue son rôle comme convenu et permet d'optimiser la surface et de préserver la vitesse.

2.6. Autre exploitation possible du découpage en blocs

A l'heure actuelle, l'ensemble des IPs synthétisés est validé par simulation. Seul le bloc de filtrage et de synchronisation de la société Thales Communication a atteint le stade de la synthèse physique pour la cible envisagée.

Il est intéressant de remarquer que la flexibilité de l'interface des IPs synthétisés par GAUT permet de rapidement changer de cible sans remettre en cause la synthèse de haut niveau. En effet les blocs rendus « patients » (processus patients) de la théorie des LIS s'adaptent automatiquement aux asynchronismes introduits par les nouveaux temps de propagation et les horloges locales. Par exemple, si les résultats de synthèse physique sont tels que la totalité des blocs ne peut tenir sur le circuit cible, alors un découpage en plusieurs circuits est possible et ne demande aucune resynthèse de haut niveau. La connectique qui sépare deux blocs synchrones sur le même circuit est identique (aux pads près) à celle qui relie ces deux mêmes blocs s'ils sont placés sur deux puces distinctes. La Figure 5-12 illustre une migration de ce type d'un SoC « impossible » vers un module de type MCM (Multi Chip Module) qui offrirait une plus grande surface de silicium. Ce changement de cible a pour conséquence de modifier la latence de calcul mais pas le débit du système s'il n'y a pas de boucles de retour entre les puces.

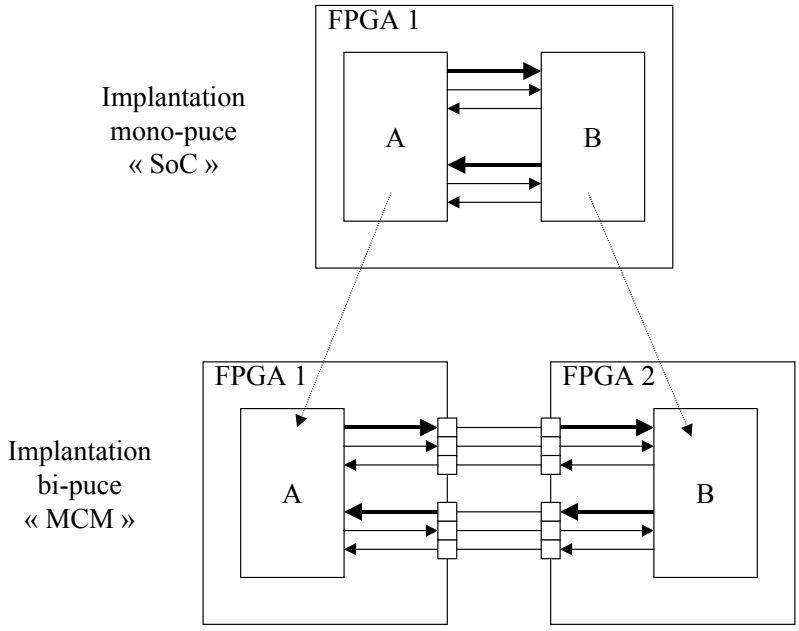


Figure 5-12 : Migration d'un SoC à un MCM

3. Conclusion

Dans ce chapitre nous avons présentés la mise au point d'un prototype de modem DVB-DSNG dans le cadre du projet ALIPTA. Ce projet a pour objectif de tester la validité d'une méthodologie de conception de systèmes qui repose sur la réutilisation d'IP de niveau algorithmique. Le prototype de modem est composé de plusieurs blocs de traitement numériques ayant de fortes contraintes de calcul et de communication.

L'introduction de la théorie des systèmes insensibles à la latence dans les circuits synthétisés par GAUT a permis de synthétiser les blocs de filtrage et synchronisation (Thales Communication), de décodage Viterbi (Sacet) et de décodage Reed-Solomon (Turboconcept/ENSTB) et des les interfacer relativement naturellement entre eux. La validité du prototype a été prouvée par simulation et au moins un bloc a passé avec succès la phase de synthèse logique et physique pour la cible Excalibur (EPXA1) retenue.

L'étude approfondie des résultats de synthèse prouve que des gains importants en surface de l'ordre de 90 % et des gains en vitesse de l'ordre de 10 à 30% (lorsque le nombre d'états dépasse quelques centaines) ont été obtenus grâce à l'implantation de la synchronisation de la communication sous la forme du *processeur de synchronisation* que nous proposons.

Optimisation de la surface, préservation des fréquences optimales des blocs, composition aisée de chaîne de traitements à base de blocs synchrones et possibilité de migration vers une solution de type MCM sont les quatre principaux avantages que le projet ALIPTA aura illustré durant ces deux dernières années.

Conclusions et Perspectives

Nous avons proposé, dans le cadre du projet PALMYRE, une architecture de plate-forme de prototypage rapide qui permet de mettre en oeuvre une méthodologie de conception de circuits au niveau système. Cette méthodologie prône la réutilisation intensive d'IPs, ou composants virtuels de niveau algorithmique, synthétisés avec l'outil GAUT. Elle met en œuvre un flot d'outils de CAO électronique d'origines industrielles et académiques qui assistent les concepteurs lors des activités de spécification du système, de raffinement matériel automatique et de validation par exécution d'un prototype.

Plate-forme de prototypage rapide	Nature des composants	Outils	Cible
Plate-forme système	Algorithmes	MCSE GAUT	Modèle système exécutable Synthèse comportementale
Plate-forme logicielle	RTOS, API Interfaces VHDL	Compilateur C/C++ Synthèse FPGA	Composants logiciel ou matériel aptes à communiquer
Plate-forme matérielle	Calcul: DSP, FPGA Comm.: CP, SDB	Câblage des prototypes	Exécution en temps réel d 'un prototype du système modélisé

Figure 5-13 : Architecture de la plate-forme PALMYRE

Cette plate-forme est composée de trois sous plates-formes. Ce sont les plates-formes système, logicielle et matérielle dont le niveau d'abstraction et la finalité diffèrent.

La plate-forme système s'adresse à deux corps de métiers distincts : les concepteurs d'algorithmes et les architectes systèmes. Les premiers se focalisent sur les calcul numériques intensifs dédiés au traitement du signal et de l'image alors que les seconds assemblent les blocs précédemment conçus. La spécification de niveau algorithmique des IPs est automatiquement raffinée en une représentation de niveau HDL par l'outil GAUT. Elle permet aux concepteurs d'IPs de ne pas avoir à se soucier d'une implantation sur une cible FPGA ou ASIC particulière. Les architectes systèmes, grâce à l'environnement MCSE peuvent combiner les IPs en un modèle système, construire une architecture exécutive et placer le modèle sur l'architecture. Les performances qui résultent de ce placement modèle/architecture sont estimables par simulation. Le modèle est ensuite converti en une implantation exécutable sur les plates-formes logicielles et matérielles. Dans le cadre de cette plate-forme, notre contribution est triple.

Tout d'abord nous avons introduit dans l'outil GAUT la théorie des systèmes insensibles à la latence pour préserver les fréquences optimales de chacun des blocs synthétisés. Nous avons proposé une nouvelle architecture de wrapper de synchronisation que nous nommons *processeur de synchronisation*. Nous avons prouvé ses meilleures performances en terme de surface et de vitesse : des gains en surface de 90 % et des gains de l'ordre de 10 à 30 % en vitesse peuvent être atteints. Nous avons de plus adjoint aux circuits synthétisés par GAUT une nouvelle architecture d'unité de mémorisation multi-bancs qui permet de supporter une méthode d'optimisation inédite : le pipelining au niveau d'algorithme. Cette première contribution permet aux circuits synthétisés par GAUT de fonctionner à la fois dans les conditions les plus contraignantes, mais aussi de naturellement s'adapter aux irrégularités de l'environnement.

Deuxièmement, nous avons proposé une plate-forme logicielle qui repose sur une API C++ de communication. Cette API permet aussi bien d'exécuter un modèle système dans l'environnement MCSE que d'exécuter un prototype sur la plate-forme matérielle. Le raffinement du système en un prototype se fait par simple spécialisation des constructeurs C++ des canaux de communication. Ainsi, les algorithmes écrits par les concepteurs d'IPs au dessus de cette API sont-ils (à l'appel des constructeurs près) totalement indépendants de l'architecture exécutive. Cette API propose, en outre, deux modes de communication : le mode bloquant (synchrone) et le mode non bloquant (asynchrone). Nous avons proposé une méthodologie de caractérisation des performances de notre API qui permet de déterminer, pour une application donnée, quelles sont les conditions matérielles et logicielles optimales en terme d'allocation mémoire, de tailles de paquets à transmettre et de style de programmation à adopter (synchrone/asynchrone). Cette deuxième contribution permet d'exploiter au mieux les performances crêtes de la plate-forme matérielle tout en restant au niveau d'abstraction « composant » grâce à l'API de la plate-forme logicielle.

Troisièmement, nous avons étudié les contraintes en terme de puissance de calcul et de communication des applications de type DVB-DSNG, répertorié les méthodologies de prototypage rapides et inventorié les plates-formes de prototypage actuelles avant de proposer une plate-forme matérielle et de justifier ses meilleures performances. Nous avons sélectionné, pour cette dernière, des nœuds de calculs de type DSP C6x de Texas Instruments et des FPGA de Xilinx connectables entre eux à l'aide de liens de communication ayant des bandes passantes qui vont de 160 Mbit/s à 3,2 Gbit/s. Les performances crêtes de cette plate-forme ainsi que sa flexibilité et son extensibilité en font le meilleur candidat actuel pour le câblage « à la demande » de toute architecture exécutive sur laquelle doit s'exécuter un prototype temps réel d'une chaîne de traitement du signal de nouvelle génération. Cette troisième contribution dote notre plate-forme de prototypage rapide des moyens de calcul et de communication commerciaux les plus performants à l'heure actuelle.

Enfin, nous avons appliqué l'outil GAUT à la synthèse d'IPs dans le cadre du projet RNRT ALIPTA. Ce projet concerne la conception, l'intégration et la validation des blocs d'une chaîne de réception DVB-DSNG auxquels ont participé les sociétés Arexsys, Turbo-Concept, SACET et Thales Communications ainsi que l'ENSTB et le LESTER. Ces expériences ont démontré l'intérêt et l'applicabilité de nos travaux dans un cadre préindustriel qui nécessite la coopération de chercheurs, de concepteurs d'IP virtuels et d'intégrateurs systèmes.

Perspectives

Bien que les résultats obtenus conduisent à des résultats de bonne qualité sur les exemples traités, plusieurs améliorations et extensions peuvent être apportées comme nous l'avons mentionné à plusieurs reprises dans ce mémoire.

En ce qui concerne la plate-forme système, l'apport de la spécification de chaînes de traitement du signal grâce à un outil tel que Matlab/Simulink ne peut être ignoré. L'intégration de Matlab en tant qu'outil d'aide à la spécification d'algorithmes peut se présenter de deux façons différentes : tout d'abord en tant qu'interface de spécification graphique supplémentaire, mais aussi en tant qu'outil d'aide à la spécification système par composition d'algorithmes précédemment développés et synthètisés avec GAUT.

En ce qui concerne la plate-forme logicielle, une meilleure prise en compte des possibilités de calcul en temps partagé des RTOS pourrait être intégrée à l'API. En effet, les notions de styles de communications synchrone ou asynchrone peuvent être unifiées en un style unique (modèle synchrone car le plus simple à programmer) dont le comportement synchrone ou asynchrone serait alors géré de façon transparente par les pilotes de périphériques (drivers) de communication.

En ce qui concerne la plate-forme matérielle, l'adjonction de structures de communication de type bus ou de matrice de commutation permettrait de prototyper des systèmes dont la reconfiguration dynamique serait alors envisageable.

De nombreuses optimisations locales des circuits synthétisés peuvent encore être apportées. Il s'agit, par exemple, de l'extension de la notion de *processeur de synchronisation* au contrôle des unités de mémorisation et de calcul. Le circuit synthétisé par GAUT devient alors un ASIP (Application Specific Integrated Processor), c'est à dire un processeur spécialisé contenant une unité de traitement super-scalaire, pipelinée (au niveau algorithme) et optimisée pour un contexte donné fonction/cadence/entrées-sorties. De même, l'extension du *processeur de synchronisation* aux architectures reconfigurables pourrait aisément se faire à l'aide de RAM au lieu de ROM pour stocker les opérations. Par rechargement de la RAM d'opérations, le scénario de synchronisation serait alors adaptable à tout nouveau comportement de l'IP encapsulé.

Enfin, puisque GAUT nous permet maintenant de synthétiser des processus patients au sens des GALS (suspensibilité et insensibilité à la latence), il serait très certainement bénéfique de mettre en oeuvre de véritables réseaux de communication asynchrone. Ces derniers auraient comme avantage de réduire la surface silicium par élimination de toutes les stations de relais et feraient, par là même, entrer notre outil dans le domaine des SoCs GALS.

Bibliographie personnelle

[BOM04a] J.-P. Delahaye, G. Gogniat, C. Roland, P. Bomel, « Software Radio and Dynamic reconfiguration on a DSP/FPGA platform », in Frequenz, no 58, ISSN 0016-1136, mai-juin 2004.

[BOM04b] S. Huet, P. Bomel, E. Casseau, B. Le Gal, O. Pasquier, « Electronic System Level to Hw/Sw Design Flow », in Proceedings of the International Embedded Solutions Event (GSPx), Santa-Clara, Californie, USA, septembre 2004.

[BOM04c] G. Corre, E. Senn, P. Bomel, N. Julien, E. Martin, « Memory Accesses Management During High Level Synthesis », in Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES'04), Stockholm, Suède, septembre 2004

[BOM04d] B. Le Gal, E. Casseau, P. Bomel, C. Jégo, N. Le Heno, E. Martin, "High Level Synthesis Assisted Rapid Prototyping for Digital Signal Processing", (ICM04), 16th Intl. Conf. on Microelectronics, Tunis, Tunisie, dec. 2004

[BOM05a] P. Bomel, E. Martin, E. Boutillon, « Synchronization Processor Synthesis for Latency Insensitive Systems » accepté à *DATE'05*, Munich, Mars 2005

[BOM05b] P. Bomel, E. Martin, E. Boutillon, "Architecture de wrapper de synchronisation pour environnement de type GALS/LIS", *Journées Francophones sur l'Adéquation Algorithme Architecture* (JFAAA'05) Dijon, Janv. 2005.

[BOM05c] P. Coussy, G. Corre, P. Bomel, E. Senn, E. Martin, « A more Efficient and Flexible DSP Design Flow from MATLAB-Simulink », accepté à ICCASP'05, Philadelphie, USA.

[BOM05d] P. Coussy, G. Corre, P. Bomel, E. Senn, E. Martin, « High-level Synthesis under I/O Timing and memory Constraints », accepté à ISCAS, mai 2005, Kobe, Japon.

Bibliographie

[ABR01] A. Abrial, J. Bouvier, M. Renaudin, P. Senn, P. Vivet, « A New Contactless Smart Card IC using an On-Chip Antenna and an Asynchronous micro-controller », IEEE Journal of Solid State Circuit, vol. 36, n° 7, juillet 2001.

[ACT04] Actel, http://www.actel.com, Mountain View, Californie, USA.

[ADE04] Adelante Technologies, A/RT Designer. http://www.adelantetech.com, Waalre, Hollande.

[ALT04] Advanced Linear devices (ALD), http://www.aldinc.com, Sunnyvale, Californie, USA.

[ALT04] Altera, http://www.altera.com, San Jose, Californie, USA
datasheets des FPGA Stratix II, Stratix GX, Stratix, Cyclone, ACEX.

[ANA04] Anadigm, http://www.anadigm.com, Crewe Cheshire, Royaume Uni.

[APT04] Aptix, www.aptix.com, Sunnyvale, Californie, USA.

[ATO04] Projet ATOMIUM, http://www.imec.be/design/atomium, IMEC.

[ARV04] Arvind, R. S. Nikhil, D. L. Rosenband, N. Dave, « High-level Synthesis : An Essential Ingredient for Designing Complex ASICs », rapport de recherche du Massachussetts Institute of Technology/CSAIL, avril 2004, en attente de publication.

[ATM04] Atmel, http://www.atmel.com/products/CMOS, San Jose, Californie, USA.

[ATM04b] Réseaux ATM, http://atmforum.com

[AXI04] Axix, www.axis.com

[BAG97] A. Baganne, « Methodologie de synthèse des unités de communication matérielles dans une approche de conception mixte logiciel/matériel (co-design) », Thèse de l'Université de Rennes I, décembre 1997.

[BAG98] A. Baganne, J.L. Philippe, E. Martin, « A Formal technique for Hardware Interface Design », in IEEE Trans. on Circuits And Systems, vol. 45, pp. 584-591, 1998.

[BAL93] S. Balemi, P. Kozak, P. Smedinga, « Discrete Events Systems Modeling and Control », Birkhauser, Basel, 1993.

[BAK90] H. B. Bakolu, « Circuits, Interconnections, and Packaging for VLSI », Addison Wesley, 1990.

[BAR00] A. Bardsley and D.A. Edwards. "The Balsa Asynchronous Circuit Synthesis System," Forum on Design Languages 2000 (FDL'2000), Université de Tuebingen, Allemagne, septembre 2000.

[BEE94] M. von der Beeck, « A Comparison of Statecharts Variants », pp. 128-148, Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, Berlin, septembre 1994.

[BER95] R. A. Bergamaschi, R. A. O'Connor, L. Stok, M. Z. Moricz, S. Prakash, A. Kuehlmann, D. S. Rao, « High-level synthesis in an industrial environnement », in IBM Journal of Research and development, vol 39, No 1 / 2, 1995.

[BER00] G. Berry, « The Foundations of ESTEREL », MIT Press, 2000.

[BND96] A. Bender, « Design of Optimal Loosely Coupled Heterogeneous Multiprocessor System », in Proc. of the IEEE Intl. Conf. on Design Automation and Test in Europe (DATE'96), pp. 275-281, 1996.

[BNI95] L. Benini and G. De Micheli, « Transformation and synthesis of FSMs for low-power gated-clock implementation », Int. Symp. on Low Power Design (ISLPED'95), pp. 21--26, Dana Point, Californie, USA, avril 1995.

[BNI02a] L. Benini and G. De Micheli, « Networks on Chip : A New Paradigm for Systems on Chip », Intl. Conf. on Design Automation and Test Europe (DATE'02), Paris, France, mars 2002.

[BNI02b] L. Benini and G. De Micheli, « Networks on Chip : A New SoC Paradigm », IEEE Computer, janvier 2002.

[BNV88] A. Benveniste, B. Le Goff, P. Le Guernic, P. Aubry, « Hybrid Dynamical Systems Theory and the Language SIGNAL », IRISA/INRIA report n°403, avril 1988.

[BNV97] A. Benveniste, P. Le Guernic, P. Aubry, « Compositionality in dataflow synchronous languages : specification and code generation », INRIA report n° 3310, novembre 1997.

[BNV99] A. Benveniste, B. Baillaud, P. Le Guernic, « From synchrony to asynchrony », INRIA report n° 3641, mai 1999.

[BNV00] A. Benveniste, B. Baillaud, P. Le Guernic, « Compositionality in dataflow synchronous languages : specification & distributed code generation », in Information and Computation, vol. 163, issue 1, ISSN 0890-5401, pp. 125-171, novembre 2000.

[BNV03] A. Benveniste, L. Carloni, P. Caspi, A. Sangiovanni-Vincentelli, « Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment », INRIA report n° 4901, aout 2003.

[BOR87] G. Boriello, R.H. Katz, « Synthesis and Optimisation of Interface Transducer Logic », in Proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'87), Santa Clara, Californie, USA, pages 274-277, novembre 1987.

[BRO01] A. Brown, « Optimisation in behavioral synthesis using hierachical expansion : module ripping », article non publié cité dans http://www.ecs.soton.ac.uk/info/people/adb

[BUC94] J.T. Buck, «Static Scheduling and code generation from dynamic dataflow graphs with integer-valued control streams », in Proc. of 28th Conf. on Signals, Systems, and Computers (ASILOMAR'94), Orange Grove, Californie, USA, novembre 1994.

[CAD04] Cadence Design Systems, www.cadence.com, San Jose, Californie, USA
Physically Knowledgeable Synthesis (PKS).

[CAM89] R. Camposano, A. Rosentiel, « Synthesizing circuits from behavioral descriptions », in IEEE Trans. on Computer Aided Design (DAC/ICAS'89), vol 8, no 2, pp. 171-180, Las Vegas, Nevada, USA, juin 1989.

[CAR99a] L. P. Carloni, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli. « A Methodology for Correct-by-Construction Latency Insensitive Design », in Proc. of ICCAD 1999. San Jose, USA.

[CAR99b] L.P Carloni, K. L. McMillan, and A.L. Sangiovanni-Vincentelli, « Latency Insensitive Protocols », in Proc. of the 11th Intl. Conf. on Computer-Aided Verification (CAV),, N. Halbwachs and D. Peled, LNCS 1633,, UC Berkeley, Cadence Design Laboratories, 12, juillet, 1999.

[CAR00] L. P. Carloni and A. L. Sangiovanni-Vincentelli, « Performance Analysis and Optimisation of Latency Insensitive Systems », in Proc. 37th design Automation Conf., IEEE Press, Piscataway, N.J., pp. 361-367, 2000.

[CAR01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, « Theory of Latency-Insensitive Design », IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 20(9):18, septembre 2001.

[CAR02] L.P. Carloni and A.L. Sangiovanni-Vincentelli, « Coping with Latency in SoC Design »,. IEEE Micro, Special Issue on Systems on Chip, 22(5):12, octobre 2002.

[CAR03] L.P. Carloni and A.L. Sangiovanni-Vincentelli, « A Formal Modeling Framework for Deploying Synchronous Designs on Distributed Architectures », First Intl. Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Architectures (FMGALS), UC Berkeley, 21, septembre, 2003.

[CAS01] E. Casseau, C. Jego, E. Martin, « Architectural Synthesis of Digital Signal Processing Applications Dedicated to Submicron Technologies », in IEEE Intl. Conf. on Electronics, Circuits and Systems (ICECS), pp. 533-536, septembre 2001.

[CAS04] M. R. Casu, L. Macchiarulo, « A New Approach to latency Insensitive Design », in Proc.of the Design and Automation Conf. (DAC04), San Diego, Californie, USA, juin 2004.

[CHA84] D. M. Chapiro, « Globally-Asynchronous Locally-Synchronous Systems », PhD Thesis, Stanford University, octobre 1984.

[CHA92] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, A. B. Kahng, « Zero skew clock routing with minimum wirelength », in IEEE Transactions on Circuits and Systems, vol. 39, pp. 799-814, novembre 1992.

[CHA03] A. Chakraborty, M. R. Greenstreet, « Efficient Self-Timed Interfaces for Crossing Clock Domains », 9th Intl. Symp.on Asynchronous Circuits and Systems (ASYNC'03), Vancouver, British Columbia, Canada, mai 2003.

[CHE01] T. Chelcea and S. M. Nowick, « Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols », Design Automation Conference (DAC'01), pages 21-26, Las Vegas, Nevada, USA, juin 2001.

[CHI93] M. Chiodo, P. Giuspo, A. Jureska, « Synthesis of mixed software-hardware implementations from CFSM specifications », in proc. IEEE Intl. Workshop on Hardware/Software Co-Design (CODES), 1993.

[CHP04] ChipExpress, http://www.chipexpress.com, Santa Clara, Californie, USA

[CHO95] P. Chou, R. Ortega, G. Borrielo, « Interface Co-Synthesis Techniques for Embedded Systems », In Proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'95), pp. 280-287, San Jose, Californie, USA, novembre 1995.

[CHU89] C. M. Chu, M. Potkonjak, M. Thaler, J. Rabaey, « HYPER: An Interactive Synthesis Environment for High performance Real Time Applications », in Proc. of the Intl. Conf. on Computer Design (ICCD'89), pp. 432-435, Boston, USA, Massachussets, octobre1989.

[CMC04] CMC, Center for Multimedia Communications, http://cmc.rice.edu, Houston, Texas, USA.

[COF04] CoFluent Design, MCSE Methodologie, http://www.cofluentdesign.com

Nantes, France.

[CO065] J. W. Cooley, J. W. Tuckey, « An algorithm for the machine calculation of complex Fourier series », Mathematics of computation, pp. 297-301, avril 1965.

[CON97a] J. Cong, Z. Pan, L. He, C. Koh, K. Khoo, « Interconnect Design for Deep Submicron Ics », in Proc. of the 1997 IEEE/ACM Intl. Conf. on Computer-aided design (ICCAD'97); Pages: 478–485, San Jose, Californie, USA, 1997, ISBN:0-8186-8200-0

[CON97b] J. Cong, « Challenges and opportunities for design innovations in nanometer technologies », SRC Working papers , Univ. of California, Los Angeles, Californie, décembre 1997.

[COR96] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakolev, « Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers », in IEICE Trans. on Information and Systems, vol. E80-D, no 3, pp. 315-325, mars 1997.

[COR03a] G. Corre, N. Julien, E. Senn, E. Martin, « Contraintes mémoire et solution architecturale pour applications TDSI », Actes du colloque GRETSI, Paris, France, septembre 2003.

[COR03b] G. Corre, N. Julien, E. Senn, E. Martin, « Ordonnancement sous contraintes de mémorisation : une optimisation efficace des ressources lors de la synthèse d'architecture », Actes des Journées Francophones d'Etudes Faible Tension Faible Consommation (FTFC'03), Paris, France, mai 2003.

[COU03] P. Coussy, « Synthèse d'Interface de Communication pour les Composants Virtuels », thèse de Doctorat de l'Université de Bretagne Sud, Lorient, France, décembre 2003.

[COW04] Coware, http://www.coware.com, San Jose, Californie, USA, « N2C ».

[CPU04] Cputech, http://www.cputech.com, Pleasanton, Californie, USA.

[CUE01] F. Cuesta, M. Auguin, E. Gresset, L. Cappela, « CODEF: a System Level Design Space Exploration Tool », in Proc. of the Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'01) 2001, Salt Lake City, UT., May 2001.

[DAV01] J.M. Daveau, G. Fernandes Marchior, T. Ben-Ismail, A. A. Jerraya, « Protocol Selection and Interface generation », in Proc. of IEEE Intl. Workshop on Hardware/Software Co-Design (CODES'01), Copenhague, Danemark, avril 2001.

[DEM88] G. De Michelli, D. Ku, « Hercules, a System for High-Level Synthesis », in Proc. IEEE Intl. Design Automation Conf. (DAC'88), pp. 483-488, Anaheim, Californie, USA, juin1988.

[DEM99] D. Demigny, M. Paindavoine, S. Weber, « A Dynamical Reconfiguration Architecture for Images Real Time Processing », in technique et Science de l'Information, numéro spécial « Reconfigrables architectures », décembre 1999.

[DEY97] S. Dey, S. Bommu, « Performance Analysis of a System of Communicating Processes », in Proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'97), San Jose, Californie, USA, novembre 1997

[DIL98] D L Dill, « Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits », an. ACM Distinguished Dissertation Series, MIT Press, 1989.

[DOB02] D. W. Dobberpuhl, « A 200-MHz 64-b dual-issue CMOS microprocessor », IEEE J. Solid-State Circuits, vol. 27, pp. 1555-1565, novembre 1992.

[DOL04] Dolphin Integration, http://www.dolphin.fr, Meylan, France

« SoC Integration Platform for Logic Virtual Components ViC », 2004,

[DUT03] S. Dutta, « Architecture and Implementation of Multi-Processor SocS for Advanced Set-Top Box and Digital TV Systems », in proc. sof the 16th Symp. on Integrated Circuits and Systems Design (SBCCI'03), Sao Paulo, Brésil, septembre 2003.

[D&R04] Design And Reuse, http://www.us.design-reuse.com , Grenoble, France

« The Catalyst of Collaborative SoC Design through SIP Exchange ».

[EDW03] D. A. Edwards, W. B. Toms, « The Status of Asynchronous Design in Industry », Information Society Technologies Programme, IST-1999-29119, février 2003.

[EIS98] H. Eisenmann, F. Johannes, « Generic Global Placement and Floorplanning », in Proc. of the Design Automation Conf. (DAC'98), San Francisco, Californie, USA, juin 1998.

[ELL00] J.P. Elliot, « Understanding Behavioral Synthesis ; A Practical Guide to High-Level design », Kluwer Academic Publishers, 2000.

[ENS04] Ecole Nationale Supérieure des Télécommunications de Bretagne (ENSTB), www.enst-bretagne.fr, Brest, France.

[EST03] Esterel Technologies, www.esterel-technologies.com, Elancourt, France

[ETS97] ETS 300 421, « Digital broadcasting systems for television, sound and data services ; Framing structure, channel coding and modulation for 11-12 GHz satellite services », ETSI, aout 1997.

[ETS98] EN 301 210, « DVB : Framing structure, channel coding and modulation for DSNG and other contribution applications by satellite », ETSI, juillet 1998.

[FIL93] D. Filo, D. C. Ku, C. N. Coelho, G. De Micheli, « Interface optimization for concurrent systems under timing constraints », in IEEE Trans. on VLSI systems, pp. 268-281, septembre 1993.

[FIS90] J. P. Fishburn, « Clock skew optimization », in IEEE Trans. on Computers, vol. 39, pp. 945-951, juillet 1990.

[FRE96] L. Freund, M. Israël, F. Rousseau, J.-M. Bergé, M. Auguin, C. Belleudy, G. Gogniat, « A codesign experiment in acoustic echo cancellation: Gmdf α », in Proc. of the IEEE Intl. Symp. on System Synthesis (ISSS'96), San Diego, Californie, USA, novembre 1996.

[FRI01] E. G. Friedman, « Clock Distribution Networks in Synchronous Digital Integrated Circuits », in Proc. of the IEEE, vol. 89, no. 5, pp. 665-692, mai 2001.

[FUH99] R.M. Fuhrer, S.M. Nowick, M. Theobald, N.K. Jha, B. Lin, L. Plana, « MINIMALIST : An Environment for the Synthesis, Verification and testabbility of Burst-Mode Asynchronous Machines », Technical Report CUCS-020-99, Computer Science Department, Columbia University, juillet 1999.

[GAJ89] D. Gajski, R. Kuhn, « New VLSI Tools », in Proc. of the IEEE Intl. Conf. on Computer, vol 16, issue 2, pp. 14-17, 1983.

[GAJ92] D. Gajski, N. Dutt, A. Wu, Y. Lin, « High-Level Synthesis : Introduction to Chip and System design », Kluwer Academic Publishers, 1992.

[GAJ94] D. Gajski, F. Vahid, S. Narayan, J. Gong, « Specification and Design of Embedded Systems », Prentice Hall, 1994.

[GAJ97] D. Gajski, et al., « Essential Issues in Codesign », in Hardware/Software Codesign Principles and Practice », Kluwer Academic Publishers, 1997.

[GAJ98] D. Gajski, R. Domer, J. Zhu, « IP-centric Methodology and design with the SpeC Language », NATO-ASI Workshop on System Level Synthesis for Electronic design, Barga, Italie, 1998.

[GAR99] J. Garside, S. Furber, S.-H. Chung, « AMULET3 revealed », in Proc. of the 5th Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC'99), IEEE Computer Society, p. 51-59, Barcelone, Espagne, avril 1999.

[GAR03] Gartner Inc., http://www.gartner.com, Stamford, Connecticut, USA.

[GAU04] Site Internet de GAUT, http://web.univ-ubs.fr/gaut/, Lorient, France.

[GAY99] O. Gay-Bellile, « Architecture programmable pour le décodage canal », thèse de doctorat de l'ENST, Brest, France, 1999.

[GET04] Get2chip, www.get2chip.com (rachetée par Cadence en avril 2003).

[GIL03] F. Gilbert, M. J. Thul, N. Wehn, « Communication Centric Architectures for Turbo-decoding on Embedded Multiprocessors », in Proc. of the Design Automation and Test Europe (DATE'03), Munich, Allemagne, mars 2003.

[GLA00] P. Glaskowski, « Pentium 4 », in Microprocessor Report, vol. 14, no. 8 aout 2000, pp. 10-13.

[GOG98] G. Gogniat, M. Auguin, L. Bianco, A. Pegatoquet, « Communication Synthesis and HW/SW Integration for Embedded System Design », in Proc. of the 6th Intl. Workshop on Hardware/Software Codesign (CODES'98), Seattle, Washington, USA, mars 1998.

[GON96] J. Gong, D. Gajski, S. Bakshi, « Model refinement for hardware/software codesign », in Proc. of the IEEE Intl. Conf. on Design Automation and Test in Europe (DATE'96), 1996.

[GOO89] G. Goosens, J. Vandewlle, H. De Man, « Loop optilisation in register-transfer scheduling for DSP-systems », in proc. of the 26th. Design Automation Conf. (DAC'89), 1989.

[GOS98] W. Gosti, A. Narayan, R.K. Brayton, A. Sangiovanni-Vincentelli, « Wireplanning in Logic Synthesis », in Proc. of the Intl. Conf. On Computer-Aided Design (ICCAD'98), pages 26-33, San Jose, Californie, USA, novembre 1998.

[GUC95] S. A. Guccione, « Programming Fine-Grained reconfigurable Architectures », thèse de doctorat de l'Université d'Austin, Texas, USA, 1995.

[GUE00] P. Guerrier, « Réseau d'Interconnexion pour Systèmes Intégrés », thèse de doctorat, Laboratoire d'Informatique de paris VI (LIP6), 2000.

[GUE03] P. Le Guernic, J.-P. Talpin, J.-C. Le Lann, « Polychrony for system design », INRIA report n° 4715, février 2003.

[GUP03a] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, « Loop Shifting and Compaction for the High-Level Synthesis of Designs with Complex Control Flow », CECS Technical Report #03-14, avril 2003.

[GUP03b] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, « SPARK : A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations », in Intl. Conf. on VLSI Design, New delhi, Inde, janvier 2003.

[HAL91] D. Harel, « Statecharts : A visul formalism for complex systems », Science of Computer programming », pages 231-274, 1987.

[HAL98] J. Hallberg, Z. Peng, « Estimation and Consideration of Interconnection Delays during High-level Synthesis », in Proc. of Euromicro'98, pp 349-356, Vesteras, Suède, aout 1998.

[HAR87] D. Harel, « Statecharts : A Visual Formalism for Complex Systems », Science of Computer Programming, vol. 8, pp. 231-274, juin 1987.

[HAU87] S. Hauck, « Asynchronous design Methodologies : an overview », in Proc. of the IEEE, Vol. 83, No 1, pp 69-93, janvier 1995.

[HAU95] S. Hauck, « Asynchronous design Methodologies : an overview », in Proc. of the IEEE, Vol. 83, No 1, pp 69-93, janvier 1995.

[HNK94] J. Henkel, R. Ernst, U. Holtman, T. Benner, « Adaptation of partitioning and high level synthesis in hardware/software co-synthesis », in proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'94), pp. 96-100, San Jose, Californie, USA, novembre1994.

[HNN94] J. L. Hennessy, D. A. Patterson, « Computer Organization and Design. The Hardware/Software Interface », Morgan Kaufmann, San Mateo, CA, USA, 1994.

[HNN96] J. L. Hennessy, D. A. Patterson, « Computer Architecture : A Quantitative Approach », Morgan Kaufmann, San Mateo, CA, USA, 1996.

[HES99] F. Hessel, P. Coste, P. LeMarrec, N.-E. Zergainoh, J.-M. Daveau, A. A. Jerraya, « Communication Interface Synthesis for Multilanguage Specifications", in IEEE Intl. Workshop on Rapid System Prototyping (RSP'99), pages 15-20, Clearwater, Floride, USA, juin 1999.

[HO99] R. Ho, K. Mai, H. Kapadia, M. Horowitz, « Interconnect Scaling Implications for CAD », in Proc.. Intl. Conf. On Computer-Aided Design (ICCAD'99), San Jose, Californie, USA, novembre 1999.

[HO01] R. Ho, K. Mai, M. Horowitz, « The Future of Wires », in Proc.. of the IEEE, vol. 89, no. 4, avril 2001.

[HOA85] C. Hoare, « Communcating Sequential Processes », Prentice Hall, 1985.

[HUN04] Hunt Engineering, www.hunteng.co.uk, Somerset, Royaume Uni.

[IBM85] « Method of deskewing dapa pulses », IBM tech, Disclosure Bull, vol. 28, pp. 2658-2659, novembre 1985.

[IBM04] IBM, http://www-306.ibm.com, « IBM PowerPC processor SoC reference flow », 2004.

[IEE85] ANSI :IEEE Standard 754-1985, « Standard for binary floating point arithmetic », 1985.

[IEE93] IEEE Standard VHDL Language reference Manual, Std 1076-1993, New York, 1993.

[IEE95] IEEE verilog , Std 1364, New York, 1995.

[IEE98] IEEE Standard VHDL Language reference Manual, Std 1076.6-1998, New York, 1993.

[IEE00] IEEE verilog , Std 1364-2000, New York, 2000.

[IET04] Institut d'Electronique et de Télécommunications de Rennes, http://www.ietr.org, Rennes, France.

[IKO04] Ikos, www.ikos.com (rachetée par Mentor Graphics).

[INR93] Institut National de Recherche en Informatique et Automatique (INRIA), Rocquencourt, France, « Projet SYNCHRONE, Les formats communs des langages synchrones », rapport n° 157, juin 1993.

[INF00] Infineon technologies, www.infineon.com, « VHDL Coding Guidelines », Chapter 1, Application Notes 1, 2000.

[INF04] Infineon technologies, www.infineon.com, Munich, Allemagne « C166S IPEVAL Kit », 2004.

[INS04a] Instat, http://www.instat.com/descriptions/topic-semiconductor.asp Scotsdale, Arizona, USA.

[INS04b] Institut National des Sciences Appliquées de Rennes, www.insa-rennes.fr Rennes, France.

[INT03] Intel, www.intel.com, Santa Clara, Californie, USA.

[ITR03] International technology Roadmap for Semiconductors, Edition 2003.

[JAN92] J. A. Brzozowski, J. C. Ebergen, « On the delay-insensitivity of gate networks », in IEEE Trans. on Computers, 41(11) :1349-1360, novembre 1992.

[JEG00] C. Jego, E. Casseau, E. Martin, « Architectural Synthesis with Interconnection Cost Control », in 15th Design of Circuits and Integrated System Conference (DCIS'00), pp. 507-512, Montpellier, France, novembre 2000.

[JEG03] C. Jego, « Etude de complexité du Reed-Solomon dans le cadre du projet ALIPTA », ENSTB, novembre 2003.

[JEO01] J . Jeon, D. Kim, D. Shin, K. Choi, « High-Level Synthesis Under Multi-Cycle Interconnect Delay », in Proc. of Asia South-Pacific design Automation Conference (ASP-DAC'01), pp. 662-667, Yokohama, Japon, janvier 2001.

[JER97a] A.-A. Jerraya, H. Ding, P. Kission, M. Rahmouni, « behavioral Synthesis and Component Reuse with VHDL », Kluwer Academic Publishers, 1997.

[JER97b] A. Jerraya, M. Romdhani, P. Le Marrec, F. Hessel, P. Coste, C. Valderrama, G. Marchioro, J. Daveau, N. Zergainoh, « Multilanguage Specification for System Design and Codesign », in System Level Synthesis , NATO ASI.

[JER02a] A.-A. Jerraya, D. Borrione, W. Cesario, F. Hessel, E. Martin, F. Rousseau, N.-E. Zergainoh, « Conception de haut niveau des systèmes monopuces », Lavoisier, ISBN 2-7462-0433-9.

[JER02b] A.-A. Jerraya, C. Landrault, E. Martin, M. Renaudin, K. Torki, « Conception logique et physique des systèmes monopuces », Lavoisier, ISBN 2-7462-0434-7.

[KAH74] G. Kahn, « The semantics of a simple language for parallel programming », in Information Processing, pages 471-475, 1974.

[KEU00] K. Keutzer, S. Malik, R. Newton, J. Rabaey, A. Sangiovanni-Vincentelli., « System level Design : Orthogonalisation of Concerns and Platform-Based Design », in IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol 19. No. 12, décembre 2000, pp. 1523-1543.

[KNH01] A. B. Kahng, « Design technology Productivity in the DSM Era », in proc. of the Asai and South Pacific Automation Conferenec (ASP-DAC'01), pp. 443-448, Yokohama, Japon, janvier 2001.

[KNH02] A. B. Kahng, G. Smith, « A New Design Cost Model For the 2001 ITRS », in proc. of the Intl. Symp. on Quality Electronic Design (ISQED'02), p. 190, San Jose, Californie, USA, mars 2002.

[KIM01] D. Kim, J. Jung, S. Lee, J. Jeon, K. Choi, « Behavior-to-Placed RTL Synthesis with Performance-Driven Placement », in Proc. Intl. Conf. On Computer-Aided Design (ICCAD'01), San Jose, Californie, USA, novembre 2001.

[KNA96] D. Knapp, « Behavioral Synthesis. Digital System Design Using the Synopsys Behavioral Compiler », Prentice Hall, 1996.

[KNU99] P. V. Knudsen, J. Madsen, « Integrating Communication Protocol Selection with HardwareSoftware Codesign », in IEEE Trans. on CAD of Integrated circuits and systems, vol. 18, N°8, 1999.

[KUU03] K. Kuusilinna, C. Chang, M. J. Ammer, B. C. Richards, R. W. Brodersen, « Design BEE : A Hardware Emulation Engine for Signal processing in Low-Power Wireless Applications », vol. 3, EURASIP 2003.

[LAH01] K. Lahiri, A. Raghunatha., S. Dey, « System-Level Analysis for Designing On-Chip Communication Architectures », in IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, Vol. 20, N°. 6, June 2001.

[LAT04] http://www.latticesemi.com, Hillsboro, Oregon, USA.

[LEE87] E. A. Lee, D. G. Messerschmitt, « Synchronous data flow », in Proc. of IEEE, 1987.

[LEE91] E. A. Lee, « Consistency in Dataflow Graphs », in IEEE Trans. on Parallel and Distributed Systems, vol. 2, avril 1991.

[LEE98] E.A. Lee and A. Sangiovanni-Vincentelli, « A Framework for Comparing Models of Computation », in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 12, pages 1217-1229. 1998.

[LEH99] H. Lehr, D. Gajski, « Modeling Custom Hardware in VHDL », technical report ICS-99-29, 1999.

[LEI83] C. Leiserson, F. Rose, J. Saxe, « Optimizing Synchronous Circuitry », Journal on VLSI and Computer Systems, pp. 41-67, 1983.

[LES04] Laboratoire d'Electronique des Systèmes Temps Réels http://web.univ-ubs.fr/lester, Lorient, France.

[LIN92] I. Lin, J. A. Ludwig, K. Eng, « Analyzing cycle stealing on synchronous circuits with level sensitive latches », in Proc. of ACM/IEEE Design Automation Conference (DAC'92), pp. 393-398, juin 1992.

[LIN94] B. Lin, S. Vercauteren, « Synthesis of concurrent system interface modules with automatic protocol conversion generation », in Proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'94), San Jose, Californie, USA, novembre 1994.

[LUC04] Lucent, www.lucent.com, New York, New York, USA.

[MAD95] J. Madsen, B. Hald, « An approach to interface synthesis », in proc. of the IEEE Intl Symp. on System Synthesis (ISSS'95), pp. 16-21, Cannes, France, septembre 1995.

[MAG04] Magma, www.magma.com, San Diego, Californie, USA
« The FixedTiming Methodology », « Gain-Based Synthesis : Speeding RTL to Silicon », white papers.

[MAN02] H. De Man, « On Nanoscale Integration and Gigascale Complexity in the post .Com World », in proc. of the IEEE Design Automation and Test Europe Conference (DATE'02), vol. 2, pp. 1530-1591, Paris, France, mars 2002.

[MAR89] A. J. Martin, « Programming in VLSI : From Communicating Processes to delay-Insensitive Circuits », In C A R Hoare (ed.) UT Year of Programming; Institute on Concurrent Programming. Addison-Wesley, 1989.

[MAR90] A. J. Martin, « The limitations to delay-insensitivity in asynchronous circuits », W. J. Dally (ed), Advanced Research in VLSI, pages 263-278, MIT Press, 1990.

[MAR97] A. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, T. Lee, « The design of an asynchronous MIPS R3000 microprocessor », in Proc. of the 17th Conf. on Advanced Research in VLSI (ARVLSI'97), p. 164-181, Ann Arbor, Michigan, USA, septembre 1997.

[MRN92] F. Maraninchi, « The Argos Language : Graphical representation of Automata and Description of Reactive Systems », in IEEE Workshop on Visual Languages, octobre 1991.

[MAR92] E. Martin, J.L. Philippe,
« Noyau de l'outil de synthèse GAUT, rapport B », 1992.

[MAR93] E. Martin, O. Santieys, J.L. Philippe, « GAUT, An Architecture Synthesis Tool for dedicated Signal Processors », in proc. IEEE Intl. European Design Automation Conference (Euro DAC'93), pp ; 14-19, Paris, France, 1993.

[MAR98] G. Martin « Design Methodologies for System Level IP », in proc. of Design Automation Conference (DAC'98), San Francisco, Californie, USA, juin 1998.

[MAR99] G. Martin et co-auteurs, « Surviving the SoC Revolution : A Guide to Platform-Based Design », Kluwer Academic Publishers, ISBN 0-306-47651-7, novembre 1999.

[MAR04] G. Martin, G. Smith, « Paltform-based SoC Design : Point and Counterpoint », in EDN, 8 juillet 2004.

[MAR04b F. Marteil, N. Julien, E. Senn, E. Martin, « A Complete Methodology for Memory Optimization in DSP Applications », in proc. of the Euromicro Symp. on Digital System Design (DSD'04), pp. 98-103, Rennes, France, septembre 2004.

[MAT01] G. Matheron, « Microelectronics and System Design, Main Challenges, Europe's positionning », in Proc. of the Design Automation and Test Europe Conf. (DATE'01), Munich, Allemagne, mars 2001.

[MAT97] D. Matzke,. "Will physical scalability sabotage performance gains?", IEEE Computer, vol. 30, no. 9, pp. 37--39, septembre 1997.

[MBD02] MBDA, E ; Campbell, « Embed with clocked logic », http://www.scism.sbu.ac.uk/ccsv/AciD-WG/Workshop2FP5/Programme/CampbellSlides.pdf , janvier 2002.

[MDA03] B. K. Madahar, I. D. Alston, D. Aulagnier, H. Schurer, M. Thomas, B. Saget, « How Rapid is Rapid prototyping ? ESPADON Results », in EURASIP, vol. 6, 2003.

[MEC96] H. Mecha, M. fernandez, F. Tirado, J. Septién, D. Mozos, « A Method for Area Estimation of Data-Path in High-level Synthesis », in IEEE Trans. on CAD of Integrated Circuits and Systems », pp. 258-265, vol 15, 1996.

[MED02] Meda+ design Automation Roadmap, http://www.medea.org, mars 2002.

[MEN04] Mentor Graphics, www.mentor.com, Wilsonville, Oregon, USA
Precision Physical Synthesis.

[MEI98] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, D. Lindqvist, H. Tenhunen, A. Postula, « Evaluating benefits of Globally Asynchronous Locally Synchronous VLSI architecture », 16th Norchip, pp 50-57, 9-10 novembre 1998.

[MON04] Montek Singh, Michael Theobald, « Generalized Lartency-Insensitive Systems for Single-Clock and Multi-Clock Architectures », in Proc. of the Intl. Conf. on Design Automation and Test Europe (DATE'04), paris, France, février 2004.

[MOO98] P.R. Moorby, D.E. Thomas, « The Verilog Hardware description Language », Hardcover, mai 1998.

[MOS96] V. Moshnyaga, K. Tamaru, « A Floorplan Based methodology for Data-Path Synthesis of Sub-micron ASIC's », in IEICE Trans. on Information and Systems, pp 1389-1395, vol. E79-D, n° 10, 1996.

[MOT02] Motorola, www.motorola.com, Schaumburg, Illinois, USA,
J. Silvey, J. Gumulja, D. Sheu, J. Scott and J. Tong, « Automatic Synthesis Tackles Power Tower », CommsDesign, avril 2002.

[MPE04] MPEG-2, http://mpeg.org

[MUR03] P. Murphy, F. Lou, A. Sabharwal, J. P. Frantz, « An FPGA Based Rapid Prototyping Platform for MIMO Systems », Asilomar Conference on Signals, Systems and Computers, novembre 2003.

[NAK02] M. Nakhle, « ECLIPSE « Environnement intégré en logiciel libre pour la conception, simulation, réalisation et mise au point des systèmes temps réels embarqués en vue d'une plate-forme française ouverte multi-métier », projet RNTL pré-compétitif, 2002
www.telecom.gouv.fr/rntl/projet/Posters-PDF/RNTL-Poster-Eclipse.pdf

[NAL04] Nallatech, www.nallatech.com, Glasgow, Royaume Uni.

[NAR94] S. Narayan, D. Gajski, "Synthesis of System-level Bus Interface", in proc. of the IEEE Intl. European Design Automation Conference (Euro DAC'94), pp 395-399, Grenoble, France, septembre1994.

[NAR95] S. Narayan, D. Gajski, « Interfacing incompatible protocols using interface process generation », in Proc. of the IEEE Intl. Design Automation Conference (DAC'95), pages. 468-473, San Francisco, Californie, USA, juin 1995.

[NES86] J. Nestor, D. Thomas, « Behavioral Synthesis with Interfaces », in Proceedings of the IEEE International Design Automation Conference (DAC'86), pages 112-115, 1986.

[NIC02] G. Nicolescu et al., « Desiderata pour la spécification et la conception des systèmes électroniques », technique et Science Informatiques, 2002.

[NOW94] S. M. Nowick, B. Coates, « UCLOCK: Automated Design of High-Performance Unclocked States Machines », in Proc. ofg the IEEE Intl. Conf. on Computer

Design, pages 434-441, Cambridge, Massachussets, USA, octobre 1994.

[OCP01] Open Core Protocol Specification, www.ocpip.org/socket/ocpspec, 2001.

[OSC04] Open SystemC Initiative, www.systemc.org

[OTT98] R. H. J. M. Otten, R. K. Brayton, « Planning for Performance », Proceedings of the 35th annual Design Automation Conf. (DAC'98), San Francisco, California, USA, juin 1998.

[PAL04] PALMYRE, https://palmyre.univ-ubs.fr

[PAN01] P. Panda, F. Catthoor, N. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandecappelle, P.G. Kjeldsberg, « Data and Memory Optimization techniques for Embedded Systems », in ACM Trans. on Design Automation of Electronic Systems (TODAES), vol. 6, issue 2, pp. 149-206, ISSN 1084-4309, avril 2001.

[PAR88] N. Park, A. C. Parker, « SEHWA : A Software Package for Synthesis of Pipelines from behavioral Specifications », in IEEE Trans. on Computer-Aided Design, vol. 7, no. 3, pp. 356-370, mars 1988.

[PAR01] J. Park, P. C. Diniz, « Synthesis of Pipelined memory Access Controllers for Streamed Data Applications on FPGA-based Computing Engines », in Proc. of the Intl. Symp. on System Synthesis (ISSS'01), Montreal, Québec, Canada, octobre 2001.

[PAR93] I. Park, K. O'Brien, A. A. Jerraya, « AMICAL : Architectural Synthesis Based on VHDL », Synthesis for control dominated circuits, Ed. G . Saucier, Publ. ELSEVIER, 1993.

[PAR99] S. Park, K. Kim, H. Chang, J ? Jeon, K. Choi, « Backward-Annotation of Post-Layout delay Information into High-Level Synthesis Process for Performance Optimization », in Proc. of Intl. Conf. on VLSI and Computer-Aided Design (ICCAD'99), pp. 25-28, San Jose, Californie, USA, octobre 1999.

[PAS98] R. Passerone, J. A. Rowson, "Automatic Synthesis of Interfaces between Incompatible Protocols", in proc. of the IEEE Intl. Design Automation Conference (DAC'98), San Francisco, Californie, USA, juin 1998.

[PHI95] J.L. Philippe, O. Sentieys, J.P. Diguet, E. Martin, « From Digital Signal Processing Specification to Layout », Logic and Architecture Synthesis : state-of-the-art and novel approaches, pp. 307-313, 1995.

[PHI02] Philips, www.semiconductors.philips.com, P87CL888, 8051 Ultra Low Power (ULP) telephony controller, avril 2002.

[PHI04] Philips Research, www.research.philips.com, Einndhoven, Hollande.

[PLA04] Projet Platon, www.telecom.gouv.fr/rnrt/gpp/FichePlaton.htm

[POW04] PowerEscape, http://www.powerescape.com, Palo Alto, Califonie, USA.

[QUA04] J. Quartana, thèse de Docteur le l'INPG, décembre 2004, « Conception de Réseaux de Communication sur Puce Asynchrones : Application aux Architectures GALS ».

[QUI04] QuickLogic, www.quicklogic.com, Sunnyvale, Californie, USA.

[RAD02] A. Radulescu, K. Goosens, « Communication services for networks on siliocon », in « Domain-Specific Processors : Systems, Architectures, Modeling, and Simulation », pp. 275-299, ed. Marcel Dekker, avril 2002.

[RAS96] « RASSP Model Year Architecture Specification », Lockeed Martin, 1996.

[RAW94] J. A. Rawson, « Hardware/Software Co-simulation », in proc. of Design Automation Conference (DAC'94), pp. 439-440, juin1994.

[REN98] M. Renaudin, P. Vivet, F. Robin, « ASPRO-216 : a standard-cell Q.D.I. 16-bits RISC asynchronous microprocessor », in proc. of the 4th Intl. Symp. on Advanced Research in Asynchronous Circuits ans Systems (ASYNC'98), pp. 22-31, San Diego, Californie, USA, juin1998.

[REN99] M. Renaudin, P. Vivet, F. Robin, « A Design Framework for Asynchronous/Synchronous Circuits Based on CHP to HDL Translation », in International Symposium on Advanced Research in Asynchronous Circuits (ASYNC'99), pp. 135-144, Barcelone, Espagne, Avril 1999.

[REN00] M. Renaudin, P. Vivet, P. Geoffroy, « ASPRO : a toy demo », in Proceedings of the 4th AciD Workshop, Grenoble, France, février 2000.

[REN02a] M. Renaudin, « Conception logique et physique des systèmes monopuces », chapitre 2 « La conception logique », Lavoisier, ISBN 2-7462-0434-7.

[REN02b] M. Renaudin, « Conception logique et physique des systèmes monopuces », chapitre 5 « La conception de circuits asynchrones », Lavoisier, ISBN 2-7462-0434-7.

[ROB97] F. Robin, M. Renaudin, G. Privat, N. Van den Bossche, « Un réseau cellulaire VLSI fonctionnellement asynchrone pour le filtrage morphologique d'images », Traitement du Signal, n° spécial, vol. 14, n° 6, Adéquation Algorithme Architecture, p. 655-664, 1997.

[RIP03] E. Rijpkema, K. Goosens, « Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip », in proc. of Design Automation and Test in Europe (DATE'03), Munich, Allemagne, mars 2003.

[SAL01] M. Es Salhiene, L. Fesquet, M. renaudin, Q. T. Ho, F. Lhuillery, « La technologie asynchrone pour la conception d'objets communiquants : une revue », in « Seminaire Objets Communiquants », rapport TIMA, ISRN TIMA—RR-01/10-11—FR, 2001.

[SAN00] A. Sangiovanni-Vincentelli, M. Sgroi, L. Lavagno, « Formal Models for Communication-based Design », in proc. of the Intl. Conf. on Concurrency Theory (CONCUR), Penn State Univ., Pennsylvanie, USA, aout 2000.

[SAN04] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, M. Sgroi, « Benefits and Challenges for Platform-Based design », in Proc of the Intl. Design Automation Conf. (DAC'04), San Diego, Californie, USA, juin 2004.

[SCH01] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. Ramakrishna Rau, D. Cronquist, M. Sivaraman, « PICO-NPA : High-Level Synthesis of Nonprogrammable Hardware Accelerators », rapport interne HP approuvé pour publication externe, HPL-2001-249, octobre 2001.

[SDR04] Software Defined Radio Forum, http://www.sdrforum.org

New Hartford, New York, USA.

[SEM04] SeMaTech International, http://www.sematech.org/resources/modeling , Modèles économiques pour usines de fabrication de semi-conducteurs.

[SHA97] Sharp Corporation,

http://sharp-world.com/products/device/limeup/ic/system-lsi/medeia.html,

« DDMP, Data-Driven Media Processor a non-Von Neumann processor ».

[SHA02] L. Shang, A. S. Kaviani, K. Bathala, « Dynamic Power Consumption in Virtex-II FPGA Family », in Proc. of the 10th ACM International Symposium on Field-Programmable Gate Arrays (FPGA'02), Monterey, Californie, USA, février 2002.

[SHI03] H. Shiba, Y. Shirato, H. Yoshioka, I. Toyoda, « Software Defined Radio Prototype (1) – System Design and Performance Evaluation », in NTT Technical Review, vol. 1, n° 4, juillet 2003.

[SIA03] Semiconductor Industry Association (SIA), www.sia-online.org/pre_statistics.cfm

[SIC03] Semiconductor Industry Capacity Supply Statistics (SICAS), Statistics Report – 3rd Quarter 2003 , SIA.

[SIL04] Silicon Dimensions, www.sidimensions.com, San Jose, Californie, USA Chip2Nite.

[SIN04] M. Singh, M. Theobald, « Generalized Latency-Insensitive Systems for Single-Clock and Multi-Clock Architectures », in Proc. of the design Automation and Test Europe Conf. (DATE'04), Paris, France, février 2004.

[SJO00] Sjogren, A.E. Myers, C.J, « Interfacing synchronous and asynchronous modules within a high-speed pipeline », in IEEE Trans. on VLSI Systems, 8(5) :573-583, octobre 2000.

[SMI98] J. Smith and G. De Micheli, « Automated composition of hard-ware components », in proc. of the IEEE Intl. Design Automation Conf. (DAC'98), San Francisco, Californie, USA, juin 1998.

[SMI04] G. Smith, Gartner Dataquest, « Platform Based Design : Does it Answer the Entire SoC Challenge ? », in Proc. of the design Automation and Test Europe Conf. (DATE'04), Paris, France, février 2004.

[STA97] J. Staunstrup, W. Wolf, « Hardware/Software Co-Design : Principles and Practice », Kulwer Academic Publishers, 1997.

[STM04] STMicroelectronics, http://www.st.com, Genève, Suisse.

[STO77] J. E. Stoy, « Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory ». Cambridge, MA, MIT Press, 1977.

[STO92] A. Stoll, P. Duzy, « High-level Synthesis from VHDL with Exact Timing Constraints », in Proc. IEEE Intl. Design Automation Conf. (DAC'92), septembre 1992.

[SUA04] P. Suaris, D. Wang, N.-C. Chou, « Smart Move: A Placement-aware Retiming and Replication Method for Field Programmable Gate Arrays », Mentor Graphics white papers, 2004.

[SUE94] B. Suessmith, G. PaapIII, « PowerPC 603 microprocessor power management », Communications of the ACM, no. 6, pp. 43-46, juin 1994.

[SUN99a] Sundance, « Comm-Port Interface, Design Specification », QCF51.

[SUN99b] Sundance, « VHDL Sundance Digital Bus Documentation », QCF42.

[SUN04] Sundance, www.sundance.com, Chesham, Royaume Uni

[SUT89] I. Sutherland, « Micropipelines », Communication of the ACM, vol. 32, n° 6, juin 1989.

[SUT99] I. Sutherland, B. Sproull, D. Harris, « Logical Effort : Designing Fast CMOS Circuits », Morgan Kaufmann Publishers, ISBN 1558605576, 1999.

[SUT01] I. Sutherland, S. Fairbanks, « GasP, A Minimal FIFO Control », IEEE Intl. Symp. On Asynchronous Circuits and Systems, pp. 46-53, 2001

[SYN00] Synopsys, www.synopsys.com, Mountain View, Californie, USA
« Why RTL-to-Placed Gates », nov. 2000.

[SYN03] Synplicity, www.synplicity.com, Sunnyvale, Californie, USA
« Gated Clock Conversion with Synplicity's Synthesis Products », Application Note.

[SYN04] Synopsys, www.synopsys.com, Saturn, Efficient and Concurrent Logical and Physical Optimization of SoC Timing, Area and Power.

[SYS04] SystemC, http://www.systemc.org, SystemC Version 2.0.

[TAK97] A. Takahashi, Y. Kajitani, « Performance and reliability driven clock scheduling of sequential logic circuits », in proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC'97), pp. 37-42, Chiba, Japon, janvier1997.

[TAK97b] A. Takamura, M. Kuwaro, M. Imai, T. Fujii, M. Ozawa, I. Fukasaru, Y. Ueno, T. Nanya, « TITAC-2: an asynchronous 32-bits microprocessor based on scalable-delay-insensitive model », in Proc. of the Intl. Conf. on Computer design (ICCD'97), p288-294, p. 208-218, Austin, Texas, USA, octobre 1997.

[TAR98] S. Tarafdar, M. Leeser, Z. Yin, « Integrating Floorplanning In Data-Transfer Based High-Level Synthesis », in Proc. of the Intl. Conference on Computer-Aided Design (ICCAD'98), pp 412-417, San Jose, Californie, USA, novembre 1998.

[TEN03] Tensilica, www.tensilica.com, Santa Clara, Californie, USA
H. Sanghavi, S. Leibson, « Throttle IP Core Power Dissipation ».

[TEN04] Tensilica, J. Sanghavi, H. Deng, T. Lu, « Logic and Physical Synthesis Methodology for High Performance VLIW/SIMD DSP Core ».

[TEX04] Texas Instruments, www.ti.com, Dallas, Texas, USA.

[TNI04] TNI-Valiosys, http://www.tni-valiosys.com, Paris, France.

[TSA93] R.-S. Tsay, « An Exact Zero Skew Clock Routing Algorithm », in IEEE Trans. on CAD/ICAS, vol. 12, no. 2, pp. 242-249, San Jose, Californie, USA, février 1993.

[TIW98] V. Tiwari, « Reducing power in high-performance microprocessors **»,** in Proc. of the 35th annual conf. on Design automation, San Francisco, California, United States, pages 732 – 737, 1998, ISBN:0-89791-964-5.

[TRI04] Triscend, www.triscend.com, Mountain View, Californie, USA.

[TRN04] Transtech, www.transtech.com, Maidenhead, Royaume Uni.

[TRQ04] Traquair, www.traquair.com, Ithaca, New York, USA.

[UBS04] Université de Bretagne Sud, www.univ-ubs.fr, Lorient/Vannes, France.

[UER98] A. Morello, V. Mignone, « Nouvelle norme DVB DSNG et contributions satellite », revue technique UER, 1998, Union Européenne de Radio-Télévision, Genève, Suisse.

[URE04] Université de Rennes 1, www.univ-rennes1.fr , Rennes, France.

[VAS01] M. Vasilko, L. Machacek, M. Matej, P. Stepien, S. Holloway, « A Rapid Prototyping Methodology and Platform for Seamless Communication Systems », in proc. of the 12th IEEE Intl. Workshop on Rapid Prototyping (RSP'01), pp. 70-76, Monterey, Californie, USA, juin 2001.

[VIL03] T. Villiger, H. Kaslin, F. K. Gurkaynak, S. Oetiker, W. Fichtner, « Self-timed Ring for Globally-Asynchronous Locally-Synchronous Systems », in proc. of the 9th Intl. Symp. on Asynchronous Circuits and Systems (ASYNC'03), Vancouver, B.C., Canada, mai 2003.

[VIS03] K. A. Vissers, « Parallel Processing Architectures for Reconfigurable Systems », in in Proc. of the Design Automation and Test Europe (DATE'03), Munich, Allemagne, mars 2003.

[VIT67] A. J. Viterbi, « Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm" » in IEEE Trans. on Information Theory, vol. IT-13, pp. 260-269, avril 1967.

[VSI01] VSI Alliance, « Virtual component interface standard », www.vsi.org.library.specs/summary.html, 2001.

[VSI04] Virtual Socket Interface Alliance, http://www.vsi.org

[WUY04] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, H. De Man, « Global communication and memory optimizing transformations for low power systems », in IEEE Intl.. Workshop on Low Power Design, pp. 203-208, avril 1994.

[WWR04] Wireless World Research Forum, http://www.wireless-world-research.org, « The Book of Vision 200x ».

[XIL04] XILINX, www.xilinx.com, San Jose, Californie, USA
datasheets des FPGA Virtex II Pro X, Virtex II Pro, Virtex II, Virtex-E extended memory, Virtex E, Virtex, Spartan 3, Spartan-IIE, Spartan-II, Spartan/XL.

[XU97] M. Xu, F. J. Kurdahi, « layout-Driven RTL Binding Techniques for High-Level Synthesis Using Accurate Estimators », in ACM Trans. on Design Automation of Electronic Systems », pp 312-343, 1997.

[YEN95] T. Yen, W. Wolf, « Communication Synthesis for Distributed Embedded Systems », in proc. of the IEEE Intl. Conf. on Computer Aided Design (ICCAD'95), San Jose, Californie, USA, novembre 1995.

[YEN96] T. YEN, W. WOLF, « Hardware-Software Co-Synthesis of Distributed Embedded Systems », Kluwer Academic Publishers, 1996.

[YEX04] Y Explorations, http://www.yxi.com, Lake Forest, Californie, USA.

[YOO01] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, A. A. Jerraya, « A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design », in Proc. of Intl. Symp. on Hardware/Software Codesign (CODES'2001), Copenhague, Danemark, avril 2001.

[YOO03] S. Yoo, A. A. Jerraya, « Introduction to Hardware Abstraction Layers for SoC », in Proc. of the Design Automation and Test Europe (DATE'03), Munich, Allemagne, mars 2003.

[YUN93] K. Y. Yun, D. L. Dill, S. M. Nowick, « Practical Generalizations of Asynchronous State machines », in European Conf. on Design Automation (EDAC'93), Paris, France, février 1993.

[YUN96] K. Y. Yun, R. P. Donohue, « Pausible Clocking : A First Step Toward Heterogeneous Systems », in Proc. of the. Intl. Conf. on Computer Design (ICCD'96), Austin, Texas, USA, octobre1996.

[YUN99] K. Y. Yun, D. L. Dill, « Automatic synthesis of extended burst-mode circuits : Part i (specification and hazard-free implementation) », in IEEE Trans. on Computer-Aided Design, 18(2) :101-117, fevrier 1999.

[ZEN04] Zenasis Technologies, www.zenasis.com, Campbell, Californie, USA « Analysis, identification and insertion of design specific ZenCells ».

Acronymes

ACFSM	**Abstract Codesign Finite State Machines**
AFSM	**Asynchronous Finite State Machines**
AL	**Architectural level**
API	**Application Program Interface**
ASIC	**Application Spécific Integrated Circuit**
ATM	**Asynchronous Transfer Mode**
BAN	**Body Area Network**
BCASH	**Bus-Cycle Accurate SHell**
BFM	**Bus Functional Model**
BSP	**Board Support Package**
CA	**Cycle Accurate**
CABA	**Cycle Accurate Bit Accurate**
CLB	**Configurable Logic Bloc**
CPU	**Central Processing Unit**
BDFG	**Boolean-controled Data Flow Graph**

CDFG	**Control Data Flow Graph**
CFSM	**Codesign Finite State Machine**
COTS	**Commercial Off The Shelf**
DDFG	**Dynamic Data Flow Graph**
DFG	**Data Flow Graph**
DPRAM	**Dual Port Randow Access Memory**
DRAM	**Dynamic Randow Access Memory**
DSM	**Deep Sub-Micron**
DSNG	**Digital Satellite News Gathering**
DSP	**Digital Signal Processing**
DVB	**Digital Video Broadcast**
ECFSM	**Extended Codesign Finite State Machines**
FPGA	**Field Programmable Gate Array**
FIFO	**First In First Out**
FSM	**Finite State Machine**
FSMC	**Finite State Machine with Coprocessor**
FSMD	**Finite State Machine with Datapath**
GALS	**Globaly Asynchronous Localy Synchronous**
GAUT	**Générateur Automatique d'Unité de Traitement**
HAL	**Hardware Abstraction Layer**
HCFSM	**Hierachical and Concurrent Finite State Machine**
HDL	**Hardware Descrition Language**

HLS	**High Level Synthesis**
ICE	**In-Circuit Emulation**
ILP	**Instruction Level Parallelism**
IP	**Intellectual Property et Internet Protocol**
ISA	**Instruction Set Architecture**
ISO	**International Standard Organisation**
ISS	**Instruction Set Simulator**
JTAG	**Join Test Acces Group, IEEE 1149.1**
LIS	**Latency Insensitive Systems**
LUT	**Look Up Table**
MCM	**Multi Chip Module**
MIMO	**Multiple Input Multiple Output**
MMDS	**Microwave Multipoint Distribution System**
MPEG	**Moving Picture Expert Group**
MPU	**Multi Processors Unit**
NOC	**Network on Chip**
OCP	**Open Core Protocol**
OFDM	**Orthogonal Frequency Division Multiplexing**
OSI	**Open Systems Interconnection**
OTA	**Over-The-Air**
PLL	**Phase Locked Loop**
QAM	**Quadrature Amplitude Modulation**

QPSK	**Quaternary Phase Shift Keying**
RPC	**Remore Procedure Call**
RTL	**Register Transfert Level**
RTOS	**Real Time Operating System**
SFG	**Signal Flow Graph**
SFSMD	**Super Finite State Machine with Datapath**
SIMD	**Simple Instruction Multiple Data**
SFG	**Signal Flow Graph**
SL	**System Level**
SLD	**System Level design**
SNG	**Satellite News Gathering**
SOC	**System On a Chip**
TDD	**Time Division Duplexing**
TLM	**Transaction Level Model**
VC	**Virtual Component**
VCI	**Virtual Component Interface**
VDSM	**Very Deep Sub-Micron**
VLIW	**Very Large Instruction Word**
VSIA	**Virtual Socket Interface Alliance**
WLAN	**Wireless Local Area Network**
WLM	**Wire Load Model**
ZSCT	**Zero Skew Clock Tree**