THÈSE D'HABILITATION DE L'UNIVERSITÉ PARIS VI

Spécialité : Informatique

Présentée par Ivan Augé
pour l'obtention d'une
habilitation à diriger des recherches
sur les thèmes :

Synthèse de haut niveau
&
Intégration des systèmes matériel/logiciel

soutenue le mardi 2 décembre 2008 devant le jury composé de

Alain Darte	Rapporteur
Alain Greiner	Président
Ahmed Jerraya	Rapporteur
Eric Martin	Rapporteur
Luciano Lavagno	Examinateur
Frédéric Pétrot	Examinateur

Table des matières

Remerciements **5**

Résumé **7**

Abstract **9**

1 Introduction **13**

2 Synthèse de haut niveau **15**
2.1 Introduction . . . 15
2.1.1 Cadre . . . 15
2.1.2 Contribution . . . 16
2.2 Description . . . 17
2.2.1 Le schéma en Y étendu . . . 17
2.2.2 Les phases de la synthèse de haut niveau . . . 17
2.2.3 Problématique générale . . . 20
2.2.4 Les différentes approches . . . 24
2.2.4.1 Les deux types de synthèse de haut niveau . . . 24
2.2.4.2 Les solutions à la problématique . . . 30
2.2.5 ALMA . . . 32
2.2.6 UGH . . . 34
2.3 Conclusion . . . 36

3 Systèmes intégrés **39**
3.1 Introduction . . . 39
3.1.1 Cadre . . . 39
3.1.2 Contribution . . . 40
3.2 Disydent . . . 41
3.2.1 Problématique générale . . . 41
3.2.2 Description . . . 47
3.2.2.1 Principe . . . 47
3.2.2.2 Outils . . . 48
3.2.2.3 Flot de conception . . . 49
3.3 Conclusion . . . 53

4 Compression sans perte adaptée au réseau **57**
4.1 Introduction . . . 57
4.1.1 Cadre . . . 57
4.1.2 Contribution . . . 57
4.2 Description . . . 57

4.2.1 Compression sans perte 57
4.2.2 Compression adaptée au réseau 59
4.3 Conclusion 60

5 Résumé des publications et travaux **63**
5.1 Livres et revues internationales avec comité de programme 63
5.2 Brevet 63
5.3 Publications et conférences internationales avec comité de programme . . . 63
5.4 Thèses 64
5.5 Rapports, Manuels, Logiciels distribués 65
5.6 Encadrement de stages 65
5.7 Autres publications et conférences 66
5.8 Bibliographie générale 66

6 Conclusion **69**

Annexes **71**

A Article sur la synthèse de haut niveau **71**

B Article sur la conception de systèmes intégrés **73**

Remerciements

Je tiens à remercier Alain Greiner pour m'avoir accueilli dans son laboratoire, pour m'avoir témoigné sa confiance en me confiant et en finançant de nombreux thèsards et stagiaires et, pour le temps qu'il a consacré sans compter à de nombreuses réunions parfois houleuses mais toujours constructives pour trouver les meilleures approches aux problèmes de synthèse de haut niveau et de conception de systèmes intégrés.

Je remercie Frédéric Pétrot pour la dizaine d'années de collaboration étroite et constructive que nous avons passée à ASIM/LIP6 pour réaliser UGH et Disydent.

Que Monsieur Alain Darte reçoive ma gratitude pour les nombreuses remarques, corrections et suggestions qui ont permis d'améliorer la qualité de ce manuscrit.

Que les nombreux stagiaires, qui ont presque toujours fait preuve de motivation et de grand investissement personnel dans les tâches que je leur ai confiées, acceptent ma gratitude.

Enfin, je remercie Messieurs Alain Darte, Ahmed Jerraya, Eric Martin et Luciano Lavagno pour l'intérêt qu'ils ont témoigné à mes travaux en participant à ce jury

Résumé

Ce document retrace mon parcours de recherche depuis ma soutenance de thèse en 1990. Il présente des travaux achevés, ainsi que des travaux en cours et d'autres encore à un stade exploratoire. Mes travaux ont porté sur des sujets assez différents.

Le domaine principal est la synthèse de haut niveau générale (non dédiée à un type d'application) sur cible silicium. De 1990 à 1996, elle tourne autour d'un outil ALMA qui ne contenait que les couches basses de la synthèse de haut niveau et qui était basé sur un concept innovatif le *FSM retiming*, contrairement aux autres outils basés sur le *DATAPATH retiming*. De 1996 à 2005, un nouvel outil complet de synthèse de haut niveau, UGH, fut développé reprenant pour les couches basses le concept de *FSM retiming* validé par ALMA. Depuis 2005, j'explore la synthèse de haut niveau sur cible FPGA.
Ces travaux sur la synthèse de haut niveau ont abouti concrètement à des logiciels opérationnels. Ils ont été le sujet d'une quinzaine de stages de M2, 3 thèses et ils ont donné lieu à une bonne dizaine de publications internationales.

La conception de systèmes intégrés sur SoC fut mon activité principale de 1997 à 2002. Elle a abouti à Disydent, une méthodologie permettant à un utilisateur d'explorer l'espace de conception et de cibler rapidement une solution en lui évitant de s'aventurer dans des voies sans issue. Cette méthodologie s'appuie sur 5 outils : un simulateur de réseau de processus de Kahn (DPN), un simulateur CABA de système (CASS), un micro-noyau multi-tâches et multi-processeurs (Mutek), un outil de synthèse de haut niveau (UGH), un générateur de système et configurateur de logiciel (DSG). La conception et la réalisation de Disydent se sont intégrées dans le projet Esprit COSY de 1998 à 2001. Elles se sont faites en étroite collaboration avec Frédéric Pétrot. Lors de cette collaboration, la conception générale (méthodologie, outils, interfaces) fut collégiale, Frédéric Pétrot s'occupa principalement de la réalisation de DPN, CASS et Mutek, pendant que moi-même, je m'occupai de UGH et DSG. Dans la période centrale, nous avons dirigé et coordonné une équipe d'une dizaine de personnes composée de stagiaires de M2, de thésards et d'ingénieurs en CDD.
Ces travaux ont abouti à la chaîne logicielle complète et opérationnelle dédiée à la conception de systèmes intégrés sur Soc. Ils ont été le sujet d'une quinzaine de stages de M2, 3 thèses et ils ont donné lieu à une vingtaine de publications internationales dont je suis co-auteur de six.

En parallèle à ces grands projets, j'ai effectué des travaux de moindre ampleur. La plupart d'entre eux furent l'objet de contrats avec des industriels. Un de ces projets fut un algorithme et son démonstrateur matériel pour la compression de données sans perte adaptée aux transmissions. Dans les transmissions, la compression est un moyen simple d'augmenter le débit d'une ligne. Le problème principal est qu'elle arrive au résultat inverse lorsque le flot de données transmis est déjà compressé.
Dans leur ensemble, ces travaux ont été le sujet d'une dizaine de stages de M2 et la compression de données sans perte aux transmissions a fait l'objet d'un brevet international.

Abstract

This document summarises my research history since my doctorate degree in 1990. It presents both completed projects and current leads. My research covered three separate subjects.

My main field of research is general (i.e : not targeting a specific type of application) high-level synthesis towards silicon designs. From 1990 to 1996 I was part of the ALMA project, a high-level synthesis tool covering only lower-level concepts of high-level synthesis. ALMA introduced the innovative concept of *FSM retiming* while comparable tools were using *Datapath retiming*. From 1996 to 2005, I took a role in the development of the UGH toolchain, a fully-fledged high-level synthesis tool using, for the lower levels of the tool, the FSM retiming concepts validated through ALMA. Since 2005, I am focusing on high-level synthesis for FPGA targets.
These works led to a range of operational solutions. They were the motivation for about 15 end of studies internships, 3 doctorate thesis and to the publication of more than 10 international publications.

From 1997 to 2002, I mostly focused on system implementation for SoC platforms. This led to the development of Disydent, a method for developing circuits through painless design space exploration, allowing the user to detect, and thus avoid, impossible development paths. This method is based on the combined use of 5 different tools : a simulator of Kahn process networks (DPN), a CABA system simulator (CASS), a multiple processor, multiple processor micro-kernel (Mutek), a high level synthesis tool (UGH), a platform generation and configuration tool (DSG).
The design and development tasks for Disydent were integrated in the 'Esprit COSY' project spanning from 1998 to 2001. They were then performed in close collaboration with Frédéric Pétrot. General design choices were performed by all as a group, while the direction of the development effort was split between Frédéric Pétrot for DPN, CASS and Mutek and myself for UGH and DSG. The team working on this project grew to reach, at its peak, 10 persons : interns, doctoral students and engineers on fixed time contracts. This work ultimately delivered a fully operational framework for SoC system design.

At the same time as these projects, other works were carried out on a smaller scale. Most of them were done under a contract with industry. Among them, one was an algorithm and hardware demonstrator for lossless data compression in networking. In networking, compression is an usual way to increase the bitrate over a communication link. The main problem being that the converse occurs reversed when compressed data are transmitted.
On the whole, these works were supported by around ten students as one of their internships and lossless data compression has been internationally patented.

Table des figures

2.1 Espace de conception. 15
2.2 Le schéma en Y étendu. 18
2.3 Gabarit structurel d'un coprocesseur. 18
2.4 Définition de la synthèse de haut niveau. 19
2.5 Cercles vicieux de la synthèse de haut niveau. 21
2.6 Schéma de communication. 22
2.7 Algorithme pour processeur et pour synthèse matérielle. 23
2.8 Un exemple de description pour la synthèse orientée données. 24
2.9 Un exemple de description pour la synthèse orientée contrôle. 25
2.10 Interfaces des différentes synthèses. 27
2.11 Quelques associations de protocole sur un canal pour la synthèse orientée données. 27
2.12 Gabarits physiques des différentes synthèses. 28
2.13 Comportements de la synthèse orientée données. 29
2.14 Vue utilisateur d'ALMA. 32
2.15 Le schéma en Y d'ALMA. 33
2.16 Vue utilisateur de UGH. 35
2.17 Schéma général du logiciel UGH. 35
2.18 Schéma en Y de CGS. 36

3.1 Une application décrite comme un graphe de tâches communicantes. 41
3.2 Illustration de la loi d'Amdahl 42
3.3 Choix de l'architecture de communication du système. 43
3.4 Espérance de déviance et indice de confiance. 45
3.5 Le flot de conception de Disydent. 49
3.6 Transformation d'un source initial en un réseau de processus de Kahn de base. 50
3.7 Parallèlisme à gros grain d'un réseau de processus de Kahn. 51

4.1 Algorithmes de compression sans perte. 58
4.2 Algorithme de compression adapté au réseau. 61

Chapitre 1

Introduction

Ce document présente les axes de recherche auxquels je me suis intéressé durant ces deux dernières décennies. Une partie s'est faite à l'ENSIIE[1], une école d'ingénieur située à Evry dans l'Essonne (91), une autre partie s'est faite au LEP[2], le laboratoire de recherche de la société Philips qui était situé à Limeil-Brevannes dans le Val-De-Marne (94) et qui est démantelé aujourd'hui et dans le département ASIM[3] du laboratoire LIP6[4] de l'université Paris VI. Ces trois lieux géographiques m'ont permis de côtoyer des gens différents, de confronter des points de vue variés. Ceci m'a donné l'occasion de diversifier mes champs d'activités sur des domaines allant de l'informatisation d'examens d'informatique, à la conception de systèmes intégrés sur puce.

Cette grande fourchette d'intérêts peut-être pressentie comme étant trop vaste dans le sens où, comme le dit l'adage, « qui trop embrasse mal étreint ». Elle peut également être vue comme légitime, car elle permet souvent d'élargir l'angle de vue ou de changer la perception d'un domaine et toujours, grâce aux cassures qu'elle impose, de donner le recul nécessaire qui évite le risque de tourner en rond et de décrocher de la réalité de sa spécialité. De plus, la polyvalence apporte une certaine sûreté à l'intuition du chercheur en lui permettant de mieux discerner le possible et en l'empêchant de s'élancer dans des voies sans issue ou sans intérêt.

Enfin les travaux de recherche que j'ai entrepris ont toujours conduit à **des outils opérationnels** pour aboutir à des résultats expérimentaux mesurables afin de prouver les qualités ou les défauts de l'approche. La réalisation de ces outils opérationnels a nécessité de lourds travaux d'ingéniérie que j'ai entrepris personnellement pour une bonne part, l'autre part ayant été réalisée par des étudiants dans le cadre de stages et de thèses.

Un aperçu de ces domaines de recherche détaillés dans les autres partie de ce document est donné ci-dessous.

Conception assistée de circuits intégrés

La conception assistée de circuits intégrés a occupé la majorité de mon temps et constitue de fait, mon domaine principal de recherche.

Mon premier axe de recherche est la synthèse de haut niveau qui consiste à implanter un programme séquentiel sur silicium. C'est une activité qui mixe la compilation et la

[1] Ecole Nationale Supérieure d'Informatique pour l'Insdustrie et l'Entreprise.
[2] Laboratoire d'Electronique de Philips.
[3] Architecture des Systèmes Intégrés et Micro-électronique.
[4] Laboratoire d'Informatique de Paris 6.

micro-architecture. On peut la définir comme un compilateur qui choisit ses variables et ses instructions de base.

La synthèse de haut niveau a été l'objet de mon doctorat en 1990 et l'approche a été validée par un outil appelé ALMA. ALMA a été amélioré et maintenu jusqu'en 1996. A cette date, un nouvel outil de synthèse de haut niveau (UGH) a été réalisé. UGH est opérationnel depuis 2000. Il est encore maintenu et est actuellement adapté à la synthèse de haut niveau sur cible FPGA. Ces travaux sont l'objet du chapitre 2.

Conception de systèmes intégrés

Dans ce domaine, mon second axe de recherche est la conception de systèmes intégrés. Elle consite à implanter une application sur silicium non pas comme un ASIC dédié mais comme une machine standard (processeurs, bus, RAM) avec quelques coprocesseurs dédiés à l'application. L'application initiale tourne alors en partie logiciellement sur les processeurs et en partie matériellement sur les coprocesseurs. La conception de systèmes intégrés est la continuité naturelle de la synthèse de haut niveau pour la réalisation des coprocesseurs. C'est une activité qui mixe l'architecture de machine, le système au sens noyau léger, simulation fonctionnelle et temporelle.

Mon activité dans la conception de systèmes intégrés a commencé en 1997 et a abouti vers 2001 à une suite logicielle complète appelée Disydent. Disydent évolue aujourd'hui pour utiliser SocLib. Ces travaux sont l'objet du chapitre 3.

Compression de données

Au cours de ces décennies, en parallèle à ces grands projets, j'ai effectué des travaux de moindre envergure. Chacun de ces travaux s'est déroulé sur des périodes allant de quelques mois à une année et ont souvent eu une composante innovante. La plupart d'entre eux furent l'objet de contrats avec des industriels.

Un de ces projets fut un algorithme et son démonstrateur matériel pour la compression de données sans perte adaptée aux transmissions. Dans les transmissions, la compression est un moyen simple d'augmenter le débit d'une ligne. Le problème principal est qu'elle arrive au résultat inverse lorsque le flot de données transmis est déjà compressé.

Ce document présente mes travaux sur la synthèse de haut niveau dans la partie 2, ceux sur la conception de systèmes intégrés dans la partie 3, ceux sur la compression de données dans la partie 4. Enfin, avant de conclure, la partie 5 liste les publications, les thèses et les logiciels distribués qui sont issus de ces travaux.

Chapitre 2

Synthèse de haut niveau

2.1 Introduction

2.1.1 Cadre

La synthèse de haut niveau consiste à produire une description au niveau transfert de registres à partir d'une description de haut niveau comme un algorithme séquentiel. Ce problème peut être comparé à un compilateur logiciel classique qui en plus génère un jeu d'instruction adapté à l'algorithme. L'intérêt de la synthèse de haut niveau réside dans le fait qu'elle mixe la compilation et la micro-architecture et qu'elle est intrinsèquement très complexe.

On sent bien que le nombre de manières dont peut être effectuée la synthèse est gigantesque. Ce nombre de solutions que l'on dénomme *espace de conception* est représenté sur la figure 2.1. Sur cette figure, les disques noirs représentent des solutions, les caractéristiques d'une solution sont sa surface de silicium (son coût) et sa vitesse qui correspond au temps nécessaire pour dérouler l'algorithme (période d'horloge nominale fois le nombre de cycles). Ces solutions se trouvent dans une partie du plan délimitée par une courbe abstraite dite de Pareto donnant les optimums (coût vitesse). La surface grisée donne les solutions acceptables. Comme le montre la figure, le nombre de solutions n'est pas unique. En effet, pour un même algorithme on a, en 1 sur la figure une solution lente et petite, en 2 une solution rapide et grosse, et en 3 un bon compromis. La difficulté de la synthèse de haut niveau résulte essentiellement de ce besoin de générer plusieurs solutions, du fait que le coût et la vitesse d'une solution ne peuvent être réellement connus que lorsque le circuit est généré (placé et routé) et que la surface hachurée est inconnue.

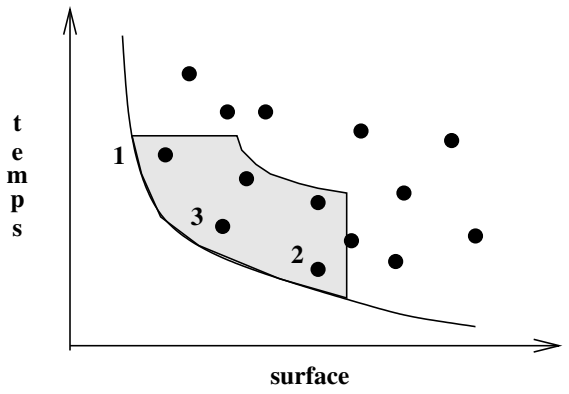


FIG. 2.1 – Espace de conception.

Il existe 2 types de synthèse de haut niveau. Le premier tel [NCG91, SDPC95, AK08, Bol08] est orienté flot de données, il consiste à générer des circuits dans lesquels le contrôle ne dépend pas des données comme les algorithmes de traitement du signal. Le second type tel [KDJ94, LKMM95, DTE02, Cyn], appelé synthèse orienté contrôle, est la synthèse non dédiée à un type d'algorithme mais d'usage général (en anglais : general purpose). Il doit être capable de traiter tous les algorithmes indépendamment de la proportion de calcul et de contrôle qu'ils contiennent. Dans cette partie, excepté le chapitre 2.2.4.1 qui explicite les différences entre les synthèses orientées données et contrôle, nous nous intéresserons uniquement à la synthèse orienté contrôle.

De 1980 à 2000, la synthèse de haut niveau était très en vogue, de nombreux outils académiques (Cathedral, Amical, Gaut) et industriels (Callas, HIS, BC, CoCentricont, Cynthesizer, CatapultC, Pico) ont vu le jour. Néanmoins aujourd'hui, son usage ne s'est toujours pas imposé dans l'industrie par manque d'efficacité de ces outils. De ce fait, la synthèse de haut niveau est encore un domaine de recherche à défricher.

2.1.2 Contribution

J'ai commencé à m'intéresser en 1987 à la synthèse de haut niveau qui fut le sujet de mon doctorat soutenu en 1990. Il déboucha sur un outil de synthèse appelé ALMA. J'ai intégré ensuite le LEP (Laboratoire d'Electronique de Philips) où j'ai continué le développement de cet outil en collaboration avec Jean-Yves Brunel et d'autres ingénieurs. En 1994 j'ai quitté le LEP qui commençait à être démantelé pour le laboratoire ASIM/LIP6 avec l'objectif de concevoir un nouvel outil de synthèse de haut niveau plus complet appelé UGH. Il fut opérationnel en 1999 et réellement terminé début 2003. Le développement de UGH fut en partie financé par le projet ESPRIT COSY dont il était un livrable.

ALMA a été utilisé au LEP pour la réalisation d'un circuit de décodage vidéo du format MPEG2, plus précisément le bloc qui analyse le flux d'entrée et active les autres blocs en fonction des types d'images. Il a été aussi utilisé pour faire des démonstrateurs au LEP tel un dictionnaire avec recherche par expressions régulières simples, un circuit de décompression sans perte adapté aux réseaux et à l'ISEP pour faire des circuits implantant la transformation d'Hadamar. ALMA a également été adapté pour faire le générateur de micro-code d'un circuit neuronal micro-programmable du LEP.

Plusieurs coprocesseurs ont été conçus avec UGH à ASIM/LIP6, un coprocesseur implantant les décodeurs de Huffman des décodeurs vidéo MPEG1/MPEG2 qui était aussi un délivrable du projet ESPRIT COSY. Quatre coprocesseurs implantant un décodeur JPEG (vld, idct, zz, iq) ont été faits pour le démonstrateur de Disydent (voir chapitre 3). On peut citer encore un circuit de DMA pour bus PCI, et différents circuits de communication et de calcul. Enfin, UGH a été utilisé en mode dégradé en enseignement en M1 sur FPGA. Actuellement, UGH est utilisé à l'ENS Lyon [DQ07], à l'INPG à Grenoble et à ORANGE-RT [FM+07, FM+08], un laboratoire de recherche et développement de France-Telecom.

Aprés ma thèse, durant laquelle j'ai créé le moteur d'ALMA, j'ai continué son développement et son amélioration avec trois autres ingénieurs (Jean-Yves Brunel, Philippe Tychon et Jean-Michel Callemyn) pendant 2 ans, puis avec Jean-Yves Brunel seulement les deux ans suivants, avec un concepteur de circuits qui concevait le bloc de contrôle d'un circuit décodeur MPEG2. Durant ces années, 4 mémoires d'ingénieurs ENSIIE ont contribué au développement d'ALMA.

UGH a été développé avec Frédéric Pétrot, deux postdocs Rajesh Bawa et Ludovic

Jacomme, un thésard François Donnet et plus d'une dizaine de stagiaires de M2 et de mémoires d'ingénieurs ENSIIE.

Ce travail a été reconnu internationalement par une publication de fond en tant que chapitre dans un livre "The New Wave of the High-Level Synthesis" [APD08] faisant le point de la synthèse de haut niveau dans le monde. Ce travail a donné lieu à 3 thèses, la mienne [Aug90], celle de Jean-Yves Brunel [Bru96] qui expose les travaux que nous avons effectués au LEP et la thèse de François Donnet [Don04] que j'ai personnellement encadrée. Il a aussi donné lieu à une dizaine de publications internationales avec comité de lecture, les principales étant [BAR+91, BAH+94, AB95, ABG+97, ABGP99, ADP03, PAD08], aux publications nationales ou autres [Aug94, Aug95, Aug95, Don02] et aux livrables COSY [AP00a, AP00b].

UGH est sûrement un des plus gros logiciels développés à ASIM/LIP6, le moteur sans compter les headers contient plus de 110.000 lignes de code réparties moitié-moitié entre le langage C et C++. J'ai personnellement créé l'architecture du logiciel et défini les interfaces des bibliothèques ainsi que les mécanismes d'échanges faisant le lien entre les différents programmes. Dans cette architecture, le soin apporté au cloisonnement a permis de faire travailler des stagiaires sur des parties bien précises sans avoir besoin de connaître les autres parties du logiciel.

2.2 Description

2.2.1 Le schéma en Y étendu

Le schéma en Y proposé par D. Gajski [HLS92] en 1992 ne peut plus représenter la diversité des outils actuels de synthèse en général et ceux de la synthèse de haut niveau en particulier. Pour cela nous l'avons étendu comme le montre la figure 2.2. Ce schéma ajoute le niveau d'abstraction "flot de contrôle" entre les niveaux d'abstraction "flot de données" et système. Il correspond à la synthèse de coprocesseurs dont la vue structurelle est un chemin de données contrôlé par une machine d'états comme l'illustre la figure 2.3.

Sur l'axe de la vue comportementale dans le niveau d'abstraction "flot de contrôle", on notera qu'il y a deux types de description impérative : la description synchronisée et la description non synchronisée. Dans la description non synchronisée, les frontières de cycles ne sont pas données. Elle correspond à une description procédurale comme les langages Pascal, C. Dans la description synchronisée, les frontières de cycles sont données et de ce fait la *fonction de transition* de l'automate de contrôle est connue. Elle peut prendre la forme d'une description procédurale comme les langages Pascal, C annotée par des pragmas donnant les frontières de cycles ou prendre la forme d'une machine d'états où à chaque état est associé des instructions impératives.

2.2.2 Les phases de la synthèse de haut niveau

Le circuit cible de la synthèse de haut niveau orienté contrôle est présenté sur la figure 2.3, c'est un circuit classique composé d'une partie contrôle (machine d'états finis) contrôlant un chemin de données. Il y a plusieurs approches pour la synthèse de haut niveau, les principales sont présentées sur la figure 2.4, a) et b) sont assez semblables et utilisées dans [PK89, BKV+96, Cam96], et c) est moins classique et utilisée dans [Sug00]. En a) la synthèse de haut niveau génère à partir de la description séquentielle non synchronisée d'un coprocesseur, une vue structurelle de celui-ci (arc 1) composée

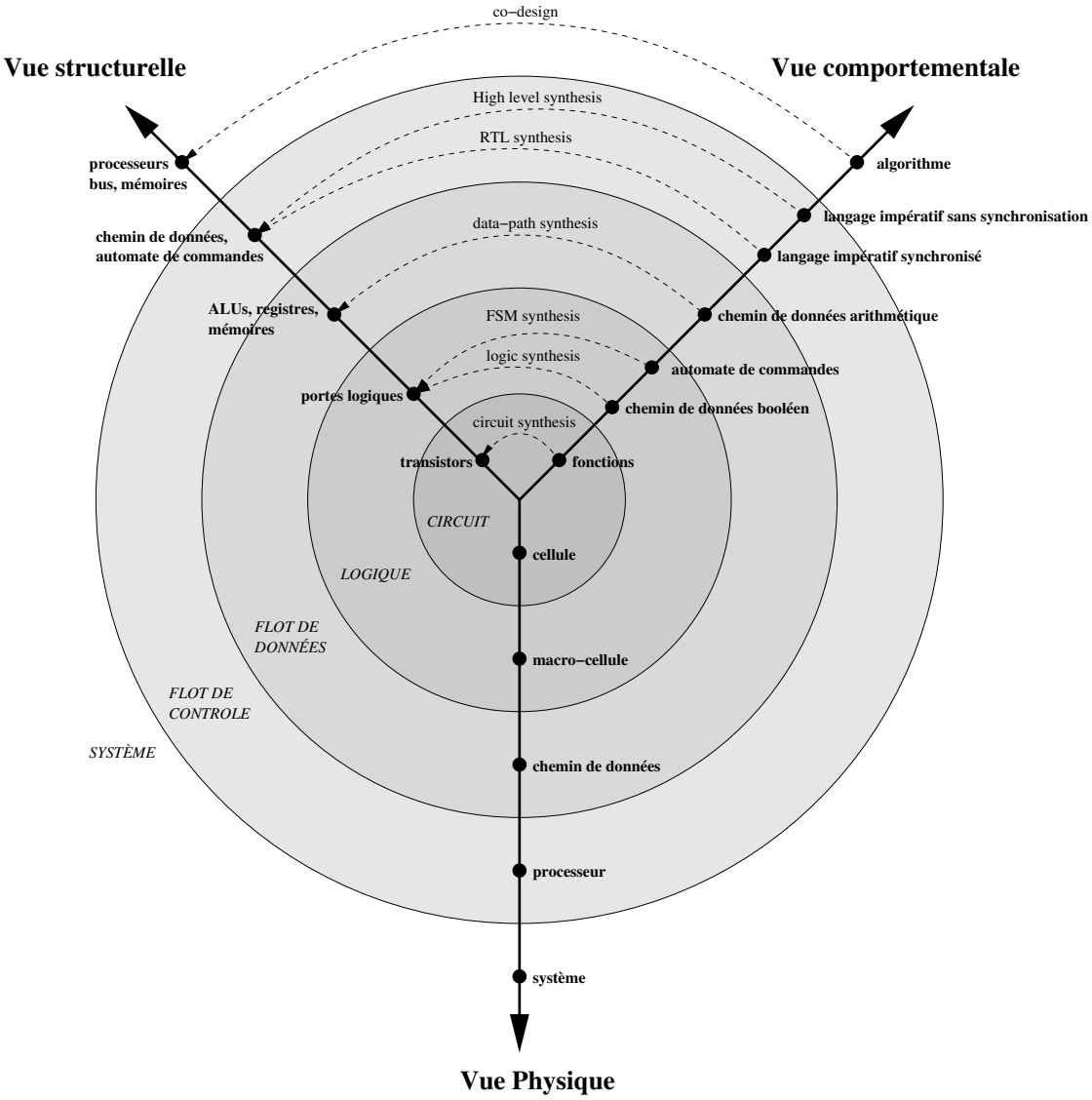


FIG. 2.2 – Le schéma en Y étendu.

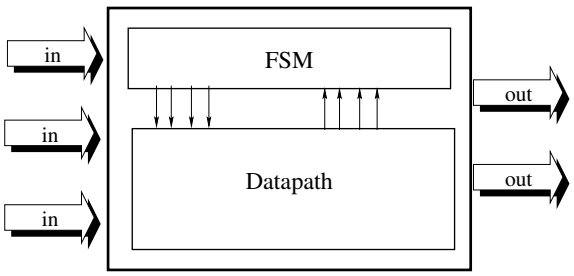
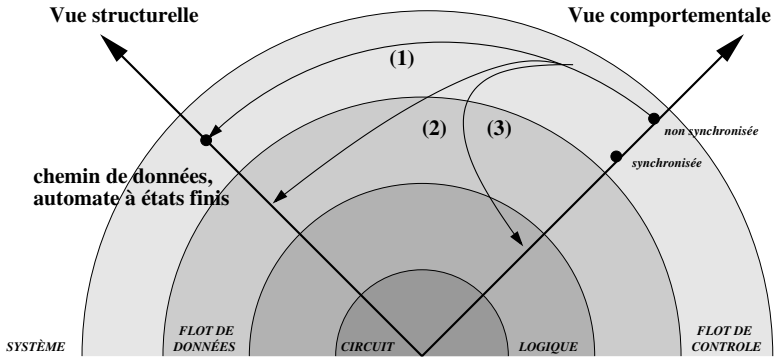
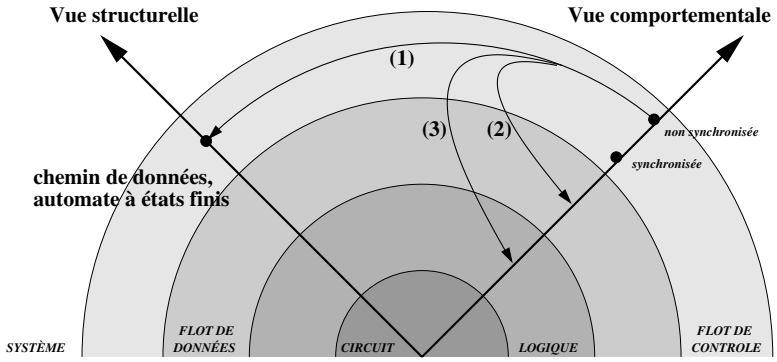


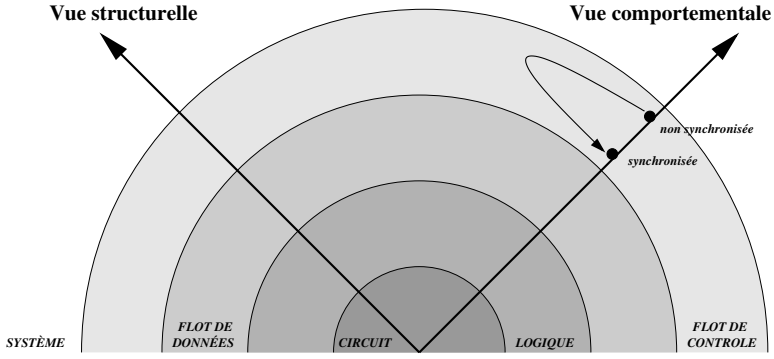
FIG. 2.3 – Gabarit structurel d'un coprocesseur.



(a)



(b)



(c)

FIG. 2.4 – Définition de la synthèse de haut niveau.

d'une machine d'états comportementale (arc 3) et un chemin de données structurel (arc 2). En b) elle génère aussi une vue structurelle du coprocesseur (arc 1) composée d'une machine d'états comportementale (arc 3) et d'un chemin de données comportemental (arc 2). Enfin en c), elle génère une description impérative synchronisée.

La synthèse de haut niveau est un processus complexe, les outils qui l'implantent sont basés sur trois phases principales.

Elaboration du CDFG

Le but de cette phase est de construire le squelette de la machine d'états qui contrôle le chemin de données du coprocesseur et, en particulier, d'identifier les branchements conditionnels qui seront présents dans l'automate.

Le graphe de flot de données et de contrôle (CDFG) est la fusion d'un graphe de flot de données (DFG) et d'un graphe de flot de contrôle (CFG). Dans le CFG, les séquences d'opérations sans branchement sont représentées par des noeuds et les branchements entre deux séquences (boucles, sauts conditionnels) par des arcs. Le DFG représente les dépendances de données à l'intérieur d'un bloc d'opérations séquentielles.

Le CDFG représente les dépendances entre toutes les opérations fonctionnelles de la description comportementale. Dans ce graphe, chaque noeud est une opération séquentielle et chaque arc symbolise une contrainte de précédence dans l'ordonnancement des opérations fonctionnelles. Les contraintes de précédence matérialisent l'impossibilité de faire une action avant une autre pour garder la cohérence globale du comportement initial.

Scheduling

L'ordonnancement consiste à fournir à chaque cycle de fonctionnement la liste des opérations à effectuer. C'est cette séquentialisation des opérations qui, de fait, va assurer que le circuit se comporte conformément à sa spécification.

Allocation

Le but de l'allocation est la définition de la structure du chemin de données du coprocesseur. L'allocation sert à déterminer le nombre d'opérateurs matériels nécessaires ainsi que leurs types et la façon dont ils sont interconnectés. L'allocation peut soit ajouter un opérateur, soit partager un opérateur déjà existant en liant cet opérateur à plusieurs opérations fonctionnelles de la description comportementale.

2.2.3 Problématique générale

Les outils de synthèse de haut niveau sont confrontés aux problèmes principaux suivants.

Cercle vicieux Les optimisations effectuées dans les trois étapes de la synthèse de haut niveau sont très liées et les décisions ne peuvent être prises indépendamment dans chaque phase sans risquer de compromettre la qualité du résultat. La figure 2.5 représente les dépendances entre les phases :

Arc 1 L'*ordonnancement* est fondé sur les DFG extraits du CDFG.

Arc 2 Le CDFG influence le nombre minimal d'opérateurs en nécessitant des opérateurs pour les branchements câblés.

Arc 3 L'allocation est fondée sur la parallélisation des opérations fonctionnelles pour déterminer le nombre de ressources matérielles nécessaires.

Arc 4 Dans la phase d'allocation et d'affectation, le partage d'un opérateur entre deux opérations fonctionnelles implique le plus souvent l'ajout de combinatoire

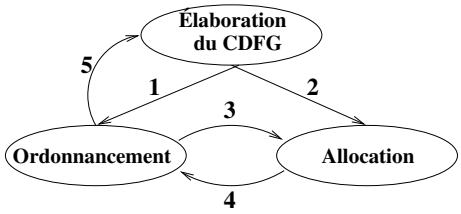


FIG. 2.5 – Cercles vicieux de la synthèse de haut niveau.

sur ses entrées (multiplexage). Le partage d'un opérateur augmente le temps de propagation de ce dernier. La phase d'allocation influence donc l'ordonnancement car elle modifie l'évaluation temporelle sur laquelle se base l'ordonnancement.

Arc 5 La détermination des blocs d'instructions séquentielles dans le CDFG se fait indépendamment des contraintes temporelles des opérateurs matériels. Un mauvais ordonnancement peut alors empêcher une parallélisation efficace des opérations fonctionnelles. L'ordonnancement doit pouvoir remettre en cause la phase d'élaboration du CDFG.

La synthèse peut toujours produire une solution de base par une démarche linéaire : construction du CDFG, ordonnancement puis allocation. Mais celle-ci est souvent inutilisable du fait des piètres caractéristiques du résultat.

Les arcs *4* et *5* de la figure 2.5 sont des phases d'optimisations des caractéristiques du circuit. Ce sont elles qui constituent les cercles vicieux.

Du fait de la complexité des algorithmes, les outils de synthèse de haut niveau sont très lents. Le travail d'exploration architecturale en est rendu plus difficile pour le concepteur.

Pour échapper aux cercles vicieux deux approches sont possibles pour traiter les dépendances croisées :

⇒ une automatisation totale de la synthèse où on itère les différentes phases de la synthèse jusqu'à obtenir une solution satisfaisante. Elle peut éventuellement impliquer des heuristiques pour converger plus rapidement vers une solution.

⇒ des informations supplémentaires fournies par l'utilisateur pour casser les cycles de dépendance et réduire le nombre de choix.

Ces heuristiques posent un problème de fiabilité (voir le paragraphe sur les estimateurs page 44) et les décisions prises en fonction de ces heuristiques doivent pouvoir être infirmées facilement par l'utilisateur.

Fréquence du coprocesseur cible Un coprocesseur n'est qu'un bloc d'un système plus vaste qui détermine la fréquence du coprocesseur. Cette fréquence est un paramètre de la synthèse de haut niveau qui doit être pris en compte très tôt. Les outils de synthèse de haut niveau prennent effectivement en entrée la fréquence cible du coprocesseur. Celle ci est prise en compte dans la phase d'ordonnancement. Mais dans cette phase le chemin de données n'est pas encore construit, donc l'ordonnancement n'est fait qu'avec les temps de propagation des opérateurs et en ignorant les temps dus à la connectique (multiprocesseurs) et au routage. De ce fait, la fréquence de fonctionnement est plus un résultat de la synthèse RT qu'une entrée. Elle est donnée par le chemin critique du chemin de données généré.

Cette approche ne permet pas de garantir que le circuit synthétisé puisse fonctionner à la fréquence voulue quand la synthèse RT échoue et même à aucune fréquence si le

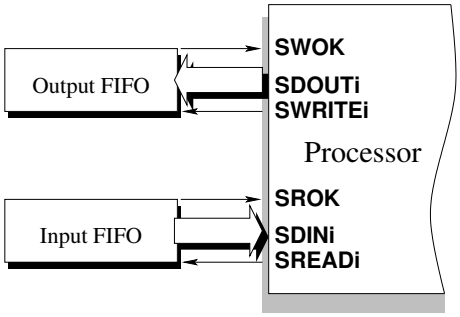
circuit possède des chaînes courtes et longues. Elle a de plus tendance à générer de très gros circuits quand la synthèse RT entre dans des techniques de procrastination pour tenir la fréquence.

Communication La question qui se pose ici est : la synthèse de haut niveau doit-elle être capable de synthétiser les communications entre le coprocesseur et le monde extérieur ? Ma réponse est non et découle principalement des considérations suivantes :

⇒ Les schémas de communication nécessitent la notion de cycle pour être définis. Ceci imposerait de réintroduire des primitives de synchronisation au langage de description.

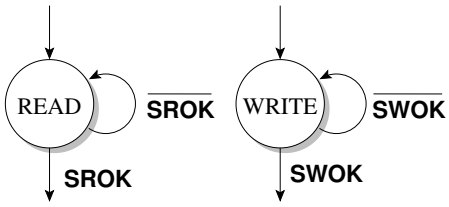
⇒ Le nombre de schémas de communication étant très vaste, il n'est pas du ressort de l'utilisateur de la synthèse de haut niveau. C'est un peu comme si on demandait à un programmeur C d'attaquer la carte réseau ou le contrôleur de disque directement.

⇒ Les communications d'un coprocesseur avec le monde extérieur doivent être fonctionnellement asynchrones. Si la synthèse de haut niveau les synthétisait, il faudrait pouvoir détecter qu'elles le sont vraiment.



Vus du maître, les signaux *SREAD* et *SWRITE* sont équivalents à des signaux de sélection, les signaux *SROK* et *SWOK* sont équivalents à des signaux d'acquittement.

a) Protocole & ports



b) automates de contrôle

lire(canal,variable)
écrire(canal,variable)

c) Primitives

FIG. 2.6 – Schéma de communication.

La synthèse de haut niveau doit donc fournir les primitives de communication comme les systèmes d'exploitation les fournissent aux langages impératifs classiques. Pour mes outils, j'ai choisi le protocole présenté sur la figure 2.6.a. Il a l'avantage d'être très simple et facilement interfaçable avec tout autre protocole. Au niveau du langage d'entrée, il fournit les primitives bloquantes *read* et *write* (figure 2.6.c) qui sont implantées par les automates de contrôle présentés sur la figure 2.6.b.

Exploration de l'espace de conception Le concepteur a une vision globale du résultat qu'il veut obtenir. Son premier objectif est d'arriver à une réalisation matérielle dont les caractéristiques physiques telles que les temps de propagation, la surface, la puissance consommée, ne soient pas rédhibitoires pour le circuit. La figure 2.1 représente un espace des solutions proposées par un outil par rapport aux critères surface et vitesse. Un *point de Pareto* (1, 2 et 3) est un point pour lequel il n'existe aucune solution meilleure atteignable par l'outil sur les axes de l'espace des solutions [Mic94]. La courbe pleine représente le résultat "idéal" estimé. Elle peut représenter, par exemple, une conception "à la main" par un concepteur expérimenté. Le

concepteur pose des contraintes sur chaque axe pour délimiter l'espace des solutions acceptables (surface en grisé sur la figure). La distance des *points de Pareto* à la courbe "idéale" nous informe sur l'efficacité de l'outil.

L'objectif de la synthèse de haut niveau est de pouvoir couvrir cet espace des solutions en obéissant aux directives du concepteur. Comme pour une démarche "à la main", le concepteur doit pouvoir contrôler les critères d'optimisation de son circuit. De plus, l'outil a pour tâche de produire le résultat le plus efficace possible dans l'espace des solutions acceptables.

Utilisateur visé Un outil est dédié à un type d'utilisateur. Il sous-entend, en effet, un certain nombre de connaissances que l'utilisateur doit posséder pour pouvoir utiliser l'outil. Dans le cas de la synthèse de haut niveau, on peut cibler :

⇒ Un algorithmicien. Dans ce cas, l'outil doit l'aider intuitivement pour trouver la bonne conception matérielle.

⇒ Un concepteur de circuit. Dans ce cas, l'outil doit l'aider à réaliser la conception qu'il désire en prenant en charge le maximum de tâches fastidieuses. De plus, l'outil doit prendre en compte ses habitudes et son savoir faire.

Les outils de synthèse de haut niveau semblent en général plutôt s'adresser à des algorithmiciens au vu 1) du langage d'entrée ([BKa93, LKMM95, DTE02, Cyn]), 2) des contraintes de synthèse qu'ils proposent, comme le nombre d'opérateurs d'un type donné ([LKMM95],[KDJ94]), ou les frontières de cycle ([BKa93, KDJ94, LKMM95, DTE02, Cyn]), 3) des informations qu'ils fournissent comme des tables d'allocations des opérations sur les opérateurs pour aider l'outil à atteindre de bons points de Pareto ([KDJ94, LKMM95]).

Un outil didactique de synthèse de haut niveau est une tâche perdue d'avance, en effet l'entrée principale est l'algorithme non synchronisé et il diffère grandement suivant qu'il est destiné à être compilé et à tourner sur un processeur standard ou qu'il est destiné à être synthétisé dans un coprocesseur matériel. Ceci est illustré par les

```
lire(A)
lire(B)
while ( A!=B ) {
  if ( A<B )
    B = B-A
  else
    A = A-B
}
ecrire (A)
```

a)PGCD pour logiciel
(1 itération par cycle)

```
lire(A)
lire(B)
while ( A!=B ) {
  if ( A<B )
    A,B = B,A
  else if ( A>4*B )
    A = A-4*B
  else if ( A>2*B )
    A = A-2*B
  else
    A = A-B
}
ecrire (A)
```

b) PGCD pour matériel
(˜3 itérations par cycle)

FIG. 2.7 – Algorithme pour processeur et pour synthèse matérielle.

algorithmes de calcul du PGCD présentés sur la figure 2.7. En effet, mettre des contraintes sur l'algorithme a) ne permettra jamais d'atteindre une implantation équivalente à celle de l'algorithme b). De plus pour écrire l'algorithme b), il faut que le concepteur ait déjà une idée très précise de son circuit.

Les deux outils de synthèse de haut niveau ALMA et UGH sont dédiés aux concepteurs de circuits, ils implantent les choix techniques présentés ci-dessus et se proposent de les valider.

2.2.4 Les différentes approches

Un outil de synthèse de haut niveau est usuellement catalogué soit orienté données (ou DSP), soit orienté contrôle (ou general purpose). Bien que ces termes soient plutôt flous et même mal choisis, ce sont ceux utilisés historiquement et nous les utiliserons dans ce document. Le sous-chapitre suivant explicite ces 2 types de synthèse et le suivant expose les réponses que des outils de ces types apportent à la problématique du chapitre précédent.

2.2.4.1 Les deux types de synthèse de haut niveau

```
// sum 4 by 4 the values of din into sum
void pe1(uint4 din[128], uint6 sum[16])
{
  int i;
  for (i=0 ; i<16 ; i+=1)
    sum[i]=0;
  for (i=0 ; i<128 ; i+=1)
    sum[i%16] += din[i];
}

// round odd value to less even one
void pe2(uint6 src[16], uint6 dest[16])
{
  int i;
  for (i=0 ; i<16 ; i+=1) {
    if ( src[i]&1 )
      dest[i] = src[i] - 1;
    else
      dest[i] = src[i];
  }
}

void coproc(uint4 din[128], uint6 dout[16])
{
  uint6 tmp[16]

  pe1(din,tmp);
  pe2(tmp,dout);
}
```

FIG. 2.8 – Un exemple de description pour la synthèse orientée données.

La figure 2.8 donne un petit exemple de description d'un coprocesseur pour l'outil CatapultC [Bol08] qui est orienté données. Le coprocesseur (fonction coproc) est décrit comme 2 processeurs élémentaires (fonctions pe1, pe2). En plus de cette description fonctionnelle, l'utilisateur doit fournir à l'outil un ensemble de contraintes dans un fichier associé que l'utilisateur doit écrire directement à la main, ou générer à partir d'un utilitaire avec une interface graphique fourni par l'outil. Les principales contraintes sont :

- Comment les paramètres de la fonction coproc sont assignés sur des ports. Dans CatapultC, chaque paramètre correspond à un canal de communication, l'utilisateur doit choisir parmi un ensemble de protocoles prédéfinis par l'outil (asynchrone (handshake) ou synchrone), celui qu'il désire sur chaque canal.
- Les fonctions qui sont à "inlinées" ou à cascadées (pipeline fonctionnel à gros grain).
- Sur la description les optimisations qu'il désire, comme telle boucle est à dérouler, ces 2 boucles sont à fussionner en une seule.
- Le débit sur les cannaux.
- Une fréquence cible.

En fonction de ces contraintes, CatapultC va générer une architecture. Dans le cas où la contrainte "fonctions cascadées" est activée, l'architecture sera semblable à celle de la figure 2.3.c (avec 2 PEs seulement) cascadant les 2 fonctions sur les coprocesseurs élémentaires (figure 2.3.b). Le protocle entre les deux coprocesseurs est choisit par l'outil. Dans le cas où la contrainte "fonctions inlinées" est activée, un débit de 1 entrée par cycle et un protocole synchrone sont demandés le coprocesseur généré se réduira à un chemin de données.

```
001 // Input stream.
002 template <class T>
003 class Ain
004 {
005   public:
006     Ain( const char* name=0 ) {}
007     sc_out<bool> rdy;
008     sc_in<bool> vld;
009     sc_in<T> data;
010     void reset() { rdy = 0; }
011     T get() {
012        wait();
013        rdy = 1;
014        do { wait(); } while ( !vld.read() );
015        rdy = 0;
016        return data.read();
017     }
018 };
019
020 // Output stream.
021 template <class T>
022 class Aout
023 {
024 public:
025   sc_in<bool> rdy;
026   sc_out<bool> vld;
027   sc_out<T> data;
028   Aout( const char* name=0 ) {}
029   void reset() {
030     vld = 0;
031     data = 0;
032   }
033   void put( const T& val ) {
034     do { wait(); } while ( !rdy.read() );
035     data.write( val );
036     vld = 1;
037     wait();
038     vld = 0;
039    }
040 };
041
042 // Coprocessor definition
043 SC_MODULE(coproc)
044 {
045   sc_in clk clk;
046   sc_in<bool> rst;
047   Ain< sc_uint<8> > din;
048   Aout< sc_uint<32> > dout;
049
050   SC_CTOR(coproc) {
051     SC_CTHREAD( behavior, clk.pos() );
052     reset_signal_is( rst, 1 );
053   }
054
055   void behavior() {
056     // reset behavior
057     din.reset();
058     dout.reset();
059     wait();
060     // init behavior
061     // wait for 2 consecutives 0
062     sc_uint<8> dummy;
063     sc_uint<8> nb=0;
064     while (nb!=2 ) {
065         dummy = din.get();
066         if (dummy == 0 )
067             nb += 1;
068         else
069             nb = 0;
070     }
071     // main loop
072     sc_uint<32> sum=0;
073     nb=0;
074     while (1) {
075       sc_uint<8> d = din.get();
076       if (d==0)
077         nb += 1;
078       else
079         nb = 0;
080       if (nb==2) {
081         sc_bool parity=0;
082         while ( sum != 0 ) {
083           if ( (sum&1)!=0 )
084               parity = !parity;
085           sum >>= 1;
086         }
087         dout.put( sum );
088         sum = parity;
089         dout.put( sum );
090         sum = 0;
091         nb = 0;
092       } else
093         sum += d;
094    }
095  }
096 };
```

FIG. 2.9 – Un exemple de description pour la synthèse orientée contrôle.

La figure 2.9 donne un exemple de description d'un coprocesseur pour l'outil Cynthesizer [Cyn] qui est orienté contrôle. L'outil utilise en entrée du C++ SystemC. Cette description merite quelques explications.

- Un canal d'entrée asynchrone équivalent à celui présenté sur la figure 2.6 est défini de façon générique des lignes 1 à 18. La généricité porte sur le type du port donnée. Un tel canal est instancié ligne 47 pour une donnée de 8 bits.
- De façon similaire, un canal de sortie est défini des lignes 20 à 40 et instancié ligne 48 pour une donnée de 32 bits.
- Les lignes 43 à 48 définissent le composant coproc ayant comme entrées clk (horloge, 1 fil), rst (reset, 1 fil), din un canal asynchrone (2 fils de contrôle et 8 fils de données) et comme sortie dout un canal asynchrone (2 fils de contrôle et 32 fils de données)
- Les lignes 49 à 52 sont de la glu SystemC.
- Les lignes 54 à 95 définissent le comportement du coprocesseur. Il se décompose en :
  1. le comportement quand le reset est actif (lignes 56 à 69).
  2. le comportement à exécuter 1 fois seulement quand le reset est relaché (lignes 61 à 69). Il attend 2 0 sur le canal d'entrée.
  3. le comportement à répèter tant que le reset n'est pas activé (lignes 74 à 92). Il lit le des données sur le canal d'entrée et les somme. Dés que deux 0 sont lus, il écrit 2 données sur le canal de sortie, la somme et son bit de parité.

En plus de cette description fonctionnelle, l'utilisateur doit fournir un ensemble de contraintes en ajoutant des pragmas dans le source. Les principales contraintes sont :

- Sur la description les optimisations qu'il désire, comme telle boucle est à dérouler.
- Les instructions conditionnelles à cabler dans le chemin de données ou à programmer dans la machine d'état (instruction wait).
- Une fréquence cible.

En fonction de ces contraintes, Cynthesizer va générer une architecture semblable à celle présentée sur la figure 2.3.a.

Ces exemples montrent intuitivement que la description orientée données fait une plus grande abstraction du matériel que la description orientée contrôle. Ci-dessous nous allons expliciter plus précisément ces différences à différents niveaux.

Modèle de communication

La figure 2.10.a montre des interfaces d'un coprocesseur pour la synthèse orientée contrôle. Une telle interface spécifie les canaux de communication, le type de chaque canal spécifie le sens du canal, la taille du bus et son protocole. Le gabarit du corps d'un tel coprocesseur est une séquence d'initialisation, suivie d'une boucle infinie.
La figure 2.10.b montre les mêmes interfaces pour la synthèse orientée données. Les canaux sont remplacés par des variables. **Attention : ceci ne signifie pas que les entrées/sorties du coprocesseur sont des tableaux mais c'est une complète abstraction des entréés/sorties comme expliqué ci-dessous**. La sémantique pour "pgcd1" est que le coprocesseur 1) lit la donnée A sur un canal, 2) lit B sur un autre canal, 3) applique l'algorithme du corps de la fonction pour calculer une valeur R en fonction de A et B, 4) puis écrit le résultat R sur un troisième canal. Il est sous entendu que le coprocesseur répète indéfiniment ce traitement. "pgcd2" est similaire à "pgcd1" mais les opérandes sont lues sur le même canal. Pour "idct", 64 données sont lues sur un canal et 64 données sont écrites sur un autre.
Pour être complet, en plus de cette interface, l'utilisateur doit associer à chaque canal un protocole à choisir parmi plusieurs protocoles prédéfinis par l'outil. Quelques protocoles sont présentés sur la figure 2.11.

```
void pgcd1(
    instream<ushort> A,
    instream<ushort> B,
    outstream<ushort> R
{
    ...
    while (1) {
        ...
    }
}
```

```
void pgcd2(
    instream<ushort> AB,
    outstream<ushort> R)
{
    ...
    while (1) {
      ...
    }
}
```

```
void idct(
    instream<pixel> PIXEL,
    outstream<ushort> COEF)
{
    ...
    while (1) {
      ...
    }
}
```

a) orientée contrôle

```
void pgcd1(
    ushort A,
    ushort B,
    ushort* R)
{
    ...
}
```

```
void pgcd2(
    ushort AB[2],
    ushort* R)
{
    ...
}
```

```
void idct(
    pixel   PIXEL[8][8],
    ushort* COEF[8][8])
{
    ...
}
```

b) orientée données

FIG. 2.10 – Interfaces des différentes synthèses.

La fonction "idc" de la figure 2.10.b definit un canal PIXEL. Les outils synthèse orientée données permettent d'associer quelques protocles prédéfinis à ce canal paramètrables à volonté. Voici quelques protocoles possibles qui illustrent cette diversité.

- un port asynchrone où les pixels arrivent un par un,
- un port asynchrone similaire au précédent avec une FIFO,
- un port asynchrone où les pixels arrivent 4 par 4,
- un port synchrone où les pixels arrivent un par un tout les 3 cycles. Dans ce cas le producteur a la charge de respecter ce débit.
- un double bloc mémoire en flip-flop de 64 pixels. Le producteur écrit dans un bloc, le consomateur lit dans l'autre et un bit permet d'inverser les deux blocs. L'idée étant de permettre au producteur et consomateur de travailler en parallèle.

Tous les outils de synthèse orientée données ne supportent pas tous ces types de protocoles.

FIG. 2.11 – Quelques associations de protocole sur un canal pour la synthèse orientée données.

De manière concrète, la synthèse orientée données impose un modèle de communication (N entrées donnent M sorties) qui restreint le nombre d'applications qu'elle peut décrire par rapport à la synthèse orientée contrôle. Voici quelques exemples que la synthèse orientée données ne pourra pas décrire du tout ou de facon incomplète ou artificielle :

- Un processeur général sauf si toutes ses instructions font un accès en lecture ou en écriture au bus données. En effet, l'entrée d'un processeur est l'instruction à exécuter. Une instruction "`move R1 R2`" n'a pas de sortie sur les bus mémoire contrairement à la l'instruction "`move R1 (R2)`". On n'a donc pas : 1 entrée donne 1 sortie.
- Un décompresseur type Huffman ou LZW à longueur de code variable.
- Pour un compresseur, une interface possible est :
  `compresse(char data, int* code, char*len)`

où `len` égal à zéro signifie que le code reçu doit être ignoré. De plus, dans cette interface, la concaténation des codes pour former un flux n'est pas traitée dans ce coprocesseur et doit être faite dans un autre composant lisant les codes générés et ce n'est pas un composant spécialement simple.

La synthèse orientée contrôle peut décrire les exemples ci-dessus. Elle ne peut néanmoins pas décrire toutes les applications en particulier celles où il est nécessaire de manipuler les delais electriques de manière fine comme les passerelles entre bus.

Modèle physique

Le modèle physique d'un coprocesseur de synthèse orientée contrôle est présenté

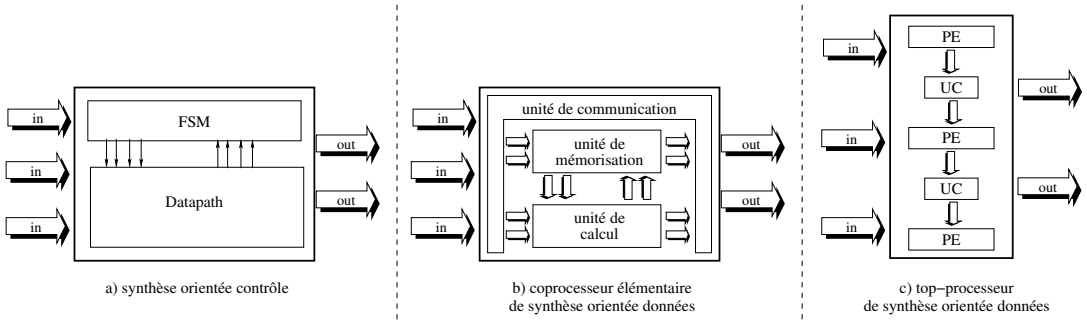


FIG. 2.12 – Gabarits physiques des différentes synthèses.

sur la figure 2.12.a. C'est un bloc matériel constitué d'une partie contrôle et d'un chemin de données sur lequel arrivent les canaux de communication.

Le modèle physique d'un coprocesseur élémentaire de synthèse orientée données est présenté sur la figure 2.12.b. C'est un bloc matériel constitué d'une unité de communication, d'une unité de mémorisation et d'une unité de calcul. Seule cette dernière est obligatoire. De plus, certains outils de synthèse orientée données tels Pico ou CatapultC permettent de décrire une application comme un chaînage de coprocesseurs élémentaires par l'intermédiaire d'une fonction qui appelle successivement les coprocesseurs élémentaires (exemple : la fonction "coproc" de la figure 2.8 définit un coprocesseur qui chaîne 2 coprocesseurs élémentaires "pe1" et "pe2" définis par les fonctions de même noms). Nous appellerons top-processeur un tel coprocesseur. La figure 2.12.c donne l'architecture d'un top-processeur. Elle est composée de coprocesseurs élémentaires (PE) communicants via des unités de communication (UC). Les outils génèrent automatiquement les unités de communication en utilisant les protocoles prédéfinis des canaux (figure 2.11). Le choix du protocole est fait en fonction du type d'échange entre le producteur et le consomateur. Par exemple si le consomateur consomme les données dans le même ordre que le producteur les produit, une FIFO est suffisante. Dans le cas contraire, une communication par un double bloc mémoire en flip-flop est plus performant.

Dans la synthèse orientée données, l'architecture cible surtout pour le top-processeur comporte de nombreux blocs générés automatiquement suivant des schémas prédéfinis et donc très éloignés de la description de l'utilisateur. Par exemple, l'unité de mémorisation peut contenir plusieurs RAMs, des FIFOs, des LIFOs qui n'ont pas de relation simple avec la description initiale. Ceci implique que l'utilisateur ne pourra pas intervenir si la netlist de portes ou de CLB générée par la synthèse de bas niveau a des problèmes de délais.

Dans la synthèse orientée contrôle, la notion de top-processeur n'existe pas. Le top-

processeur est à la charge du concepteur. C'est à lui de définir son interconnection de coprocesseurs élémentaires et leurs protocoles de communication.

Modèle des contraintes

Dans la synthèse orientée contrôle stricte (description non synchronisée), la contrainte usuelle est la taille du coprocesseur élémentaire. Sa vitesse sera connue par simulation du coprocesseur généré. Cependant pour pouvoir jouer sur la vitesse la plupart des outils proposent des contraintes temporelles. Elles consistent à indiquer qu'une séquence d'instructions du source doit s'exécuter en au moins *min* cycles et au plus *max* cycles. Si tout le code est annoté et que les bornes sont égales, on arrive presque à une description synchronisée.

Dans la synthèse orientée données, les contraintes de base sont les latences sur les ports de sorties et les débits. Les contraintes auxiliaires sont l'allocation des variables (registres ou mémoire), la taille et le nombre de bancs mémoire, les protocoles de communication.

Modèle comportemental

Pour la synthèse orientée contrôle, le modèle comportemental est un CDFG. Dans ce cas la synthèse doit optimiser un ensemble de DFG sans oublier les conditions permettant de passer d'un DFG à un autre.

Pour la synthèse orientée données, le modèle comportemental est défini dans

```
void pgcd(
    uchar A,
    uchar B,
    uchar*R
{
  while (A!=B) {
    if (A<B)
      B -= A;
    else
      A -= B;
  }
  R = A;
}
```

a)

```
void pgcd(
    uchar A,
    uchar B,
    uchar*R
{
  for (i=0 ; i<255 ; i++) {
    if (A<B)
      B -= A;
    else if (B<A)
      A -= B;
  }
  R = A;
}
```

b)

```
void pgcd(
    uint A,
    uint B,
    uint*R
{
  for (i=0 ; i<(1<<31)-1
           ; i++) {
    if (A<B)
      B -= A;
    else if (B<A)
      A -= B;
  }
  R = A;
}
```

c)

FIG. 2.13 – Comportements de la synthèse orientée données.

la littérature comme une chaîne de traitements ne dépendant pas des données. Cette définition est trop imprécise et partiellement fausse. En effet, le coprocesseur élémentaire ci-dessous a un traitement qui dépend des données (la donnée est-elle paire ?) et n'est pourtant pas incompatible avec la synthèse orientée données.

```
int pair(int in) { return in&1? in-1 ; in; }
```

Nous préférons définir le modèle comportemental comme un DFG donnant chaque sortie en fonction des entrées. De plus ce DFG doit être raisonnable dans le sens qu'il doit être manipulable et synthétisable. La figure 2.13.a est un exemple de comportement non modélisable par un DFG car on ne peut pas créer un DFG donnant R en fonction de A et B. La figure 2.13.b est un exemple de comportement modélisable par un DFG. En effet en déroulant 255 fois la boucle on obtient un DFG donnant R en fonction de A et B. Ce DFG est raisonnable, il est certe conséquent mais encore manipulable et synthétisable. La figure 2.13.c est un exemple de comportement modélisable par un DFG mais non raisonnable. En effet, en déroulant 2^32 fois (plus de 4 milliards) la boucle on obtient bien un DFG donnant R

en fonction de A et B. Mais sa taille fait qu'il n'est ni manipulable ni synthètisable. Certains outils relachent cette contrainte en définissant le modèle comportemental comme un DFG à répéter un certain nombre de fois acceptant ainsi les exemples donnés sur les figures 2.13.a et c.

Ainsi la synthèse orientée données consiste à optimiser et à synthétiser un seul DFG sur lequel arrive un flot de données. Le but de ces outils est de faire apparaître le maximum de parallélisme dans le DFG généré, puis de générer le matériel correspondant à ce DFG, et enfin d'optimiser l'allocation de l'unité mémoire et de communication pour alimenter le matériel avec un débit maximum. Les techniques utilisées pour générer le DFG sont le cablage systématique des instructions conditionnelles (si), la fusion de boucles séquentielles[1] le déroulement (total ou partiel) ou le repliement des boucles et même des transformations polyédriques pour les boucles imbriquées.
La synthèse orientée contrôle consiste à générer un chemin de données adapté à un ensemble de DFGs sans notion de flot de données. Le but de ces outils est de traduire plus fidèlement la description initiale et de coller plus au matériel comme par exemple l'utilisation de macro-cellules spécifiques.

2.2.4.2 Les solutions à la problématique

Ce chapitre expose les solutions apportées par les outils de synthèse de haut niveau tant orientée données que contrôle à la problématique présentée au chapitre 2.2.3 (page 20).

Cercle vicieux Pour la synthèse orientée contrôle, les cercles vicieux (voir figure 2.5) existent et les contraintes supplémentaires nécessaires pour les briser sont présentées ci-dessous dans le paragraphe "Exploration de l'espace de conception".
Pour la synthèse orientée données, les restrictions sur les modèles de données et de comportement changent l'état "Elaboration du CDFG" en "Elaboration du CFG" ce qui supprime les arcs 2 et 5 et de ce fait, un des cercles vicieux.

Fréquence du coprocesseur cible La plupart des outils de synthèse sinon tous utilisent pour la phase ordonnancement une fréquence cible et une table donnant les temps de propagation des opérateurs au mieux en tenant compte de la taille en bits de l'opérateur. Puis ils passent à la phase allocation (arc 3 de la figure 2.5) qui génère le RTL du coprocesseur. Enfin la fréquence cible devient une contrainte de la synthèse RTL.
Comme on le voit l'arc 4 de la figure 2.5 est ignoré et rien ne garantit que la synthèse RTL arrivera à tenir la contrainte de la fréquence cible et si elle y arrive à quel coût. En effet, entre le modèle ordonnancé et le coprocesseur généré par la synthèse RTL, il y a d'une part beaucoup de connectique en plus et d'autre part des délais supplémentaires dus aux capacités de routage pour les ASICs ou aux niveaux de routage pour les FPGAs. La fréquence réelle du coprocesseur demeure donc bien une sortie de la synthèse.
Pour la synthèse orientée données, [GMC06] indique que l'outil de synthèse CatapultC [Bol08] permet de diminuer par 2 ou 3 le temps de développement par rapport à une conception "manuelle". Les auteurs précisent que leurs mesures s'arrêtent à l'obtention d'un RTL synthétisable. Un RTL synthétisable signifie 1) que la simulation fonctionnelle de la description fonctionnelle fonctionne, 2) que la description est compréhensible par les outils de synthèse de bas niveau. Un RTL synthétisable ne signifie pas que la description RTL synthétisée fonctionnera en

[1] exemple : `for (i=0 ; i<5 ; i+=1) t1[i]=0 ; for (i=0 ; i<10 ; i+=1) t2[i]=i ;` ⟶ `for (i=0 ; i<10 ; i+=1) { if (i<5) t1[i]=0 ; t2[i]=i ; }`

simulation electro-temporelle à la fréquence cible. Il est dommage que les auteurs ne soient pas aller jusqu'au bout de l'experience et n'aient pas quantifié les temps nécessaires pour aller d'un RTL synthétisable à un circuit tournant à une fréquence cible dans les cas d'une conception "manuelle" et avec un outil de synthèse de haut niveau.

C'est à ce niveau que réside la principale différence entre les outils de synthèse tant orientés données que contrôle et ceux que j'ai développés. Les premiers chaînent élaboration, ordonnancement, allocation, synthèse RTL (fréquence cible donnée par la synthèse RTL) alors que ces derniers chaînent élaboration, allocation, synthèse RTL, ordonnancement (pour la fréquence cible).

Communication Pour la synthèse orientée contrôle les outils tels CoCentric de Synopsys [DTE02] ou Cynthesizer de Forte [Cyn] demande à l'utilisateur de décrire les protocoles de communication, introduisant des primitives de synchronisation dans le langage. Pour la synthèse orientée données la plupart tels Gaut [CCB05], Pico [AK08] ou CatapultC [Bol08] proposent des protocoles (voir figure 2.11) que l'utilisateur n'a plus qu'à instancier.

Exploration de l'espace de conception Pour la synthèse orientée contrôle, 2 méthodes sont principalement utilisées. La première que je qualifierai de didactique consiste 1) à fournir les tables d'allocations des opérations sur les opérateurs et des variables sur les registres, 2) de permettre à l'utilisateur de modifier ces tables, 3) de relancer l'outil en tenant compte de ces changements, 4) et de recommencer le cycle. La seconde que je qualifierai de synchronisée consiste à poser des frontières de cycle dans la description initiale pour orienter l'outil vers une solution.
Le principal inconvénient de la méthode didactique est qu'elle est inadaptée aux concepteurs de CI car elle leur demande de changer leurs habitudes d'une part et qu'elle est trop compliquée pour les algorithmiciens d'autre part. Le principal inconvénient de la méthode synchronisée bien que très efficace est qu'elle s'éloigne de la synthèse de haut niveau.

Pour la synthèse orientée données, [GMC06] indique que l'outil de synthèse CatapultC [Bol08] permet d'explorer l'espace de conception et d'arriver à une solution comparable à une solution "manuelle". De plus les auteurs estiment que l'utilisation de l'outil de synthèse de haut niveau diminue le temps de conception d'un facteur de 2 ou 3. Ceci n'a rien d'étonnant, les restrictions sur le modèle de données et le modèle comportemental ainsi que les contraintes de latence et de débit limitent grandement l'espace de conception. La contrepartie étant que la synthèse orientée données ne peut s'appliquer qu'à certains types de descriptions (voir page 27).

Utilisateur visé Pour la synthèse orientée contrôle, un outil tel Cynthesizer [Cyn], par le choix de SystemC comme entrée est délibérément destiné aux concepteurs de circuits. De plus, ces outils [KDJ94, DTE02, Cyn] ont besoin de primitives de synchronisation pour pouvoir décrire les communications, ceci les dédie aux concepteurs de circuits.

Pour la synthèse orientée données, les contraintes de latence et de débit et le pipeline fonctionnel sont des notions tout à fait compréhensibles par un algorithmicien. Pour augmenter le débit dans Pico [AK08] ou CatapultC [Bol08], il lui sera demandé de modifier la description initiale pour faire apparaître plus de coprocesseurs élémentaires dans son top-processeur. Pour diminuer la latence, il lui sera demandé de modifier celle-ci pour fusionner les algorithmes des différents coprocesseurs élémentaires. Il aura certes du mal à interpréter les nombreuses tables d'allocation de bas niveau générées par ces outils, mais elles ne sont pas réellement nécessaires à leur utilisation.

2.2.5 ALMA

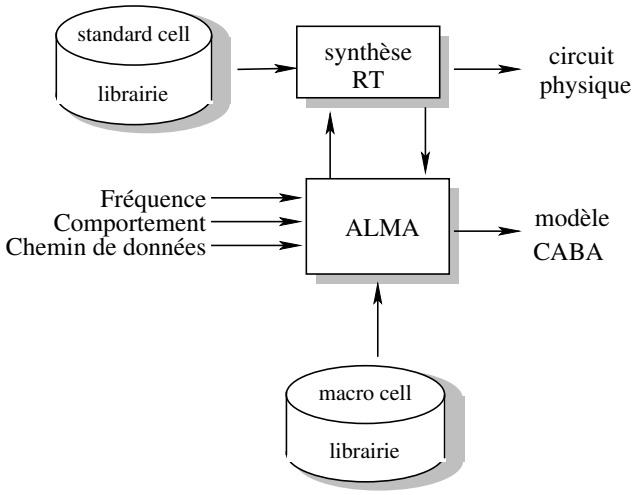


FIG. 2.14 – Vue utilisateur d'ALMA.

La vue utilisateur d'ALMA est donnée sur la figure 2.14. L'utilisateur doit fournir d'une part son chemin de données comme une interconnection de macro-cellules pré-définies par l'outil et d'autre part son comportement dans un langage spécifique. Ce langage est une description non synchronisée avec des instructions de contrôle (si, tant que, ...) et des instructions de transferts de registres du chemin de données. Pour obtenir le circuit, il doit fournir la fréquence désirée, ALMA génère alors l'automate de contrôle en tenant compte précisément des temps de propagation, de hold et de setup des opérateurs du chemin de données. Il fournit tous les fichiers nécessaires à l'outil de synthèse de bas niveau[2] pour que ce dernier génère la netlist physique du coprocesseur. En plus du circuit physique, ALMA génère un modèle C CABA (Cycle Accurate and Bit Accurate) qui permet des simulations rapides du coprocesseur tant au niveau fonctionnel que temporel.

Le schéma en Y d'ALMA est donné sur la figure 2.15. ALMA commence par générer (arc 1) le chemin de données défini comme une interconnection de macro-cellules ALMA en une interconnection de macro-celulles de l'outil de synthèse RT, les macro-cellules de l'outil de synthèse RT étant caractérisées temporellement, il calcule les caractéristiques temporelles des macro-cellules ALMA (arc 2a). Puis ALMA prend le comportement et crée une machine d'états avec une instruction RT par cycle (arc 3). Enfin tenant compte des caractéristiques temporelles des macro-cellules, ALMA réordonnance la machine d'états précédente pour l'adapter à la fréquence donnée par l'utilisateur (arc 4). De plus, si l'utilisateur a déjà placé/routé le chemin de données, ALMA tiendra compte des capacités des interconnections exactes pour caractériser les macro-cellules ALMA (arc 2b).

Dans ALMA, le concept innovant est le réordonnancement de l'automate de contrôle pour adapter le circuit à une fréquence. Nous l'avons nommé "FSM retiming" par analogie au "datapath retiming" qui résout le même problème mais en retaillant le chemin de données. Le FSM retiming consiste intuitivement à 1) extraire de la machine d'états initiale les basic-blocs (suite d'états sans branchement), à 2) reconstituer la séquence d'instructions au niveau de transfert de registres du chemin de données, à 3) réordonnancer ces instructions RT en tenant compte des contraintes temporelles des opérateurs du chemin de données et

[2] Cet outil était VLSI qui a été acheté par COMPASS à la fin des années 1990.

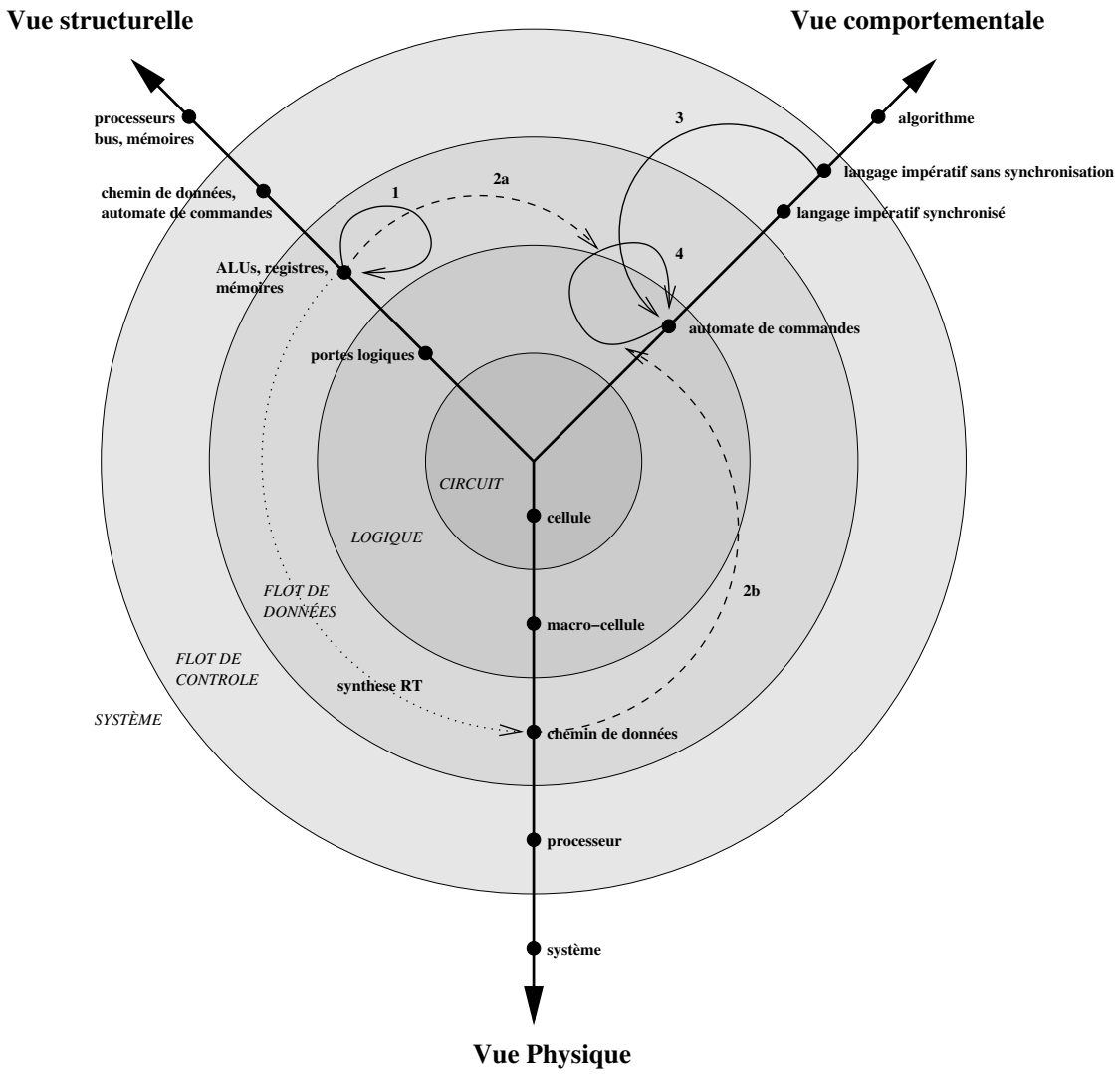


FIG. 2.15 – Le schéma en Y d'ALMA.

de la fréquence désirée. Une description détaillée des algorithmes utilisés est donnée dans l'annexe A. Dans cette annexe la section 5.4 (page A.19) définit 3 niveaux d'optimisation en fonctions du degré de réordonnancement des instructions RT de la machine d'états initiale. Ceux-ci sont brièvement décrits ci-dessous par odre d'optimisation croissante et le dernier garantissant un des meilleurs ordonnancements possibles :

l'ordre résolu L'ordre des instructions RT de la machine d'états initiale n'est pas remis en cause.

ordre concourant L'ordre des instructions RT qui partagent une ressource combinatoire du chemin de données est remis en cause.

ordre séquentiel Il remet en cause les instructions RT qui partagent une ressource combinatoire ou séquentielle du chemin de données.

ALMA travaille au niveau de l'"ordre séquentiel" ce qui signifie qu'il fournit l'ordonnancement le plus rapide possible pour le comportement RT defini par la machine d'états initiale et pour les contraintes électroniques du chemin de données

Les avantages du FSM retiming par rapport au datapath retiming sont les suivants :

1. Détection des problèmes des chaînes courtes et longues. C'est un problème électronique qui arrive quand les temps de setup de 2 unités de mémorisation sont disproportionnés (File de registres et simple registre). Quand ce problème arrive sur un circuit, il induit que le circuit ne fonctinnera pas quelle que soit la fréquence. Les solutions électroniques usuelles sont un modification de la machine d'états ou l'ajout de buffers sur le setup court pour le ralonger.
2. Fonctionnement quelle que soit la fréquence désirée dans les limites imposées d'une part par la fréquence maximale de fonctionnement des éléments mémorisants du chemin de données et d'autre part par la synthèse de la machine d'états par la synthèse logique à la fréquence désirée.
3. Insensible au faux chemin du chemin de données. Ce sont des chemins de registres à registres qui existent dans le chemin de données mais que l'automate de contrôle n'ouvrira jamais.
4. Faible coût pour des fréquences raisonnables car il n'ajoute en général que quelques états.

Le seul désavantage du FSM retiming par rapport au datapath retiming est qu'il nécessite des communications asynchrones car il modifie l'automate de contrôle. Mais ceci n'est pas gênant pour ALMA car les communications sont, à la base, asynchrones.

2.2.6 UGH

La vue utilisateur de UGH est donnée sur la figure 2.16. Elle est de prime abord similaire à la figure 2.14 mais le comportement est devenu un programme C et le chemin de données est remplacé par des contraintes. L'utilisateur doit donc fournir d'une part une contrainte de synthèse qui est un brouillon plus ou moins élaboré du chemin de données appelé Draft Data-Path (DDP). Il est créé à partir de macro-cellules pré-définies par l'outil. D'autre part, il doit fournir un comportement sous la forme d'un programme dont la syntaxe est un sous-ensemble du C enrichi de types prédéfinis pour supporter des descriptions précises au niveau du bit (Bit Accurate). Pour obtenir le circuit, il doit fournir la fréquence désirée, UGH génère alors le chemin de données et l'automate de contrôle en

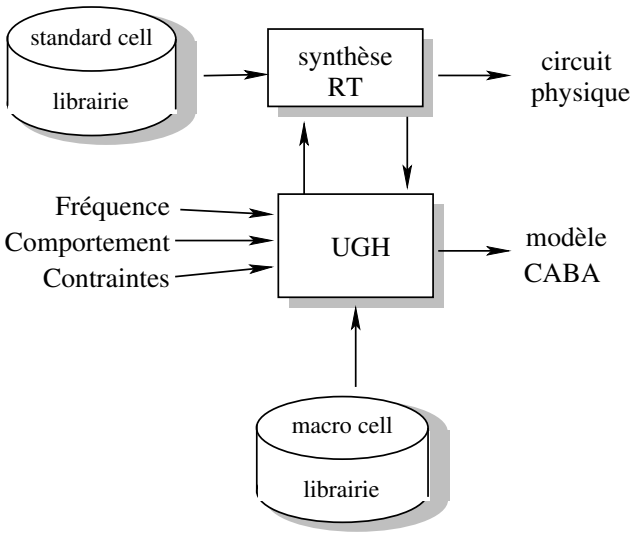


FIG. 2.16 – Vue utilisateur de UGH.

tenant compte précisément des temps de propagation, de hold et de setup des opérateurs du chemin de données. Il fournit tous les fichiers nécessaires à l'outil de synthèse de bas niveau[3] pour que ce dernier génère la netlist physique du coprocesseur. En plus du circuit physique UGH génère un modèle C CABA (Cycle Accurate and Bit Accurate) qui permet des simulations rapides du coprocesseur tant au niveau fonctionnel que temporel.

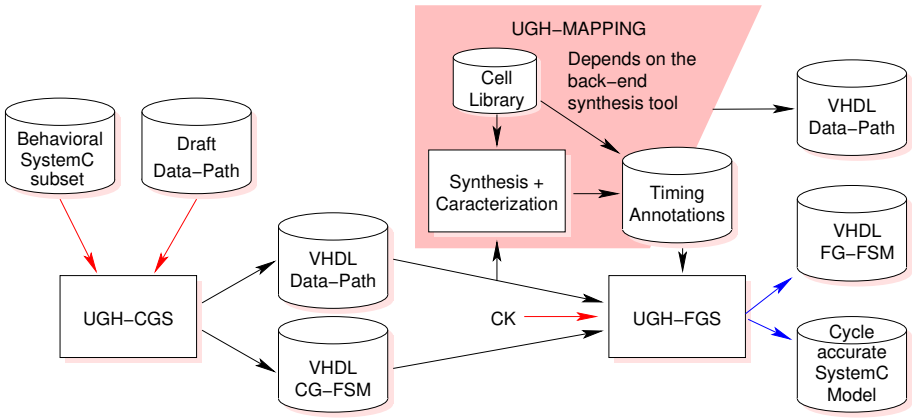


FIG. 2.17 – Schéma général du logiciel UGH.

Le schéma fonctionnel de UGH est présenté sur la figure 2.17. La synthèse se déroule en deux grandes étapes indépendantes que nous avons nommées ordonnancement à gros grain (CGS : Coarse Grain Scheduling) et (FGS : Fine Grain Scheduling).

CGS Comme le montre la figure 2.17 et le schéma en Y de la figure 2.18, CGS produit une vue structurelle du coprocesseur (arc 1) composée d'un chemin de données (arc 2) supportant les instructions du comportement en fonction des contraintes exprimées dans le DDP d'une part et d'autre part d'un automate de contrôle (arc 3) implantant ces instructions.
Le chemin de données est une interconnection de macro-cellules que CGS décrit comportementalement (arc 4).

[3] Cet outil est Synopsys.

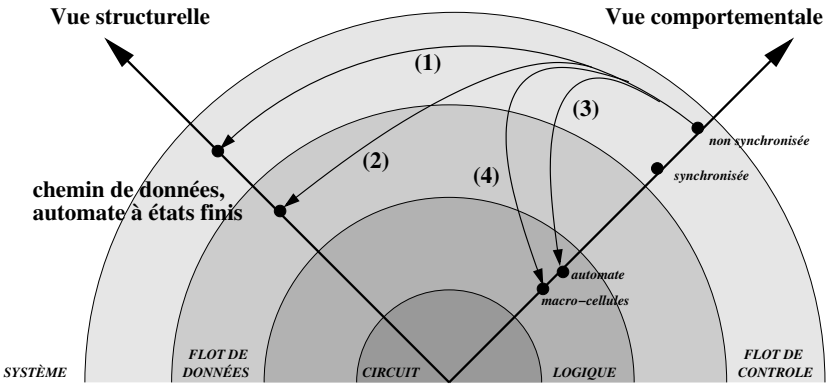


FIG. 2.18 – Schéma en Y de CGS.

La machine d'états est dite à gros grain, en effet, elle est calculée indépendamment de la fréquence visée en posant arbitrairement que les temps de propagation des cellules avec propagation de retenue (additionneurs, soustracteurs, ...) sont d'un cycle, que ceux des multiplieurs sont de 2 cycles et que ceux de tout le reste sont nuls.
Le rôle de CGS est de construire le chemin de données et d'ordonnancer les instructions en maximisant le parallélisme. [DQ07] a prouvé que le problème sous-jacent à l'algorithme utilisé est NP-complet et l'a amélioré.

FGS FGS est une réimplantation du "FSM retiming" d'ALMA et est donné sur la figure 2.15. Il diffère de celui-ci par le fait que d'une part l'arc 2b n'a pas été réimplanté et que d'autre part il ne travaille qu'au niveau de l'"ordre résolu" (voir page 34 les ordres résolu, concourant et séquentiel).
L'ordonnancement de FGS est donc moins performant que celui d'ALMA. Ceci n'est pas un problème car CGS a déjà ordonné les instructions RT au niveau "ordre séquentiel".

L'originalité de UGH par rapport aux autres outils de synthèse de haut niveau est l'expression des contraintes architecturales sous la forme d'un DDP. La contrainte minimale imposée par l'outil est l'allocation des variables (respectivement : des tableaux) du comportement sur des registres (respectivement : des RAM ou files de registres) du DDP. Ceci permet à l'utilisateur d'obtenir une implantation matérielle très proche de celle qu'il visait dans le cas des contraintes minimales et quasiment celle qu'il visait s'il complète le DPP.

2.3 Conclusion

ALMA a été testé en grandeur nature pour réaliser le bloc qui analyse le flux d'entrée et active les autres blocs en fonction des types d'images d'un circuit intégré décodant des flux MPEG1/2 en flux pour téléviseurs (NTSC, PAL, SECAM). Cette expérimentation a prouvé que le concept de FSM retiming est bien adapté à l'intégration d'un élément dans un système. Il a été aussi utilisé pour réaliser plusieurs prototypes de circuits tel un dictionnaire cascadable[4], un décompresseur sans perte adapté au réseau, et plusieurs filtres 2D basés sur la transformation d'Hadamar. Ces différentes expérimentations ont prouvé qu'une conception avec ALMA est bien plus rapide qu'une conception "manuelle".

[4] Ce processeur contient des mots, on peut en ajouter et rechercher si un mot existe, ou les mots correspondants à un patron (bo*r* → bonjour, bord, ...). Il est cascadable dans le sens qu'il est implanté comme un noeud d'arbre binaire. Le processeur propage les requètes venant de son père à ses fils et les réponses dans l'autre sens.

De nombreuses réalisations ont été faites avec UGH, les principales sont : une implantant les décodeurs de Huffman des décodeurs vidéo MPEG1/2, quatre implantant les tâches d'un décodeur JPEG (vld, idct, zz, iq), une implantant un algorithme de reconnaissance de visages. Ses réalisations ont montrées que UGH est très efficace en temps CPU nécessaire à la synthèse par rapport aux autres outils et en temps de développement pour l'utilisateur. Ses réalisations montrent que UGH traite aussi bien les applications à flot de contrôle que celles à flot de données. Cependant les applications à flot de données demandent un travail d'architecture au concepteur.

Reprenons point par point la problématique présentée au chapitre 2.2.3 (page 20) au jour de ces expérimentations.

Cercle vicieux Grâce au DDP et à sa contrainte minimale (allocation des éléments de mémorisation), les cercles vicieux n'existent plus. L'algorithme de CGS peut construire en parallèle le chemin de données et l'automate de contrôle. Bien que cet algorithme utilise un mécanisme pour lequel déterminer la faisabilité est NP complet, en pratique il converge très rapidement vers une solution qui est une solution acceptable, ceci est dû au fait que la contrainte minimale est très forte et limite grandement le nombre de degrés de liberté. Sur plusieurs exemples, François Donnet a montré dans [Don04] que la solution trouvée est à quelques pour cent en vitesse et en surface d'une solution optimale. Enfin notons que cet algorithme n'a rien de commun avec les algorithmes usuels de la synthèse de haut niveau qui construisent d'abord la machine d'états (ordonnancement) et ensuite le chemin de données.

Fréquence du coprocesseur cible ALMA comme UGH permettent d'adapter le coprocesseur au système dans lequel il sera intégré. Ceci est fait grâce au "FSM retiming" qui de manière générale ne coûte rien en surface contrairement au "DataPath retiming".

Communication Le protocole de base utilisé n'a posé aucun problème pour intégrer les coprocesseurs dans leurs systèmes respectifs. Au niveau de la synthèse l'utilisation de primitives prédéfinies de lecture et d'écriture bloquantes d'une donnée se traduisent facilement et ont effectivement évité d'introduire des primitives de synchronisation. Cependant, pour le concepteur elles sont trop limitatives. En effet, elles excluent de fait d'exécuter plusieurs opérations d'entrée/sortie en parallèle et de pipeliner lecture, traitement, écriture pour les applications travaillant par bloc. Cette remarque ne remet pas en cause les avantages de la prédéfinition des entrées et sorties pour la synthèse de haut niveau mais signifie juste que les primitives d'entrée/sortie de UGH sont trop restrictives et doivent être enrichies.

Exploration de l'espace de conception Grâce au DPP, UGH peut placer les points de Pareto quasiment sur la courbe "idéale" (figure 2.1). Il permet donc bien d'explorer l'espace de conception et il est un outil très efficace.

Utilisateur visé ALMA et UGH sont dédiés aux concepteurs de circuits. L'avantage de UGH par rapport aux autres outils de synthèse de haut niveau est qu'il ne change pas ses habitudes. En effet, le DDP n'est rien d'autre que le croquis qu'il fait à la main avant d'entrer son chemin de données dans un outil de CAD de circuits. UGH demande donc au concepteur de circuits de lui donner les grandes lignes de son chemin de données et l'automate de contrôle sous la forme d'un programme C. Cette partie intéressante du travail de conception est laissée au concepteur, UGH ne s'occupant que des tâches fastidieuses de la réalisation.

La réalisation de UGH réellement opérationnelle prouve que le concept de "FSM retiming" et l'"allocation des variables sur les registres par le concepteur" sont une solution viable pour la synthèse de haut niveau d'usage général. Bien sûr, UGH n'est pas parfait, de nombreuses améliorations peuvent être envisagées comme l'ajout de composants et de primitives d'entrée/sortie travaillant par bloc ou un traitement plus automatique des boucles (dépliement, repliement). Cependant celles-ci sont à un autre niveau et peuvent être envisagées avant d'entrer dans UGH, en source vers source par exemple.

Actuellement, je m'intéresse à l'adaptation de UGH pour des cibles FPGA. Le problème est que le "FSM retiming" un des principes de base de UGH n'est plus directement applicable sur les cibles FPGA. En effet, les temps de propagation, setup et hold des éléments du chemin de données d'un coprocesseur dépendent du placement routage dans le FPGA du système complet et ce placement changera si on change l'automate de contrôle du coprocesseur.
En 2006, une ébauche de solution a été trouvée. Dans le principe, il s'agit de générer une machine d'états où l'on peut insérer des cycles d'attente avant chaque état. Les nombres de cycles d'attente de chaque état sont stockés dans une RAM. Le contenu de cette RAM fait partie du bit-stream qui programme le FPGA et peut donc être initialisé en fonction du placement routage du système. En d'autre terme, cette solution est un "FGS retiming" réalisé dynamiquement par étalonnage lors de l'exécution.
En 2007, cette solution a été testée. On a d'abord conçu l'architecture de la machine d'états avec un nombre de cycles d'attente programmable par état et ajouté à UGH un driver la générant. Les coprocesseurs du décodeur JPEG ont été expérimentés. En définissant le nombre de cycles d'attente manuellement, on a réussi à les faire tourner sur FPGA. Cette expérimentation montre que le VHDL généré par UGH est correct et que l'approche est donc envisageable. Par contre, on ne sait pas à ce jour comment remplir cette RAM automatiquement.

Ce débroussaillage de UGH-FPGA ouvre un domaine de recherche prometteur.

Chapitre 3

Systèmes intégrés

3.1 Introduction

3.1.1 Cadre

La dénomination «Systèmes intégrés », bien que couramment admise, est ambiguë. En effet, par le terme « système » on entend généralement en informatique un matériel prédéfini sur lequel s'exécutent des applications très variées. Dans le cas qui nous intéresse, on part d'une application unique et bien spécifiée et on cherche à la réaliser à la fois en logiciel et en matériel sous contrainte de coût, puissance consommée, performance, etc, sans idées préconçues sur la nature logicielle ou matérielle de telle ou telle partie. L'intérêt de la conception de systèmes intégrés réside dans la variété des domaines qu'elle intègre. En effet elle mixe l'architecture de machine (assemblage de processeurs, mémoires, bus), système au sens noyau à installer sur les processeurs, conception matérielle et synthèse de haut niveau pour ajouter facilement des coprocesseurs, parallélisation d'application et communication, simulation fonctionnelle et temporelle du système intégré.

Vu ce grand nombre de domaines, les outils d'aide à la conception de systèmes intégrés ont deux écueils principaux à éviter. Soit ils se restreignent à un domaine et se pose alors le problème de leur interfaçage avec les autres outils, soit ils couvrent l'ensemble de la conception des systèmes intégrés et ils risquent d'être d'une trop grande complexité, ce qui les rend pratiquement inutilisables. Cette partie présente Disydent une suite logicielle complète dédiée à la conception de systèmes intégrés.

Les progrès technologiques (loi de Moore) des circuits ont permis dans les années 1990 d'intégrer un système intégré sur une seule puce que l'on appelle SoC (System-On-Chip). Durant ces années 1990, on pensait que les systèmes intégrés sur SoC étaient l'avenir pour tous les systèmes embarqués. Tout nouvel équipement léger comme un PDA[1] (Personal Digital Assistant) ou un capteur intelligent donnerait lieu à un SoC. On pensait donc que la croissance du nombre de systèmes intégrés sur SoC (conception conjointe matérielle et logicielle) serait exponentielle. Aujourd'hui on peut dire que les systèmes intégrés sur SoC ont été victimes de leur succès. En effet, le coût de réalisation physique d'un SoC (le circuit) étant extrêmement élevé, les industriels pouvant faire des affaires en modifiant et adaptant le logiciel d'un SoC existant, la durée de vie d'un SoC (le circuit) est très longue et l'apparition de nouveaux systèmes intégrés sur SoC relativement faible. Cette montée en puissance de l'utilisation des SoC avec leur réutilisation par reprogrammation a eu pour conséquence collatérale de diminuer drastiquement le nombre de réalisations de circuits spécialisés (ASIC : Application Specific Integrated Circuit) et est une des raisons de la

[1] Ce terme désigne tout équipement électronique numérique transportable dans une poche de veste : agenda, téléphone, appareil photographique, console de jeux, ...

crise touchant la micro-électronique.
Cette note industrielle pessimiste n'entâche en rien l'intérêt de la conception de systèmes intégrés. Pour la cible silicium (ASIC), elle introduit juste la contrainte supplémentaire que la plate-forme est plus figée que dans les années 1990. Et avec les nouveaux FPGA qui intègrent des processeurs, des unités de calcul en plus de leurs arbres de CLB (matrice de cellules programmables), la conception de systèmes intégrés sur cible FPGA est un domaine plein d'avenir.

3.1.2 Contribution

De 1996 à 2002, mon activité principale de recherche tourna autour de la conception d'outils dédiés à la conception de systèmes intégrés sur SoC. Cette recherche aboutit concrètement à Disydent une suite logicielle couvrant l'ensemble du processus de la conception de systèmes intégrés. Disydent est distribué [AP02web] depuis 2002 gratuitement sous licence GNU à l'adresse `www-asim.lip6.fr/disydent` .

Cette réalisation a été fortement corrélée au projet Esprit COSY (*COsimulation and SYnthesis*). Certains composants en particulier les schémas de communications, les blocs matériels, la synthèse de haut niveau faisaient partie intégrante de COSY et leurs développements ont été directement financés par le projet. D'autres comme le simulateur CABA ont été fortement inspirés de leur équivalent dans COSY, mais réécris et améliorés pour des raisons de licence.
J'ai été l'initiateur officieux de ce projet. En effet, j'avais quitté Philips 3 ans avant pour le laboratoire ASIM/LIP6 tout en gardant de bonnes relations avec ce premier, le projet est né de la volonté de Philips représenté par Jean-Yves Brunel de mixer l'outil de synthèse de haut niveau que j'avais développé lorsque j'étais employé par Philips avec un simulateur CABA développé par une équipe du centre de recherche de Philips à Eindhoven. J'ai ensuite participé activement à la définition du projet. Une fois que le projet fut accepté, j'ai été co-responsable avec Frédéric Pétrot du côté de ASIM/LIP6 de son suivi et de la coordination avec les partenaires au travers de réunions trimestrielles. Enfin à la fin du projet, je fus aussi co-responsable de la remises des livrables dus par ASIM/LIP6.

Grâce au financement apporté par le projet COSY, pendant environ quatre ans, Frédéric Pétrot et moi même avons assuré en parallèle la gestion du projet COSY ainsi que la conception et le développement de **Disydent** une suite logicielle complète et fonctionnelle dédiée à la conception de systèmes intégrés sur SoC. Concrètement nous avons coordonné et encadré pendant ces années une équipe d'une dizaine de personnes composée de stagiaires de M2, de thésards et d'ingénieurs en CDD.
Dans cette co-gestion, la spécification globale de Disydent, la définition générale des composants ainsi que leurs interfaces ont été collégiales. La réalisation des composants matériels a été sous-traitée à des ingénieurs en CDD, Frédéric Pétrot a développé et encadré la réalisation du simulateur CABA et du noyau et personnellement j'ai développé et encadré la réalisation de l'outil de synthèse de haut niveau.

Ce travail a été reconnu internationalement par une publication de fond dans la revue "IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems" [APD05]. Ce travail a donné lieu à trois thèses [Hom01, Don04, Gom06]. Il a aussi donné lieu à des publications internationales avec comité de lecture [ABG+97, APH01, HPA01a, HPA01b, ADP03], aux publications nationales [HP+A00, Don02] et aux livrables COSY [AP00a, AP00b].

3.2 Disydent

3.2.1 Problématique générale

Le point de départ de la conception d'applications intégrées est une application existante déjà validée sur une machine standard Unix ou Windows telle par exemple un décodeur MPEG ou un algorithme de reconnaissance de visages. Elle est décrite dans un langage séquentiel usuellement le C. On désire la migrer dans un autre environnement comme un PDA pour le décodeur MPEG ou comme une WEBCAM pour la reconnaissance de visage pour réaliser un capteur intelligent. Ce nouvel environnement induit une contrainte de consommation qui empêche d'avoir la puissance de calcul de la machine initiale. Une première solution serait de faire un circuit spécifique pour l'application mais elle n'est économiquement pas viable pour des raisons de coût. En effet, la conception du circuit demande un gros investissement et cet investissement sera peu ou prou à refaire pour intégrer l'application dans un environnement différent.

L'approche "système intégré" consiste à faire tourner l'application sur un ou plusieurs petits processeurs avec éventuellement quelques coprocesseurs les plus génériques possibles qui fournissent une accélération matérielle. Le résulat final est du côté physique une puce unique (SoC) qui intègre tout ce matériel, et du côté logiciel le noyau et l'application. Cette approche diminue d'une part l'investissement initial car les processeurs existent déjà et d'autre part elle permet sa réutilisation. En effet les parties logicielles peuvent être modifiées et les coprocesseurs génériques sont des IPs qui peuvent être réutilisés dans d'autres systèmes intégrés. En contrepartie de cette souplesse, cette approche introduit la contrainte de performance temporelle car l'architecture matérielle ciblée n'a pas la puissance de calcul des machines standards. Cette contrainte temporelle peut être le nombre minimal d'images à décoder par seconde pour le décodeur MPEG et le temps pour extraire les composantes caractéristiques d'un visage pour la WEBCAM intelligente.

Une personne se lançant dans une conception système intégré se trouve confrontée aux problèmes principaux suivants :

Parallélisation La structure du système cible nécessite une réécriture de l'application pour faire apparaître du parallélisme à gros grain entre tâches [Kah74, Hoa85], comme illustré sur la figure 3.1. Ce modèle permet, en effet, de mettre en évidence le gain que

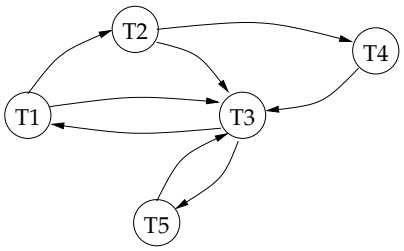


FIG. 3.1 – Une application décrite comme un graphe de tâches communicantes.

l'on aura en réalisant une tâche d'une manière ou d'une autre, car on peut avoir d'une part l'accélération fournie par telle ou telle implantation (accélération matérielle) et d'autre part le recouvrement d'exécution des différentes tâches entre elles et avec les flux de données (accélération par pipeline). L'utilité de ce modèle découle directement de la loi d'Amdahl [Amd67], qui nous indique que seule l'utilisation du parallélisme effectif peut permettre un gain significatif en performance pour une application.

La figure 3.2 illustre en pratique les limitations qu'impose la loi d'Amdahl. En a), nous avons la spécification séquentielle. En b), une partie de la spécification est

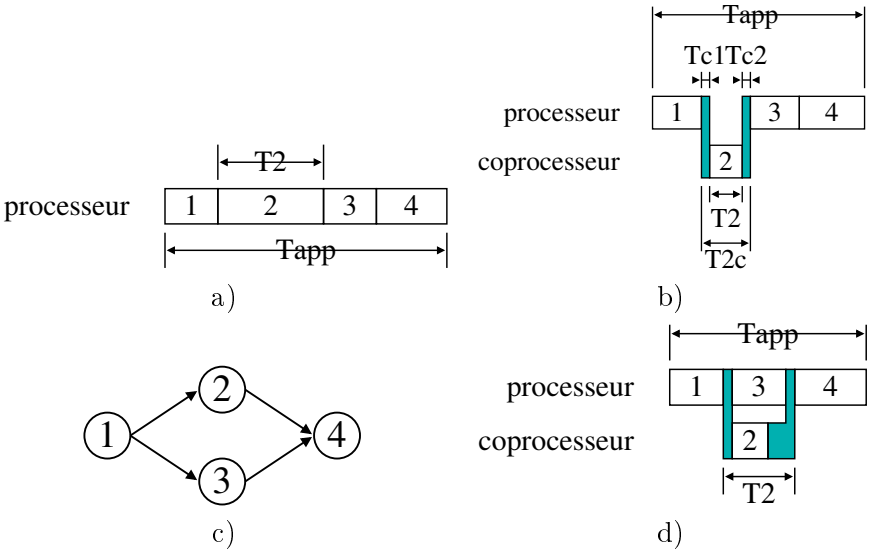


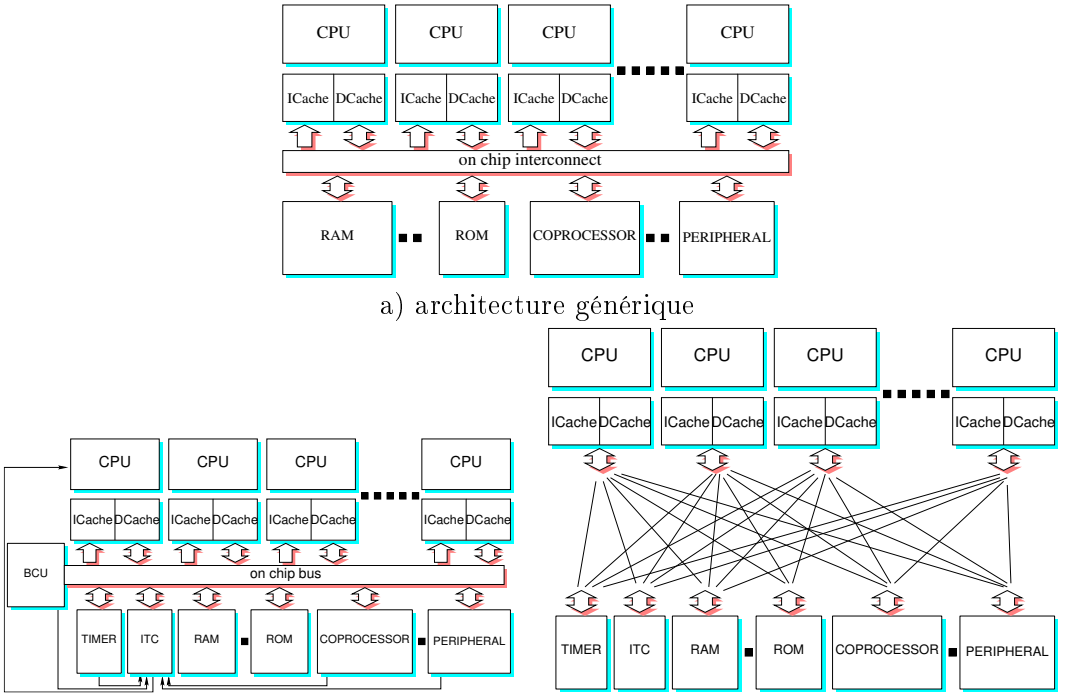
FIG. 3.2 – Illustration de la loi d'Amdahl

exécutée sur un accélérateur matériel, donc plus rapidement, mais l'on paye un coût pour la communication qui rend le gain négligeable ou même ralentit le système. En effet, Amdahl nous enseigne que le temps maximum que l'on peut gagner est le temps de la partie que l'on a accélérée. Si on fait apparaître explicitement les dépendances de données, comme en c), alors on a un recouvrement des temps d'exécution des différentes tâches mais aussi des temps de communications, si le matériel le permet, comme cela est illustré en d).

Plate-forme cible La plate-forme comprend d'une part l'architecture matérielle dont les éléments sont le nombre et les types de processeurs, le nombre et les types de co-processeurs et l'organisation des interconnections et d'autre part le logiciel de base c'est-à-dire les noyaux tournant sur les processeurs faisant l'interface entre le matériel et l'application.

Au niveau de l'architecture matérielle le choix de l'interconnection est particulièrement important car il définit les capacités du système en bande passante et donc son adéquation à une classe d'applications. La figure 3.3 illustre deux implantations (b et c) totalement différentes d'une même architecture générique (a). Le choix de l'architecture et le partitionnement ne peuvent pas être fait séparément selon [KM98]. Le choix du bus de communication, par exemple, dépend du débit qui est nécessaire aux différents éléments pour communiquer et des tâches affectées à ces composants. À l'inverse, suivant le débit du bus choisi, l'affectation des tâches aux éléments du système sera ou non réalisable. Néanmoins repartir de rien pour chaque conception de système intégré est très proche de la conception d'un circuit spécifique à l'application et est économiquement peu viable. L'utilisation d'une plate-forme cible soit prédéfinie, soit générique, est une nécessité économique et est de plus en plus utilisée bien qu'elle limite les degrés de liberté de la conception et que selon Wayne Wolf [Wol01] « L'approche de la conception à base de plate-forme est en quelque sorte l'antithèse de la co-conception matérielle/logicielle ! »

Les communications entre les tâches de l'application sont aussi un problème crucial. Les types de communication possibles sont présentés sur le tableau 3.1 suivant la nature logicielle ou matérielle de l'émetteur et du récepteur et si le bus de la plate-forme est utilisé ou pas. Pour chaque type de communication, le nombre de protocoles



a) architecture générique

b) architecture mono-bus

c) architecture à réseau d'interconnections

FIG. 3.3 – Choix de l'architecture de communication du système.

type	émetteur	récepteur	via
SW → SW	logiciel	logiciel	bus
SW → HW esclave	logiciel	matériel	bus
SW → HW maître	logiciel	matériel	bus
HW esclave → SW	matériel	logiciel	bus
HW maître → SW	matériel	logiciel	bus
HW maître → HW esclave	matériel	matériel	bus
HW esclave → HW maître	matériel	matériel	bus
HW → HW	matériel	matériel	direct

TAB. 3.1 – types de communication entre tâches

possibles est très grand et ces protocoles requièrent pour les tâches logicielles un noyau supportant quelques ressources de base comme les tampons de mémoire partagée et les sémaphores et pour les tâches matérielles des ressources physiques telles que les interruptions, les DMA (Direct Access Memory) ainsi qu'éventuellement leur gestion au niveau du noyau. Dans la conception de systèmes intégrés, le concepteur peut décider de migrer une tâche du logiciel vers le matériel ce qui nécessite la modification de toutes les primitives de communication dans les tâches logicielles communiquant avec la tâche migrée. Ce travail laborieux est un frein à l'exploration de l'espace de conception.

Dans la conception de système intégré, le choix des schémas de communication peut être laissé au concepteur ou les schémas de communication peuvent faire partie de la définition de la plate-forme cible en tant que composants matériels et de supports logiciels au niveau du noyau. Cette alternative est tout à fait similaire au choix de la plate-forme. En effet, on peut arguer que les schémas de communication dépendent de l'application, mais l'investissement pour les concevoir, les implanter et les valider

logiciellement et matériellement est démesuré par rapport au gain de performance obtenu par l'utilisation de composants de communications prédéfinis optimisés.
De plus, la définition des schémas de communication comme partie intégrante de la plate-forme permet aux outils d'aide à la conception de systèmes intégrés d'en tenir compte (mesurer les temps dus aux communications, déterminer des risques d'interblocage, générer le code appelant les primitives de communication, ...) et de les soulager des problèmes de synthèse des communications.

Exploration de l'espace de conception et validation Pour que la conception de système intégré soit viable, il faut que le concepteur puisse facilement explorer l'espace de conception qui est immense. Cette immensité est due aux nombreux degrés de liberté décrits dans les deux items précédents comme le découpage en tâches, l'affectation des tâches en logiciel ou matériel, le choix ou paramétrage de la plate-forme. Explorer signifie concrètement décrire un point de l'espace puis le valider. Il y a deux niveaux de validation. Le premier est fonctionnel : le système intégré est-il bien équivalent à l'application séquentielle initiale ? Le second est la validation des contraintes, la plus ardue étant la contrainte temporelle : le système intégré tourne-t-il assez vite ?
La vérification fonctionnelle consiste à vérifier si la réorganisation des sources séquentielles initiales de l'application en tâches communicantes est correcte. Ceci peut être fait sur une machine standard par simulation. En effet, il ne s'agit pas ici de prouver que l'application est correcte mais une équivalence et l'application initiale peut fournir autant de jeux d'essais que nécessaires et les vitesses d'exécution des deux modèles sont du même ordre de grandeur. Cependant *cette approche par simulation n'est valide que si les résultats produits par le modèle parallèle ne dépendent pas de l'ordonnancement des tâches* [Par95].
La vérification temporelle consiste à mesurer le temps mis par le système intégré pour réaliser des fonctions caractéristiques. A ce niveau interviennent de nombreux aléas tels que les défauts de cache des processeurs, les temps de communication, les conflits de prises de bus, les famines des coprocesseurs, ... Ces aléas ne sont pas modélisables et une approche par simulation temporelle est nécessaire. Si l'on se place au niveau porte, l'exploration de l'espace de conception devient impossible à cause d'abord du temps nécessaire pour générer le modèle de simulation et ensuite à cause de la lenteur de simulation, il faut déjà compter au moins plusieurs heures rien que pour booter les noyaux. Cette lenteur est proportionnelle au nombre de processeurs de la plate-forme. Pour cette validation temporelle, le concepteur doit disposer d'un simulateur au niveau cycle de la plate-forme (CABA : Cycle Accurate Bit Accurate) [Jen91] plus rapide de plusieurs magnitudes par rapport au niveau porte et plus lent d'une seule magnitude par rapport à la simulation fonctionnelle sur une machine standard. Ce simulateur CABA doit aussi éventuellement pouvoir s'interfacer avec un simulateur au niveau porte pour les coprocesseurs.

Interface Dans ce chapitre, le terme *interface* est pris au sens large. On évoquera ici, les interactions au niveau conceptuel entre l'utilisateur et les outils d'aide à la conception de systèmes intégrés.

- Les estimateurs : les outils d'aide à la conception de systèmes intégrés sont usuellement de grands consommateurs d'estimateurs.
  Par exemple, les performances temporelles de l'application sur l'architecture cible peuvent être déterminées par une simulation sur une machine standard avec un estimateur qui convertit les temps d'exécution de la machine standard en temps d'exécution de la machine cible ou qui annote le code de l'application à la manière d'un profileur. Ces outils indiquent uniquement une espérance de la déviance tournant autour de 0.2. La complexité [MM+97] de l'exécution d'une application

Attention : cette figure n'est pas un traité sur la prévision météo mais une explication probabiliste de ce qui se cache sous l'expression usuelle "*Cet estimateur est fiable à 80%*".

Quand Météo-France fait une prévision météorologique de 5 sur une échelle de 1 à 9, il donne un indice confiance d'environ 0,65 si cette prévision est à 3 jours.
Cela signifie grosso-modo qu'il y a une probabilité de 0,65 pour que la météo soit 4, 5 ou 6 et une probabilité de 0.35 pour que la météo soit 1, 2, 3, 7, 8 ou 9.

En posant X la variable aléatoire donnant la météo, en considérant que la météo est aléatoire[a] dans chaque groupe (eg : $P_i = 1/6, \forall\, i \in \{1,2,3,7,8,9\}$ et $P_i = 1/3, \forall\, i \in \{4,5,6\}$), alors l'espérance de la variable Y donnant la déviance de X lorsque X=5 est :

$$\begin{aligned} E(Y)_{X=5} &= 0,35/6\ .\ (|1-5| + |2-5| + |3-5|) \\ &+ 0,65/3\ .\ (|4-5| + |5-5| + |6-5|) \\ &+ 0,35/6\ .\ (|7-5| + |8-5| + |9-5|) \\ &= 1.48 \end{aligned}$$

De même, on a : $E(Y)_{X=1} = 2,1$; $E(Y)_{X=2} = 2,0$; $E(Y)_{X=3} = 1,7$; $E(Y)_{X=4} = 1,5$; $E(Y)_{X=5} = 1,4$; $E(Y)_{X=6} = 1,5$; $E(Y)_{X=7} = 1,7$; $E(Y)_{X=8} = 2,0$; $E(Y)_{X=9} = 2,1$.
soit en faisant la moyenne on a E(Y)=1.8. En tenant compte de l'échelle à 9 éléments, on peut dire que l'erreur de la variable X est de 20%.

Donc Météo-France pourrait annoncer plus glorieusement sa prévision avec une erreur de **20%** ou bien une fiabilité de **80%** au lieu d'un indice de confiance de seulement **0,65**.

[a] c'est une grossière approximation mais suffisante pour ce rappel de probalité.

FIG. 3.4 – Espérance de déviance et indice de confiance.

multi-tâches tournant sur une machine multi-processeurs où interviennent entre autre la taille des caches, l'ordonnancement des tâches (noyau), les conflits sur le bus, laisse à penser que l'indice de confiance doit être assez faible comme le montre la figure 3.4. Choisir une architecture sur un tel estimateur peut conduire avec une probabilité non négligeable à une voie sans issue ou à une solution surdimensionnée.

Un autre exemple classique est utilisé dans la synthèse de haut niveau. Lors du choix de l'architecture du chemin de données plusieurs solutions sont essayées et un estimateur choisit la meilleure. Ici les concepteurs de l'outil avouent qu'ils sont incapables de faire un estimateur juste et affirment que ce n'est pas grave car leur estimateur est utilisé comme une relation d'ordre. Ils assurent cependant que l'estimateur la respecte ! L'indice de confiance n'est pas mentionné et il ne doit pas être très élevé vu le nombre important de paramètres mis en jeu tels le nombre de cellules et de connections et plus gênant l'outil de placement-routage de bas niveau. De ce fait, la probabilité que l'outil fasse un très mauvais choix n'est pas négligeable.

D'abord les estimateurs doivent être utilisés avec parcimonie. Ensuite, lors de l'utilisation d'un estimateur, si on ne peut garantir une espérance faible de la déviance avec un indice de confiance élevé, il est indispensable de prévoir un mécanisme **simple** permettant à l'utilisateur de se substituer au choix de l'estimateur.

– Le degré de compétence : la conception de systèmes intégrés couvre de fait les domaines architecture de machines, système au sens noyau, architecture matérielle, algorithmique parallèle et génie logiciel. Une suite logicielle dédiée à la conception de système va avoir des outils couvrant ces différents domaines ou à cheval sur plusieurs. Si un outil nécessite que l'utilisateur soit un expert dans chacun des domaines, il sera peut être génial mais inutilisable par le commun des mortels. Pour chaque outil, il est important de définir le domaine de compétence de l'utilisateur final.

  Par exemple, un algorithmicien est la personne adéquate pour découper l'application séquentielle en tâches concurrentes communicantes ou changer la représentation des réels en virgule fixe. On peut en plus lui demander de tester ses algorithmes sur une plate-forme générique et de choisir les paramètres génériques. On ne peut pas, par contre, lui demander de connecter les composants d'une machine ou de réaliser un coprocesseur matériel. Donc si on considère que la tâche de découpage en processus communicants est du ressort d'un algorithmicien, on a besoin d'un outil qui génère un simulateur temporel du système intégré en fonction des paramètres génériques.

  Prenons une analogie : avant le WEB basé sur HTML, il y a eu une expérience conceptuellement identique mais basée sur SGML. SGML est un langage de description complet de document avec feuille de style. Pour mettre un document SGML sur cette ancienne toile, il fallait le compiler avant de pouvoir le visualiser avec un browser. En comparaison, HTML est très pauvre (la documentation de HTML dans ses premières versions tenait sur quelques pages) et de plus un document HTML est visualisable immédiatement dans un browser même si le document contient des erreurs HTML. L'échec de SGML et le succès de HTML s'expliquent par la vitesse du retour sur l'investissement attendue par l'utilisateur. En effet, les concepteurs de cette expérience avaient sous-entendu qu'un utilisateur lambda savait premièrement ce qu'était un compilateur, et deuxièmement qu'il était prêt à s'investir dans l'apprentissage d'un langage avant de pouvoir réaliser un document. La fin de l'histoire est qu'au cours du temps, HTML a largement atteint la

complexité de SGML avec les XML et des nouveaux outils WEB comme JAVASCRIPT, PHP, AJAX l'ont même pulvérisé mais à vitesse de retour sur investissement constante car PHP s'adresse à des utilisateurs qui connaissent déjà HTML. Lors de la définition d'un outil pour la conception de systèmes intégrés il faut maximiser cette vitesse de retour sur l'investissement attendue par l'utilisateur. Pour un outil cela consiste 1) à limiter aux fondamentaux du domaine les connaissances requises à son utilisation, 2) ne pas changer les habitudes de l'utilisateur, et 3) à éviter que la connaissance interne de l'outil soit nécessaire pour son utilisation. Pour le 1) on peut par exemple supposer qu'un algorithmicien puissent développer en C++ mais il est plus osé de lui demander de développer en VHDL. Pour le 2) un outil de synthèse de haut niveau sera d'autant mieux accepté par un micro-architecte qu'il suit le schéma usuel de la conception de circuit. Le syndrome de la boîte noire illustre le 3). Il est demandé au concepteur de rentrer les vues de la boîte noire pour les différents outils, ce qui nécessite au mieux la compréhension du fonctionnement des outils quand la vue est décrite par un fichier de configuration et au pire leur implantation précise quand la vue est décrite par un programme lié (éditeur de lien) avec les bibliothèques de l'outil.

– La complexité : des problèmes principaux précédents, il résulte un écueil supplémentaire pour le concepteur. En effet, il va se trouver face à un nombre important de modèles de son application comme le modèle séquentiel initial, le modèle de simulation fonctionnel, les modèles pour le système intégré suivant que telle tâche est implantée de manière logicielle ou matérielle et éventuellement les modèles pour le simulateur CABA pour les coprocesseurs. Il a alors la tâche ardue de maintenir la cohérence entre ces modèles.

Disydent qui est une suite logicielle complète dédiée à la conception de systèmes intégrés est l'objet du chapitre suivant. Elle implante les choix techniques fondamentaux présentés ci-dessus et se propose de les valider. Cette implantation résout également les problèmes soulevés dans ce chapitre.

3.2.2 Description

3.2.2.1 Principe

Disydent est un environnement de développement pour concevoir des systèmes intégrés à base de plate-forme. Soit un triplet (*système*,*application*,*contraintes*) dans lequel le *système* existe et l'*application* n'est pas encore supportée par le *système*, le problème est d'ajouter au *système* l'*application* en respectant les *contraintes*. Par système existant, on entend un ensemble de composants matériels et logiciels opérationnels, mais au niveau matériel, le système n'est pas figé, il peut être enrichi par l'ajout de processeurs et de coprocesseurs.

Pour réaliser une implantation de son triplet (*système*,*application*,*contraintes*), un utilisateur doit fournir les deux entrées : l'application et le système. Les contraintes ne sont pas des entrées de Disydent mais des propriétés (surface, rapidité, consommation) que la solution doit satisfaire. La première entrée est un modèle de l'application, elle se présente sous la forme d'un réseau restreint de processus de Kahn où chaque processus est décrit en langage C. Un réseau restreint de processus de Kahn est un ensemble de processus séquentiels qui communiquent au travers de FIFOs de taille finie avec un unique producteur et un unique consommateur. De plus la lecture sur une FIFO vide est bloquante ainsi que l'écriture sur une FIFO pleine. Parks dans [Par95] a prouvé que ces réseaux ont un comportement totalement indépendant de l'ordonnancement des processus si les tailles des FIFOs sont suffisantes. Par la suite, nous appellerons ces réseaux KPN (Kahn Process Network).

La seconde entrée est une plate-forme basée sur celle présentée sur la figure 3.3.b. C'est une plate-forme dont les composants sont connectés à un bus, ses principaux paramètres sont le nombre de processeurs avec la taille de leurs caches, la taille des mémoires. Elle peut de plus être enrichie par des coprocesseurs.

La spécification de l'application par un KPN ne permet pas de décrire toutes les applications. Un utilisateur peut se sentir frustré d'être obligé d'utiliser des FIFO pour communiquer au lieu d'autres mécanismes qu'il juge plus performants comme la mémoire partagée ou des sémaphores. En fait, les FIFOs de Disydent, (décrites dans [Hom01]) sont implémentées par des composants solides et très optimisés qui utilisent en interne ces mécanismes et même du DMA (direct memory access). De notre point de vue, concevoir en utilisant les FIFO de Disydent est plus un gain de temps qu'une frustation et évite d'implanter et réimplanter des mécanismes similaires de communication. Cependant Disydent n'empêche pas un utilisateur de quitter le monde déterministe des réseaux restreints de processus de Kahn par exemple en communiquant au moyen de variables globales dans les tâches logicielles avec des instructions de lecture et l'écriture dont l'ordre dépend de l'ordonnancement des tâches ou en effectuant des tests non bloquants pour savoir si une FIFO en lecture est vide ou si une FIFO en écriture est pleine.

Le fait que le système généré ne soit qu'une extension d'une plate-forme paramétrée peut être vu comme une limitation. Cependant la prise en compte de l'architecture cible très tôt dans le processus de développement libère le concepteur de la conception matérielle, de l'ajout de drivers aux noyaux et de nombreux problèmes logiciels comme la cross-compilation, le portage de bibliothèques ou de co-simulation (VHDL et C). De plus, ceci reflète parfaitement ce que l'on trouve aujourd'hui dans le monde industriel. Les sociétés de semiconducteurs fabriquent des plate-formes pour les intégrateurs de système qui utilisent aussi des noyaux produits par d'autres sociétés spécialisées.

3.2.2.2 Outils

Disydent est basé sur 5 outils qui permettent à l'utilisateur de passer de spécifications fonctionnelles à une implantation réelle. Ceux-ci sont :

DPN DPN est une librairie C qui implante les communications FIFO des réseaux de processus de Kahn au dessus des threads POSIX.

Plate-forme générique Le gabarit de la plate-forme est donné sur la figure 3.3.b. Le matériel est composé d'un ou plusieurs processeurs MIPS R3000 avec leurs caches, d'un PI-bus [PI96] comme interconnect, d'un contrôleur d'interruption, de RAM et ROM. Il peut être étendu en connectant au bus, des interfaces FIFO sur lesquelles on branche des coprocesseurs.
Le logiciel de base est un micro-noyau appelé Mutek qui implante les threads POSIX et est multi-processeurs.

CASS CASS est un simulateur précis au niveau cycle d'un ensemble de composants matériels (RAM, BUS, processeur, cache, ...) interconnectés. Chaque composant doit être modèlisé au niveau CABA par une description en langage C. On appelle module CASS cette description d'un composant. Disydent fournit un module CASS pour tous les composants matériels de la plate-forme générique.

UGH UGH est un outil de synthèse de haut niveau, il prend en entrée un programme C et une période d'horloge. Il produit un modèle VHDL synthétisable et un module de simulation CASS. Il est décrit dans le chapitre 2.2.6 page 34 de ce document.

DSG DSG est la clef de voûte de Disydent. Il prend en entrée un réseau de processus de Kahn et une implantation de celui-ci sur le système. L'implantation comprend :

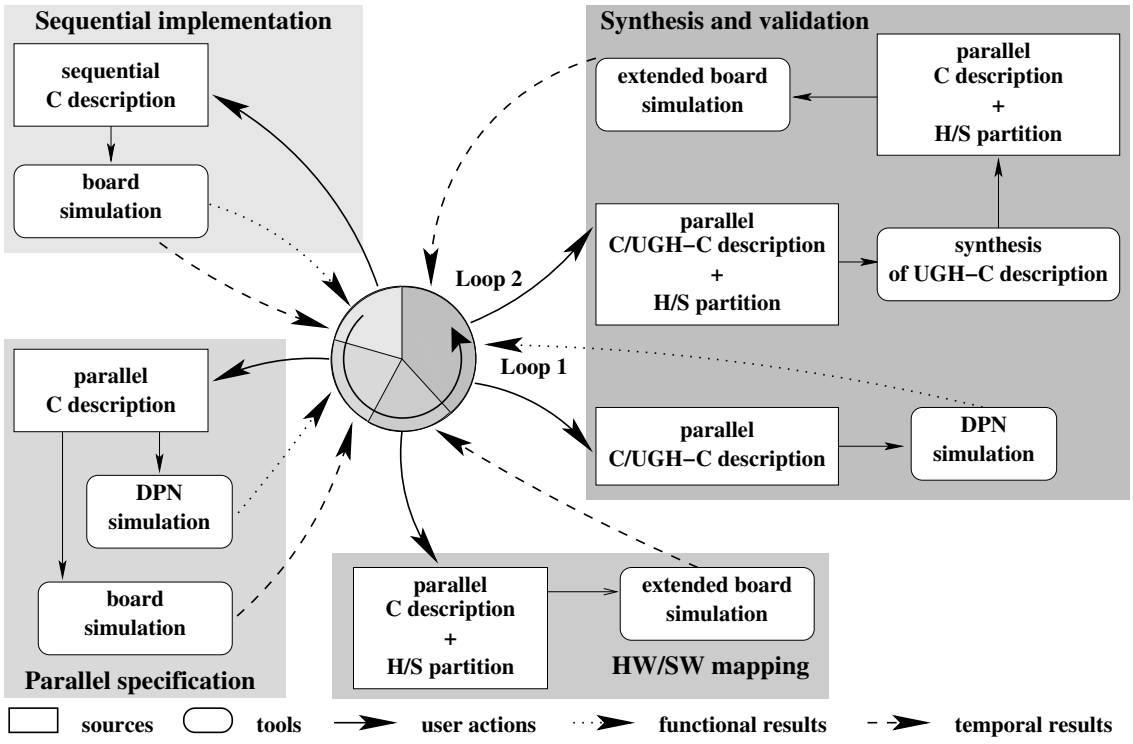


FIG. 3.5 – Le flot de conception de Disydent.

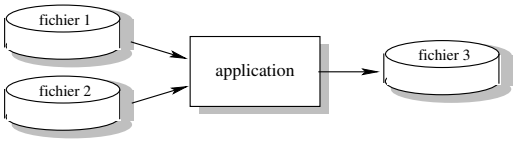
- Le nombre de processeurs et la taille de leurs caches.
- La profondeur de chaque FIFO.
- Pour chaque processus du réseau de processus de Kahn, une projection logicielle, matérielle ou matérielle exploratoire (voir la migration exploratoire page 52).

Il génère d'une part le simulateur fonctionnel DPN de l'application et d'autre part le simulateur fonctionnel et temporel CASS du système. Ce dernier comprend la configuration de la plate-forme matérielle et le logiciel tournant sur la plate-forme.

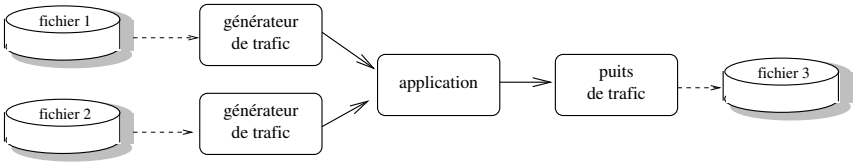
3.2.2.3 Flot de conception

Pour résoudre un problème (*système*,*application*,*contraintes*), Disydent propose une méthode en 4 étapes principales qui est schématisée sur la figure 3.5. Le rond central représente le concepteur. L'*implantation séquentielle* est le programme C sans aucun parallélisme qui spécifie l'application à ajouter au système. L'*implantation parallèle* est une réécriture de l'application comme un réseau de processus de Kahn. Elle doit faire ressortir le parallélisme à gros grain de l'application. Dans l'étape *projection matérielle/logicielle*, le concepteur doit décider quelles tâches du réseau de processus de Kahn sont à migrer sur du matériel dans un coprocesseur. Enfin la dernière étape, *synthèse et validation*, comprend la synthèse des coprocesseurs implantant les tâches matérielles et la validation temporelle du triplet initial.

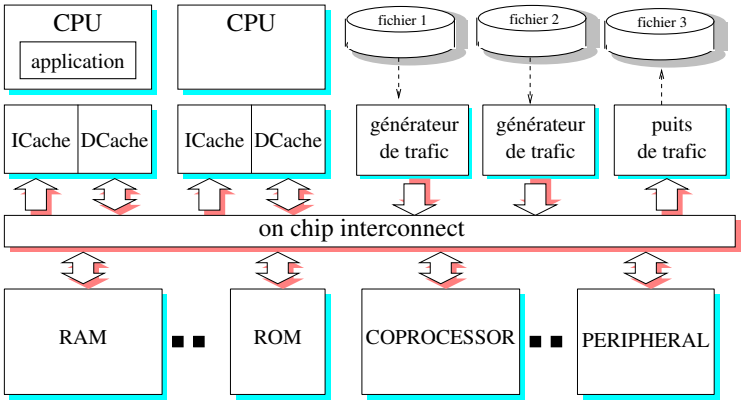
Comme le montre la figure 3.5, le concepteur joue un rôle central. Disydent ne se substitue pas au concepteur, il se contente de lui fournir à chaque étape des informations pertinentes pour l'aider à choisir entre valider l'étape courante et donc passer à l'étape suivante, ou retravailler l'étape en cours pour la valider, ou rejeter l'étape courante pour revenir à l'étape précédente. Cette approche pas à pas dans la conception permet à l'utilisateur d'élaguer très tôt l'espace de conception et d'aboutir rapidement soit à une solution viable ou à une absence de solution en lui évitant de s'engouffrer dans des voies sans issue.



a) source initial



b) réseau de processus de Kahn



c) implantation sur la plate-forme

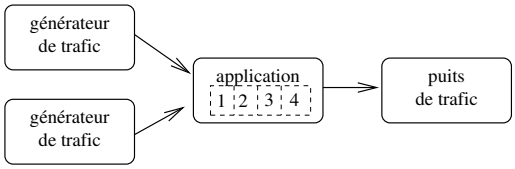
FIG. 3.6 – Transformation d'un source initial en un réseau de processus de Kahn de base.

Implantation séquentielle Lors de cette phase, le concepteur adapte d'abord la description de base de l'application conçue pour des machines standards au système. Cela consiste à remplacer dans le code initial (figure 3.6.a) les entrées/sorties sur fichier par les primitives de lecture/sortie sur FIFO de DPN et à créer les processus générateurs et puits de trafic (figure 3.6.b). Un générateur de trafic est une tâche qui fournit les données à l'application. Un puits de trafic est une tâche qui récupère les données résultats de l'application. Pour un décodeur MPEG, le générateur peut être un périphérique qui recoit un flux vidéo sur un canal hertzien ou autre, le puits de trafic peut être un le pixmap d'un écran LCD. Sur une machine standard, il peut vérifier grâce à DPN que les deux descriptions sont équivalentes. Si les générateurs et puits de trafic n'existent pas déjà sur le système, il doit alors les ajouter au système au niveau physique et au niveau CASS.

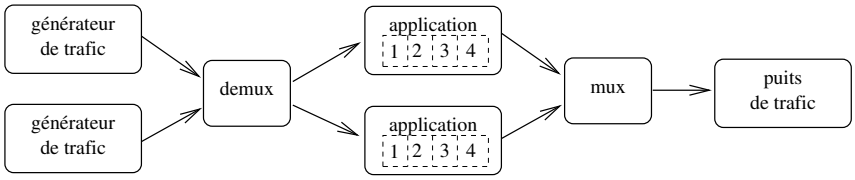
Ensuite, il place une suite de points de mesure dans le processus principal du KPN et exécute ce KPN sur le système au travers du simulateur CASS avec un jeu d'essai. Les processus générateurs et puits de trafic sont alors instanciés sur des composants physiques du système comme le montre la figure 3.6.c. Il obtient alors le nombre exact de cycles mis par le système pour exécuter le jeu d'essai ainsi que les nombres de cycles passés entre chaque point de mesure.

En comparant sa contrainte temporelle avec le nombre total de cycles, il obtient immédiatement et de manière fiable la difficulté du problème (*système*,*application*, *contraintes*). Les nombres de cycles entre les points de mesure lui donnent un profil précis de son application très utile pour l'étape suivante. Ce profil est bien plus précis qu'un profil classique de l'application originale obtenu sur une machine standard car

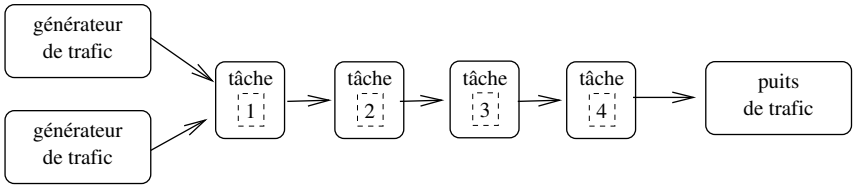
il est obtenu sans altération du programme cible et il tient compte des défauts de cache, des temps d'attente. Il est absolument exact.



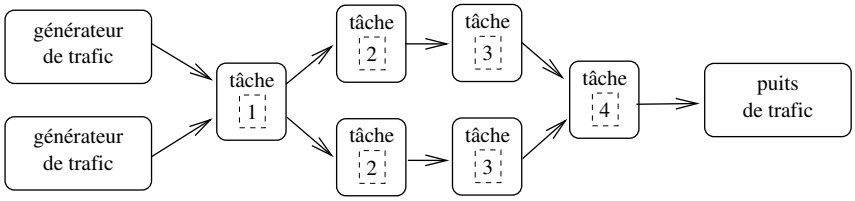
a) modèle séquentiel



b) modèle parallèle pur



c) modèle parallèle pipeliné



d) modèle parallèle mixte

FIG. 3.7 – Parallèlisme à gros grain d'un réseau de processus de Kahn.

Implantation parallèle En utilisant le profil obtenu à l'étape précédente, le concepteur décompose la tâche principale en processus pour faire apparaître du parallélisme. Sur la figure 3.7 le concepteur a isolé quatre tâches (figure 3.7.a), il peut opter pour du parallélisme pur (figure 3.7.b), pour un parallélisme temporel (figure 3.7.c), ou un mixte des deux (figure 3.7.d). Concrètement, cette décomposition consiste en une réorganisation du code comme dupliquer les variables globales et les initialiser dans chaque processus à partir de FIFOs d'entrée.

Pour choisir le découpage le plus prometteur, il doit d'abord valider le modèle parallèle sur une machine standard au travers de DPN. Puis il lance des exécutions du modèle parallèle sur le système avec CASS en faisant varier le nombre de processeurs du système. Pour chaque exécution, il obtient les informations : nombre total de cycles, le nombre de cycles passés dans chaque processus, le nombre de cycles passés dans le système comprenant les commutations de tâches et les commutations, la charge de bus. Il peut aussi obtenir d'autres informations de profil en plaçant des points de mesure comme à l'étape précédente.

A ce stade, il se peut que le problème (*système*,*application*,*contraintes*) ait une solution sinon il faut envisager une accélération matérielle, choisir le découpage le plus

prometteur et passer à l'étape suivante.

Projection matérielle/logicielle La conception d'un coprocesseur matériel est très coûteuse en terme de main-d'oeuvre et de ce fait elle ne doit être entreprise qu'avec une probabilité de quasi 100% qu'elle serve réellement à quelque chose. Le coût de la conception est au moins proportionnel au facteur d'accélération attendu. Ce facteur d'accélération doit être connu avant la réalisation du coprocesseur, en effet la réalisation d'un coprocesseur d'accélération d'un facteur n ne sert à rien pour la conception d'un accélérateur matériel d'un facteur $2n$. En effet, le chemin de données est à repenser complètement et même probablement l'algorithme. L'objectif de cette phase est de sélectionner parmi les tâches du KPN choisi à l'étape précédente, celle ou celles dont une accélération matérielle permet d'avoir une solution au problème (*système*, *application*,*contraintes*), et pour chacune de ces tâches les caractéristiques temporelles minimales nécessaires.

Pour cela Disydent propose une approche de migration exploratoire. Concrètement, l'utilisateur indique l'ensemble des tâches qu'il veut matériellement, en indiquant pour chacune d'elles un nombre de cycles d'attente NCA. Puis il exécute son modèle sur le système au travers de CASS. Pendant cette exécution, les tâches de l'ensemble vont se dérouler dans des coprocesseurs branchés sur le bus de la plate-forme. Le modèle temporel de ces coprocesseurs est au plus une lecture ou une écriture par cycle et NCA cycles entre 2 opérations d'entrée/sortie. Ainsi un NCA nul donne la borne supérieure de performance tenant compte des délais induits par les communications pour une vitesse de calcul maximale d'un cycle. CASS donnant les mêmes informations de profil qu'aux étapes précédentes, un concepteur de circuit peut déterminer les tâches bonnes candidates à la migration d'une part et d'autre part son expérience et les NBA lui indiquent si le développement des coprocesseurs est jouable ainsi que sa difficulté.

Au niveau implantation, cette migration exploratoire se fait automatiquement sans modification de code. La migration matérielle d'une tâche intègre le code source du processus KPN sans aucune modification grâce à un wrapper DPN vers CASS. Le wrapper ne diminue pas les performances des simulations CASS, il les améliore même car le code de la tâche n'est plus émulé par le module CASS d'un processeur mais tourne directement sur le processeur de la machine faisant tourner le similateur CASS.

Synthèse et validation Le concepteur doit créer pour chaque tâche sélectionnée à l'étape précédente la description C et le DDP (draft of the data-path) pour l'outil de synthèse UGH. Bien qu'il puisse, en théorie, partir de la description C du processus du KPN, ceci n'est pas possible en pratique. En effet, d'une part, il doit fournir un travail de conception de circuit pour trouver du parallélisme à grain fin, celui-ci étant absent de la description KPN, et d'autre part, il doit tailler les variables et les opérateurs au bit près afin de diminuer le coût de surface et de consommation du coprocesseur. Ceci fait qu'il doit réécrire la tâche de manière complètement différente.

Dans Disydent, cette nouvelle description en C synthétisable par UGH peut être utilisée au niveau DPN ce qui permet de la valider sur une machine standard. Une fois cette validation comportementale faite, UGH synthétise le coprocesseur et en plus du modèle VHDL synthétisable il génère un modèle CABA du coprocesseur directement utilisable par CASS. De ce fait, le concepteur peut valider temporellement au cycle près la projection matérielle/logicielle au travers de CASS. A ce stade, suivant les résultats de la validation temporelle, il peut soit retravailler l'architecture des coprocesseurs, soit revenir à l'étape précédente. Ce dernier cas implique qu'il a fait

précédemment une grosse erreur de jugement.

3.3 Conclusion

Disydent a été testé avec succès sur l'exemple significatif du MJPEG. Le MJPEG (Motion JPEG) consiste à décoder un flux vidéo composé d'une suite d'images de même taille au format JPEG.
Reprenons point par point la problématique présentée au chapitre 3.2.1 (page 41) au jour de ces expérimentations.

Parallélisation Les réseaux restreints de processus de Kahn ont permis de décrire sans problème ces 2 exemples et ont permis de trouver suffisamment de parallélisme pour tenir les contraintes temporelles. La sémantique solide des réseaux restreints de processus de Kahn permet une validation fonctionnelle sûre du modèle parallèle avec DPN directement sur une machine standard à une grande vitesse.
Dans cette modélisation, le seul bémol détecté est l'absence de variable globale de configuration. En effet, dans l'exemple du MJPEG il faut propager la taille de l'image à tous les processus du KPN au moyen d'une FIFO propre, ce qui n'est pas très économique et un peu pénible pour le concepteur.

Plate-forme cible Le système cible conçu (matériel, noyau et communication) s'est comporté tout à fait convenablement lors de ces expérimentations. Aucun goulet d'étranglement au niveau des communications n'a été détecté, ce qui valide l'intégration des communications à la définition du système et les avantages qu'elle apporte. Les principaux sont : le gain de temps (4 semaines pour le MJPEG avec deux coprocesseurs), l'automatisation de la génération des différents simulateurs et du logiciel intégré tenant compte du partitionnement logiciel/matériel.

Ces performances sont dues essentiellement à l'existence du système qui, lui, a par contre demandé plusieurs hommes/an de développement mais qui est factorisé avec toutes les résolutions de problèmes (*système*,*application*,*contraintes*) avec ce système.
La configuration de Disydent pour une autre plate-forme est une tâche compliquée. D'abord elle nécessite la définition du système cible. Ensuite il faut décrire au niveau RTL tous ses composants matériels. Enfin les modèles de simulation CABA pour CASS doivent être écrits et un noyau POSIX doit être adapté au système.

Exploration de l'espace de conception et validation Disydent permet d'atteindre tous les points de l'espace de conception. La méthode proposée permet de cibler très rapidement une solution à un triplet (*système*,*application*,*contraintes*). Les points forts sont :

1. Les différents niveaux de simulation (DPN pour la validation fonctionnelle, CABA pour la validation temporelle et surtout la simulation CABA de la migration exploratoire) qui retournent les informations permettant au concepteur de détecter très tôt les voies sans issue.
2. L'intégration des outils entre eux et des outils au système. Une modification faite, il ne faut au maximum que quelques minutes pour lancer la simulation qui l'infirmera ou l'affirmera.
3. Les résultats fournis par les simulations sont exacts avec un indice de confiance de 1 par rapport au système réel. Comme la validation temporelle ne passe pas par des estimateurs, il est garanti qu'une solution trouvée à un triplet (*système*,*application*,*contraintes*) sera une solution réelle et qu'une solution limite fonctionnera. Ce dernier point est particulièrement important car il évite au

outil outil	utilisateur de base	installateur ou intégrateur
DPN	KPN & C & pipe & thread POSIX	
CASS	C & compilation C	VHDL & sémantique CASS des automates
MUTEX	transparent	C & thread POSIX
UGH	C & micro-architecture	VHDL & Synopsys & outils de CAD de bas niveau
DSG	KPN & board	

TAB. 3.2 – Degrés de compétence des outils

concepteur de prendre des gardes qui conduisent à des solutions surdimensionnées.

Enfin, cette exploration de l'espace de conception basée sur des résultats solides de simulation est due à la rapidité de CASS qui est de l'ordre d'une magnitude par rapport à l'exécution de l'application directement sur une machine standard.

Interface

- Les estimateurs : Dans la suite logicielle Disydent, il n'y a qu'un seul estimateur. Il se trouve dans les profondeurs de l'outil de synthèse UGH et il estime lors de la construction du chemin de données sur quel opérateur il faut placer une opération du source C pour minimiser la taille globale du chemin de données. Son espérance de déviation et sa confiance sont inconnues, mais cette dernière ne doit pas être très élevée. Ce qui devait arriver est arrivé en 2008 lors de la réalisation d'un circuit de reconnaissance de visage [FM+07]. L'estimateur a trouvé judicieux de placer l'incrémenteur sur un additionneur un peu chargé ce qui ajoute un cycle d'attente à la boucle. L'utilisateur a corrigé cette déviance en spécifiant dans le DDP le chemin que l'outil doit utiliser pour ce compteur.
- Le degré de compétence : Le degré de compétence pour utiliser chacun des outils est présenté sur la table 3.2. Elle montre que pour l'utilisation de l'outil il se limite au domaine cible et pour l'initialisation (configurer l'outil à une plate-forme) à des fondamentaux généraux ou du domaine. Ainsi un utilisateur aura un retour sur investissement rapide.
- La complexité : La complexité de Disydent se réduit à 2 descriptions pour chaque processus du KPN, une est utilisée quand le processus tourne en logiciel, l'autre est utilisée par l'outil de synthèse UGH. Bien que la description UGH-C puisse être aussi utilisée quand le processus tourne logiciellement, en pratique cette utilisation est significativement moins bonne que la description initiale. En effet, les algorithmes pour le matériel sont très différents de ceux pour le logiciel. D'une part, ils mettent en valeur du parallélisme à grain fin qui complique le code et d'autre part ils effectuent les calculs au niveau bit (exemple : compteur de 4 bits) qui sont faciles à réaliser en matériel mais qui sont coûteux en logiciel car ils nécessitent l'emploi de masques.
  Le concepteur de systèmes intégrés aura donc à maintenir ces 2 vues dès qu'il passera un processus en matériel.
  Comme DSG génère automatiquement les programmes principaux à partir du KPN et de la projection logiciel/matériel pour les différents niveaux de simulation ainsi que les commandes de génération des simulateurs correspondants, le concepteur

est totalement déchargé de ces tâches.

Notre réalisation de Disydent, suite logicielle complète et réellement opérationnelle, prouve que le fait de fixer tôt le système (plate-forme générique, schéma de communication et noyau) permet d'automatiser de nombreuses étapes de la conception des systèmes intégrés et de masquer complètement de nombreux domaines sous-jacents à l'utilisateur final.

Au laboratoire ASIM/LIP6, l'expertise acquise par Disydent est prolongée actuellement autour de deux axes principaux. Le premier est la réalisation d'un outil de conception de systèmes intégrés basé sur SocLib pour les composants, sur SystemC pour la simulation CABA et sur un générateur de plate-forme plus général. Le second est de migrer la cible des systèmes intégrés du silicium vers les FPGA.

Chapitre 4

Compression sans perte adaptée au réseau

4.1 Introduction

4.1.1 Cadre

Ce projet concerne la compression sans perte adaptée aux réseaux. Il s'agit de compresser un flux de données à la volée avant son émission et de le décompresser à la volée à la réception afin d'augmenter le débit de la ligne de transmission. Le type des données est inconnu, par type on entend du texte, du son, des exécutables, etc, et ce type n'est pas constant dans le flux.
Les problèmes de cette compression sont qu'il ne faut pas introduire de latence d'une part et d'autre part qu'il ne faut pas aboutir à une diminution du débit quand les données sont non compressibles.

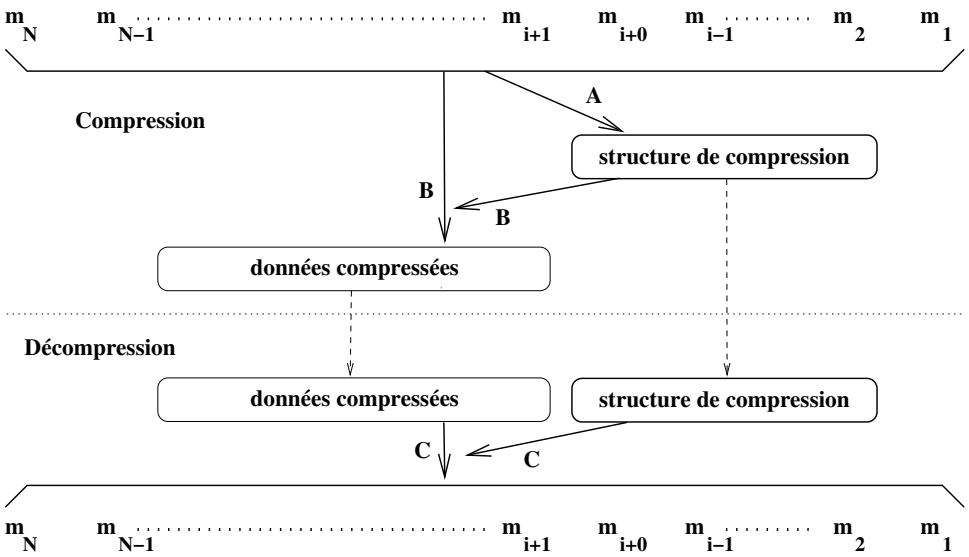
4.1.2 Contribution

La compression de données adaptée aux réseaux est une étude que j'ai commencée en 1993 au LEP (Laboratoire d'Electronique de Philips) et que j'ai continuée à ASIM/LIP6 sous la forme d'un contrat de recherche avec le LEP. Le commanditaire de cette étude était TRT (Télécommunications Radioélectriques Téléphoniques) pour ses commutateurs X25. Cette étude a abouti à un algorithme, un prototype matériel réalisé avec l'outil de synthèse de haut niveau ALMA (voir chapitre 2), et a donné lieu à un stage de DEA. Ce travail a été reconnu internationalement par une publication [AB95] et un brevet [AA96].
Enfin, sur ce sujet, en 1998, j'ai passé plusieurs demi-journées à conseiller M. Karar du LPNHE qui travaillait sur la conception du détecteur de l'accélérateur de particules SOLEIL. En effet, les capteurs de la chambre de collisions fournissent tellement de données qu'il fallait les compresser.

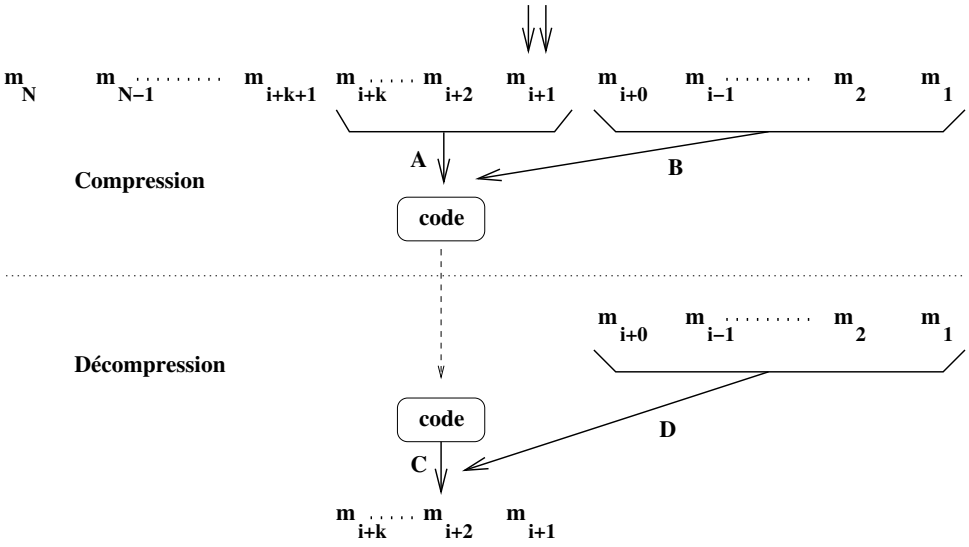
4.2 Description

4.2.1 Compression sans perte

Soit $\{s_M, s_{M-1}, \ldots, s_2, s_1\}$ un alphabet de M symboles et un message "m_N,-$m_{N-1}, \ldots, m_2, m_1$" de N symboles de cet alphabet. En pratique, si la taille du message est trop grande, il sera soit découpé en blocs de N symboles ou une fenêtre de N symboles sera



a) Algorithme global



b) Algorithme à contexte

FIG. 4.1 – Algorithmes de compression sans perte.

déplacée sur le message. Il y a deux grande classes d'algorithmes de compression sans perte.

Les premiers comme Huffman [Huf52] sont présentés sur la figure 4.1.a. Un algorithme de ce type commence par mouliner le message pour construire une structure de compression (arc A), puis il compresse le message en utilisant cette structure (arcs B) et, enfin il envoie la structure de compression et les données compressées. Le décompresseur utilise la structure de compression pour interpréter les données compressées (arcs C) et regénérer le message initial. Dans Huffman, la structure de compression est une table donnant la fréquence de chaque symbole, et les données compressées sont une suite de codes où chaque code correspond à un symbole et la taille (en bit) d'un code est inversement proportionnelle à la fréquence du symbole.
Nous appellerons ce type d'algorithme **algorithme global**.

Les seconds tels gzip, LZW [ZL78] sont présentés sur la figure 4.1.b. Ces algorithmes définissent un ensemble $\{c_{M'}, c_{M'-1}, \ldots, c_2, c_1\}$ de M' codes avec $M' > M$. De plus, les M premiers codes correspondent aus symboles de l'alphabet (c_i correspond à s_i). Cette figure représente une itération de l'agorithme. On se trouve dans la situation représentée par la double flèche verticale où les i premiers caractères ont déjà été compressés par le compresseur et décompressés par le décompresseur.
Le compresseur génère un code regroupant k symboles (arc A) en utilisant le contexte défini par les i premiers symboles du message (arc B). Le décompresseur a lui aussi connaissance de ce contexte (arc D) et peut donc interpréter le code (arc C) pour regénérer les k symboles. L'itération suivante reprend sur le symbole m_{i+k+1}. Notons enfin que k est au moins égal à 1 car pour chaque symbole il existe un code correspondant comme indiqué dans le préambule de ce paragraphe.
Ces algorithmes diffèrent dans la manière de définir les codes regroupant plusieurs symboles. Par exemple le code peut indiquer la répétition de k fois le même symbole ou, une sous-suite du contexte (e.g. : les k symboles précédents le symbole l du contexte courant soit $m_{l+k-1}, \ldots, m_{l+1}, m_l$).
Nous appellerons ce type d'algorithme **algorithme à contexte**.

Entre les deux approches, les algorithmes à contexte sont en général plus efficaces en taux de compression. Ceci est dû au coût de la structure de compression qu'il est difficile de compenser. Les algorithmes globaux l'emportent quand la structure de compression peut être définie statiquement. C'est le cas quand le type des données à compresser est connu, c'est le cas par exemple de texte en anglais ou dans MPEG où les fréquences des coefficients générés par la double transformation de Fourier sont assez stables.

4.2.2 Compression adaptée au réseau

Nous appelons ici compression adaptée au réseau un mécanisme de compression qui se place au niveau 2 du modèle OSI (liaison de données). Une conséquence majeure est qu'à ce niveau, **le type de données est inconnu**. En particulier, ce sera une succession de type de données dépendant des applications tournant dans la couche OSI 7 (application). Ces types allant de données très compressibles comme un texte en français ou un programme VHDL, à des données non compressibles comme un fichier déjà compressé, en passant par des données moyennement compressibles comme un exécutable.

Les algorithmes globaux sont rejetés car ils induisent une latence importante due à l'impossibilité d'utiliser une structure de compression statique. Ceci nécessite d'attendre N

symboles pour calculer la structure de compression avant de pouvoir commencer à émettre.

La latence des algorithmes à contexte se réduit à quelques symboles plus le temps de génération du code à transmettre. Pour ce temps de génération, il existe des algorithmes le rendant acceptable pour des taux de compression convenables et en dernier ressort il peut être accéléré matériellement. La latence n'est donc pas un problème rédhibitoire des algorithmes à contexte.
Supposons maintenant que les données passant sur la ligne soient incompressibles. Sur la figure 4.1.b, k vaut 1 à chaque itération, un code sera donc émis pour chaque m_i. La taille en bit d'un code est $\log_2(M')$ qu'il faut comparer à celle d'un symbole qui est $\log_2(M)$. En posant $M' = \lambda M$ le taux de compression[1] vaut $1 + \frac{\log(\lambda)}{\log(M)}$ soit un taux de compression supérieur à 1. Pour des valeurs usuelles de M et M', on a : $1,38$ pour $M = 256$ et $M' = 4\,M$, $1,5$ pour $M = 256$ et $M' = 8\,M$. Ce taux de compression supérieur à 1 pose un sérieux problème quant à l'utilisation de ce type d'algorithme tel quel dans un réseau.

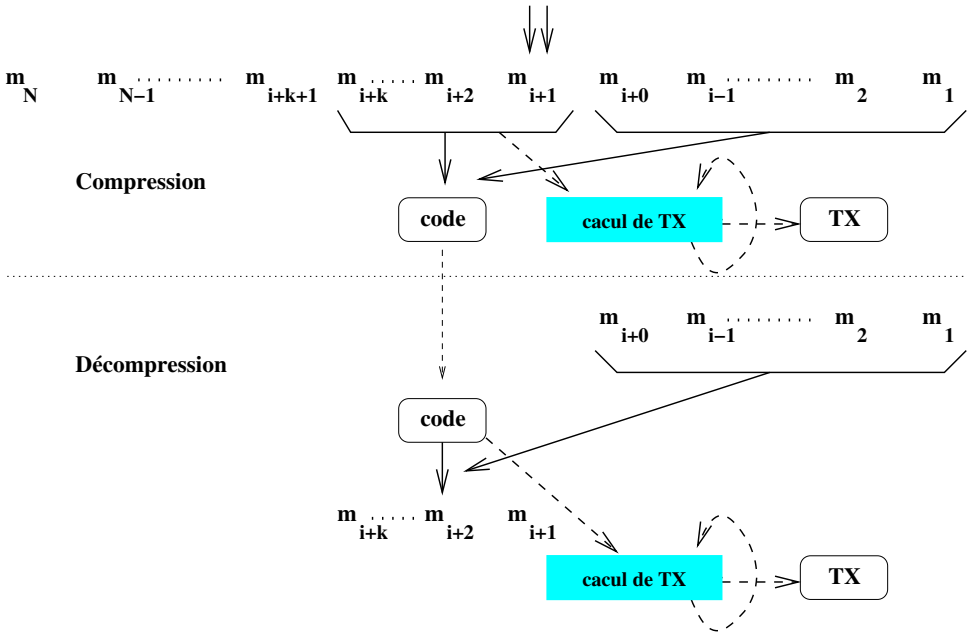
Pour pallier ce problème, il faut que le compresseur se mette en mode transparent dans les zones de données non compressibles et se remette en mode compression en dehors de ces zones. Pour que ceci fonctionne, il faut que le décompresseur soit au courant des changements de mode du compresseur. Le compresseur peut indiquer le passage du mode compresseur au mode transparent en réservant un des M' codes pour cet effet. Par contre, une fois en mode transparent, il n'a plus de moyen d'indiquer qu'il repasse en mode compression.

La solution à ce problème est présentée sur la figure 4.2. En a) le système est en mode compression, le compresseur et le décompresseur calculent en permanence TX, le taux de compression des X derniers symboles émis. Quand ce taux dépasse un certain seuil, les deux passent en mode transparent.
En b) le système est en mode transparent. Le compresseur continue à compresser mais jette les codes calculés à la corbeille et transmet les symboles tels quels et continue à calculer TX. Le décompresseur quant à lui se met à compresser les données et jette aussi les codes calculés à la corbeille. Cette compression lui permet aussi de calculer TX. Le compresseur et le décompresseur se remettent alors simultanément en mode compression dès que TX repasse le seuil.

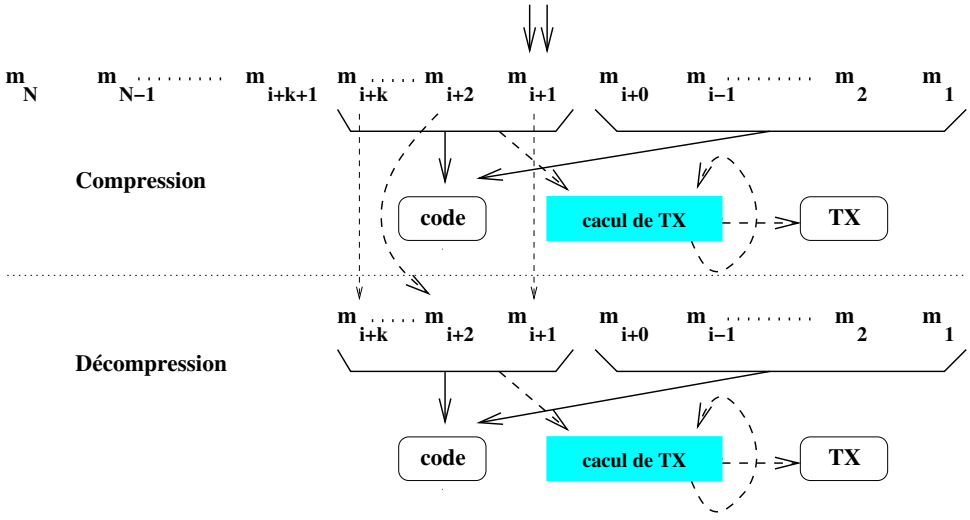
4.3 Conclusion

La compression adaptée au réseau a été testée logiciellement et fonctionne très bien. L'innovation consistant à synchroniser le compresseur et le décompresseur en faisant compresser le décompresseur a été étendue pour faire un système multi-algorithmes permettant de mieux s'adapter aux données en transit. Par exemple, on a simultanément l'algorithme à mode transparent, un algorithme à contexte court, un algorithme à contexte moyen et un algorithme à contexte long, ... Les taux de compression de tous ces algorithmes sont calculés simultanément et celui donnant le meilleur taux est celui utilisé pour la transmission.
Enfin, un modèle matériel générique en λ de cet algorithme avec M valant 256 et $M' = \lambda M$ a été développé avec ALMA (voir section 2.2.5).

[1] défini par $\frac{volume\ final}{volume\ initial}$



a) Mode compression



b) Mode transparent

FIG. 4.2 – Algorithme de compression adapté au réseau.

Chapitre 5

Résumé des publications et travaux

5.1 Livres et revues internationales avec comité de programme

Ces papiers sont donnés respectivement dans les annexes A et B.

[APD08] Ivan Augé and Frédéric Pétrot. *Chapter 10 : User Guided High Level Synthesis.* In *The New Wave of the High-Level Synthesis*, Publisher Springer, pages 172–196, 2008.

[APD05] Ivan Augé, Frédéric Pétrot, François Donnet and Pascal Gomez. *Platform-based design from parallel C specifications.* In *IEEE Transaction on CAD of Integrated Circuits and Systems*, volume 24, numéro 12, pages 1811–1826, December 2005.

5.2 Brevet

[AA96] Ivan Augé and A. Aglan
Brevet international sur un algorithme de compression et de son implantation matériel, February 1996

5.3 Publications et conférences internationales avec comité de programme

[PAD08] Frédéric Pétrot and Ivan Augé. *User Guided High Level Synthesis.* In *The New Wave of the High-Level Synthesis, DATE Workshop* IEEE, Munich, Germany, Invited talk, March 2008.

[FM+08] Nicolas Farrugia, Franck Mamalet, Frédéric Pétrot, Ivan Augé and Michel Paindavoine. *A convolutional neural network hardware architecture designed with the UGH tool.* In *The New Wave of the High-Level Synthesis, DATE Workshop* IEEE, Munich, Germany, March 2008.

[AP+04] Ivan Augé, Frédéric Pétrot, François Donnet, Étienne Faure et Pascal Gomez *Disydent : un environnement pour la conception de systèmes numériques synchrones.* In *Premier congrès international de signaux, circuits et systèmes*, pages 71–77, Monastir, Tunisie, March 2004.

[ADP03] Ivan Augé, François Donnet and Frédéric Pétrot. *Retiming Finite State Machines to Control Hardened Data-Paths.* In *16th Symposium on Integrated Circuits and Systems Design*, page 41–47, Sao Paulo, Brazil, September 2003.

[APH01] Ivan Augé, Frédéric Pétrot and Denis Hommais. *A pragmatic approach to the design of embedded systems.* In *Design Automation and Test in Europe*, pages 170–174, Munich, Germany, March 2001. IEEE.

[HPA01a] Denis Hommais, Frédéric Pétrot and Ivan Augé. *A practical toolbox for system level communication synthesis.* In *9th International Symposium on Hardware/Software Co-design*, pages 48–53, Copenhagen, Dennemark, April 2001.

[HPA01b] Denis Hommais, Frédéric Pétrot and Ivan Augé. *A toolbox to map system level communications on hw/sw architectures.* In *12th International Workshop on Rapid System Prototyping*, pages 77–82, June 2001.

[ABGP99] Ivan Augé, Jean-Yves Brunel, Alain Greiner and Frédéric Pétrot. *Fine grain scheduling.* In *IEEE 25th Euromicro*, Vesteras, Sweden, September 1999.

[ABG+97] Ivan Augé, Rajesh K. Bawa, Pierre Guerrier, Alain Greiner, Ludovic Jacomme and Frédéric Pétrot. *User guided high level synthesis.* In Ricardo Reis and Luc Claensen, editors, *VLSI : Integrated Systems on Silicon*, pages 464–475, Gramado, Brazil, August 1997. IFIP, Chapman & Hall.

[AB95] Ivan Augé and Jean-Yves Brunel. *Lossless Data Compression for Embedded Applications in Networking*, In *IEEE 21th Euromicro*, Como, Italy, August 1995.

[BAH+94] Jean-Yves Brunel, Ivan Augé, Marc Hervieu, Pascal Bourquin and Patrick Renaud. *Quantitative Design of a Scalable MicroSystem Using ALMA.* In *IEEE* $5^n d$ *International Workshop on Rapid System Prototyping*, Grenoble, June 1994.

[ABG+91] Ivan Augé, Jean-Yves Brunel, Jean-Michel Callemyn, Philippe Tychon and Michel Israel. *The design of circuits with no timing references using the ALMA environment* In *IEEE* 5^{th} *Annual European Computer Conference*, pages 596–598, Bologne, Italy,1991

[BAR+91] Jean-Yves Brunel, Ivan Augé, Xavier Redon and Philippe Tychon. *System prototyping with ALMA.* In *IEEE* $2^n d$ *International Workshop on Rapid System Prototyping*, Research Triangle Park USA, June 1991.

[BGT+90] Jean-Yves Brunel, Jean-Michel Callemyn, Philippe Tychon, Ivan Augé and Michel Israel. *ALMA : A design methodology for Microsystem Architecture.* In 16^{th} *European Solid-State Circuits Conference*, pages 65–68, Grenoble, September 1990

[ABI+89] Ivan Augé, Jean-Yves Brunel, Michel Israel and Philippe Tychon. *ALMA : Advanced Library for Micro Architecure.* In *Congres IASTED 89*, pages 155–157, Grindelwald, Switzerland, February 1989,

[IA89] Michel Israel and Ivan Augé. *New trends in ASICs CAD tools.* In *The Fourth International Symposium on Computer and Informations*, pages 691–700, Cesçme, Turquie, November 1989.

5.4 Thèses

[Gom06] Pascal Gomez *Outils pour le déploiement d'applications avec des contraintes de temps réel sur des multiprocesseurs sur puces. Thèse de l'Université Pierre et Marie Curie* soutenue en décembre 2006.
Thèse résultant du noyau temps réel du projet Disydent.

[Don04] François Donnet. *Synthèse de co-processeurs pour systèmes intégrés. Thèse de l'Université Pierre et Marie Curie* soutenue en Janvier 2004.
Thèse que j'ai dirigée personnellement.

[Hom01] Denis Hommais. *Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel. Thèse de l'Université Pierre et Marie Curie* soutenue en Septembre 2001.
Thèse résultant des schémas de communication et du simulateur CABA CASS du projet Disydent. Participation à l'encadrement de cette thèse dont la direction était assurée par Fréderic Pétrot.

[Bru96] Jean-Yves Brunel. *Synthèse d'architecture de circuits intégrés : Étude de l'ordonnancement au niveau transfert de registres pour prendre en compte les contraintes temporelles imposées par les bibliothèques d'opérateurs disponibles. Thèse de l'Université Pierre et Marie Curie* soutenue en décembre 1996.
Thèse exposant les travaux que Jean-Yves Brunel et moi même avons effectués de 1990 à 1994 au Laboratoire d'électronique de Philips.

[Aug90] Ivan Augé. *ALMA : Approche d'un langage pour la Micro-Architecture. Thèse du CNAM* soutenue en Janvier 1990.

5.5 Rapports, Manuels, Logiciels distribués

[AL06web] Ivan Augé et Vincent Leligeour. *Site de distribution de YaKa* , `http://yaka.ensiie.fr/`, 150 pages, *Un outil de gestion de parc informatique et une distribution Linux* .

[AP06web] Ivan Augé et Olivier Pons. *Site de distribution de FoKon* , `http://yaka.ensiie.fr/fokon`, 25 pages, *Un outil d'automatisation d'examens informatiques.*

[AP06web] Ivan Augé. *Site de distribution des A tools*, `http://yaka.ensiie.fr/ta`, 60 pages, *Une bibliothèque C/C++ d'intéret général, un système de documentation de code, un préprocésseur C++, un générateur de makefile, un debugeur mémoire.*

[AP02web] Ivan Augé et Frédéric Pétrot. *Site de distribution de Disydent et Ugh*, `http://www-asim.lip6.fr/recherche/disydent` 100 pages de documentation sur DPN, CASS, UGH, DSG, 25 pages de tutorial sur Disydent.

[AP00a] Ivan Augé et Frédéric Pétrot. Document COSY D6.4.1, 87 pages.

[AP00b] Ivan Augé et Frédéric Pétrot. Document COSY D5.4.1, 14 pages.

5.6 Encadrement de stages

Au cours de ces presque 20 années de recherche, j'ai encadré de nombreux étudiants qui ont participé activement à mes travaux.
Le plus grand contingent (14 étudiants) fut constitué par des mémoires d'ingénieur de l'ENSIIE de $3^{ième}$ d'une durée de 6 mois. Le second contingent (7 étudiants) fut constitué par des stages de M2 ASIME de durée également de 6 mois. Sur ces sujets, j'ai aussi encadré 8 stages d'entreprise d'étudiants de l'ENSIIE de $1^{ère}$ ou de $2^{ième}$ année d'une durée de 3 mois.
Ceci fait donc que j'ai encadré en moyenne un 1,5 stages par an, avec quelques mauvaises années à 0 et quelques bonnes années à trois ou quatre.

5.7 Autres publications et conférences

[AP+04b] Ivan Augé, Frédéric Pétrot, François Donnet, Étienne Faure et Pascal Gomez *Intégration sur plate-forme matérielle/logicielle de spécifications 'C' parallèles.* Dans *Annales des télécommunications*, volume 59, pages 807–837, Paris, France, 2004.

[Don02] François Donnet. *Représentation des opérateurs du langage C pour la synthèse haut-niveau.* Dans *Actes du troisième colloque CAO de circuits et systèmes*, pages 3–6, Paris, France, Mai 2002.

[HP+A00] Denis Hommais, Frédéric Pétrot et Ivan Augé. *Une approche pour la conception des systèmes intégrés.* Dans *IIIèmes Journées Nationales du Réseau Doctoral de Micro-électronique*, Montpellier, France, Mai 2000, pages 76–77.

[Aug95] Ivan Augé *High Level Synthesis.* Conférence à *l'université de Porto-Alegre*, Août 1997.

[Aug95] Ivan Augé. *Ordonnancement fin sur "netlist" physique.* Conférences à *Journée GT CAO Architecture du GDR ANM*, Ecole Polytechnique, Mars 1995.

[Aug94] Ivan Augé. *Introduction à la synthèse de haut niveau.* Université d'été MASI/ENST, durée 6 heures, septembre 1994.

[IA89] M. Israel, I. Augé. *Un tour d'horizon des outils de CAO pour ASICs* Journées professionnelles SMtique, CONNectique ATELIERS ASIC, proceedings pages 250-254, Paris, Avril 1989.

5.8 Bibliographie générale

[Huf52] D. A. Huffman. *A method for the construction of minimum redundancy codes.* In *Proceedings of the IRE*, pages 1098–1101, 1952.

[Amd67] G.M. Amdahl. *Validity of the single-processor approach to achieving large scale computing capabilities.* In *AFIPS Conference Proceedings*, volume 30, pages 483–485, Reston, Va., April 1967. AFIPS Press.

[Kah74] Gilles Kahn. *The semantics of a simple language for parallel programming.* In *Information Processing 74*, pages 471–475, Stockolm, Sweden, August 1974.

[ZL78] Jacob Ziv and Abraham Lempel *Compression of Individual Sequences via Variable-Rate Coding.* In *IEEE Transactions on Information Theory*, Vol. 24, No. 5. (1978), pages 530–536.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes. Prentice Hall*, 1985.

[PK89] Pierre G. Paulin and John P. Knight. *Algorithms for high-level synthesis.* In *IEEE Design and Test*, pages 18–31, décembre 1989.

[NCG91] S. Note, F. Catthoor, and G. Goosens. *Cathedral-iii architecture-driven high-level synthesis for high throughput dsp applications.* In *28th ACM/IEEE Design Automation Conference - DAC 91*, 1991.

[Jen91] G. Jennings. *A case against event driven simulation for digital system design.* In *24th Annual Simulation Symposium*, pages 170–175, April 1991.

[BKa93] J. Biesenack, M. Koster and others *The Siemens High-Level Synthesis System CALLAS.* In *IEEE Transactions on Very Large Scale Integration Systems*, volume 1, pages 244–275, 1993.

[KDJ94] Polen Kission, Hong Ding and Ahmed A. Jerraya. *Structured design methodology for high-level design.* In *31st ACM/IEEE Design Automation Conference*, 1994.

[Mic94] Giovanni De Micheli. *Synthesis and Optimisation of Digital Circuit*, McGraw-Hill, chapter 1, page 23, 1994.

[LKMM95] T. Ly, D. Knapp, R. Miller and D. Mac Millen. *Scheduling using behavioral templates.* In *32th ACM/IEEE Design Automation Conference - DAC'95*, juin 1995.

[SDPC95] O. Sentieys, J.Ph. Diguet, J.-L. Philippe, and D. Chillet. *Gaut : A high level synthesis tool, dedicated to real time signal processing application.* In *EURO-DAC 95*, Septembre 1995.

[Par95] Thomas M. Parks. *Bounded Scheduling of Process Networks.* PhD thesis, Electronics Research Laboratory, Berkeley, 1995.

[PI96] *Draft Standard OMI 324 : PI-Bus.* Open Microprocessor Initiative, Tech. Rep., Dec. 1996, rev. 0.3d.

[BKV+96] E. Berrebi, P. Kission, S. Vernalde, S. De Troch, J.C. Herluison, J. Fréhel, A.A. Jerraya and I. Bolsens. *Combined control flow dominated and data flow dominated high-level synthesis.* In *33rd Design Automation Conference*, 1996.

[Cam96] Raul Camposano. *Behavioral synthesis.* In *33rd Design Automation Conference*, 1996.

[HLS92] Daniel D. Gajski, Nikil Dutt, Allen Wu and Steve Lin. *High Level Synthesis, Introduction to Chip and System Design.* In *Kuwer Academic Publisher*, 1992.

[MM+97] Sharad Malik, Margaret Martonosi and Yau-Tsun Steven Li *Static Timing Analysis of Embedded Software.* In *ACM Design Automation Conference*, pages 147–152, June, 1997.

[KM98] Peter Voigt Knudsen and Jan Madsen. *Communication estimation for hardware/software codesign.* In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign (Codes/CASHE)*, 1998.

[Sug00] Zoltán Sugár. *Synthèse comportementale basée sur l'ordonnancement.* Thèse de l'Institut National Polytechnique de Grenoble, mai 2000.

[JCM01] Christophe Jego, Emmanuel Casseau and Eric Martin. *Architectural synthesis of digital signal processing applications dedicated to submicron technologies.* In *IEEE International Conference on Electronics Circuits and Systems, ICECS 01*, pages "533–536", Septembre 2001.

[Wol01] Wayne Wolf. *Codesign, looking beyond and forward.* In *CODES 01*, Copenhagen, Dennemark, April 2001.

[DTE02] H. Dawid, S. Thiel, H. Elders-Boll, M. Vaupel, E. Geesmann, M. Vellachi and M. Antweiler *Efficient Design Flow from System Level to Hardware in CoCentric System Studio.* In *Design, Automation and Test in Europe*, pages 3-7, mars 2002

[CCB05] G. Corre, P. Coussy, P. Bomel, E. Senn and E. Martin *Synthèse Comportementale Sous Contraintes de Communication et de Placement Mémoire pour les composants du TDSI* In *HAL - CCSd - CNRS*, 2005

[GMC06] Y. Guo, D. McCain, J. R. Cavallaro, and A. Takach *Rapid Industrial Prototyping and SoC Design of 3G/4G Wireless Systems Using an HLS Methodology* In *EURASIP Journal on Embedded Systems*, volume 2006, pages 1–25, 2006

[FM+07] Nicolas Farrugia, Franck Mamalet, Sébastien Roux, Fan Yang and Michel Paindavoine. *A Parallel Face Detection System Implemented on FPGA.* In *International Symposium on Circuits and Systems (ISCAS 2007)*, IEEE, New Orleans, Louisiana, USA, pages 3704–3707, May 2007.

[DQ07] Alain Darte and Clément Quinson *Scheduling Register-Allocated Codes in User-Guided High-Level.* In *18th Int. Conf. on Application-specific Systems, Architectures and Processors*, IEEE, Montreal, Canada, pages 219–224, July 2007.

[Bol08] T. Bollaert *Chapter 3 : Catapult Synthesis : A Practical Introduction to Interactive C Synthesis* In *The New Wave of the High-Level Synthesis*, Publisher Springer, pages 29–52, 2008.

[AK08] S. Aditya and V. Kathail *Chapter 4 : Algorithmic Synthesis Using PICO : An Integrated Framework for Application Engine Synthesis and Verification from High Level C Algorithms.* In *The New Wave of the High-Level Synthesis*, Publisher Springer, pages 53–74, 2008.

[Cyn] Forte's Cynthesizer *Outil de synthèse de haut niveau partant de SystemC.* In *http ://www.forteds.com*, 2008

Chapitre 6

Conclusion

Dans l'introduction de ce document, j'ai exprimé le fait que j'ai été amené à travailler dans de nombreux domaines. J'espère avoir convaincu le lecteur au travers de ce document que j'ai acquis un grand degré d'expertise dans la synthèse de haut niveau, dans les domaines liés à la conception des systèmes intégrés sur SoC.

Par ailleurs je tiens à souligner que ces travaux, bien que j'en ai été l'instigateur ou le co-instigateur ont toujours été un travail d'équipe. La taille de ces équipes a varié d'une poignée de personnes, à treize personnes au plus fort du projet Esprit COSY. Ces équipes ont été constituées d'une trentaine de stagiaires de M2 ou équivalents, de postdocs, d'ingénieurs en CDD et de thésards. J'ai assuré personnellement la gestion et la coordination des travaux de ces équipes avec réussite puisqu'ils ont toujours abouti à des logiciels opérationnels d'une part et d'autre part pour les projets objets de contrats, j'ai toujours réussi à respecter les échéances.

D'autre part, ces projets ont donné lieu à plusieurs dizaines de publications internationales et 4 thèses. Parmi ces publications, je suis co-auteur d'une vingtaine dont une dans la revue de référence sur la conception assistée par ordinateur de circuits (2006), une dans un livre de synthèse sur la synthèse de haut niveau (2008) et un brevet international. Parmi ces thèses, une ([Bru96]) expose en partie des travaux que j'ai personnellement effectués pendant plusieurs années avec son auteur, pour deux autres d'entre elles ([Hom01, Gom06]) j'ai grandement influencé et participé à leurs déroulements car elles sont directement issues du projet Disydent dont j'étais le co-reponsable, enfin pour une ([Don04]), j'ai été le directeur officiel et réel. Cette dernière thèse fut un succès tant au niveau respect de l'échéance de 3 ans, qu'au niveau du résultat qui est encore utilisé aujourd'hui.

Lors de ces travaux, je me suis toujours attaché à réaliser non pas de simples prototypes de démonstration mais des logiciels complètement opérationnels avec documentation. Par exemple le logiciel UGH de synthèse de haut niveau a été validé en vérifiant les circuits obtenus par des simulations électro-temporelles au niveau porte et plus récemment en les faisant tourner sur FPGA.

Cette exigence de qualité m'a permis d'avancer sur des bases solides et de proposer à la communauté sous licence GNU 4 logiciels. Elle a aussi permis une utilisation de ces logiciels en interne à ASIM/LIP6 mais aussi à l'extérieur. UGH est utilisé actuellement à l'ENS Lyon et dans un laboratoire de recherche de France-Telecom à Grenoble. Cette utilisation permet de les améliorer.

En conclusion, les questions qui me préoccupent actuellement sont la synthèse de haut niveau sur cible FPGA que j'ai déjà commencée à débroussailler en 2007, l'adaptation de la conception de systèmes intégrés (Disydent) à SocLib et SystèmeC qui est en cours depuis plusieurs années. J'espère que l'avenir me donnera la possibilité et les moyens de concrétiser ces perspectives.

Annexe A

Article sur la synthèse de haut niveau

Annexe B

Article sur la conception de systèmes intégrés