Contribution à la gestion dynamique de ressource reconfigurable intégrée au sein d'un MPSoC

Daniel Chillet

ANNÉE 2010

HDR/ UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Européenne de Bretagne

Mention : Traitement du Signal

présentée par

Daniel Chillet

préparée dans l'équipe CAIRN

de l'unité de recherche IRISA/ENSSAT

soutenue à l'ENS Cachan Antenne de Rennes

le 8 juin 2010.

---

devant le jury composé de :

Yvon TRINQUET
Professeur des Universités, Univ. de Nantes, président ;

Michel AUGUIN
Directeur de Recherche CNRS, LEAT, Nice, rapporteur ;

Jean-Luc DEKEYSER
Professeur des Universités, Univ. de Lille , rapporteur ;

Jean Didier LEGAT
Professeur, Université Catholique de Louvain, rapporteur ;

Jean-Luc PHILIPPE
Professeur des Universités, Univ. de Bretagne Sud, examinateur ;

Patrice QUINTON
Professeur des Universités, Univ. de Rennes 1, examinateur ;

Olivier SENTIEYS
Professeur des Universités, Univ. de Rennes 1, examinateur.

Contribution à la gestion dynamique de ressource reconfigurable intégrée au sein d'un MPSoC.

Remerciements

J'adresse mes plus respectueux remerciements à Messieurs Jean Didier LEGAT, Professeur à l'Université Catholique de Louvain La Neuve (Belgique), Michel AUGUIN, Directeur de Recherche CNRS au LEAT - Sophia Antipolis et Jean Luc DEKEYSER, Professeur des Universités de l'Université de Lille 1, pour avoir accepté d'être rapporteurs de mon Habilitation à Diriger des Recherches.

Je remercie très chaleureusement Monsieur Yvon TRINQUET, Professeur des Universités de l'Université de Nantes d'avoir accepté de présider le jury de mon HDR.

Je tiens à exprimer mes sincères remerciements à Messieurs Jean Luc PHILIPPE, Professeur des Universités de l'Université de Bretagne Sud, et Patrice QUINTON, Professeur des Universités de l'Université de Rennes 1 de m'avoir fait le plaisir de participer à mon jury d'HDR.

J'adresse des remerciements particuliers à Olivier SENTIEYS, responsable de l'équipe CAIRN, qui a suivi avec intérêt l'ensemble des travaux que j'ai pu mener depuis plus d'une dizaine d'années et qui a su à la fois soutenir, favoriser et critiquer ces travaux pour qu'ils mènent à cette soutenance. Qu'il en soit remercié très sincèrement.

Bien évidemment, ces travaux sont le fruit de discussions et de réflexions où la part du travail d'équipe est importante, aussi je voudrais adresser un remerciement collectif à l'ensemble des membres de l'équipe CAIRN avec qui ces discussions ont eu lieu, autour d'un café ou lors de diverses réunions formelles ou informelles. Un remerciement plus particulier à Sébastien PILLEMENT avec qui je partage une partie des thèmes développés dans ce manuscrit, ainsi qu'à Olivier BERDER, Emmanuel CASSEAU, Didier DEMIGNY, Hélène DUBOIS, Michel GUITTON, Daniel MENARD, Romuald ROCHER, Pascal SCALART et Arnaud TISSERAND pour leurs contributions diverses, amicales et/ou professionnelles, m'ayant permis de soutenir cette HDR.

Que ce soit aussi remercié l'ensemble des collègues de l'ENSSAT, ainsi que les personnels administratif et technique qui, par leur travail quotidien, facilitent nos travaux et dont on oublie souvent la part importante dans la vie de l'école.

Je remercie également Delphine VERBYST pour sa relecture attentive, et qui m'a permis d'améliorer le document dans son ensemble.

Finalement, je n'oublie pas ceux qui ont accepté de me donner du temps pour rédiger et préparer cette HDR. Un grand merci à Cécile ainsi qu'à Alice et Karl qui ont participé, à leur manière, à l'aboutissement de ce document.

Avant-propos

Doctorant au LASTI[1] entre 1994 et 1996, puis Maître de conférences à l'ENSSAT[2] (Université de Rennes 1), j'ai intégré l'équipe Signal et Architecture du laboratoire LASTI en tant que permanent en septembre 1997. Mon activité de recherche était alors liée à l'axe de recherche principal de l'équipe : la définition de méthodologies de conception pour des circuits et systèmes intégrés. Ces développements méthodologiques ont alors intégré l'environnement GAUT[3] constitué d'une chaîne d'outils logiciels permettant d'aborder, à un haut niveau d'abstraction (à partir d'un langage tel que VHDL[4] ou C), la conception de la partie opérative d'une application de traitement du signal. L'outil ciblait, à ce moment là, des architectures de type ASIC[5] pour lesquelles l'optimisation principale concernait la surface du circuit.

Ma contribution durant mes travaux de thèse a consisté à proposer une méthodologie de conception de la partie mémorisation d'un circuit spécialisé. La méthodologie développée permet de définir l'unité de mémorisation sous contrainte d'une séquence de transferts de données [A1, A15]. En pratique, une phase de synthèse a été ajoutée dans le flot de conception global d'un circuit ASIC. Nous avons développé l'outil logiciel GAUM[6], pour compléter la chaîne de conception de GAUT et lui permettre de couvrir la conception de la partie mémoire pour des circuits spécifiques.

J'ai ensuite poursuivi ces travaux en intégrant la contrainte de consommation d'énergie dans notre méthodologie. La prise en compte de cette nouvelle dimension nous a conduits à remettre en cause l'ordre dans lequel les différentes phases de conception étaient abordées. En effet, jusqu'alors, la principale difficulté de conception était liée à la définition d'une architecture opérant les calculs de manière efficace. Or, l'évolution des applications (accroissement très important des volumes de données manipulées) et l'évolution technologique (augmentation de la densité d'intégration des transistors) posent la question de l'ordre classique des contraintes de synthèse. Alors que la surface du circuit final était un élément primordial, elle est devenue un facteur beaucoup moins critique pour les nouvelles technologies. À l'inverse, la consommation des circuits est devenue le "point chaud" de la conception d'un système et plus particulièrement de la partie mémoire. Nos travaux ont alors porté sur la conception des différentes unités d'un système, et donc de l'unité de mémorisation, sous contrainte de minimisation de la consommation d'énergie. Nous avons défini une méthodologie permettant de mettre en place une hiérarchie mémoire interne au système et nous avons montré que cette solution améliore considérablement la consommation énergétique [A37] par rapport à une solution mémoire sans hiérarchie.

L'évolution des circuits vers les systèmes sur puce (SoC[7]) a eu un impact important sur les méthodologies de conception. L'élargissement de l'espace de conception, et donc de celui des solutions potentielles, a considérablement augmenté au point qu'il est actuellement inconcevable d'envisager une exploration exhaustive des solutions. De plus, la notion de conception s'est, elle aussi, élargie pour englober maintenant la couche "middleware" des systèmes à concevoir. Enfin, notons le recours accru à la reconfiguration au sein de ces systèmes afin de fournir un

[1] LASTI : Laboratoire d'Analyse des Systèmes de Traitement de l'Information
[2] ENSSAT : École Nationale Supérieure de Sciences Appliquées et de Technologie
[3] GAUT : Génération Automatique des Unités de Traitement
[4] VHDL : Vhsic Hardware Description Language
[5] ASIC : Applied Specific Integrated Circuit
[6] GAUM : Génération Automatique d'Unité de Mémorisation
[7] SoC : System-on-Chip

support d'évolution pour les applications dont les spécifications doivent suivre les besoins des utilisateurs toujours plus avides de nouveautés. Dans ce contexte, mes travaux s'inscrivent dans plusieurs axes de recherche. Le premier concerne l'étude des zones reconfigurables au sein d'un système sur puce, et en particulier leur gestion au travers de services spécifiques fournis par le système d'exploitation. Le second concerne de façon plus spécifique l'ordonnanceur d'un système sur puce confronté au contrôle de l'exécution de nombreuses tâches sur de multiples cibles d'exécution de natures différentes. Finalement l'axe de recherche concernant l'organisation mémoire au sens large pour les SoC a été poursuivi tant il est vrai que ce problème reste entier dans ces systèmes.

Ces axes de recherche s'intègrent pleinement dans les thèmes de recherche de l'équipe CAIRN[8] (ex R2D2[9]) dont les activités d'articulent autour de la conception de SoC et plus particulièrement de SoC reconfigurable (RSoC[10]).

Au sein de l'équipe CAIRN, je mène des activités sur plusieurs axes de recherche. Ce document décrit mes activités de recherche depuis l'obtention de mon doctorat. Après avoir situé le contexte de mes travaux et balayé l'ensemble de mes implications dans la première partie de ce document, je détaille dans la seconde partie trois axes de recherche qui sont actuellement en développement au sein de l'équipe et dans lesquels je suis fortement impliqué.

[8] CAIRN : energy efficient Computing ArchItectures with embedded RecoNfigurable resources
[9] R2D2 : Reconfigurable and Retargetable Digital Design
[10] RSoC : Reconfigurable System-on-Chip

Table des matières

Première partie

Positionnement des travaux et CV étendu

Introduction

Contexte des travaux

Dans le contexte de la conception des circuits intégrés, l'accroissement très important de la densité d'intégration a placé, depuis quelques années, les concepteurs en position d'intégrer des systèmes complets sur une seule puce de silicium. De l'ère de la conception de systèmes SSI[11], nous sommes actuellement passés dans l'ère de la conception de MPSoC[12]. L'évolution vers ce type de systèmes a remis en cause les méthodes ainsi que les outils de conception. En effet, la complexité des systèmes à concevoir (du niveau applicatif jusqu'au niveau circuit) ne permet plus la maîtrise complète par une seule équipe de concepteurs, mais nécessite la collaboration de nombreuses compétences couvrant les différents aspects mis en œuvre. De nos jours, la conception de SoC s'apparente à un "jeu d'assemblage" de blocs pré-conçus dont il faut s'assurer de la cohérence et du bon interfaçage avec le reste du système. Ainsi, une multitude de blocs est envisagée afin de couvrir l'ensemble des besoins fonctionnels de l'application : cœurs de processeurs généralistes, cœurs de processeurs de traitement du signal, blocs d'interfaces, ...Cela passe aussi par le développement de blocs de traitements spécifiques pour des besoins applicatifs très particuliers ou pour répondre à des contraintes très spécifiques. Enfin, des blocs de logiques reconfigurables sont de plus en plus fréquemment mis à la disposition du concepteur afin qu'il puisse créer des chemins de données spécifiques pour enchaîner des traitements plus ou moins occasionnels.
Face à la multitude de blocs fonctionnels disponibles, le concepteur doit disposer des moyens d'explorer le vaste espace de solutions qui s'offrent à lui. Les méthodes de conception doivent donc proposer des solutions permettant d'aborder rationnellement l'ensemble des parties d'un tel système, des aspects logiciels aux aspects matériels, en passant par le système d'exploitation.

Les axes de recherche de l'équipe CAIRN couvrent ces différents aspects grâce à ses membres dont les compétences sont très complémentaires. Sur la figure 1, illustrant ces différentes compétences, mes interventions et implications, tant au niveau enseignement qu'au niveau recherche, concernent les compétences "Architecture des circuits intégrés" et dans une moindre mesure les aspects "Informatique".

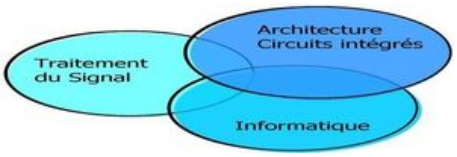


FIG. *1 : Compétences des membres de CAIRN*

Concernant les aspects "Architecture", je me suis impliqué dans plusieurs travaux de recherche couvrant la conception de systèmes à logique multi-valente, les réseaux de capteurs, la conception de hiérarchies mémoire spécifiques, la reconfiguration appliquée à la hiérarchie mémoire, la gestion de la zone reconfigurable au sein d'un RSoC et enfin la gestion des communications au sein de cette même zone.
Concernant les aspects "Informatique", mes contributions adressent certains aspects du système d'exploitation concernant notamment l'ordonnancement de tâches sur cibles d'exécution hétérogènes par réseaux de neurones ainsi que sur la définition de services spécifiques pour la gestion de zones reconfigurables au sein des SoC.

[11] SSI : Small Scale Integration
[12] MPSoC : Multi-Processor System on Chip

Dans la section suivante, je présente brièvement les différentes activités de recherche que j'ai abordées depuis l'obtention de mon doctorat. Le chapitre suivant présente, sous forme d'un Curriculum Vitæ, mes activités d'encadrements de thèses, de stages de DEAs/Masters, ainsi que mes implications dans des projets de recherche. Puis, la seconde partie de ce document expose trois thèmes particuliers qui sont actuellement étudiés au sein de l'équipe et qui sont inclus dans des projets de recherche nationaux (OverSoC[13] et FosFor[14]). Finalement, la dernière partie conclut ce document par quelques réflexions sur mes activités d'enseignement et de recherche.

Résumé des travaux

Depuis l'obtention de mon doctorat en janvier 1997, j'ai participé à différents travaux de recherche soit au travers de l'encadrement de thèses et/ou de stages de master, soit en développant mes propres activités au sein de l'équipe. Dans les paragraphes suivants, je présente tout d'abord rapidement les travaux qui sont encore en cours au sein de l'équipe. Je présente ensuite les principales activités sur lesquelles je suis intervenu. J'indique pour chaque thème abordé, les encadrements de thèses et/ou de masters assurés. De nombreux stages, d'ingénieurs et/ou de techniciens, sont venus compléter ces travaux, ils ne sont pas tous mentionnés dans ce document.

Hiérarchie mémoire au sein des SoC

L'évolution récente des SoC montre que la mémoire au sein de ces systèmes prend une place de plus en plus importante. En effet, les systèmes qui sont développés couvrent le spectre des applications de communication au sens large pour lesquelles, même si les traitements sont complexes, le point critique relève du stockage des données. Deux principales contraintes relèvent directement de cette fonctionnalité : les performances temporelles et la consommation énergétique.
Concernant les performances temporelles, l'augmentation du volume de données à stocker et les contraintes de traitement tendent naturellement à proposer des solutions de stockage hiérarchique. Concernant les performances énergétiques, la faible activité des mémoires (activité de la cellule mémoire ramenée à l'activité globale de la mémoire) associée aux courants de fuite de plus en plus importants pour les technologies sub-microniques engendre une consommation statique importante (au regard de la consommation dynamique).
Nous avons montré que la mise en place d'une organisation hiérarchique permet de répondre efficacement au problème de la consommation énergétique. De plus, compte tenu du déterminisme important des applications ciblées, il est possible de maîtriser les performances temporelles de cette hiérarchie en anticipant les besoins des blocs de traitement [A32]. Enfin, l'application du concept de reconfiguration dynamique aux mémoires laisse entrevoir un potentiel de gain important sur la consommation énergétique [A8].

Sur ce thème de recherche, je co-encadre les travaux des personnes suivantes :
- Lahcène ABDELOUEL, enseignant-chercheur Algérien en poste à l'ENP[15] d'Alger. Le sujet des travaux aborde la conception de hiérarchie mémoire pour un SoC. Il s'agit principalement de définir la structure hiérarchique en fonction des contraintes de l'application, et de définir le stockage des données au sein de cette hiérarchie.
- Erwan GRACE, étudiant en dernière année de thèse dont les travaux sont menés dans le cadre d'une collaboration avec le CEA[16] LIST[17]. Le sujet aborde la définition d'une structure reconfigurable de mémorisation sous contrainte de performance et de consommation. Il s'agit de définir la structure, le stockage des données et de définir le contrôle de l'ensemble afin d'assurer l'ensemble des requêtes de lectures et écritures des blocs de traitement.

J'ai aussi encadré les travaux d'Erwan GRACE durant son stage de master STI[18] au sein de l'équipe. Son travail

[13] OverSoC : OS validation and exploration methodology for reconfigurable Systems on Chip
[14] Flexible OS FOr Reconfigurable platform
[15] ENP : École Nationale Polytechnique
[16] CEA : Commissariat à l'Énergie Atomique
[17] LIST : Laboratoire d'Intégration des Systèmes et des Technologies
[18] STI : Signal Télécommunication Image

de stage a notamment permis de faire une étude de l'état de l'art des problématiques de mémorisation au sein des SoC et de positionner les travaux de thèse que nous envisagions alors.
Enfin, je co-encadre les travaux de Amine DIDIOUI depuis février 2010 dont l'objectif consiste à modéliser la consommation des mémoires sous différentes tensions d'alimentation. Il s'agira notamment d'étudier le gain énergétique apporté par une gestion dynamique de celles-ci.

Reconfiguration au sein des SoC et impact sur l'OS

L'évolution très importante des besoins applicatifs ne laisse guère de temps aux concepteurs pour développer des SoC spécifiques à chaque application. De plus, pour amortir les coûts engendrés par la conception, il est de plus en plus important de s'assurer que le système conçu puisse évoluer pour ne pas être très rapidement obsolète. L'une des réponses à ces problématiques est la mise à disposition de logique reconfigurable au sein du système. En offrant à la fois souplesse et performance, ce type de logique peut prendre en charge des enchaînements de traitements intensifs. Toutefois, pour être efficace (en temps et en énergie), cette logique doit pouvoir être "manipulée" comme toutes les autres parties matérielles du système. Il s'agit alors d'élargir les services proposés par le système d'exploitation (OS[19]) aux tâches afin de prendre en compte les spécificités de cette logique reconfigurable.
De plus, hormis les problèmes de configurations, de préemptions et de contrôles de l'exécution, le problème de l'acheminement des données aux tâches est crucial. Un "réseau" d'interconnexions doit être disponible au sein de cette zone et la nature reconfigurable de cette zone conduit inévitablement à la définition d'un réseau flexible et reconfigurable.

Nous menons actuellement des travaux sur ces problématiques. Nos travaux visent notamment à re-visiter les services d'un OS afin de les faire évoluer, si nécessaire, pour la prise en charge efficace de ce type de ressource reconfigurable. Conjointement, nous nous intéressons à la définition d'un réseau flexible pour l'acheminement des données au sein même de la zone reconfigurable. Cette étude s'intéresse à la définition de l'ossature d'un tel réseau et des éléments de "routage".

Sur ce thème de recherche, j'ai été responsable scientifique du projet OverSoC pour l'équipe CAIRN. Ce projet a été labélisé par l'ANR en tant que projet de type ARA[20] SSIA[21]. Il a débuté en décembre 2005 pour une durée de 3 ans, il a permis de formaliser des travaux entamés avec l'équipe Architecture du laboratoire ETIS[22] de l'ENSEA (UMR CNRS 8051) et de l'équipe SYEL[23] du laboratoire LISIF de l'Université Pierre et Marie Curie.
Dans ce contexte, j'ai co-endacré les travaux des étudiants suivants :

- Lallit GARG, étudiant du master STI de Rennes 1. Ces travaux, qui se sont déroulés de avril à août 2006, ont permis de faire un état de l'art des solutions proposées dans la littérature sur le thème des réseaux pour RSoC.
- Karim HAROUAT, étudiant de 3ième année d'école d'ingénieur ENSSAT. Durant ce projet technologique réalisé de septembre 2006 à mars 2007, nous avons défini un premier niveau de description d'une zone reconfigurable en SystemC. Une modélisation UML[24] de la zone reconfigurable a été définie et a ensuite servi de base à la description SystemC. J'ai pris en charge la suite de ces travaux en enrichissant progressivement la description SystemC.
- Umer FAROOQ, étudiant de master Sciences technologies et santé, mention sciences et technologies de l'information et de la communication de l'université de Nice, spécialité systèmes embarqués. Ces travaux de master, qui se sont déroulés de avril à août 2007, ont porté sur la définition d'une première architecture de communication flexible pour une cible d'exécution reconfigurable. Ces travaux ont donné lieu à une description en VHDL de l'ensemble des éléments de la structure.

[19] OS : Operating System
[20] ARA : Actions de Recherche Amont
[21] SSIA : Sécurité, Systèmes embarqués & Intelligence Ambiante
[22] ETIS : Equipes Traitement de l'Information et Systèmes
[23] SYEL : Systèmes Electroniques
[24] UML : Unified Modeling Language

– Ronan GUGUIN, étudiant de 3ième année d'école d'ingénieur ENSSAT. L'objectif de ce projet technologique a consisté à poursuivre les travaux de Umer FAROOQ. Le travail a notamment permis de transcrire en SystemC la solution précédemment définie. Cette transcription permet la modélisation de la dynamicité de la zone reconfigurable, ce qui n'était pas possible avec le langage VHDL.
– Ludovic DEVAUX, étudiant de master STI de Rennes 1. Les travaux ont débuté en mars 2008, et se poursuivent actuellement en thèse. Il s'agit d'approfondir l'ensemble des problématiques relatives à la mise en place d'une structure de communication flexible au sein d'un SoC. On étudie notamment la partie contrôle de la solution proposée.

L'ensemble de ces travaux est inclus dans les projets ANR OverSoC (achevé en juin 2009) et FosFor (planifié de janvier 2008 à décembre 2010). Le chapitre 2 présente en détail ces travaux.

Ordonnancement de tâches au sein d'un SoC multi-processeurs hétérogène

La constitution matérielle des SoC est très largement basée sur une multitude de blocs dont les fonctionnalités sont diverses et variées. Du point de vue du développeur d'applications, cette diversité peut se traduire par la disponibilité de plusieurs cibles d'exécution potentielles pour une même tâche. Le contrôle de l'exécution des tâches sur ces différentes cibles doit pouvoir se faire au travers de l'appel des services du système d'exploitation. L'ordonnancement, qui est l'un des services principaux de l'OS, manipule donc naturellement des cibles d'exécution hétérogènes et doit satisfaire les contraintes applicatives, qui peuvent être de performances temporelles ou encore, et c'est souvent le cas pour des systèmes embarqués, de performances énergétiques.
Dans ce contexte, nous avons proposé un ordonnancement original s'appuyant sur une réalisation matérielle à base de réseaux de neurones. Notre proposition étend les travaux de Cardeira et Mammeri en proposant une nouvelle fonction d'énergie. Cette fonction d'énergie se traduit par une nouvelle règle de construction du réseau de neurones permettant le contrôle de l'ordonnancement des tâches dans un contexte d'hétérogénéité de la plate-forme matérielle (le SoC) et d'instanciations multiples de chaque tâche sur plusieurs cibles d'exécutions.

Sur ce thème, bien que n'étant pas officiellement encadrant des travaux de thèse d'Imène BENKERMI [1](travaux soutenus en janvier 2007), j'ai très largement participé à la définition de la fonction d'énergie permettant d'assurer la convergence du réseau de neurones dans le cas d'une architecture multi-processeurs hétérogène. Depuis la soutenance de Mlle BENKERMI, j'ai pris en charge ce thème de recherche et j'ai notamment proposé deux nouvelles structures dont les objectifs concernent principalement la diminution du nombre de neurones pour modéliser la solution et la définition d'un réseau de neurones reconfigurable mieux adapté à la gestion d'un RSoC. Finalement, j'encadre Antoine EICHE depuis octobre 2008 pour des travaux dont l'objectif consiste à étudier la prise en compte du placement des tâches (ordonnancement spatial) au sein d'une architecture reconfigurable. Ces travaux seront présentés dans le chapitre 3. Une partie de ces travaux devrait être mise en valeur dans le projet ANR FosFor.

Problématique de la consommation des circuits

Au sein de l'équipe, les premiers travaux concernant la conception de circuits faible consommation ont été menés en reprenant le flot de conception de haut niveau de GAUT. Il s'agissait alors de modifier le flot de synthèse afin de cibler des cœurs de processeurs spécifiques (ASIP[25]). L'objectif concernait alors la définition d'un cœur de processeur capable d'exécuter un ensemble de traitements, tout en assurant une consommation énergétique minimale. Ces travaux ont permis de développer des estimateurs de consommation pour plusieurs niveaux de descriptions. Les éléments classiques des unités de traitement ont été caractérisés, permettant d'estimer l'architecture globale synthétisée. Finalement, un estimateur du jeu d'instructions du cœur de processeur a été développé. La conception de systèmes faible consommation s'est ensuite poursuivie par des travaux concernant les aspects mémoire, ainsi que les architectures reconfigurables qui semblaient être de bonnes candidates d'un point de vue énergétique.

Sur ce thème, j'ai participé à l'encadrement des travaux de thèse de Jean Gabriel COUSIN concernant la conception de cœurs d'ASIP [2]. J'ai ensuite été le tuteur pédagogique de Raphaël DAVID qui a montré, dans ses travaux de

[25] ASIP : Applied Specific Integrated Processor

thèse, qu'une architecture reconfigurable à gros grain permettait un gain énergétique important.
Depuis janvier 2009, je suis impliqué dans le projet ANR Open-People dont l'objectif concerne la modélisation de la consommation énergétique d'un SoC complet. Au sein de ce projet, l'implication de l'équipe Cairn concerne l'étude de la consommation des architectures reconfigurables. A ce titre, j'encadre depuis octobre 2009 Robin BONAMY dont les travaux de thèse sont financés par ce projet. Ces travaux ont pour objectif la définition de modèles de consommation des applications sur une zone reconfigurable, ces modèles viendront alimenter la bibliothèque de modèles de la plateforme Open-People.

Autres implications

Parallèlement aux activités mentionnées ci-dessus, j'ai participé à l'encadrement de plusieurs doctorants, étudiants de master et élèves ingénieurs sur des travaux de recherche ne relevant pas directement des problématiques précédentes. Seule une brève description de ces travaux est donnée dans les sections qui suivent afin de concentrer le discours du document sur l'axe de la conception de *System-on-Chip* et sur des travaux plus récents.

Estimation du placement/routage afin de guider la synthèse de haut niveau

Pour anticiper les problèmes de conception engendrés par l'allongement des délais de propagation dans les interconnexions des circuits sub-microniques, nous avons réalisé, il y a quelques années, des travaux visant à estimer le placement/routage d'un circuit afin de guider la synthèse comportementale. A l'époque, l'obtention des performances réelles du système à concevoir souffrait des nombreuses phases de placement/routage, phases longues et difficilement ré-itérables compte tenu du temps de calcul nécessaire. L'idée des travaux que nous avons alors menés a consisté à estimer le placement/routage du circuit de façon rapide afin d'obtenir une classification relative des solutions architecturales possibles. L'estimateur de "floorplan", ou plan de masse, que nous avons développé s'appuie sur la description de l'architecture définie par la synthèse de haut niveau et sur les caractéristiques, notamment surfaciques, des blocs : opérateurs, registres et autres éléments instanciés. En fait, chaque bloc est décrit par une fonction de formes indiquant les différents couples $(largeur, hauteur)$ des instanciations possibles. L'architecture est modélisée par un graphe dont les nœuds représentent les blocs, et les arcs représentent l'existence d'une connexion entre deux blocs. Sur la base de ces modélisations, plusieurs méthodes d'estimation de placement ont été étudiées et outillées. Il est clair que l'intérêt de cette estimation réside dans la possibilité de comparer les différentes solutions architecturales, mais que l'erreur d'estimation, concernant notamment les temps de propagation par rapport au circuit final, est importante.

J'ai co-encadré, avec Olivier SENTIEYS, les travaux de thèse de Raofeng YU [3], avec entre autres la mise au point de l'une des techniques de placement de blocs ainsi que le développement des outils logiciels associés. Un outil d'estimation du "layout" ainsi qu'un outil de visualisation ont été développés. Ces travaux ont été soutenus en juin 2002.

Logique Multi-Valuée

Nés de contacts avec une société américaine, Omnibase, nos travaux sur la logique multi-valuée (MVL[26]) ont eu pour objectif d'étudier la faisabilité de circuits à partir d'une logique à base de transistors à enrichissement et à appauvrissement. Sur ce thème, j'ai participé à l'encadrement des travaux de thèse de Ekué KINVI-BOH dont la soutenance a eu lieu en novembre 2006 [4]. Une méthodologie de conception spécifique a été mise en place en proposant notamment la définition d'un paquetage VHDL permettant la manipulation de signaux véhiculant des données multi-valuées. La modélisation et la caractérisation d'opérateurs ont ainsi pu être menées dans le but d'évaluer l'intérêt de ce type de logique par rapport à la logique binaire. Des comparaisons entre des fonctions ternaires et leur "équivalent" binaire ont été réalisées autant que faire se peut, sachant qu'il est délicat de comparer des opérateurs élémentaires dont les fonctionnalités ne sont pas complètement identiques. Lors de ces travaux,

[26] MVL : Multiple Valued Logic

un atelier de spécification et de simulation de tables de vérité a été développé, MVL-FrameWork. Il permet la spécification d'une table de vérité quelconque, sa simplification par un outil externe "mvsis" de l'université de Berkley, puis la simulation des équations logiques obtenues.
Globalement, les résultats obtenus montrent que l'intérêt de la logique multi-valuée est assez limité, ceci est en grande partie dû au coût des convertisseurs pour passer du monde binaire au monde ternaire et vice versa. De plus, d'un point de vue consommation, le bilan est mitigé puisque s'il est possible d'obtenir un gain pour certaines fonctions ternaires, celui-ci peut être annulé si l'équivalent binaire est placé dans des conditions de fonctionnement optimal. En effet, pour des circuits ternaires alimentés avec deux tensions Vdd et $Vdd/2$, si l'équivalent binaire est alimenté avec la tension Vdd alors il y a un gain en consommation, mais au détriment d'une plus grande sensibilité au bruit (marge de bruit réduite d'un facteur 2). Par contre, si l'équivalent binaire est alimenté avec la tension $Vdd/2$ alors la consommation est moindre pour le circuit binaire.

Logique asynchrone

Dans le cadre de la synthèse comportementale, nous avons étudié le potentiel de la logique asynchrone vis-à-vis de la traditionnelle logique synchrone. L'objectif de ces travaux a été de montrer qu'il existait un potentiel d'optimisation important en considérant non plus le pire cas du temps de traversée des opérateurs, mais les temps réels de fonctionnement de ceux-ci. Les optimisations ciblées étaient bien évidemment temporelles mais aussi et surtout énergétiques. En effet, l'évolution technologique a, depuis quelques années, tendance à ré-équilibrer les rôles des consommations statique et dynamique au sein des SoC. Si la consommation statique pouvait sans trop de problème être négligée, cela n'est plus le cas pour les technologies actuelles. Or, lorsque l'on effectue une conception synchrone, le concepteur se base sur les pires temps de traversée, ce qui peut finalement conduire à une consommation statique importante. Les circuits asynchrones permettent d'éliminer la "marge" nécessaire au bon fonctionnement des circuits synchrones, cela se traduit par des performances temporelles moyennes et non pire cas. Il est alors possible d'exploiter l'intervalle de temps ainsi "récupéré" pour mettre en place des techniques de contrôle dynamique de la tension d'alimentation (DVS[27]), voir même de couper complètement l'alimentation d'une partie du circuit pour annuler les courants de fuite et donc annuler la consommation statique.

Sur ce thème de recherche, j'ai co-encadré les travaux de thèse de Joseph DEDOU dont la soutenance a eu lieu en 2000 [5]. Une méthodologie de conception d'ASIC asynchrone a été développée. Cette méthodologie a été outillée avec entre autres un outil de simulation de graphes flot de données basés sur des opérations asynchrones dont le fonctionnement est spécifié dans une bibliothèque d'opérateurs développée par ailleurs. Ces travaux ont montré qu'il existait un potentiel intéressant du point de vue temporel, de l'ordre de 60% de gain. En exploitant ce gain de temps, une réduction de consommation statique de l'ordre de $2/5$ est alors envisageable.

[27] DVS : Dynamic Voltage Scaling

Curriculum Vitae

Maître de conférences de l'Université de Rennes 1

Informations personnelles

Daniel CHILLET
Né le 19 avril 1968
Marié, 2 enfants
2, rue Saint Yves
22 420 le Vieux Marché
tél : +33 (0)2.96.38.31.71

Coordonnées professionnelles

IRISA/CAIRN - ENSSAT
6 Rue de Kérampont
BP 80518, 22305 Lannion
Téléphone : +33 (0)2.96.46.90.69
Télécopie : +33 (0)2.96.46.90.75
Daniel.Chillet@enssat.fr
Daniel.Chillet@irisa.fr
http ://cairn.enssat.fr/

Fonctions

09/1994 → 08/1996 :	Doctorant sur bourse régionale, région Bretagne.
09/1996 → 08/1997 :	ATER à l'ENSSAT à Lannion, Université de Rennes 1 en section 61 du CNU.
09/1997 → 09/2004 :	Maître de Conférences à l'ENSSAT à Lannion - Université de Rennes 1 en section 61 du CNU.
09/2004 → 08/2006 :	En délégation INRIA.
Depuis 09/2006 :	Maître de Conférences à l'ENSSAT à Lannion - Université de Rennes 1 en section 61 du CNU.

Formation

1994-1997 :	DOCTORAT	en Traitement du Signal et Télécommunication.
	Titre :	Méthodologie de conception architecturale des mémoires pour circuits dédiés au traitement du signal temps réel
	Laboratoire :	LASTI - ENSSAT Université de Rennes 1
	Directeur de thèse :	E.Martin, LESTER, Université de Bretagne Sud, Lorient
	Thèse soutenue le	9 janvier 1997 à l'ENSSAT devant le jury composé de :
	Président du jury :	M.Corazza, LASTI, ENSSAT, Université de Rennes 1, Lannion
	Rapporteurs :	J.P.Calvez, IRESTE, Université de Nantes
		F.Catthoor, IMEC, Université de Louvain, Belgique
	Examinateurs :	E.Martin, LESTER, Université de Bretagne Sud, Lorient
		O.Sentieys, LASTI, ENSSAT, Université de Rennes 1, Lannion
		J.L.Philippe, LESTER, Université de Bretagne Sud, Lorient
1993-1994 :	DEA STIR : Signal Télécommunication Image Radar, Université de Rennes 1, option Signal et Architecture, mention Assez Bien.	
1992-1993 :	*Service National : Scientifique du contingent, Base aérienne de Reims (BA 112).*	
1989-1992 :	Diplôme d'ingénieur ENSSAT : École Nationale Supérieure de Sciences Appliquées et de Technologie, option EII (Électronique et Informatique Industrielle), à Lannion (22).	

Activités d'enseignement

Mes activités d'enseignement ont débuté durant mes travaux de thèse avec une participation à des encadrements de travaux dirigés et de travaux pratiques à l'IUT de Lorient ainsi qu'à l'ENSSAT. Par la suite, alors que je terminais mes travaux de thèse, j'ai pris en charge des cours magistraux, et notamment des cours de présentation du langage VHDL et d'architectures des processeurs. Mes enseignements se sont ensuite étendus aux aspects temps réel et méthodologique (SART[28]) pour des élèves ingénieurs.
J'ai assuré l'enseignement d'un module du CNAM concernant l'architecture des ordinateurs durant trois années. De plus, j'assure depuis cinq ans des enseignements à TélécomBretagne (Enst Bretagne) autour du sujet de la conception faible consommation des systèmes sur puce. Finalement, j'ai occasionnellement assuré des enseignements du langage VHDL pour des entreprises au travers de la structure Jessica Ouest.

Les items suivants résument mes activités d'enseignement par cycle d'étude.

En 1 er cycle :
- Logique combinatoire et séquentielle ;
- Initiation à l'informatique, langage C ;

En 2 ème cycle :
- Logique combinatoire et séquentielle ;
- Langage VHDL ;
- Architectures avancées des processeurs ;
- Temps réel et méthodologie SART ;
- Conception d'ASIC ;

En 3 ème cycle :
- Architectures avancées des processeurs ;
- Multiprocesseurs, MPSoC ;

[28] SART : Structured Analysis for Real Time

Autres :

- Formation Jessica ”VHDL” aux entreprises ;
- Vacataire CNAM en 2000-2001, 2002-2003 et 2004-2005, module de 50 heures , ”Architecture des systèmes informatiques” ;
- Vacations à l’ENST Bretagne en 2005-2006, 2006-2007, 2007-2008, 2008-2009 et 2009-2010, module ”Gestion intelligente de l’énergie : aspects matériels et logiciels” pour des étudiants de niveau Bac+5.

Dans le cadre de mes enseignements, j’ai développé et encadré les développements de plusieurs outils logiciels à but pédagogique afin d’illustrer des enseignements d’architectures des processeurs et de temps réel.
Concernant l’architecture des processeurs, les outils développés JSimVEM, JSimRISC, VPS et JSimCache constituent une plate forme pour des travaux pratiques. Ces outils sont utilisés par les étudiants de niveau Bac+3, Bac+4 et Bac+5 et permettent d’approfondir le fonctionnement des techniques avancées mises en œuvre dans les processeurs RISC[29] récents : les techniques de prédiction de branchements, de branchements conditionnels, de transferts conditionnels sont quelques uns des mécanismes étudiés en cours et abordés en travaux pratiques sur des exemples concrets. Ces outils ont été développés au sein de l’ENSSAT et sont téléchargeables sur le site de l’équipe à l’adresse : `http ://cairn.enssat.fr`. À titre d’exemple, l’interface de l’outil JSimRisc est présentée à la figure 2.

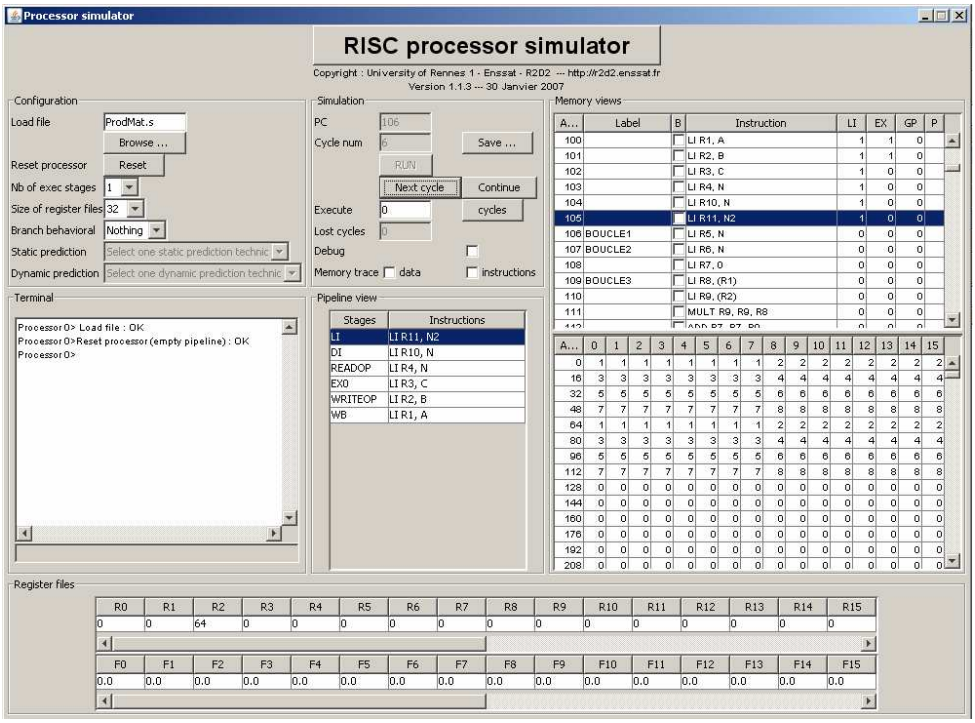


FIG. *2 : Interface graphique de l’outil de simulation de processeurs RISC (JSimRisc) développé dans le but d’illustrer mes enseignements d’architectures avancées de processeurs. Ce simulateur permet de mieux appréhender le fonctionnement pipeline d’un processeur et de cerner clairement les aléas d’exécution. Des techniques de prédiction de branchements sont aussi incluses et il est possible de visualiser leur efficacité lors de l’exécution d’applications simples. Pour des raisons de compatibilité, le langage assembleur retenu pour JSimRisc est très proche de l’outil DlxView modélisant l’architecture Dlx servant de support à l’ouvrage de Hennessy et Patterson, ”bible” pour l’architecture des processeurs [6].*

[29] RISC : Reduced Instruction Set Computer

Encadrements de thèses et de DEA/Masters recherche

Depuis l'obtention de mon doctorat, j'ai participé à l'encadrement de plusieurs thèses, stages de DEAs/Masters et stages d'ingénieurs. Les sujets sur lesquels je suis intervenu couvrent différents aspects de la conception de systèmes, comme par exemple les problèmes de consommation, d'interconnexions au sein de zones reconfigurables, d'ordonnancement de tâches sur machines hétérogènes, de gestion de la reconfiguration dynamique, etc.
Les listes ci-dessous indiquent uniquement les encadrements de thèses et de DEA/Masters. La figure 3 reprend ces encadrements et montre une activité régulière.

Encadrements de stages de DEA/Masters

Karim NOUAR : stage de DEA STIR de l'Université de Rennes 1, en 1998-1999,
Taux d'encadrement : 20 %, Directeur des travaux : Olivier Sentieys.
Sujet : *Impact des technologies submicroniques sur les méthodes de conception des systèmes intégrés.*

Mickaël CARTRON : stage de DEA STIR de l'Université de Rennes 1, en 2003-2002,
Taux d'encadrement : 20 %, Directeur des travaux : Olivier Sentieys.
Sujet : *Optimisations énergétiques d'une plate-forme adaptée aux réseaux de capteurs.*

Imène CHAIEB : stage de DEA Systèmes de communication ENIT, Tunis, en 2003-2004,
Taux d'encadrement : 100%, Directeur des travaux : Daniel Chillet.
Sujet : *Etude de l'application Mpeg2 et proposition de parallélisation en vue de son implémentation "multi-threadée".*

Erwan GRACE : stage de MASTER STI de l'Université de Rennes 1, en 2004-2005,
Taux d'encadrement : 80%, Directeur des travaux : Daniel Chillet.
Sujet : *Etude du concept de reconfiguration dynamique appliqué à la mémoire d'un SoC.*

Lallit GARG : stage de MASTER STI de l'Université de Rennes 1, en 2005-2006,
Taux d'encadrement : 50%, Directeur des travaux : Daniel Chillet.
Sujet : *Etude des solutions pour la réalisation d'un interconnect efficace au sein d'une zone reconfigurable de SoC.*

Umer FAROOQ : stage de MASTER Sciences technologies et santé, mention sciences et technologies de l'informations et de la communication de l'université de Nice, en 2006-2007,
Taux d'encadrement : 50%, Directeur des travaux : Daniel Chillet.
Sujet : *Proposition d'une architecture d'interconnect pour la gestion des interconnexions au sein d'une zone reconfigurable.*

Ludovic DEVAUX : stage de MASTER SISEA, en 2007-2008,
Taux d'encadrement : 50%, Directeur des travaux : Sébastien Pillement.
Sujet : *Etude et implémentation du service de communication d'un OS pour la gestion d'un réseau flexible au sein d'un SoC reconfigurable.*

Rachid DRIF : stage de MASTER SISEA, en 2008-2009,
Taux d'encadrement : 50%, Directeur des travaux : Daniel Chillet.
Sujet : *Etude de la consommation des circuits reconfigurables FPGA.*

Ferhat ABBAS : stage de MASTER SISEA, en 2009-2010,
Taux d'encadrement : 50%, Directeur des travaux : Daniel Chillet.
Sujet : *Etude des langages de modélisation de systèmes en vue de la modélisation de la consommation.*

Amine DIDIOUI : stage de MASTER SISEA, en 2009-2010,
Taux d'encadrement : 30%, Directeur des travaux : Olivier Sentieys.
Sujet : *Modélisation de la consommation des mémoires au sein des systems-on-chip.*

Encadrements de thèses

J.Gabriel COUSIN : thèse soutenue en septembre 1999,
Taux d'encadrement : 20 %, Directeur de thèse : Michel Corazza.
Sujet : *Méthodologie de conception de cœurs de processeurs spécifiques : mise en œuvre sous contraintes, estimation de la consommation.*

Joseph DEDOU : thèse soutenue en octobre 2000,
Taux d'encadrement : 30 %, Directeur de thèse : Olivier Sentieys.
Sujet : *Synthèse de haut niveau d'architectures asynchrones en traitement numérique du signal.*

Raofeng YU : thèse soutenue en juin 2002,
Taux d'encadrement : 70 %, Directeur de thèse : Olivier Sentieys.
Sujet : *Estimation de haut niveau du placement et des interconnexions dans les circuits VLSI submicroniques.*

Ekué KINVI-BOH : thèse soutenue en novembre 2006 ,
Taux d'encadrement : 30 %, Directeur de thèse : Olivier Sentieys.
Sujet : *Conception de circuits en logique ternaire : de la caractérisation au niveau transistor à la modélisation architecturale.*

Imène BENKERMI : thèse soutenue en janvier 2007,
Pour ces travaux, bien que non encadrant, je suis intervenu de façon importante pour les aspects modélisation et développement. Directeur de thèse : Olivier Sentieys.
Sujet : *Modèle et algorithme d'ordonnancement pour architectures reconfigurables dynamiquement.*

Erwan GRACE : thèse à soutenir en 2010,
Taux d'encadrement : 50 %, Directeur de thèse : Olivier Sentieys.
Travaux réalisés en collaboration avec le CEA Saclay.
Sujet : *Hiérarchie mémoire reconfigurable.*

Ludovic DEVAUX : thèse à soutenir en 2011,
Ces travaux font partie du projet ANR FosFor et bien que non encadrant, je suis impliqué dans le déroulement des travaux de thèse. Directeur de thèse : Didier Demigny.
Sujet : *Interconnect flexible pour systèmes sur puce reconfigurable.*

Antoine EICHE : thèse à soutenir en 2011,
Taux d'encadrement : 70 %, Directeur de thèse : Olivier Sentieys.
Sujet : *Ordonnancement temps-réel pour architectures hétérogènes reconfigurables à partir de structures de réseaux de neurones.*

Robin BONAMY : thèse à soutenir en 2012,
Taux d'encadrement : 70 %, Directeur de thèse : Olivier Sentieys.
Sujet : *Modélisation de la consommation des circuits reconfigurables et gestion de la reconfiguration dynamique d'un point de vue énergétique.*

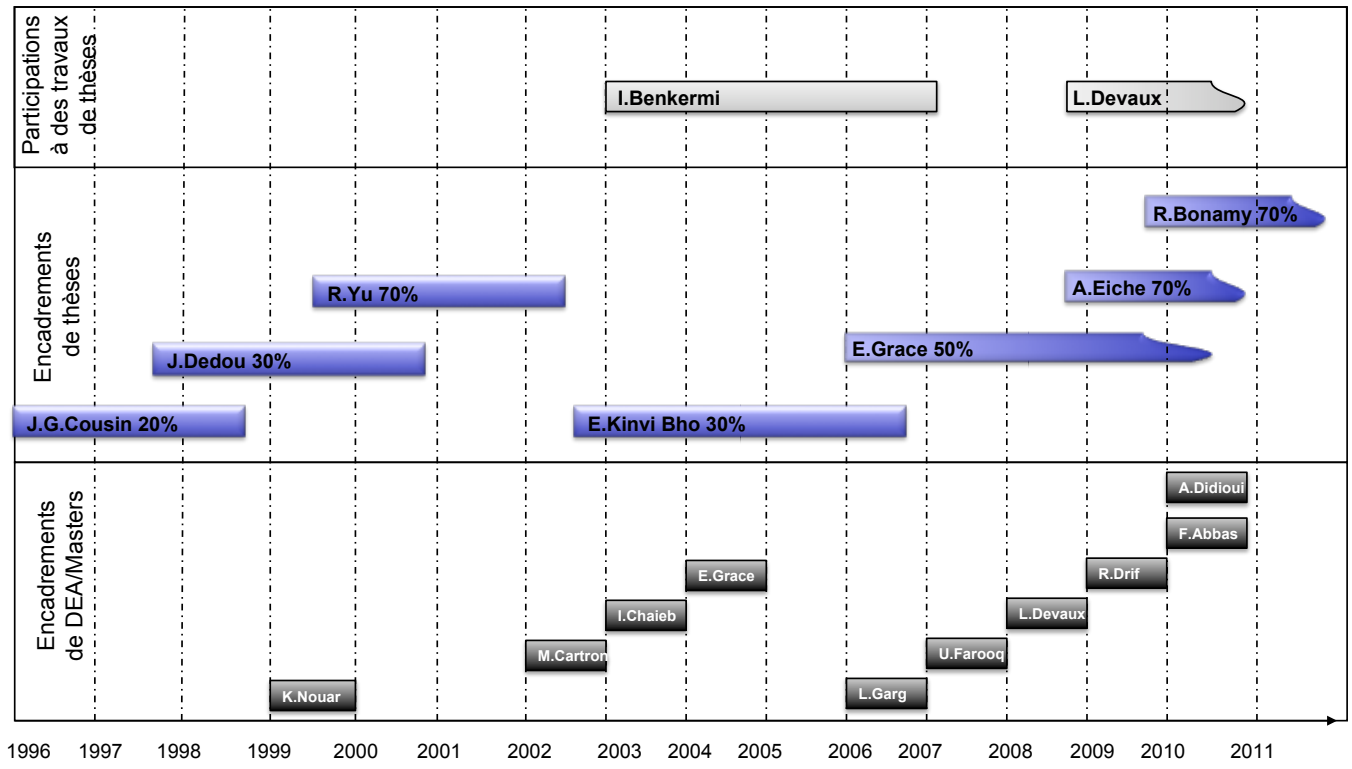


FIG. *3 : Résumé de mes encadrements de thèses et de DEAs/Masters.*

Relations scientifiques

Au cours de mes travaux de recherche, j'ai participé à des contrats de recherche et des projets en relation avec des partenaires universitaires et industriels. Au travers de ces projets, j'ai poursuivi des activités autour d'axes de recherche en cours, mais ils ont aussi été l'occasion de développer de nouveaux axes notamment autour des aspects systèmes d'exploitation pour les architectures reconfigurables. Les projets auxquels j'ai participé sont représentés sur la figure 4.

Participations à des contrats de recherche

- Contrat de recherche avec Thomson CSF de Brest : *Réalisation d'une étude relative aux techniques FPGA-ASIC*, 1995. Il s'agissait de montrer l'apport des techniques de synthèse d'architectures par rapport aux capacités des circuits FPGA. Mes travaux de thèse ont notamment été présentés à Thomson dont les applications cibles présentaient un volume de données important.

- Projet ITR (Informatique Télécommunication et Réseaux) *MOCAT* : *Modélisation et Outils pour la Conception d'Architectures pour les Télécommunications*. En collaboration avec Comatlas (Rennes), TNI (Brest), l'IRISA-INRIA (Rennes) et le LESTER-UBS (Lorient). Ce projet s'est déroulé de janvier 1997 à septembre 1998. L'objectif de MOCAT consistait à expérimenter de nouvelles méthodes de conception permettant d'accélérer le passage de la définition d'une application à sa réalisation sur un ou plusieurs circuits intégrés. Une application réelle, un décodeur de Viterbi, fournie par Comatlas, a été abordée avec d'une part des outils commerciaux de conception de circuits, et d'autre part, avec des logiciels prototypes (Alpha, Gaut) et industriels (Sildex) développés par les partenaires du projet.

- Projet MVL : Multi-Value Logic, en collaboration avec la société Omnibase : *Architectures et circuits en technologie Sus-Loc*. Il s'agissait principalement d'étudier l'intérêt de la logique multi-valente dans le cadre de la

conception de systèmes et notamment en tentant de réaliser une comparaison entre deux cœurs de processeur, l'un en fonctionnant en binaire et l'autre en ternaire. Les résultats ont montré que s'il existe un intérêt à travailler en MVL, notamment d'un point de vue consommation, il faut que l'ensemble de la chaîne manipule ce format de codage, pour limiter l'impact énergétique des convertisseurs. Les études ont aussi montré qu'il était délicat de comparer des fonctions ternaires et binaires puisqu'il n'existe pas d'équivalent simple pour toutes les fonctions ternaires de base.

- Projet PHRASE STSI du ministère de l'industrie, en collaboration avec STMicroelectronics et l'UBO, de 2001 à 2003 : *Définition d'une plate-forme intégrée parallèle et hétérogène, Architecture et Logiciels*. Il s'agissait de définir à la fois une architecture et une méthodologie de conception pour une architecture fortement parallèle.

Participations à des projets nationaux

- Projet Télécom CNRS : *MACGTT : Méthode d'aide à la conception des terminaux de télécommunications*. Les partenaires de ce projet étaient l'I3S de Sophia Antipolis à Nice et le LESTER de l'Université de Bretagne Sud. Le projet s'est déroulé de Janvier 2000 à Décembre 2001 et a abordé la mise en commun d'outils de conception disponible au sein des différentes équipes et la définition d'un flot de conception général couvrant à la fois les aspects co-design et les aspects synthèse d'architectures.

- Projet RNRT MILPAT : *Méthodologie et développement pour les InteLlectuals Properties (IP*[30]*) pour Applications Telecoms*. Projet Exploratoire 1, Thème 4.a, appel d'offre du 21/09/2998, en collaboration avec France Télécom R&D et le LESTER, de 1999 à 2001. Le travail réalisé dans le cadre de ce projet a consisté d'une part à formaliser la méthode de spécification d'un IP comportemental et d'illustrer cette démarche à travers l'établissement d'une bibliothèque de base de composants virtuels dédiés au traitement du signal. Ces travaux ont donné lieu aux développements de deux outils au sein de l'équipe : IPDesigner, et IPCompiler. IPDesigner permet la définition d'un IP à partir d'un résultat de synthèse d'architectures, le second outil permet de fixer les paramètres applicatifs et de produire le code VHDL synthétisable du bloc IP en dérivant les paramètres applicatifs en paramètres architecturaux.

- Projet SOCLIB : *Mise à disposition d'une librairie libre de modèles de simulation interopérables de cœurs d'IP à destination des industriels et des laboratoires de recherche académiques*. Projet regroupant 4 industriels, et 11 laboratoires académique. Projet soutenu par le CNRS et qui a été labélisé par l'ANR en 2006.

- Projet RNTL OSGAR : *Outils de Synthèse Générique pour Architectures Reconfigurables*. Projet Exploratoire, thème 1, en collaboration avec le CEA, la société TNI Valiosys et l'Université de Bretagne Occidentale, 2002-2005. L'objectif général de ce projet a consisté à étudier et à développer une chaîne d'outils de synthèse de haut niveau capable de prendre en charge la description d'une application en langage C et de produire une description pour un ou plusieurs circuits reconfigurables de type FPGA.

- Projet ANR OversSoC : *Méthodologie de validation et d'exploration des interactions entre les OS et les systèmes reconfigurables sur puce*. Projet ANR ARA SSIA (Actions de Recherche Amont : Sécurité, Systèmes embarqués et Intelligence Ambiante), labélisé en 2005. En collaboration avec l'équipe Architecture du laboratoire ETIS (UMR CNRS 8051) et le groupe SYEL du laboratoire LISIF. Ce projet a débuté en décembre 2005 et s'est terminé en juin 2009. L'objectif de ce projet a consisté à proposer un flot d'exploration pour un système sur puce reconfigurable en prenant en charge l'exploration des services du système d'exploitation. J'étais le responsable scientifique de l'équipe CAIRN pour ce projet.

- Projet ANR FosFor : *Flexible Operating System FOr Reconfigurable platform*. Projet ANR Architecture du futur, labélisé en 2007. En collaboration avec l'équipe Architecture du laboratoire ETIS (UMR CNRS 8051), le LEAT (UMR 6071) et Thales Research and Technology. Ce projet a débuté en janvier 2008 et aborde les as-

[30] IP :InteLlectuals Properties

pects architecturaux des systèmes de type multi-processeurs sur puce en étudiant notamment l'implémentation matérielle de certains services d'un système d'exploitation.

- Projet ANR Open-People : *Plateforme ouverte pour l'estimation et l'optimisation de la consommation en puissance et en énergie*. Projet ANR Systèmes Embarqués et Grandes Infrastructures, labélisé en 2008. En collaboration avec le LabSticc (UBS), le LORIA (Inria Nancy Grand Est), le LEAT (UMR 6071), le LIFL (Inria Lille) et Thales (Colombes). Ce projet a débuté en avril 2009 et aborde les aspects estimation et optimisation de la consommation des systèmes sur puce. L'objectif global du projet consiste à développer une plate forme matérielle et logicielle permettant l'implémentation d'applications sur des cibles matérielles en vue d'obtenir une estimation de la consommation énergétique. Je suis le responsable scientifique de l'équipe CAIRN pour ce projet.

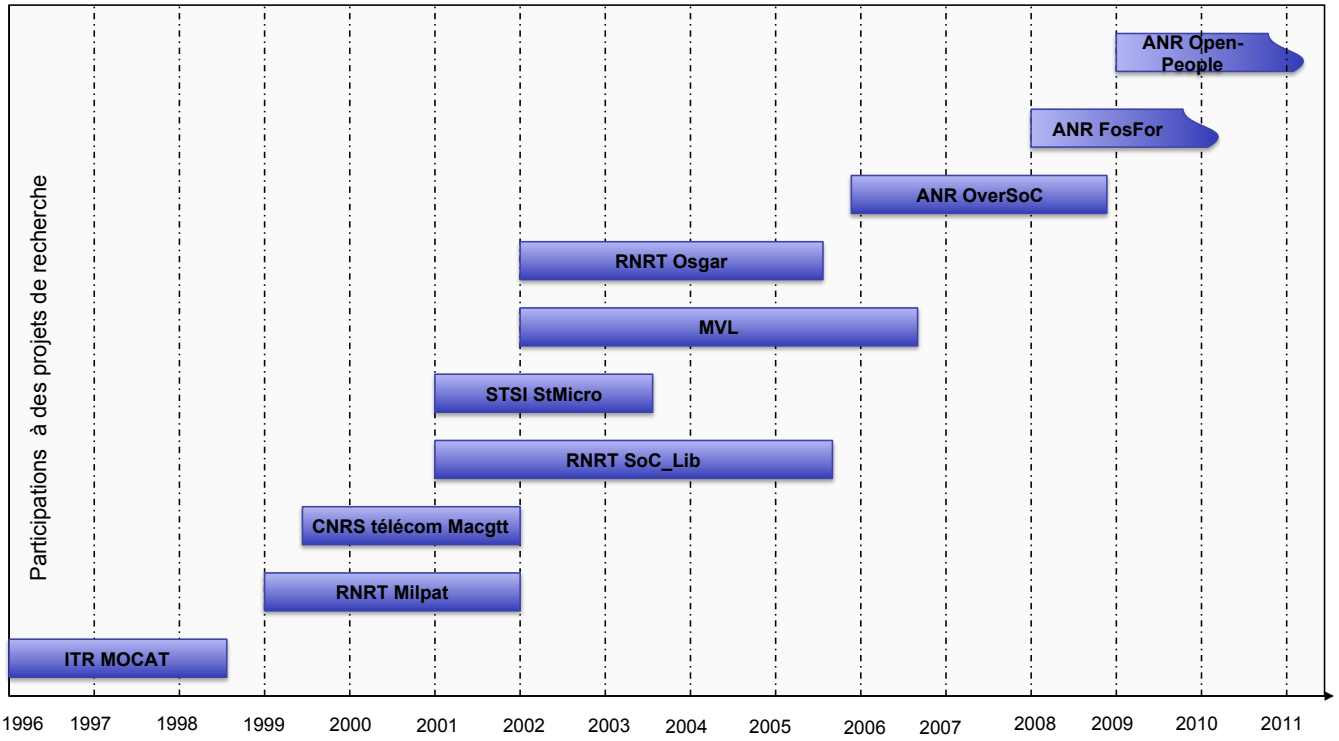


FIG. *4 : Résumé de mes participations à des projets de recherches*

Expertises auprès d'entreprises

L'expertise de l'équipe dans le domaine de l'électronique au sens large, nous conduit régulièrement à la réalisation d'expertises pour des entreprises au travers des actions Jessica Ouest. J'ai personnellement pris en charge plusieurs études allant de la rédaction de cahiers des charges à la proposition de solutions techniques pour différentes entreprises. Ces études sont un moyen intéressant de mieux appréhender les problématiques de petites et moyennes entreprises qui souhaitent faire évoluer leurs équipements vers plus d'automatisme, plus de qualités, etc. C'est aussi l'occasion de constater que nos travaux de recherche sont souvent complètement inconnus de ce type d'entreprises et au delà de la simple prestation, les discussions et échanges sur nos activités réciproques sont toujours intéressants pour les deux partenaires.

Activités diverses concernant la recherche

Au delà des activités de recherche proprement dite, je participe à la "vie" de l'ENSSAT et de l'Université au travers de mon implication dans diverses instances (Conseil de direction, Conseil Scientifique, Commission de spécialistes, Conseil National des Universités, etc).
De même, j'ai participé et participe à plusieurs actions émanant notamment du CNRS (GDRs). Ces actions, principalement articulées autour de réunions, sont un vecteur d'échanges important qu'il convient de ne pas négliger, tant pour la qualité des discussions qu'elles engendrent au travers de la confrontation des idées de chacun que pour les projets de recherche qui peuvent en émaner.

- Membre IEEE, depuis janvier 1997 ;
- Membre nommé du Conseil National des Universités, section 61 (CNU 61) depuis décembre 2009 ;
- Membre élu du Conseil Scientifique de l'Enssat, élu en octobre 1999, réélu en octobre 2003, réélu en octobre 2007 ;
- Membre élu de la Commission de Spécialistes 61° section de l'Universités de Rennes 1, titulaire de septembre 2001 à septembre 2004, suppléant depuis septembre 2004 ;
- Membre élu de la Commission de Spécialistes 61° section de l'Universités de Bretagne Sud, titulaire de septembre 2001 à septembre 2004, suppléant depuis septembre 2004 ;
- Membre du Conseil d'Equipe du LASTI de octobre 1998 à décembre 2003 ;
- Titulaire d'une Prime d'Encadrement Doctoral et de Recherche, de septembre 2000 à septembre 2004, renouvellement acquis en septembre 2004 ;
- Membre des comités de lecture de DCIS (2007, 2008, 2009, 2010) NewCas (2008, 2009, 2010) ;
- Membre du comité d'organisation de DASIP 2007, 2008, 2009 (Workshop on Design and Architectures for Signal and Image Processing) ;
- Membre du comité de programme/lecture de MajecStic (MAnifestation des JEunes Chercheurs en Sciences et Technologies de l'Information et de la Communication) 2005, 2006, 2007 et 2008 ;
- Conférencier concernant la problématique de la conception de systèmes faible consommation aux écoles d'été *Archi'03* (Roscoff, 2003), *Ecofac'06* (Nice, 2006), *EcoFac'10* (Plestin les Grèves, 2010) ;
- Webmaster du site internet du Groupe Signal et Architecture du LASTI depuis juin 2000 et de l'équipe R2D2 de 2003 à 2007 ;

Autres activités

- Membre du club EEA, depuis janvier 1997 ;
- Membre élu du conseil de direction de l'Enssat de octobre 2003 à octobre 2006 ;
- Responsable pédagogique de la deuxième année Enssat, option EII (Eletronique et Informatique Industrielle), de septembre 2001 à Juin 2004 ;

- Responsable pédagogique de la troisième année Enssat, option EII (Eletronique et Informatique Industrielle), depuis septembre 2006 ;
- Conseils d'expertises pour plusieurs PME/PMI dans le cadre des actions Jessica Ouest ;
- Responsable placement pour les diplômés Enssat depuis septembre 1999 ;
- Responsable de l'Observatoire de l'Enssat depuis septembre 2006 ;
- Membre de la cellule TICE de l'Enssat depuis 2009 ;

Bibliographie personnelle

[A1] D. Chillet, *Méthodologie de conception architecturale des mémoires pour circuits dédiés au traitement du signal temps réel*, thèse de doctorat, ENSSAT-Université de Rennes, Janvier 1997.

Articles en revues internationales et nationales

[A2] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, "Flexible interconnection network for dynamically and partially reconfigurable architectures", *International Journal on Reconfigurable Computing (IJRC)*, 2010.

[A3] B. Miramond, E. Huck, F. Verdier, A. Benkhelifa, B. Granado, M. Aichouch, J. C. Prevotet, D. Chillet, S. Pillement, "OveRSoC : a Framework for the Exploration of RTOS for RSoC Platforms", *International Journal of Reconfigurable Computing*, 2010.

[A4] A. Benkhelifa, D. Chillet, J. Denoulet, B. Granado, B. Miramond, S. Pillement, J. C. Prevotet, F. Verdier, "Operating Systems in Reconfigurable Platforms : a Survey", *Submit to ACM Transactions in Embedded Computing Systems (TECS)*, 2009.

[A5] D. Chillet, S. Pillement, O. Sentieys, "Ordonnancement de tâches par réseaux de neurones pour architectures de SoC hétérogènes", *Traitement du signal 26*, 1, 2009, p. 77–89.

[A6] D. Chillet, S. Pillement, O. Sentieys, "RANN : A Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable System-on-Chip", *Submit to ACM Transactions on Reconfigurable Technology and Systems*, 2009.

[A7] D. Chillet, S. Pillement, O. Sentieys, "Real-Time Scheduling on Heterogeneous SoC Architectures Using Inhibitor Neurons in a Neural Network", *Submitted to Journal of Systems Architecture*, 2009.

[A8] D. Chillet, R. David, E. Grace, O. Sentieys, "Structure mémoire reconfigurable : vers une structure de stockage faible consommation", *RSTI – Technique et Science Informatiques 27*, 1-2, 2008, p. 183 – 204.

[A9] D. Menard, D. Chillet, O. Sentieys, "Floating-to-fixed-point Conversion for Digital Signal Processors", *EURASIP Journal on Applied Signal Processing, Special Issue Design Methods for DSP Systems*, 1, 2006, p. 1–15.

[A10] D. Menard, T. Saïdi, D. Chillet, O. Sentieys, "Implantation d'algorithmes spécifiés en virgule flottante dans les DSP virgule fixe", *Technique et Science Informatiques 22*, 2, 2003, p. 783–809.

[A11] R. David, D. Chillet, S. Pillement, O. Sentieys, "SOC Design Methodologies", *218*, Kluwer Academic Publishers, 2002, ch. A Dynamically Reconfigurable Architecture for Low-Power Multimedia terminals, p. 51–62.

[A12] J. Diguet, D. Chillet, O. Sentieys, "A Framework for High Level Estimations of Signal Processing Implementations", *Journal of VLSI System for Signal, Image and Video Technology 25*, 3, Juillet 2000.

[A13] H. Dubois, D. Chillet, J. Philippe, O. Sentieys, "Teaching Hardware/Software System Codesign using High-Level CAD tools : a case study in image synthesis", *IEEE Transactions on Education 43*, 3, Aout 2000.

[A14] D. Chillet, J. Philippe, O. Sentieys, H. Dubois, "Conception des unités mémoire pour des applications de traitement du signal temps réel", *Traitement du Signal 14*, 1997.

[A15] D. Chillet, J. Diguet, J. Philippe, O. Sentieys, "Méthodologie de conception des unités mémoires appliquée au traitement du signal temps réel", *Technique et Science Informatiques 16*, 1997.

Communications dans des conférences internationales et nationales

[A16] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, "DRAFT : Flexible interconnection network for dynamically reconfigurable architectures", Proc. of the IEEE International Conference on Field-Programmable Technology (FPT'09), Sydney, Australia, december 2009.

[A17] Y. Oliva-Venegas, J.-C. Prevotet, F. Nouvel, S. Pillement, D. Chillet, "Exploration for Dynamic Reconfiguration Management", *in : Sophia Antipolis MicroElectronics, SAME 2009*, September 2009.

[A18] L. Devaux, D. Chillet, S. Pillement, D. Demigny, "Flexible Communication Support For Dynamically Reconfigurable FPGA", *in : Proc. of the Southern Programmable Logic Conference*, p. 65–70, Sao-Carlos, Brazil, april 2009.

[A19] A. Eiche, D. Chillet, S. Pillement, O. Sentieys, "Flot d'ordonnancement pour architecture reconfigurable", *in : Proc. of the Symposium en Architecture de machines (SympA'13)*, Toulouse, France, September 2009.

[A20] A. Eiche, D. Chillet, S. Pillement, O. Sentieys, "Flot d'Ordonnancement Temps Réel d'un Ensemble de Tâches Matérielles pour Architecture Reconfigurable", *in : Workshop GDR SoCSiP*, June 2009.

[A21] S. Pillement, D. Chillet, Y. Oliva, J. C. Prevotet, "High-Level Exploration for Dynamic Reconfiguration Management", *in : Proc. of the International Conference on Engineering of Reconfigurable Systems & Algorithms, ERSA 2009*, CSREA Press, Las Vegas, Nevada, USA, June 2009.

[A22] S. Pillement, D. Chillet, "High-level Model of Dynamically Reconfigurable Architectures", *in : Proc. of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Nice, France, September 2009.

[A23] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, "Réseau d'interconnexion flexible pour architecture reconfigurable dynamiquement et partiellement", *in : Proc. of the Symposium en Architecture de machines (SympA'13)*, Toulouse, France, September 2009.

[A24] J. C. Prevotet, A. Benkhelifa, B. Granado, E. Huck, B. Miramond, F. Verdier, D. Chillet, S. Pillement, "A Framework for the Exploration of RTOS Dedicated to the Management of Hardware Reconfigurable Resources", *in : RECONFIG '08 : Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*, IEEE Computer Society, p. 61–66, Washington, DC, USA, 2008.

[A25] E. Grace, R. David, D. Chillet, O. Sentieys, "MOREA : A Memory-Oriented Reconfigurable Embedded Architecture", *in : Conference on Design and Architectures for Signal and Image Processing*, p. 124–131, Université Libre de Bruxelles, Belgium, November 24 -26 2008.

[A26] D. Chillet, S. Pillement, O. Sentieys, "Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable SoC", *in : Conference on Design and Architectures for Signal and Image Processing*, p. 92–99, Université Libre de Bruxelles, Belgium, November 24 -26 2008.

[A27] D. Chillet, I. Benkermi, S. Pillement, O. Sentieys, "Hardware Task Scheduling for Heteregeneous SoC Architectures", *in : EUSIPCO 2007*, 2007.

[A28] D. Chillet, S. Pillement, O. Sentieys, "A Neural Network Model for Real-Time Scheduling on Heterogeneous SoC Architectures", *in : International Joint Conference on Neural Networks, IJCNN 2007*, Orlando, Floride, august, 12-17 2007.

[A29] D. Chillet, S. Pillement, O. Sentieys, "Vers une implémentation matérielle d'un réseau de neurones pour le service d'ordonnancement des tâches au sein d'un SoC", *in : GRETSI 2007*, 2007.

[A30] F. Hannig, H. Dutta, A. Kupriyanov, J. Teich, R. Schaffer, S. Siegel, R. Merker, R. Keryell, B. Pottier, D. Chillet, D. Menard, O. Sentieys, "Co-Design of Massively Parallel Embedded Processor Architectures", *in : Workshop ReCoSoC 2005, Reconfigurable Communication-Centric SoCs*, Montpellier, France, 2005.

[A31] F. Verdier, J. Prévotet, A. Benkhelifa, D. Chillet, S. Pillement, "Exploring RTOS issues with a high-level model of a reconfigurable SoC platform", *in : Workshop ReCoSoC 2005, Reconfigurable Communication-Centric SoCs*, Montpellier, France, 2005.

[A32] D. Chillet, L. Abdelouel, O. Sentieys, "Modèle générique de hiérarchie mémoire pour l'exploration architecturale", *in : Symposium en Architecture de Machines, SympA'2005*, Le Croisic, Presqu'île de Guérande, Avril 2005.

[A33] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J. Prevotet, F. Verdier, "Modélisation niveau système de SoC reconfigurables", *in : Symposium en Architecture de Machines, SympA'2005*, Le Croisic, Presqu'île de Guérande, Avril 2005.

[A34] L. Abdelouel, D. Chillet, O. Sentieys, "Synthèse de l'interconnexion des mémoires dans un contexte multiprocesseurs", *in : Soumis à Majestic 2005, Troisième MAnifestation des JEunes Chercheurs dans les domaines STIC*, Rennes, France, 2005.

[A35] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J. Prévotet, F. Verdier, "System-Level Modelling for Reconfigurable SoCs", *in : DCIS'05, XX Conference on Design of Circuits and Integrated Systems Lisboa - Portugal*, November 23-25 2005.

[A36] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J. Prevotet, F. Verdier, "System-Level Modelling for Reconfigurable SoCs", *in : 17th Euromicro Conference on Real Time Systems*, Palma de Mallorca, Balearic Islands, Spain, Juillet 2005.

[A37] D. Saillé, D. Chillet, O. Sentieys, "Construction d'une hiérarchie mémoire faible consommation", *in : Colloque Faible Tension Faible Consommation FTFC'03*, Paris, Mai 2003.

[A38] D. Menard, D. Chillet, F. Charot, O. Sentieys, "Automatic Floating-point to Fixed-point Conversion for DSP Code Generation", *in : International Conference on Compilers, Architectures and Synthesis for Embedded Systems 2002 (CASES 2002)*, Grenoble, October 2002.

[A39] O. Sentieys, S. Pillement, D. Chillet, "Behavioral IP Specification and Integration Framework for High-Level Design Reuse", *in : ISQED 2002 : 3rd International Symposium on Quality Electronic Design*, p. 388–393, San Jose, California, USA, Mars 2002.

[A40] R. David, D. Chillet, S. Pillement, O. Sentieys, "A Compilation Framework for a Dynamically Reconfigurable Architecture", *in : 12th International Workshop on Field-Programmable Logic and application*, p. 1058–1067, Montpellier, France, Septembre 2002.

[A41] R. David, D. Chillet, S. Pillement, O. Sentieys, "A Compilation Framework for a Dynamically Reconfigurable Architecture", *in : 12th IEEE International Conference on Field Programmable Logic and Applications, FPL 2002*, *2438*, Lecture Notes in Computer Sciences, Springer CS P, Septembre 2002.

[A42] R. David, D. Chillet, S. Pillement, O. Sentieys, "DART : A Dynamically Reconfigurable Architecture dealing with Next Generation Telecommunications Constraints", *in : 9th IEEE Reconfigurable Architecture Workshop RAW*, IEEE CS Press, p. 0156–0164, Avril 2002.

[A43] R. David, D. Chillet, S. Pillement, O. Sentieys, "Flot de Conception pour Plateforme Reconfigurable", *in : Colloque CAO 2002*, p. 79–82, Avril 2002.

[A44] R. David, D. Chillet, S. Pillement, O. Sentieys, "A High-Performance dynamically reconfigurable embedded architecture", *in : Sophia Antipolis Forum on Microelectronics, SAME'2002*, Montpellier, France, Septembre 2002.

[A45] R. David, D. Chillet, S. Pillement, O. Sentieys, "Mapping Future Generation Mobile Telecommunication Applications on a Dynamically Reconfigurable Architecture", *in : 27th International Conference on Acoustics, Speech, and Signal Processing, ICASSP'2002*, Orlando, Florida, USA, Mai 2002.

[A46] D. Chillet, S. Pillement, O. Sentieys, al., "Vers une approche unifiée pour la conception globale des terminaux de télécommunications", *in : JFAAA*, Gabes, Tunisie, Decembre 2002.

[A47] S. Pillement, D. Chillet, O. Sentieys, "A Virtual Component for Motion Estimation Algorithm", *in : ERSA'02 : 2002 International Conference on Engineering of Reconfigurable Systems and Algorithms*, Juin 2002.

[A48] R. David, S. Pillement, O. Sentieys, D. Chillet, "Architectures Enfouies Reconfigurables Dynamiquement", *in : Symposium en Architectures Nouvelles de Machines SYMPA'7*, p. 23–32, Paris, France, Avril 2001.

[A49] S. Pillement, O. Sentieys, D. Chillet, E. Casseau, P. Coussy, E. Martin, G. Savaton, S. Roux, "Design and synthesis of behavioral level virtual components", *in : 11th IFIP Int. Conference on VLSI and System On Chip, VLSI-SOC'2001*, Montpellier, France, Decembre 2001.

[A50] R. David, D. Chillet, S. Pillement, O. Sentieys, "A Dynamically Reconfigurable Architecture for Low-Power Multimedia terminals", *in : VLSI-SOC'01*, Montpellier, France, Decembre 2001.

[A51] D. Chillet, "Evolution des processeurs", *in : Journées de la Science*, Enssat, Lannion, Octobre 2001. Conférence invitée.

[A52] D. Saillé, D. Chillet, O. Sentieys, "Modélisation de la consommation pour les mémoires SRAM", *in : Colloque Faible Tension Faible Consommation FTFC'01*, Paris, 2001.

[A53] S. Pillement, O. Sentieys, D. Chillet, "Vers la définition de composants virtuels au niveau algorithmique", *in : GRETSI'01*, Toulouse, France, Septembre 2001.

[A54] D. Chillet, R. Yu, H. Dubois, O. Sentieys, "Conception haut niveau de circuits integrés : prise en compte des problèmes liés aux interconnexions", *in : 5th AAA Workshop on Algorithm Architecture Adequation*, INRIA Rocquencourt, FRANCE, Janvier 2000.

[A55] J.-G. Cousin, O. Sentieys, D. Chillet, "Multi-algorithm ASIP Synthesis and Power Estimation for DSP Applications", *in : IEEE International Symposium on circuits and systems ISCAS'2000*, Geneva, SW, Mai 2000.

[A56] J. Dedou, D. Chillet, O. Sentieys, "Behavioral Synthesis of Asynchronous Systems : a methodology", *in : International Symposium on Circuits and Systems*, Orlando, USA, Juillet 1999. Articles dans des Conférences Internationales avec Comité de Lecture.

[A57] O. J. Dedou, D. Chillet, O. Sentieys, "Behavioral synthesis of systems : a methodology", *in : IEEE International Symposium on circuits and systems ISCAS'99*, Orlando, USA, may 30 - june 2 1999.

[A58] J. Cousin, D. Chillet, O. Sentieys, "Conception de cœurs d'ASIP : une méthodologie", *in : Symposium en Architectures Nouvelles de Machines : Sympa'5*, Septembre 1999.

[A59] D. Chillet, O. Sentieys, M. Corazza, "Memory Unit Design for Real Time DSP Applications", *in : IEEE Great Lakes Symposium on VLSI GLSV'99*, Ann Arbor, Michigan, USA, Mars 1999.

[A60] J. Dedou, D. Chillet, O. Sentieys, "Synthèse Architecturale des systèmes asynchrones", *in : Colloque Gretsi*, Vannes, France, Septembre 1999.

[A61] J. Dedou, D. Chillet, O. Sentieys, "Synthèse de Haut Niveau des Systèmes Asynchrones", *in : Symposium en Architectures Nouvelles de Machines : Sympa'5*, Juin 1999.

[A62] D. Saillé, M. Denoual, D. Chillet, O. Sentieys, "Un outil d'estimation de la consommation intégrant les caractéristiques du signal", *in : Colloque CAO (Aix-en-Provence)*, p. 140–145, Mai 1999.

[A63] D. Chillet, O. Sentieys, H. Dubois, "Accès simplifié à la synthèse architecturale par Télé-CAO", *in : Journées pédagogiques du CNFM, St Malo*, décembre 1998.

[A64] J.-G. Cousin, D. Chillet, O. Sentieys, "ASIP Design and Power Estimation for DSP Applications", *in : Sophia Antipolis Conference on Microelectronics*, SAME'98, Octobre 1998.

[A65] O. J. Dedou, D. Chillet, O. Sentieys, "Asynchronous Timing Model for High Level Synthesis for DSP Applications", *in : European Signal Processing IX : Theories and applications*, *1*, EUSIPCO-98, p. 475–478, Septembre 1998.

[A66] "Conception de circuits intégrés basse consommation", *in : Journées thématiques sur l'informatique et l'électronique embarqée*, Brest, Octobre 1998. Conférence invitée.

[A67] O. Sentieys, D. Chillet, H. Chuberre, "Convertisseur A/N SigmaDelta", *in : Cinquièmes journées pédagogiques CNFM*, St Malo, Decembre 1998. Articles dans des Conférences sans Comité de Lecture.

[A68] J. Cousin, D. Chillet, O. Sentieys, "Conception de circuits dédiés à une classe d'applications (ASIP) : compromis consommation - performances - flexibilité", *in : Journées d'études SEE - Faible Tension - Faible Consommation*, Paris, Novembre 1997.

[A69] J. Cousin, D. Chillet, O. Sentieys, "Conception de circuits dédiés à une classe d'applications (ASIP) : compromis consommation - performances - flexibilité", *in : Seizième colloque Gretsi sur le traitement du signal et des images*, Grenoble, Septembre 1997.

[A70] D. Chillet, O. Sentieys, "Conception haut niveau des unités de mémorisation", *in : Colloque CAO de circuits intégrés et systèms*, p. 213–216, Grenoble - Villars de Lans, Janvier 1997.

[A71] J. Diguet, O. Sentieys, D. Chillet, E. Martin, "Estimation probabiliste de la complexité de circuits VLSI pour le traitement du signal", *in : Seiziéme colloque Gretsi sur le traitement du signal et des images*, Grenoble, Septembre 1997.

[A72] D. Chillet, O. Sentieys, M. Corazza, "Synthèse des unités mémoire pour le traitement du signal", *in : Seizième colloque Gretsi sur le traitement du signal et des images*, Grenoble, Septembre 1997.

[A73] J. Diguet, O. Sentieys, D. Chillet, J. Philippe, "VLSI High Level Synthesis of Fast Exact Least Mean Square Algorithms", *in : ICASSP'97, 34th IEEE International Conference on Acoustic Speech, and Signal Processing*, Munich, Germany, Avril 1997.

[A74] D. Chillet, J. Philippe, O. Sentieys, H. Dubois, "Architectures des Unités Mémoires pour des Algorithmes de Traitement du Signal Temps Réel", *in : Symposium Architectures Nouvelles de Machines*, IRISA (éditeur), p. 87–96, Rennes, France, 1996.

[A75] O. Sentieys, J. Philippe, D. Chillet, H. Dubois, "Enseigner la conception de systèmes électroniques grâce aux outils de CAO : Application la synthèse d'images", *in : Quatrième journées pédagogiques du Comité National de Formation en Micro électronique : outils de simulation et outils de conception*, Decembre 1996.

[A76] J. Philippe, D. Chillet, O. Sentieys, J. Diguet, "Memory Aspects in Signal Processing and HLS Tool : Some Results", *in : European Signal Processing Conference*, Trieste, Itaky, Septembre 1996.

[A77] O. Sentieys, D. Chillet, J. Diguet, J. Philippe, "Memory Module Selection for High Level Synthesis", *in : VLSI Signal Processing IX*, IEEE Press, 1996.

[A78] D. Chillet, J. Philippe, O. Sentieys, H. Dubois, "Méthodologie de conception des unités de mémorisation pour des algorithmes de traitement du signal", *in : Conférence AAA*, CNES, p. 29–36, Toulouse, France, Janvier 1996.

[A79] D. Chillet, "Les Problèmes de mémorisation dans les algorithmes de filtrage adaptatif", *in : GDR 134, GT6-2 Filtrage Adaptatif et VLSI*, Janvier 1995. Articles dans des Conférences sans Comité de Lecture.

[A80] D. Chillet, "GAUT : spécification VHDL pour la synthèse architecturale", *in : Journée Franco VHDL, université Parix X, IUT GEII Ville d'Avray*, Novembre 1994.

Rapports de recherche et de contrats

[A81] O. Sentieys, D. Chillet, X. Lerouzic, "Evaluation d'outils de synthèse d'architecture pour les applications de traitment du signal", *rapport de recherche*, Rapport interne LASTI, Juillet 1997.

[A82] D. Chillet, "Conception des unités de mémorisation pour un outil de synthèse architecturale dédiée au traitement du signal", *Rapport de fin de 2ème année de thèse*, LASTI-ENSSAT, Septembre 1996, Communication a Diffusion Restreinte.

[A83] D. Chillet, "Étude des problèmes de mémorisation de données en vu du développement d'un outil de synthèse d'unité de mémorisation", *rapport de recherche*, Septembre 1995, Communication a Diffusion Restreinte.

[A84] O. Sentieys, D. Chillet, J. Philippe, E. Martin, "Réalisation d'une étude relative aux techniques et technologique FPGA-ASIC", *Rapport de contrat d'étude entre thomson csf/brest et l'enssat*, LASTI-ENSSAT, Avril 1995.

[A85] D. Chillet, "Contribution à la synthèse des unités de mémorisation pour GAUT", *Stage de dea stir*, LASTI-ENSSAT, Septembre 1994.

[A86] D. Chillet, ”Réalisation d’une passerelle entre GAUT et le langage de description VHDL”, *Stage de 3ieme annee enssat*, LASTI-ENSSAT, Septembre 1992, Communication a Diffusion Restreinte.

Œuvres audiovisuelles

[A87] D. Menard, D. Chillet, O. Sentieys, ”Étude de faisabilité d’un récepteur satellitaire - Partie 2”, Rapport jessica pour la société Sofreavia, jun 2000.

[A88] D. Chillet, D. Ménard, O. Sentieys, ”Étude de faisabilité d’un récepteur satellitaire : partie 1”, Rapport Jessica pour la société Sofreavia (Issy Les Moulineaux), 2000, Rapports de Conseil aux Entreprises.

[A89] D. Chillet, M. Guitton, ”Etude d’un système de conversion et de transfert en temps réel”, Rapport Jessica pour la société Cylor Technologies (Pabu), 2000, Rapports de Conseil aux Entreprises.

[A90] O. Sentieys, D. Chillet, D. Ménard, ”CPU embarquée pour la télésurveillance”, Rapport Jessica pour la société TIMEAT (Rennes), 1999, Rapports de Conseil aux Entreprises.

Deuxième partie

Synthèse des travaux

Chapitre 1

Hiérarchie mémoire au sein des SoC

L'évolution architecturale des *System-on-Chip* accorde de plus en plus de place à la fonction de mémorisation (voir figure 1.1). Cela est en partie dû au besoin grandissant en stockage des applications. Pour être efficace, l'augmentation importante du volume de stockage ne peut se faire uniquement linéairement, c'est-à-dire en augmentant simplement l'espace d'adressage. La recherche d'un compromis entre l'efficacité et l'espace de stockage conduit les concepteurs à proposer des organisations complexes, généralement hiérarchiques.

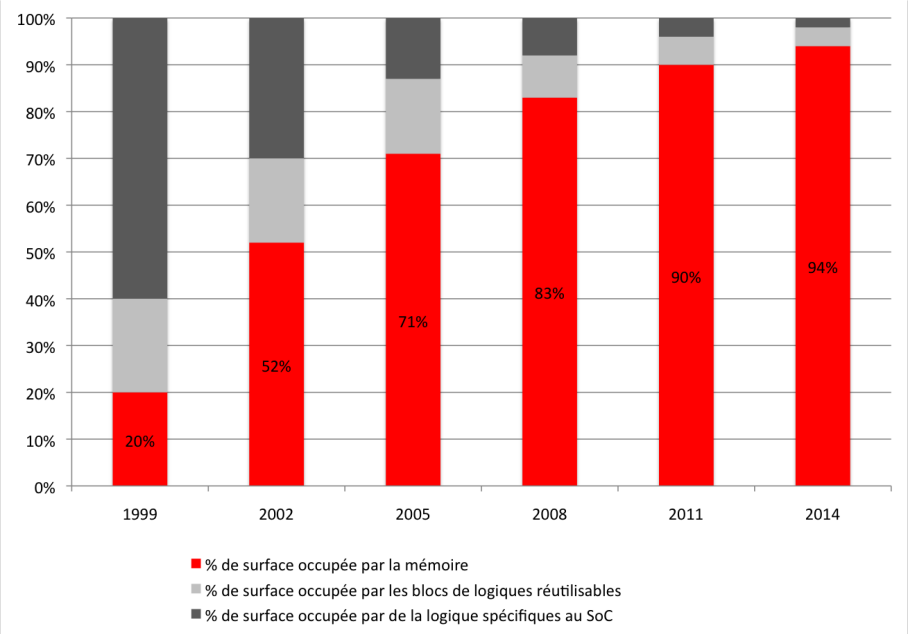


FIG. *1.1 : Part de la mémoire dans les circuits en fonction de la technologie (source ITRS). Cette tendance à embarquer de plus en plus de mémoire au sein des circuits est assez similaire pour la conception des processeurs. Après une évolution très importante de la complexité des cœurs de processeur, il semble que l'on soit passé dans une aire où cette complexité croit plus lentement. La densité d'intégration continuant à croître, la question qui se pose est alors de savoir que faire des centaines de millions de transistors supplémentaires intégrés dans les circuits. Cette question trouve deux réponses principales qui sont la multiplication des cœurs de traitement, qui permet d'accroître les performances sans complexifier le cœur de traitement et l'augmentation du volume de mémoire embarquée qui permet de réduire les temps d'accès aux données.*

Parallèlement à cette évolution, la contrainte énergétique est devenue de plus en plus critique et s'est reportée naturellement sur cette fonction de mémorisation. La conception de cette fonction doit alors proposer des solutions pour limiter au mieux l'énergie tout en proposant des performances importantes. Des solutions originales doivent être proposées, reposant sur des concepts différents des classiques hiérarchies mémoire à base de mémoires caches

n'offrant que peu de garantie en terme de performances.

Dans ce chapitre, nous présentons, un tour d'horizon des techniques de conception et d'optimisation mises en place pour les hiérarchies mémoire. Nous présentons ensuite deux propositions d'organisation mémoire. La première proposition concerne la mise en place d'une structure mémoire hiérarchique. La seconde étudie une structure mémoire distribuée au sein de l'ensemble d'un SoC. Dans cette proposition, l'espace mémoire est re-localisé au plus proche des ressources de calcul et une attention toute particulière est portée sur la génération des adresses pour des motifs de traitements particuliers. La seconde étude illustre le concept de reconfiguration appliqué à la structure mémoire. En complément des travaux réalisés autour des structures reconfigurables de calcul, il s'agit ici de montrer qu'il existe un potentiel d'optimisation énergétique important sur la fonctionnalité mémoire en appliquant ce principe.

La première proposition fait actuellement l'objet de travaux de recherche menés par Erwan Grace. Ces travaux sont réalisés en collaboration avec le CEA LIST et devraient eux aussi être soutenus dans le courant de l'année universitaire.

1.1 Techniques de conception et d'optimisation des hiérarchies mémoire

En considérant les différentes possibilités et architectures des éléments de mémorisation offertes par les composants d'un système sur puce (blocs de mémoires embarqués, mémoires distribuées dans les éléments de traitement comme les FPGA, mémoires caches des processeurs, etc), on observe que les choix proposés pour la réalisation d'une architecture mémoire sont très riches et variés. Par conséquent, la mise en place d'une hiérarchie de mémoire est un processus complexe, d'autant plus qu'elle dépend de manière étroite du type d'application traitée. Le choix des paramètres architecturaux relève de nombreux compromis effectués entre performances, consommation et flexibilité.

Le nombre de travaux ayant pour objectif de concevoir une architecture mémoire optimisée ou d'améliorer la gestion de la hiérarchie des mémoires est en rapport avec l'importance des enjeux. On peut distinguer trois grandes approches :

1. La première approche consiste à adapter le code de l'application aux contraintes et mécanismes d'accès d'une architecture cible prédéfinie et d'une hiérarchie mémoire fixe. Les techniques d'optimisation utilisées vont des transformations de codes [7] aux techniques d'ordonnancement des accès mémoire [8] et/ou de placement des données en mémoire [9]. L'architecture étant pré-fixée, cette approche peut alors être considérée comme logicielle. Elle regroupe les travaux concernant les mémoires caches ainsi que les travaux visant à effectuer des transformations de codes en vue de minimiser les accès aux données.

2. La seconde approche consiste à construire, pour une application donnée, une architecture mémoire dédiée. La hiérarchie mémoire est donc *taillée* sur mesure et le choix des paramètres s'effectue par rapport à des fonctions de coût s'appuyant sur des modèles caractérisant les performances, la surface et la consommation des mémoires en fonction de leurs paramètres architecturaux, algorithmiques et technologiques (librairie mémoire) [10, 11]. Cette approche peut être qualifiée de matérielle, elle est mieux adaptée aux applications dont les séquences d'accès sont connues *a priori*.

3. Enfin, la troisième approche combine les deux premières et ambitionne l'optimisation conjointe de l'architecture mémoire et du code de l'application. En effet, l'application et l'architecture sont fortement interdépendantes : une allocation optimale des données ne peut être faite sans prendre en compte l'architecture des unités mémoire cibles. De la même manière, le choix d'une architecture mémoire efficace ne peut être effectué sans prendre en compte les besoins spécifiques de l'application [12]. Il s'agit ici d'une démarche de conception conjointe où logiciel et matériel sont conçus en parfaite adéquation. Notons que cette démarche est globalement reconnue pour être complexe et difficilement automatisable.

1.1.1 Approches logicielles

Les travaux qui sont menés dans ce contexte concernent des phases d'optimisation. Il s'agit notamment soit d'optimiser le code de l'application pour limiter le nombre de transferts entre la mémoire et les éléments de calcul (par exemple le processeur), soit de modifier les structures de données pour augmenter les localités spatiales et temporelles qui sont nécessaires pour la mise en place d'une hiérarchie mémoire efficace. Une analyse des accès aux données est donc primordiale et celle-ci dépend fortement de la capacité à détecter les dépendances de données dans le code des applications. Différentes représentations ou abstractions des dépendances ont été proposées. Certains modèles permettent d'analyser finement la circulation des informations au cours du déroulement du programme [13]. Ainsi, il est par exemple possible de repérer les données non réutilisées, ou de décrire la manière dont les opérations utilisant les mêmes données seront regroupées dans le temps.
Du point de vue de la transformation de codes, l'analyse des boucles a été très largement abordée et il existe un nombre très important de transformations du code source portant sur les boucles. Des transformations comme la permutation, l'échange, la distribution et l'inversion des boucles sont souvent appliquées aussi bien pour la compilation parallèle que pour la synthèse de circuits. Ces techniques ont de bonnes propriétés vis-à-vis des tailles des mémoires, de la bande passante et par voie de conséquence sur la consommation énergétique globale de l'application [14].
Des environnements de conception comme MMAlpha [15] développé à Rennes, Atomium de l'IMEC [16] ou PiCo[1] de HP/Synfora [17] s'appuient sur ces techniques pour améliorer les localités spatiale et temporelle des programmes en ré-ordonnançant les accès mémoire [18, 19].
Il faut toutefois noter que ce problème de transformation de code a été démontré comme étant NP-complet, notamment dans [20]. Cette complexité justifie l'emploi d'heuristiques, d'algorithmes et de techniques de programmation, pour guider le choix des transformations à appliquer [21, 22].
Concernant les techniques de ré-organisation des structures de données, l'objectif concerne aussi l'augmentation des localités spatiale et temporelle des accès. Les regroupements de structures de type tableaux permettent notamment d'agir efficacement sur la localité spatiale [23]. De même, le placement en mémoire de données allouées dynamiquement [24] ainsi que les techniques d'alignement de données en mémoire peuvent augmenter cette caractéristique de l'application [25]. On citera encore les techniques de compression de données [26] et/ou de code [27] ainsi que les techniques d'optimisation de la largeur des données [28] qui visent à limiter l'empreinte mémoire occupée par l'application.
Finalement, la gestion logicielle des mémoires *scratchpad*[2] a montré un réel intérêt à la fois d'un point de vue temporel mais aussi d'un point de vue énergétique [29, 30]. L'intérêt pour les mémoires *scratchpad* s'est surtout fait ressentir dans le contexte des systèmes sur puce où les contraintes énergétiques sont souvent présentes [31].

1.1.2 Approches matérielles

L'objectif de ces approches est d'adapter la hiérarchie mémoire à une ou à nombre restreint d'applications, afin d'obtenir une architecture mémoire efficace permettant d'atteindre les meilleures performances, tout en respectant les contraintes imposées. La notion de performance ne se limite pas seulement aux aspects temporels mais englobe très souvent la notion de consommation énergétique. Les méthodes qui sont développées pour fournir une solution se basent en général sur un profil d'exécution de l'application considérée. Les solutions proposées s'orientent très souvent vers du partitionnement de la mémoire en plusieurs bancs (*sub-banking*) afin de répondre notamment aux accès simultanés nécessaires pour l'application [32, 33, 34].
Des travaux visant à étudier le partitionnement des caches ont aussi été réalisés [35]. Il s'agit entre autres de limiter la consommation du système mémoire global en essayant de limiter au maximum les transferts entre la mémoire centrale et les niveaux de hiérarchies les plus rapides. Les configurations, au sens large, des caches peuvent aussi avoir un impact non négligeable sur les performances des applications [36]. Dans ce cadre, notons l'utilisation de caches des victimes qui a un impact très favorable sur la consommation énergétique du système mémoire [37].
La prise en compte des contraintes liées au stockage des données a aussi été abordée dans les flots de synthèse de

[1] PiCo : Program In Chip Out
[2] SP : ScratchPad, mémoire directement *mappée* dans l'espace adressable.

haut niveau. Les opérations d'accès à la mémoire sont modélisées dans le graphe flot de données [38] et l'ordonnancement tient alors compte de ces opérations particulières [39]. Ce sont principalement le nombre de conflits d'accès ainsi que l'espace de stockage nécessaire qui sont utilisés comme métriques [40]. Lorsque la contrainte énergétique devient prépondérante, il est alors possible de travailler sur l'activité des mémoires en concentrant le maximum de transferts dans un intervalle précis, laissant apparaître alors des intervalles sans activité durant lesquels la ou les mémoires peuvent être placées en mode veille (voire même en mode *off*) [41].
Notons enfin que l'ordre de la synthèse des différentes unités du système peut lui aussi être remis en cause. C'est le cas des travaux présentés dans [42] où l'auteur propose un ordonnancement des accès mémoire spécifique afin de limiter les conflits d'accès, puis impose ces contraintes à la synthèse des autres unités.
La construction de l'unité de mémorisation passe aussi par la définition des générateurs d'adresses. De nombreuses études se sont intéressées à cette problématique. Des structures plus ou moins complexes sont proposées, parfois associées aux structures de données manipulées (tableaux par exemple) [A1, 43].
Finalement, la conception du système de stockage dans le contexte des SoC est maintenant assez largement explorée. On retrouve alors des problématiques relatives aux multi-processeurs et notamment des problèmes de congestion d'accès à la ressource mémoire lorsque celle-ci est sollicitée par plusieurs requêtes. Globalement, ces problématiques sont traitées par des approches statiques associées à des phases de compilation, ou par des approches dynamiques par le biais d'allocations mémoire au cours du temps.

1.1.3 Approches conjointes matérielle/logicielle

Les approches conjointes semblent être des solutions intéressantes pour parvenir à produire à la fois un code limitant les besoins en stockage (nombre de données à stocker et nombre d'accès à assurer) et une architecture mémoire performante. L'approche la plus connue est sans aucun doute celle développée par l'IMEC. Cette approche, DTSE [44] traite du problème de la mémorisation en un certain nombre d'étapes, 6 au total, qui englobent les transformations de code, la définition de la structure hiérarchique mémoire ainsi que le placement des données dans cette structure. La méthodologie développée a été partiellement outillée dans l'outil Atomium. Les nombreuses étapes de la méthodologie DTSE augmentent la complexité de calcul des bornes des boucles, des index de tableaux et des conditions utilisées pour le contrôle, ce qui n'est pas sans conséquence sur la complexité des générateurs d'adresses. Une méthodologie séparée, appelée ADOPT, a été spécialement développée pour simplifier les expressions des calculs d'adresses et d'indexation des tableaux, en utilisant également des transformations de source à source de programmes et/ou en générant un circuit spécifique chargé de la gestion des adresses.
Enfin, l'architecture mémoire et la spécification exécutable sont alors définies pour une synthèse *hardware* et/ou une compilation logicielle.
Notons l'existence de solutions spécifiques basées sur la mise en place de mémoires intelligentes. Il s'agit de placer dans ces blocs mémoire de la logique de traitement afin de distribuer une partie des traitements dans la mémoire elle-même [45, 46]. Cette approche permet de réduire de façon importante la bande passante nécessaire et a un impact important sur les performances mais aussi sur la consommation énergétique du système. On peut citer des travaux plus orientés vers le traitement des vecteurs [47] avec une adaptation des algorithmes en général nécessaires pour profiter pleinement de l'architecture mémoire.

1.1.4 Bilan des différentes approches

Les travaux consacrés aux problèmes de mise en œuvre d'une architecture mémoire optimisée ont suscité l'intérêt de plusieurs communautés de recherche (architecture, compilation, parallélisme, calcul haute performance, etc). Cette problématique prend toute sa dimension dans le contexte des systèmes sur puce pour applications à fort volume de données (applications multimédia et de télécommunication). En effet, pour ces systèmes, les contraintes de conception sont aussi diverses (vitesse, consommation, surface), que variables dans leur criticité. Or la mémoire est l'une des parties les plus critiques et les performances globales de l'application sont fortement dépendantes des performances de cette unité.
Le traitement de l'ensemble des problèmes relatifs aux mémoires nécessitera sûrement encore beaucoup d'efforts. Le processus de conception d'une hiérarchie mémoire optimisée est très complexe et ne pourra se faire qu'avec

une bonne méthodologie de conception, associée à des outils d'exploration et de synthèse travaillant à tous les niveaux de conception.
Pour tenter de répondre à ces différentes problématiques, nous avons entamé des travaux afin de définir des solutions architecturales et méthodologiques pour la conception de hiérarchies mémoire au sein des SoC. Nous présentons ces solutions dans les sections suivantes.

1.2 Vers la définition d'une architecture mémoire distribuée

L'étude précédente s'est intéressée à la mise en place d'une solution mémoire centralisée au sein d'un SoC. Cette solution, si elle peut avoir un intérêt dans le cas où le nombre d'unités de calcul est faible, risque de devenir très rapidement inefficace dans le cas où ce nombre est important, de l'ordre de plusieurs dizaines. Un autre modèle de stockage doit alors être envisagé afin d'éviter les risques d'engorgement liés à la centralisation. Ce point est d'ailleurs de plus en plus critique compte tenu de l'évolution des systèmes actuels vers des systèmes dits *multi-processors on chip* ou *multi-cores* (la figure 1.2 recense quelques systèmes répondant à ce modèle). À titre d'exemple, on peut citer les projets de développements de systèmes *multi-processors* de la société Tilera et le projet Polaris d'Intel ayant permis le développement du processeur Tera-scale. Ces projets illustrent bien la tendance générale consistant à développer et/ou à étudier des architectures contenant plusieurs dizaines de cœurs de traitement. Dans ces projets, les solutions de stockage mises en œuvre s'articulent autour de hiérarchies mémoire relativement classiques : cache(s) de niveau 1 pour les données et pour les instructions, puis cache de niveau 2 pour partager des données entre *cores*, et éventuellement cache de niveau 3 pour augmenter les performances globales du système.
Dans ce contexte où l'augmentation de performances passe par la multiplication des cœurs de traitement, nous étudions actuellement une organisation mémoire pour une architecture fortement parallèle. L'objectif est de proposer une solution qui soit en mesure de répondre aux requêtes des différents éléments de traitement tout en limitant la consommation énergétique.

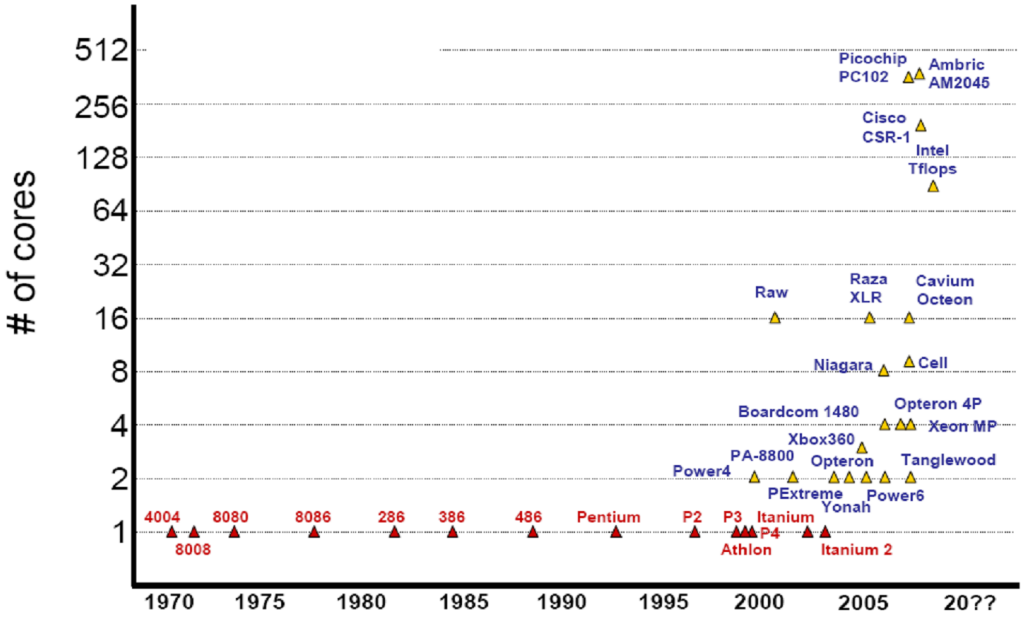


FIG. *1.2 : Evolution des processeurs vers des machines multi-cores [48], source ITRS 2009. L'évolution actuelle s'appuie sur plus de parallélisme. Les systèmes sont construits sur la base de nombreux éléments de traitement, et la notion de systèmes* multi-cores *est de plus en plus présente, à la fois dans les systèmes hautes performances mais aussi, et de plus en plus, dans les processeurs classiques.*

1.2.1 Architecture proposée

En ce qui concerne les systèmes hautes performances, nous avons entamé des travaux visant à étudier une architecture distribuée à la fois pour les traitements, mais aussi pour le stockage. Ces travaux font l'objet d'une thèse dont la première partie a consisté à définir l'architecture générale du système. Globalement, cette architecture est organisée autour de *tuiles* de calculs reconfigurables. L'aspect reconfiguration s'applique ici à la fois aux unités de calcul à proprement parler, et à la partie mémorisation.
Compte tenu de notre expérience des concepts de reconfiguration pour les chemins de données, nous avons bâti une architecture reprenant ce modèle de calcul. Aussi, une tuile de calcul est composée de quatre chemins de données reconfigurables s'appuyant sur le modèle de DART [49]. Chaque *processing unit* est un chemin de données à gros grains, reconfigurable dynamiquement. L'organisation mémoire s'appuie alors sur cette structure et propose un stockage de données local, avec des générateurs d'adresses permettant des accès réguliers et irréguliers, et un stockage global permettant des échanges entre les chemins de données reconfigurables. La figure 1.3 présente l'architecture dans sa globalité et donne un aperçu du contenu d'une tuile de traitement.

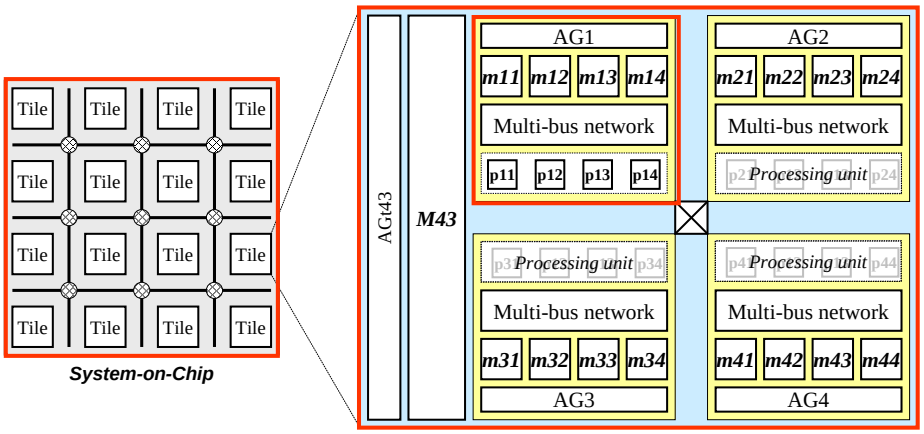


FIG. *1.3 : L'architecture globale est un système* multi-cores *(tuiles). Chaque* core *est composé d'unités de calculs reconfigurables répondant au modèle de l'architecture DART (processing unit). Pour chaque* core *de traitement, une organisation mémoire est mise en place (mémoires* `mij`*) et des générateurs d'adresses sont associés (*`AGi`*) afin de produire les séquences d'adresses régulières ou irrégulières. Une mémoire de tuile (*`Mnm`*) permet de stocker des données partagées entre plusieurs* processing units.

1.2.2 Générateurs d'adresses

Si la structure sous forme de tuile est relativement classique, notre architecture se distingue principalement par la génération des séquences d'adresses. Nous avons, en effet, concentré nos premiers efforts sur la définition des générateurs d'adresses en vue de proposer un support d'exécution complet pour des applications comportant des accès mémoire réguliers et irréguliers. Les accès réguliers que nous souhaitons supporter sont relatifs à des parcours de vecteurs, de tableaux, d'images. On citera, par exemple, les séquences d'adresses en lignes, en colonnes ou encore en *zig zag*. Les séquences irrégulières concernent des accès dont les adresses dépendent des données et où il n'est pas possible de prévoir la séquence, c'est le cas par exemple des calculs d'histogrammes. À partir de ces besoins, nous avons défini une structure de générateurs d'adresses composée d'une partie permettant de générer des accès réguliers, à faible coût et à haute fréquence, et d'une autre partie permettant de générer des accès irréguliers. La figure 1.4 présente l'architecture de ces générateurs d'adresses. Ils sont composés d'un cœur de processeur, et d'un assemblage de modules dits *stepper*. Le cœur de processeur assure la génération des adresses

irrégulières d'une part, et a en charge la configuration des *steppers* et des multiplexeurs assurant les séquences d'adresses régulières d'autre part.

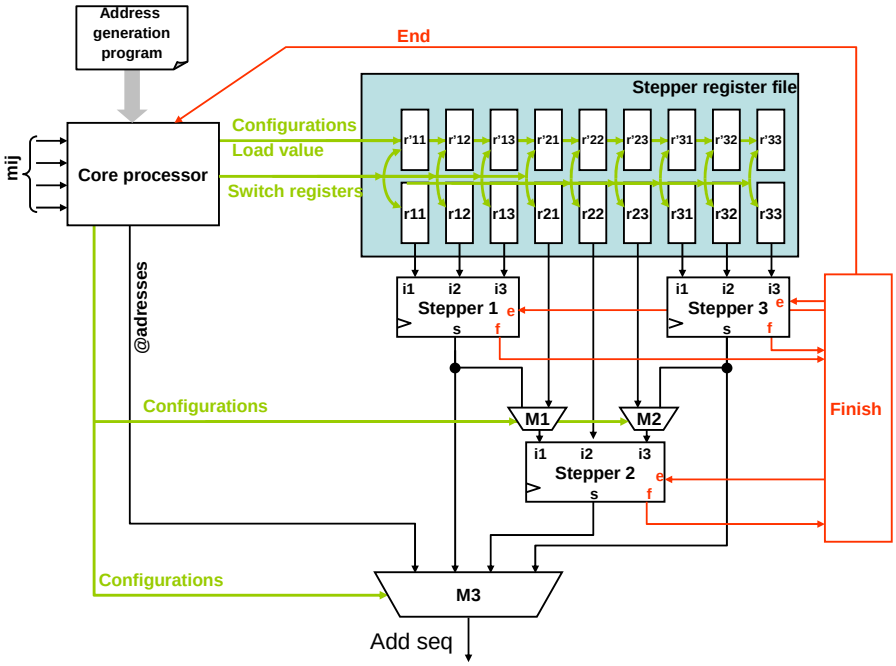


FIG. *1.4 : Les générateurs d'adresses sont composés de deux éléments dont les objectifs concernent la génération de séquences d'accès soit régulières soit irrégulières. Le* core *de processeur prend en charge la génération de séquences irrégulières. La structure à base de* steppers *assure la génération de séquences régulières. La mise en cascade des* steppers *permet de générer des séquences plus ou moins complexes.*

Génération de séquences d'adresses irrégulières

Pour produire les séquences d'adresses irrégulières, c'est-à-dire dépendantes des données, nous proposons d'utiliser un cœur de processeur. En dotant ce processeur de la possibilité de lire (et éventuellement d'écrire) le contenu des mémoires de données, celui-ci est alors en mesure de produire toute séquence non prédictible, dont les adresses sont calculées par des traitements. Ces traitements font appel à des opérations arithmétiques relativement simples du type *addition* d'une adresse de base et d'un décalage. Le cœur de traitement que nous avons défini dispose du jeu d'instructions présenté dans le listing 1.1.

Dans ce jeu d'instructions, RA_i un registre d'adresses interne au processeur, et m_{jk} la mémoire locale du chemin de données associé. Ces opérations sont assurées par le cœur de processeur défini à la figure 1.5.

Ce jeu d'instructions dispose aussi d'un mécanisme permettant de répéter une instruction durant plusieurs cycles. Pour cela, chaque instruction peut être précédée du mnémonique `REP(nbcycles)`. Par exemple, l'instruction `OUTADD` peut être répétée durant 16 cycles en écrivant :

`LOAD`	RA_i, <valeur>	chargement d'un registre d'adresses avec une valeur immédiate
`ADD`	RA_i, RA_j, RA_k	addition du contenu de deux registres, résultat en registre
`ADD`	RA_i, RA_j, m_{jk}	addition du contenu d'un registre et d'une donnée, résultat en registre
`SUB`	RA_i, RA_j, RA_k	soustraction du contenu de deux registres, résultat en registre
`SUB`	RA_i, RA_j, m_{jk}	soustraction du contenu d'un registre et d'une donnée, résultat en registre
`OUTADD`	RA_i, RA_j	addition du contenu de deux registres, résultat sur le bus d'adresses
`OUTADD`	RA_i, m_{jk}	addition du contenu d'un registre et d'une donnée, résultat sur le bus d'adresses
`OUTSUB`	RA_i, RA_j	soustraction du contenu de deux registres, résultat sur le bus d'adresses
`OUTSUB`	RA_i, m_{jk}	soustraction du contenu d'un registre et d'une donnée, résultat sur le bus d'adresses
`REVERSE`	RA_i, RA_j	bit reverse d'un registre, résultat en registre
`OUTREVERSE`	RA_i	bit reverse d'un registre, résultat sur le bus d'adresses
`LOAD`	RA_i, m_{jk}	chargement d'une valeur dans un registre d'adresses interne au processeur

Listing 1.1 : Jeu d'instructions du cœur de processeur du générateur d'adresses. A partir de ces instructions, le processeur est capable de produire toute séquence d'adresses. Le mécanisme proposé ici permet de préparer la prochaine séquence à générer en parallèle avec la production de la séquence courante.

`REP(16) OUTADD`	RA_i, m_{jk}	16 répétitions de l'addition du contenu d'un registre et d'une donnée, résultat sur le bus d'adresses

Listing 1.2 : Exemple de codage permettant une répétition d'une instruction. Ce codage permet de limiter les phases de chargement, décodage d'instruction et exécution d'instruction par le cœur de processeur. Ce mécanisme permet alors de limiter le surcoût énergétique du générateur d'adresses.

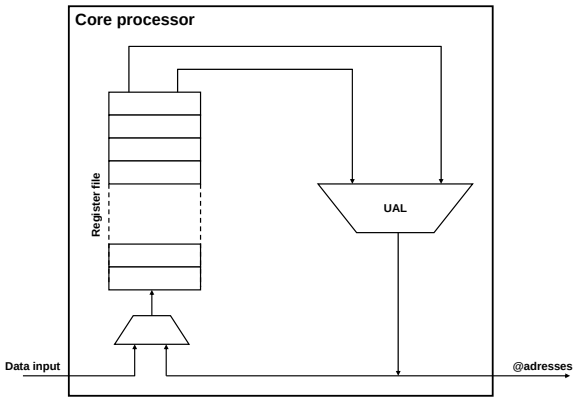


FIG. *1.5 : Le cœur de traitement du processeur est constitué d'une file de registres et d'une unité arithmétique et logique. Des opérations simples peuvent être réalisées via l'UAL. Les résultats de ces opérations peuvent soit être stockés dans un registre, soit envoyés vers le bus d'adresses afin d'accéder à la mémoire.*

Génération de séquences d'adresses régulières

Pour les séquences régulières, la configuration consiste principalement à fixer les entrées i_1, i_2 et i_3 des *steppers* (correspondant respectivement aux fonctions suivantes : $i_1 = @base$, $i2 = step$, $i_3 = @limit$) et à piloter les

multiplexeurs (M_1, M_2 et M_3) pour éventuellement cascader les *steppers*. Les *steppers* disposent d'une horloge et des entrées *enable* et *reset*. La fonction assurée par un *stepper* peut être décrite par l'algorithme 1.6.

```
Faire
        Si reset'event alors
                adresse = @base;
        Fsi ;
        Si clock'event alors
                adresse = adresse + step;
        Fsi ;
Tant Que (adresse ≤ limit) ;
end ='1';
```

Listing 1.3 : Description algorithmique du comportement d'un stepper

Sur la base de ce fonctionnement et par assemblage en cascade via les multiplexeurs, il est alors possible de configurer cette partie du générateur d'adresses pour générer des séquences relativement complexes. Par exemple, la figure 1.6 présente les deux configurations à appliquer aux *steppers* afin de générer la séquence d'adresses permettant d'accéder à une image en mode *zigzag*.

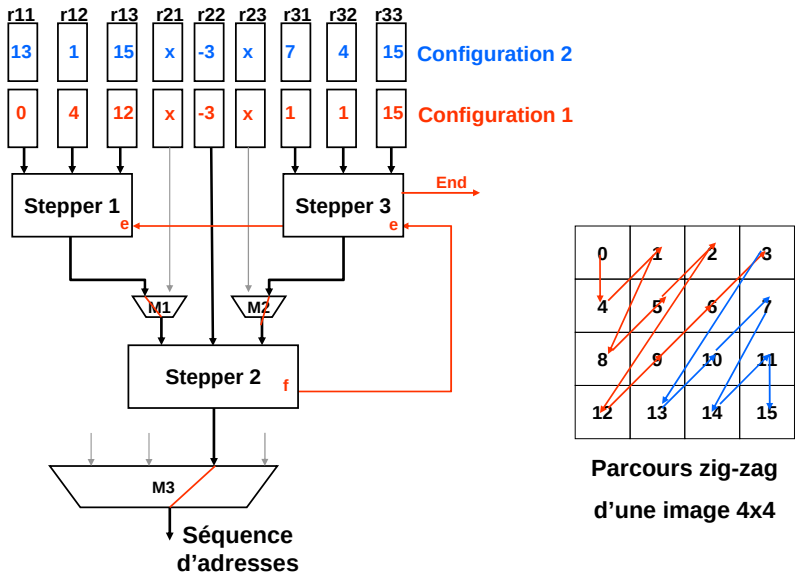


FIG. *1.6 : Exemple de configurations des* steppers *afin de générer une séquence d'adresses de type* zig zag *pour une image. Dans ce cas, la séquence d'adresses* $0, 4, 1, 8, 5, 2, 12, 9, 6, 3$ *est générée par la première configuration des registres des* steppers*, alors que la séquence d'adresses* $13, 10, 7, 14, 11, 15$ *est générée par la seconde configuration. Les* steppers 1 *et* 3 *permettent de générer les bornes minimum et maximum du* stepper 2 *qui se charge alors de générer les adresses en remontant sur la diagonale.*

Le dimensionnement des *steppers* ainsi que des registres dépend étroitement des tailles des structures de données. Les *steppers* doivent être dimensionnés pour le cas maximum, ce qui se traduit pour les registres r_{x1} et r_{x3} par des tailles permettant d'adresser la totalité de la mémoire locale.
En ce qui concerne les entrées des registres r_{x2} correspondant au pas d'incrémentation, elles peuvent être de tailles plus faibles, une taille de 8 bits semble raisonnable et suffisante.
Globalement, pour assurer la configuration de la partie produisant les adresses régulières, le cœur de processeur

doit être doté des instructions suivantes :

`STORE`	r_{xy}, <valeur>	configuration du registre spécifique xy
`STORE`	$r_{.y}$, <valeur>	configuration du registre y de tous les *steppers*
`STORE`	$r_{x.}$, <valeur>	configuration de tous les registres du *stepper* x
`CFGmux`	<valeur>	configuration des multiplexeurs (valeur sur 4 bits)
`CFGend`	<valeur>	configuration du bloc de détection de fin de séquence
`WAITFOR`	<nbcycles>	suspend l'exécution durant un certain nombre de cycles, permet de produire une séquence d'adresses régulière pendant nbcycles

Listing 1.4 : Complément du jeu d'instructions du cœur de processeur du générateur d'adresses. Ces instructions permettent de charger les registres des steppers *afin de définir une séquence régulière d'adresses. Une fois le chargement effectué, le processeur peut switcher les registres puis laisser le générateur d'adresses exécuter sa séquence.*

La première instruction permet de stocker une valeur particulière dans n'importe lequel des registres d'entrée des *steppers*. La seconde instruction permet de stocker la même valeur dans le registre y de tous les *steppers*. La troisième instruction permet de configurer tous les registres d'un *stepper* avec la même valeur. La quatrième instruction permet de positionner les multiplexeurs pour la génération de séquences d'adresses complexes. La cinquième instruction permet de définir les conditions d'arrêt des *steppers*. Le bloc *finish* de la figure 1.4 permet de réaliser une combinaison logique des entrées et de produire les activations des *steppers*.
Finalement, pour permettre un fonctionnement le plus fluide possible, nous avons mis en place un mode de configuration transparent vis-à-vis de la génération des adresses. Cette transparence est assurée par la présence de deux jeux de registres de configurations des *steppers*. Dans l'exemple présenté à la figure 1.6, la première configuration est maintenue durant 10 cycles. Durant ces 10 cycles, le processeur prépare la configuration suivante dans le second jeu de registres. Il faut au processeur 9 cycles pour configurer les 9 registres des *steppers*. Le basculement d'un jeu de registres à l'autre s'effectue par l'instruction suivante :

`SWr`	basculement de registres configuration/entrées *steppers*

Listing 1.5 : Instruction permettant de switcher *d'un jeu de registres à l'autre en vue de la production de la nouvelle séquence d'adresses.*

Exemple de générateurs d'adresses

Pour l'exemple du calcul d'histogramme d'une image, en supposant que l'image originale soit contenue dans la mémoire locale m_{11} et que l'histogramme à calculer soit contenu dans la mémoire m_{12}, alors il est possible de synchroniser les deux générateurs g_{11} et g_{12} afin qu'ils assurent le calcul de cet histogramme. La figure 1.7 présente la structure de la génération d'adresses à mettre en place pour assurer l'accès aux occurrences de chaque valeur de pixels.

Synchronisation des générateurs d'adresses

Comme nous venons de le voir dans la section précédente, les générateurs d'adresses doivent pouvoir être synchronisés. Compte tenu de la structure d'une tuile de traitement, constituée de quatre chemins de données reconfigurables, il semble raisonnable de définir une synchronisation locale matérielle calquée sur la synchronisation des unités de traitement. La solution envisagée actuellement consiste à reporter la synchronisation des générateurs d'adresses sur un contrôleur global dont le rôle consistera à envoyer les signaux de *start* afin de déclencher les

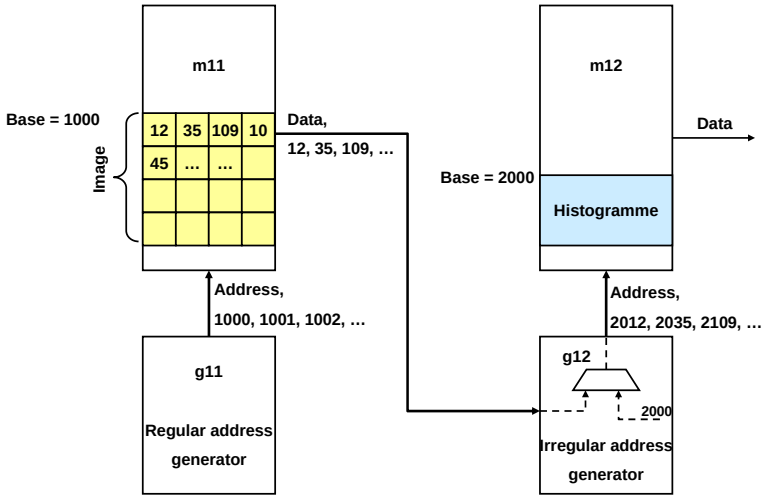


FIG. *1.7 : Illustration de la génération des adresses pour un calcul d'histogramme d'image. Les valeurs des pixels de l'image sont extraites successivement de la mémoire m_{11} et utilisées pour construire l'adresse de l'occurrence de la valeur en question dans l'histogramme. Le générateur g_{11} est configuré afin de produire une séquence d'adresses régulière, alors que le second générateur g_{12} est configuré afin de produire une séquence dépendante de la valeur lue.*

exécutions des programmes de génération des adresses. Cette solution permet d'envisager des fonctionnements complètement indépendants de chaque générateur d'adresses, et offre donc une souplesse intéressante. Toutefois, il faut noter qu'il est fort probable que l'instanciation des traitements sur un chemin de données reconfigurable ne mette pas en jeu des traitements (*threads*) indépendants. En effet, si tel était le cas, alors la prédictibilité des temps de calcul serait impossible. Dans ce cas, il semble que les générateurs d'adresses puissent bénéficier d'un mécanisme de synchronisation commune. Pour factoriser le contrôle des générateurs d'adresses nous proposons d'utiliser une structure de contrôle VLIW[3]. La structure est bâtie sur un ensemble de quatre instructions simples (instructions définies précédemment) permettant d'adresser les quatre mémoires m_{ij}. Dans le cas du calcul de l'histogramme sur une image, le programme de génération d'adresses qui doit être écrit est alors le suivant :

```
|| STORE r11, 1000          || -                          || -                          || -
|| STORE r12, 1              || -                          || -                          || -
|| STORE r13, 1015           || -                          || -                          || -
|| CFGmux B0001              || LOAD RA0, 2000             || -                          || -
|| WAITFOR 16                || REP(16) OUTADD RA0, m11    || -                          || -
```

Listing 1.6 : Exemple de programme de génération des adresses pour le calcul d'un histogramme. La structure VLIW permet de configurer les deux générateurs d'adresses nécessaires en parallèle et permet aussi d'assurer la synchronisation. L'inconvénient majeur de ce codage concerne la taille du code qui contient de nombreuses instructions `NOP` *notées ici par un tiret.*

Dans l'exemple ci-dessus, seuls deux générateurs d'adresses sur quatre sont utilisés pour produire l'histogramme. Dans ce cas, le code VLIW de génération des adresses est coûteux compte tenu de la nécessité de spécifier les instructions de chaque générateur. Pour limiter ce coût, une technique visant à rendre les instructions VLIW à taille variable peut être appliquée. Dans ce cas, un bloc de *dispatching* des instructions vers les bons générateurs d'adresses est nécessaire et permet de compresser de façon importante le programme de génération des adresses.

[3]VLIW : Very Long Instruction Word.

1.2.3 Travaux en cours sur cette étude

Dans l'état actuel de nos travaux, l'architecture complète a été proposée et une description VHDL de l'ensemble a été faite. Une phase de validation est actuellement en cours de réalisation, avec notamment pour objectif d'extraire les caractéristiques de fonctionnement de l'architecture. La phase de validation envisagée consiste en une chaîne de traitements nécessitant différentes phases traitements et des échanges de données important entre chaque phase.

1.3 Application du concept de reconfiguration à la structure mémoire

Dans la littérature, de nombreuses techniques, aussi bien algorithmiques que matérielles, ont été proposées pour optimiser le système de mémorisation des architectures embarquées. Parmi ces dernières, l'introduction du principe de reconfiguration matérielle au sein d'une hiérarchie mémoire semble être une piste intéressante. Ce principe de reconfiguration, consistant à modifier au cours de l'exécution la structure et l'agencement des ressources d'une architecture [50], a déjà été largement étudié dans le cadre de l'optimisation des structures de calcul [51, 52], voire de contrôle [53]. Récemment, certains travaux ont donc conduit à l'exploration de ce type de solutions pour l'optimisation de la hiérarchie mémoire. L'idée sous-jacente est que différents algorithmes requièrent différents types de transferts de données (de processeur à processeur, de processeur à mémoire et inversement, de mémoire à mémoire), d'accès mémoire (réguliers tels que l'accès aux éléments successifs d'un tableau, ou irréguliers) et des capacités de stockage variables. Il est donc indispensable pour disposer d'une mémoire performante d'adapter à la fois son architecture interne et son interconnexion avec les autres acteurs du système, à ces différents cas de figure. Dans cette section, nous présentons le concept de reconfiguration appliqué à la fonction de mémorisation. Nous introduisons le concept de hiérarchie mémoire "virtuelle" et nous montrons que ce type de structure peut être une partie de la réponse à l'augmentation de la consommation d'énergie.

1.3.1 Modélisation de structure mémoire reconfigurable

Le concept de reconfiguration que nous avons étudié se concentre sur l'axe de la minimisation de la consommation. Notre proposition s'articule autour d'une structure de mémorisation reconfigurable au sein de laquelle une "hiérarchie virtuelle" est définie. Nous avons proposé un modèle dans lequel les bancs mémoire peuvent être "déplacés" au sein de la "hiérarchie virtuelle" au gré des besoins de l'application. Nous appelons "hiérarchie virtuelle" une structure mémoire constituée de plusieurs bancs mémoire dont il est possible de contrôler la tension d'alimentation de chacun des bancs de façon indépendante. Il est reconnu que les changements de tension d'alimentation ont un impact sur les temps d'accès ainsi que sur la consommation, notamment statique. Ainsi, un abaissement de la tension d'alimentation d'un banc mémoire, allongera son temps d'accès en lecture et en écriture, mais diminuera de façon importante sa consommation. Nous avons montré dans [A8] que ce modèle associé à un contrôle approprié permet d'envisager un gain en consommation important.

Typiquement, les différents niveaux de mémorisation dans une hiérarchie mémoire se distinguent par leur latence d'accès, leur taille respective, mais aussi par le coût énergétique des accès mémoire. Ainsi, une mémoire est associée à un niveau déterminé de la hiérarchie mémoire en fonction de sa capacité de stockage, de son temps d'accès et donc de sa consommation. Les paramètres étant fixés lors de la fabrication du circuit, la mémoire restera associée à un même niveau de hiérarchie mémoire tout au long du cycle de fonctionnement du composant.
Compte tenu de l'impact de la tension d'alimentation et de la fréquence de fonctionnement des composants sur leur consommation, de nombreux efforts ont été fournis pour autoriser un ajustement dynamique des paramètres en fonction des besoins applicatifs. Ainsi, il devient désormais possible de modifier, en quelques cycles de fonctionnement, les performances et l'efficacité énergétique d'une architecture.

Dans ce contexte, nous supposons que chaque banc mémoire dispose de plusieurs points de fonctionnement pouvant être sélectionnés indépendamment. Ces points de fonctionnement varient d'un fonctionnement sous tension

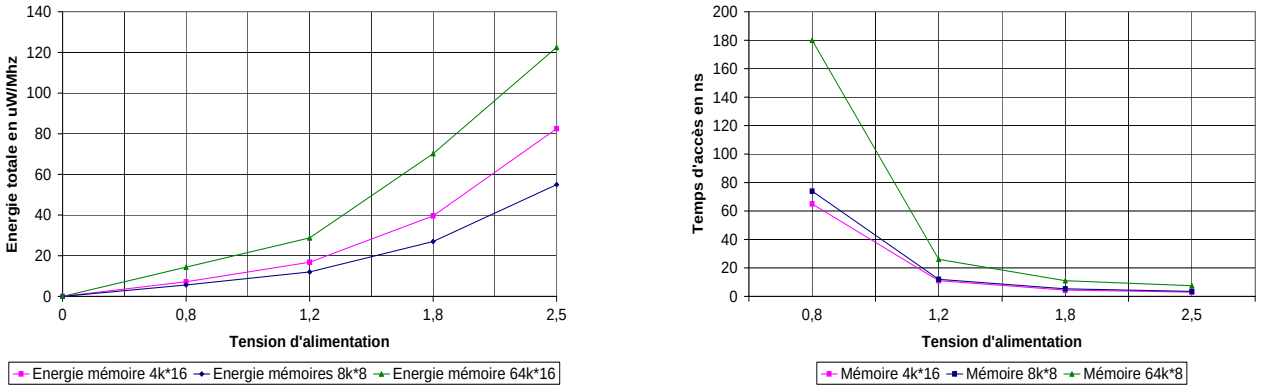


FIG. *1.8 : Évolution des temps d'accès et de l'énergie totale de mémoires SRAM en fonction de la tension d'alimentation pour différentes mémoires en technologie* $0,25\mu m$ *[54] (consommation des cellules et de l'ensemble du dispositif de décodage et d'amplification des sorties). En dépit d'une technologie ancienne, on peut observer que l'effet de la tension d'alimentation sur le temps d'accès et la puissance sont importants. En ce qui concerne la tension d'alimentation, le passage sous le seuil de 1,2 volt conduit à un ralentissement de la mémoire important. En dessous de la tension de 0,8 volt, le temps d'accès n'est pas significatif puisque sous cette tension la mémoire ne fournit plus la fonctionnalité attendue. La tension de 1,2 volt est alors considérée comme étant la tension minimale de rétention de l'information. En ce qui concerne l'énergie consommée par la mémoire, celle-ci augmente avec la tension d'alimentation.*

nominale à un fonctionnement en mode *off* correspondant à la suppression de la tension d'alimentation. À ces différents modes d'alimentation, nous associons des caractéristiques de temps de cycle et de consommation, obtenues par le biais de courbes caractéristiques. Un exemple de caractéristiques est donné sur les courbes de la figure 1.8. Ces courbes sont données pour une tension nominale de $2,5V$. La technologie $0,25\mu m$ mise en œuvre ici n'engendre qu'un faible courant de fuite (*leakage*). Par contre pour une technologie $90nm$, le *leakage* aura une influence beaucoup plus importante sur la consommation, en particulier sur la consommation statique. En effet, l'augmentation des courants de fuite dans les technologies récentes a rééquilibré la répartition des consommations dynamique et statique (on estime qu'environ 50% de la consommation est (ou sera) due à ces courants de fuite). De plus, on peut noter une variation importante du temps d'accès pour des tensions inférieures à $1,3V$.

1.3.2 Modes de fonctionnement de la structure mémoire reconfigurable

À partir de cette bibliothèque de mémoires enrichies de ces modes de fonctionnement, nous avons proposé la construction d'une structure mémoire dans laquelle il est possible de "déplacer" virtuellement les mémoires par adaptation de leur tension d'alimentation. En faisant varier cette tension, il devient en effet possible de faire varier dynamiquement les paramètres de consommation et de temps d'accès, et donc de modifier dynamiquement le "positionnement virtuel" d'une mémoire dans la hiérarchie.
La consommation dans une hiérarchie mémoire classique est en grande partie engendrée par les transferts de blocs de données entre les différents niveaux de la hiérarchie mémoire. Par la virtualisation de la hiérarchie mémoire, ces transferts de données entre différents niveaux de la "hiérarchie" sont limités, de même que le "gaspillage" d'énergie inhérent à ces transferts. Ainsi, cette structure permet de ne conserver que la consommation réellement nécessaire aux transferts des données vers les ressources de calcul, et donc d'optimiser le coût de l'accès aux données dans les systèmes embarqués.
Soit une hiérarchie à deux niveaux (voir schéma de la figure 1.9.a) entre lesquels il existe les transferts suivants :
– transferts de $D_{n+1,n}$ données entre les niveaux $n+1$ et n ;

– transferts de $D_{n,r}$ données depuis le niveau n vers les ressources de calcul ;
l'énergie globale pour un système classique Eg_c engendrée par ces transferts est donnée par :

$$\begin{aligned} Eg_c &= Ed_c + Es_c \\ &= D_{n,n+1} \times (Ed_n + Ed_{n+1}) + D_{n,r} \times Ed_n + T_{algo} \times (Ps_n + Ps_{n+1}) \end{aligned} \tag{1.1}$$

avec Ed et Es représentant respectivement les énergies dynamique et statique, Ed_i l'énergie dynamique par accès à une donnée du niveau i et Es_i l'énergie statique dissipée au niveau i lors des transferts. L'énergie Es_i est obtenue à partir de la puissance statique Ps_i de la mémoire de niveau i, et du temps d'exécution global de l'algorithme T_{algo}. On suppose, dans cette configuration, que l'ensemble des données est stocké dans la mémoire de niveau $n+1$ (le niveau 2 sur la figure 1.9.a).

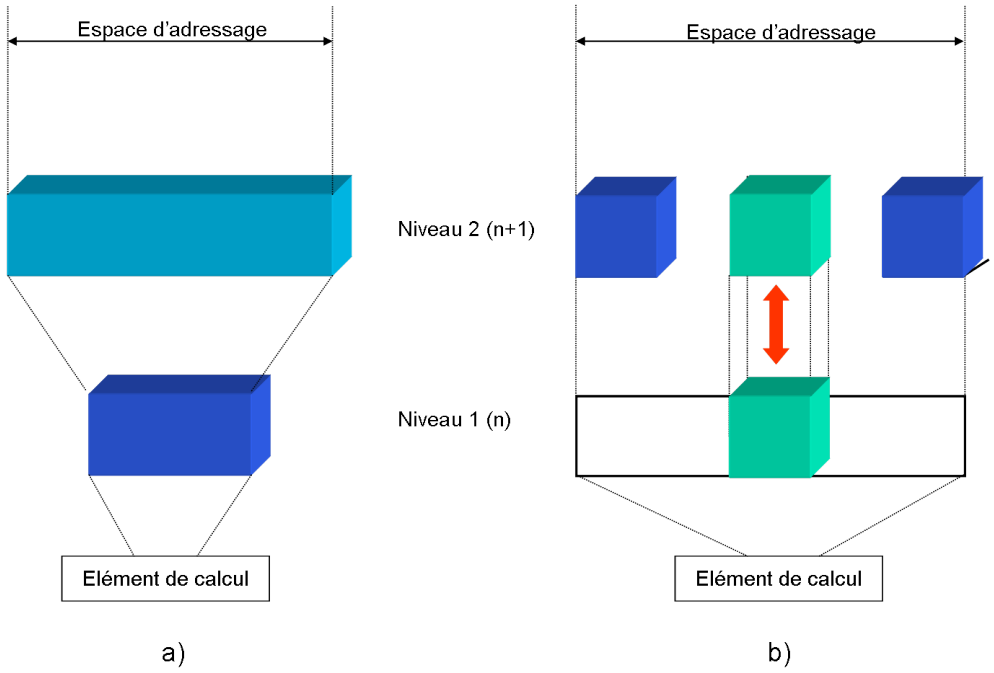


FIG. *1.9 : Exemple d'organisation mémoire ; a) hiérarchie classique ; b) hiérarchie virtuelle reconfigurable. Dans le cas classique, les données transitent du niveau le plus haut (le plus éloigné des ressources de calcul) vers le niveau le plus bas, et le plus rapide. Les ressources accèdent alors aux données avec un temps d'accès très faible. Dans le cas d'une hiérarchie virtuelle, les données ne transitent pas d'un niveau vers un autre, mais le changement de tension d'alimentation permet de modifier le temps d'accès. Une réduction de la tension d'alimentation permet de ralentir la mémoire, et donc de la "repousser" virtuellement dans un niveau plus éloigné. Ce ralentissement est réalisé au bénéfice d'une réduction de la consommation énergétique importante. Inversement, pour accéder à une donnée rapidement, la tension d'alimentation de la mémoire concernée doit être augmentée, au prix d'une augmentation de la consommation énergétique.*

Dans un système dans lequel les données ne migreraient plus de niveau en niveau (voir schéma de la figure 1.9.b) mais pour lequel il serait possible de faire varier la tension d'alimentation et donc les caractéristiques de temps de cycle et de consommation, l'énergie globale Eg_h devient :

$$\begin{aligned} Eg_h &= Ed_h + Es_h \\ &= D_{n,r} \times Ed_n + Ps_n \times \delta T_n + Ps_{n+1} \times [T_{algo} - \delta T_n] \end{aligned} \tag{1.2}$$

avec δT_n le temps pendant lequel la mémoire contenant les données accédées est placée au niveau n et T_{algo} le temps d'exécution global de l'algorithme. Dans cette configuration, les données sont supposées distribuées de façon adéquate au sein des différents bancs mémoire de la structure.
Dans le cas où l'algorithme est exécuté sans interruption à la cadence T_{algo}, la puissance moyenne consommée est donnée par :

$$Pg_h = Eg_h / T_{algo} \tag{1.3}$$

Il est à noter dans l'équation 1.3, que le terme correspondant aux transferts de données entre les niveaux de mémoire a disparu et que la consommation statique du niveau de mémoire n n'est à prendre en compte que pendant la durée pendant laquelle la mémoire a été placée virtuellement à ce niveau. Il découle de cette équation une problématique, relative aux outils de développement, consistant à trouver des configurations pour lesquelles le temps δT_n est court devant l'intervalle de temps global d'exécution de l'algorithme T_{algo}.
Pour illustrer l'intérêt de ce concept, nous avons étudié le cas de l'application du calcul de la DCT. Sur cette application, nous avons montré que ce principe de fonctionnement permettrait un gain à la fois en puissance statique et en puissance dynamique [A8]. Les expressions de ces gains sont alors les suivantes :

$$Gain_d = \frac{N^2 \left(2 \times Vdd_1^2 + Vdd_2^2\right)}{N^2 \times Vdd_1^2} = \frac{2 \times Kv^2 + 1}{Kv^2} \tag{1.4}$$

$$Gain_s = \frac{N/8 \times T_{dct} \left(Vdd_1 + Vdd_2\right)}{T_{dct} \left(Vdd_1 + Vdd_2(N/8 - 1)\right)} = \frac{N/8(Kv + 1)}{Kv + N/8 - 1} \tag{1.5}$$

avec N la taille de l'image sur laquelle on applique le calcul de la DCT, Vdd_i la tension d'alimentation du niveau de mémoire i, T_{dct} le temps de calcul total de la DCT et Kv le rapport des tensions Vdd_1/Vdd_2.
En considérant une image de taille égale à $N = 512$ et des tensions d'alimentation égales à $Vdd_1 = 2.5V$ et $Vdd_2 = 0.8V$, notre étude théorique montre qu'il existe les potentiels de gains en consommation suivants :

$$Gain_d \simeq 2 \tag{1.6}$$

$$Gain_s \simeq 4 \tag{1.7}$$

Une nouvelle fois, les travaux de master que nous réalisons actuellement devraient nous permettre de montrer que selon la valeur de Kv, ce principe de fonctionnement permet une meilleure maîtrise de la consommation de la partie mémoire des systèmes sur puce.

Notons que l'utilisation des différents modes d'alimentation pourrait être exploitée dans la hiérarchie mémoire présentée précédemment afin de limiter, lorsque cela est possible, la consommation énergétique globale du système mémoire. Dans ce cas, un contrôle spécifique devrait être mis en place pour gérer les mises en repos des différents bancs mémoire, une stratégie globale de gestion devant alors être définie afin de conserver de hautes performances tout en limitant l'empreinte énergétique du système.

1.4 Conclusion et perspectives

La conception de SoC embarqués complexes passe par la définition d'organisations mémoire innovantes et efficaces. Les solutions classiques de type mémoires caches ne sont guère adaptées compte tenu notamment des contraintes temps réel et faible consommation inhérentes à ces systèmes. Dans ce contexte, nous avons étudié cette problématique et nous l'avons abordée sous deux angles différents.

– Le premier concerne la définition d'une organisation distribuée des mémoires au sein d'un système de type multi-cœurs de traitement. La relocalisation des mémoires au plus près des éléments de traitement permet de *désengorger* la mémoire centrale. Dans ces travaux, nous avons abordé la définition de l'architecture générale, ainsi que l'interconnexion globale du système mémoire. Nous avons aussi proposé une structure de générateurs d'adresses permettant de produire des séquences d'accès pour des données régulières ou irrégulières. Cette étude est en cours et devrait permettre de proposer une solution de stockage performante. Ces travaux sont réalisés en collaboration avec le Laboratoire d'Intégration des Systèmes et des Technologies du CEA (CEA LIST).
– Le second concerne l'étude du potentiel d'optimisation énergétique qu'il est possible d'atteindre au travers du contrôle de la tension d'alimentation des mémoires. Notre proposition s'appuie sur l'adaptation dynamique de la hiérarchie mémoire, grâce à sa virtualisation. L'appartenance d'une mémoire à un niveau virtuel de la hiérarchie est déterminée par ses temps d'accès et sa consommation par accès. La position des mémoires dans les différents niveaux de la hiérarchie mémoire est contrôlée par la modification dynamique des paramètres de consommation et de temps d'accès des mémoires, mis en œuvre par des techniques de variation de la tension d'alimentation. Cette étude nous a permis de montrer qu'il existait un intérêt pour diminuer l'énergie globale du système mémoire, toutefois, nous devons maintenant poursuivre en étudiant les mécanismes de contrôle permettant d'atteindre cette minimisation énergétique.

Globalement, il est clair que les aspects mémorisation dans les systèmes sur puce sont un enjeu majeur des prochaines années. Le volume de données manipulées par les applications ne cessant de croître, les développeurs logiciels souhaitent avoir à leur disposition des quantités mémoire toujours plus importante, tout en conservant des temps d'accès très courts. Toutefois, d'un point de vue du concepteur matériel, une recherche de compromis doit être faite, et cela passe par des organisations mémoires complexes et innovantes. Nous avons montré, au travers de ce chapitre, deux pistes dont on peut facilement imaginer qu'elles profiteraient l'une de l'autre. L'organisation hiérarchique des mémoires, si elle permet de répondre à des exigences de performances, n'est pas une solution à elle seule pour les aspects consommation énergétique. De nombreuses études proposent de modifier dynamiquement les tensions d'alimentation pour limiter la consommation de mémoires non utiles à un instant donné. Pour la hiérarchie que nous proposons et dans un contexte où les tâches applicatives seraient connues à l'avance, l'ajout d'un contrôle sur la tension d'alimentation permettrait d'atteindre cet objectif. Nous allons poursuivre nos travaux dans ce sens, notamment en affinant nos connaissances des modes de fonctionnement des mémoires et en étudiant le couplage de notre organisation mémoire avec un contrôle des alimentations des mémoires.

Chapitre 2

Reconfiguration au sein des SoC et impact sur l'OS

2.1 Contexte

L'augmentation de la complexité des algorithmes de traitement du signal et de l'image, notamment dans les applications embarquées, nécessite des puissances de calcul importantes pour satisfaire l'ensemble des contraintes. L'une des contraintes critiques est sans aucun doute la notion de temps réel liée à l'exécution, mais des contraintes de qualité de services sont aussi, et de plus en plus souvent, à prendre en compte. Ce contexte complexe conduit les concepteurs à s'orienter vers des architectures matérielles composées de différentes unités de calcul opérant en parallèle et présentant des capacités de calcul différentes. La conception de l'architecture offre alors des possibilités d'optimisations importantes que le concepteur doit être en mesure d'évaluer afin de sélectionner la meilleure solution au sens de l'application à implémenter. Les systèmes sur puce, ou SoC, développés actuellement sont une réponse élégante par le potentiel d'intégration important qu'ils proposent. S'ils ouvrent une voie de conception tout à fait intéressante, ces systèmes entraînent avec eux un ensemble de problématiques complexes. L'hétérogénéité potentielle des unités de calcul pour les tâches de l'application est, à elle seule, une caractéristique importante compte tenu du degré de liberté qu'elle offre. Ce degré de liberté s'accroît un peu plus lorsque l'on envisage de placer au sein de ces systèmes des zones de logique reconfigurable dynamiquement (ARD[1]). En effet, la présence d'unités reconfigurables au sein du SoC offre des capacités d'adaptation de l'architecture à différents traitements tout en maintenant des hautes performances d'exécution. Souplesse et performance font de ce type de plate-formes des cibles très intéressantes pour des applications "gourmandes" en puissance de calculs et de plus en plus dynamiques, le terme de *Platform-based Design* [55] a été introduit pour ce type de plate-formes. La flexibilité de ces architectures est (ou sera à court terme) assurée, en fonction de la technologie reconfigurable employée, par la possibilité d'allouer dynamiquement différents opérateurs de traitement au sein même de l'accélérateur reconfigurable. On parlera alors de Reconfigurable SoC, RSoC.

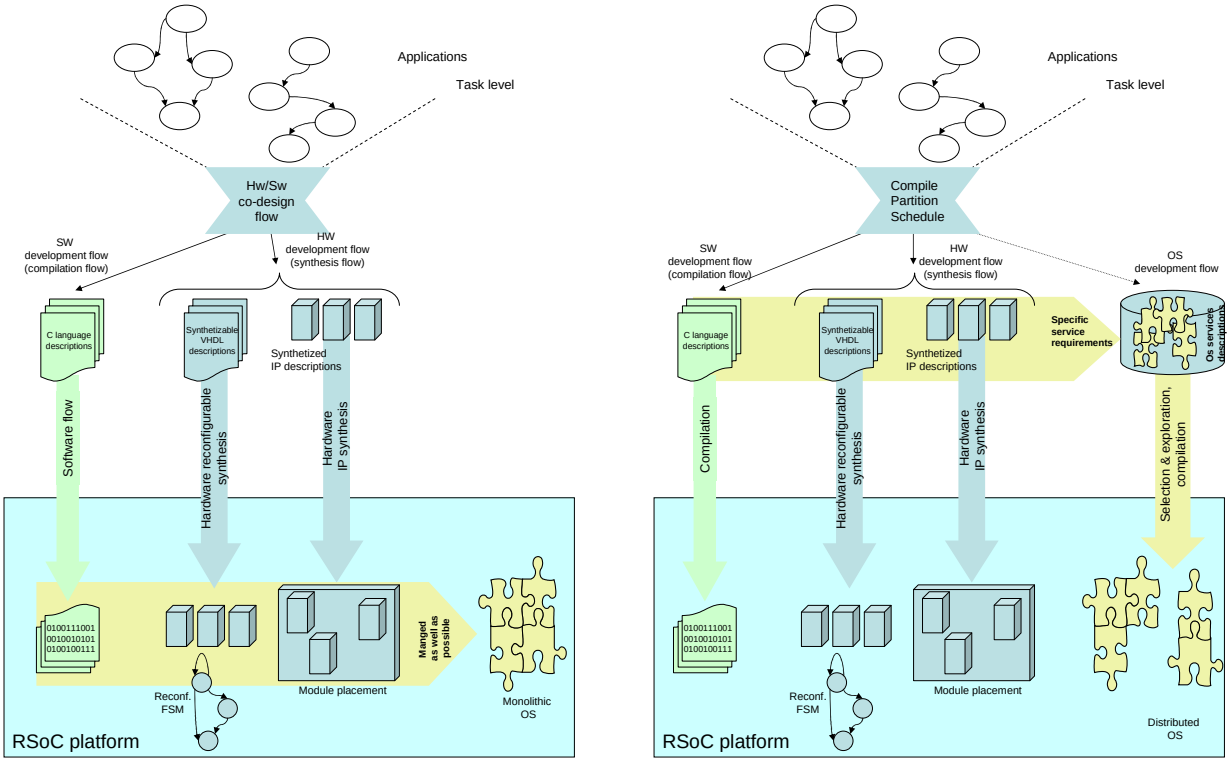
La volonté d'utiliser efficacement ce type d'architectures pose la question d'une gestion la plus souple possible de l'ensemble de la plate-forme. L'une des solutions envisageables pour faciliter la gestion des zones reconfigurables consiste à se "reposer" sur la notion de système d'exploitation (OS) et de ses services. Ceux-ci doivent alors au minimum permettre de centraliser la gestion de la mémoire, d'ordonnancer sous différentes contraintes les tâches à exécuter, d'assurer le partage des moyens de communication et d'offrir aux concepteurs un modèle de déploiement d'applications qui soit indépendant de la technologie et de la structure matérielle sous jacente.

La thématique des systèmes d'exploitation pour les RSoC (RTOS[2]) fait l'objet de nombreuses recherches. Pour le cas particulier de la gestion d'architectures reconfigurables, une identification des services d'un système d'exploitation temps réel a été proposée dans [56]. Depuis, des RTOS pour le reconfigurable ont été implémentés suivant deux concepts principaux. Le premier est l'utilisation d'un RTOS standard (RTAI , RTLinux, VxWorks, ...) auquel des services pour la gestion du RSoC [57] sont ajoutés. Le deuxième est l'utilisation d'un RTOS spécifique,

[1] ARD : Accélérateur Reconfigurable Dynamiquement

[2] RTOS : Real Time Operating System.

c'est-à-dire un ensemble de fonctions logicielles (et matérielles) nécessaires à la gestion du RSoC [58, 59]. Globalement, il semble que si la première solution a l'avantage de la simplicité, elle repose sur des noyaux assez monolithiques. La seconde approche est beaucoup plus dédiée et par conséquent peut-être plus en adéquation avec les besoins réels de l'application.
Depuis plusieurs années, nous menons des travaux sur cette thématique générale des RSoC. Ces travaux se sont inscrits en partie dans le projet ANR OverSoC. L'objectif global de ce projet a consisté à définir une méthodologie d'exploration pour la conception d'une plate-forme RSoC. Alors que dans un flot de conception classique l'application est implémentée sur le RSoC en s'appuyant sur la présence d'un OS existant (voir figure 2.1.a), nous avons proposé de remonter la conception de l'OS au même niveau que les autres éléments de la plate-forme (voir figure 2.1.b). Cette approche permet alors au concepteur d'explorer le triplet {Application, Architecture, Système d'exploitation}.



a) b)

FIG. *2.1 : Représentation schématique des flots de conception d'un RSoC. a) Le flot classique permet l'exploration des différents éléments de l'architecture et s'appuie au final sur un système d'exploitation éventuellement adapté et/ou équipé des pilotes nécessaires au contrôle des ressources de calcul. b) Le flot proposé dans OverSoC consiste à remonter le système d'exploitation dans le niveau de conception où l'exploration est réalisée. Nous avons proposé d'inclure l'OS dans cette étape, ce qui signifie que les services de l'OS sont alors vus comme des éléments à explorer. Leur définition, leur(s) instanciation(s), leurs caractéristiques sont considérées comme des éléments qui impactent le système final et qui, à ce titre, ne peuvent pas être imposées ou figées lors de la conception.*

Dans les sections suivantes, nous présentons tout d'abord le modèle de plate-formes envisagée dans le projet OverSoC. Nous indiquons ensuite les différents niveaux de représentation des éléments et le fonctionnement global de

la plate-forme. Nous discutons alors des évolutions de services nécessaires pour une gestion efficace de la plate-forme par le système d'exploitation, et plus particulièrement une gestion efficace de la zone reconfigurable. Ces éléments étant définis, nous nous focalisons ensuite sur deux études particulières. La première concerne la modélisation hiérarchique de la zone reconfigurable elle même. Il s'agit ici de définir plusieurs niveaux de description de la zone reconfigurable et de fournir les niveaux de description SystemC associés. Cette phase a été un élément important du projet OverSoC pour lequel une plate-forme d'exploration devait être fournie, incluant tous les éléments d'un RSoC. La seconde concerne l'étude des ressources de communication au sein de la zone reconfigurable. Nous détaillerons en quoi ce point est critique dans un RSoC et nous commenterons nos travaux sur cette problématique. Notons que ces travaux font partie intégrante du projet FosFor labélisé par l'ANR en septembre 2007 et encore en cours actuellement.

2.2 Modélisation de la plate-forme RSoC

Avant d'entamer nos travaux au sein du projet OverSoC, nous avons souhaité disposer d'une description sans équivoque d'une plate-forme de type RSoC. L'objectif de cette modélisation consiste à fixer, au minimum, le périmètre de chaque élément de la plate-forme, qu'il soit matériel et/ou logiciel.
La modélisation que nous avons proposée prend en compte la, ou les, zone(s) de logique reconfigurable dynamiquement [A33, 60]. Le modèle que nous avons proposé est générique et peut supporter des structures reconfigurables très variées, comme :

– des zones reconfigurables à grains fins de type FPGA pour les applications orientées niveau bit (architecture FPSLIC [61] par exemple) ;
– des zones à grains élevés pour les applications multimédia (SoC XPP [62] avec des traitements arithmétiques principalement) ;
– des zones mixtes associant chemins de données configurables et zones FPGA (architecture DART par exemple [63]).

La mise à disposition de nombreuses cibles d'exécution impose, si l'on veut un traitement efficace de l'application, que celle-ci soit décomposée en tâches. Pour des raisons de flexibilité d'instanciation sur la plate-forme, chaque tâche peut être décrite de manière logicielle et/ou matérielle.

– Une tâche logicielle est prise en charge par une unité d'exécution programmable pré-existante dans la plate-forme ou configurée au sein de la zone reconfigurable ;
– Une tâche matérielle s'exécute quant à elle sur une cible matérielle pré-existante dans la plate-forme ou configurée au sein de la zone reconfigurable.

L'instanciation des tâches sur le RSoC peut alors être représentée par le schéma de la figure 2.2.

2.2.1 Gestion de la plate-forme

La gestion de la plate-forme passe, entre autres, par l'assignation des tâches aux différentes ressources de calcul (matérielle et/ou logicielle). Sur la figure 2.2, les phases d'assignation sont représentées par les flèches accompagnées d'une notation `C`, `D`, `E` et `L`, correspondant à l'ensemble des cas possibles :

– Chaque tâche applicative (niveau 3) doit au moins disposer d'une description (flèche notée `D`) de niveau 2 ou 1. Dans l'exemple de la figure 2.2, les tâches applicatives, respectivement 1 et 3, sont décrites, respectivement par des tâches de niveaux 1 et 2. Chaque tâche applicative peut aussi être décrite par une ou plusieurs tâches de niveau 1 et/ou 2. C'est le cas de la tâche applicative 2 qui dispose de trois descriptions :
  – une description de niveau 2 : description logicielle sous forme d'un programme pour une cible d'exécution "processeur" ;
  – deux descriptions de niveau 1 pour deux cibles d'exécution matérielle différentes (avec éventuellement des performances différentes).
– L'assignation d'une tâche de niveau 2 sur une cible d'exécution "processeur" correspond à une phase d'exécution, notée `E` sur la figure 2.2. Cette phase est classiquement réalisée au travers d'un changement de contexte

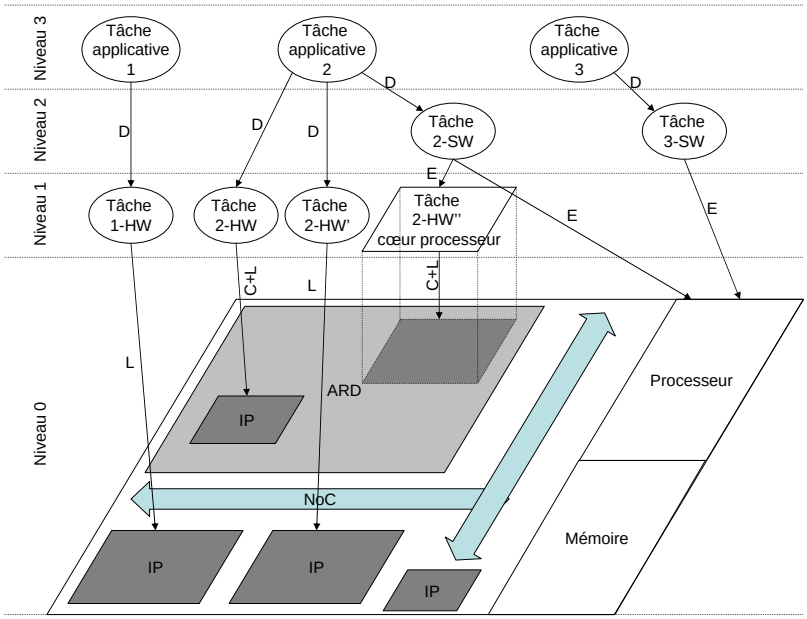


FIG. *2.2 : Modèle de la plate-forme et exemple d'instanciation de tâches matérielles (HW) et logicielles (SW). Le modèle est ici décomposé en 4 niveaux. Le premier niveau (niveau 0) correspondant à la plate-forme matérielle proprement dite. Cette plate-forme est constituée des différents blocs précédemment cités. Le niveau 1 définit les tâches qui s'exécutent directement sur les ressources. Les tâches matérielles sont simplement "lancées" (L) sur les blocs matériels dédiés, soit configurées sur des cibles d'exécution reconfigurables. Notons que les tâches matérielles peuvent éventuellement prendre en charge l'exécution de tâches logicielles, comme c'est le cas pour une ressource de type processeur qui est placée sur la ressource reconfigurable. Dans ce cas, il est tout d'abord nécessaire de prévoir une configuration (C) de la ressource processeur, pour ensuite "lancer" (L) la tâche logicielle sur cette ressource. Le niveau 2 définit les tâches logicielles découlant directement de la description (D) d'une tâche applicative. Finalement, le niveau 3 définit les tâches applicatives, ce niveau définit la fonctionnalité des tâches de l'application alors que le niveau 2 décrit les tâches au niveau algorithmique.*

permettant à ladite tâche d'obtenir la ressource de calcul. Ici la ressource d'exécution est soit le processeur, soit un cœur de processeur préalablement instancié sur l'ARD.

– L'assignation d'une tâche de niveau 1 sur la plate-forme prend deux formes possibles, en fonction de la cible matérielle envisagée :
  – soit il s'agit d'une cible de type "bloc spécialisé" (bloc IP), et dans ce cas l'assignation se résume au "lancement" du bloc (démarrage, noté `L` sur la figure 2.2) ;
  – soit il s'agit d'une cible reconfigurable, et dans ce cas l'assignation doit passer au préalable par une phase de configuration (allocation) de la zone reconfigurable (notée `C` dans la figure 2.2) avant d'effectuer le "lancement" (L) de la tâche sur la zone configurée (allouée).

Pour obtenir une gestion efficace de la zone reconfigurable, il est nécessaire de se poser la question de la préemption des tâches matérielles. Proposer la préemption pour ce type de tâches permet d'envisager la gestion des priorités des tâches arrivant en cours d'exécution. La préemption permet aussi d'offrir des possibilités de migration de tâches matérielles. Toutefois, pour les tâches matérielles, contrairement aux tâches logicielles, la notion de contexte d'exécution est critique, notamment à cause du volume d'information que celui-ci peut occuper. Aussi la majorité des travaux considère que la préemption d'une tâche matérielle ne peut être réalisée qu'à des instants particuliers. Ces instants, ou points de reprise, doivent être spécifiés par le concepteur.

2.2.2 Le modèle général

Pour prendre en compte à la fois le modèle architectural et le modèle de gestion définis précédemment, nous avons proposé une modélisation globale incluant tous les "objets" mis en œuvre dans ce type de plate-formes : de l'application au support physique d'exécution. Le langage graphique UML a été choisi afin d'avoir une représentation claire des différents éléments. Plus particulièrement, nous nous appuyons sur une représentation à l'aide d'un diagramme de classes UML, tel que le montre la figure 2.3. Bien que nous n'ayons pas retenu une modélisation au travers d'un ADL "classique" [64], le choix d'UML était suffisant pour ce projet et nous a permis d'identifier les différentes parties du système global.

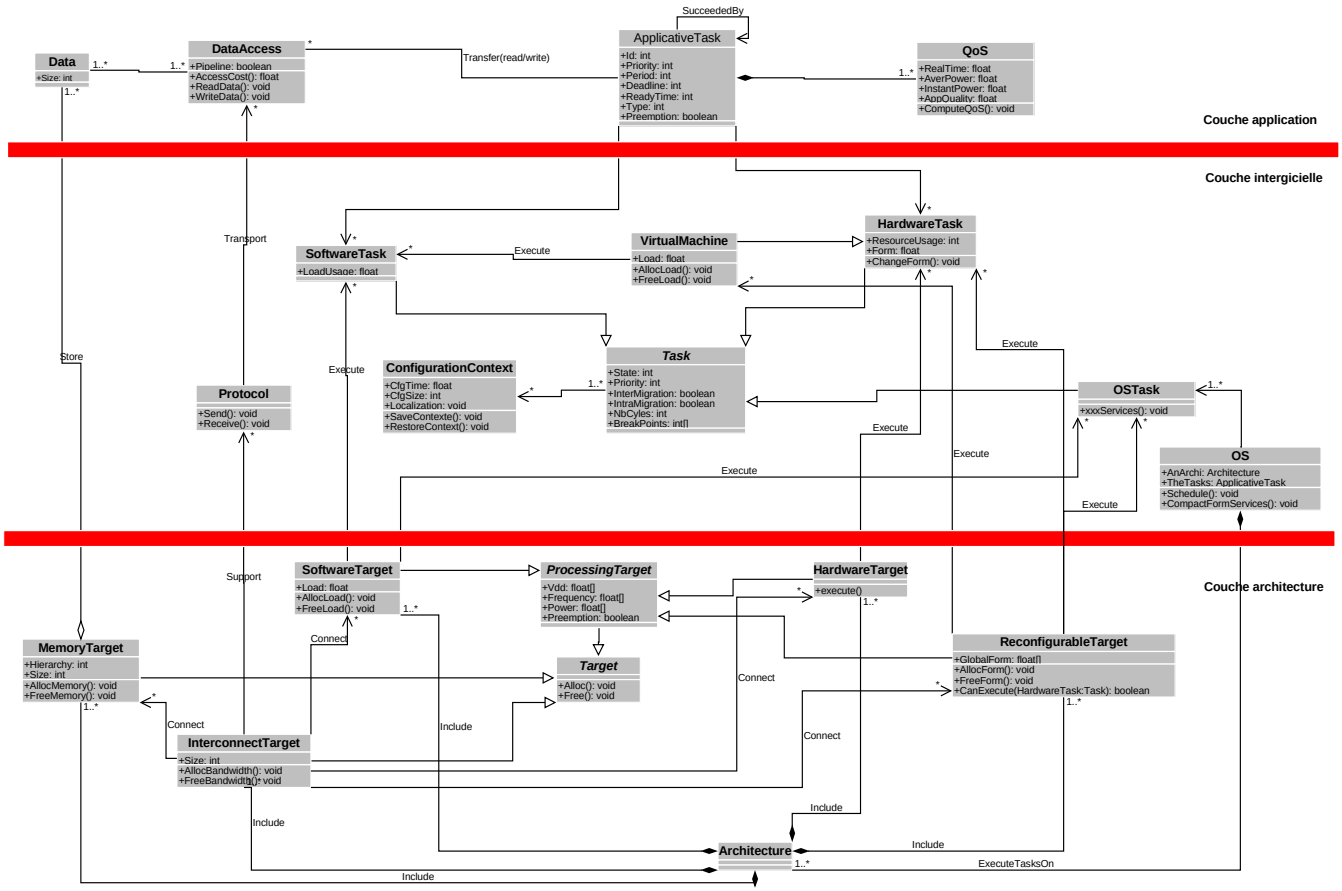


FIG. *2.3 : Diagramme de classes du modèle global de la plate-forme RSoC. Ce diagramme reprend les différentes couches présentées dans la figure 2.2. Ce diagramme fait apparaître l'ensemble des acteurs du système en indiquant les interactions existantes entre ceux-ci. Le système d'exploitation est ici très simplifié et l'ensemble des services a été résumé à un seul* objet *appelé OS dans la figure. L'objectif de cette modélisation consiste surtout à identifier clairement le périmètre de chaque élément. La couche physique, ou* architecture*, est parfaitement bien délimitée et modélise les cibles d'exécution, les éléments de stockage et enfin le support de communication. La couche* application *s'intéresse uniquement à la vue du développeur. Finalement, la couche intergicielle permet d'abstraire le support d'exécution et propose des services à l'applicatif.*

La modélisation que nous proposons est scindée en trois couches, correspondant à des niveaux différents de la plate-forme. Comme nous l'avons dit précédemment, les niveaux 1 et 2 de la figure 2.2 sont relatifs au système

d'exploitation. Aussi ces deux niveaux sont englobés dans la couche intergicielle représentée au centre de la figure 2.3.
Notons que la découpe en couches proposée permet d'englober tous les niveaux définis précédemment, tout en permettant une spécification de l'application hors de toutes considérations matérielles. La modélisation assure donc l'indépendance de la spécification de l'application vis-à-vis de l'architecture.

La couche application

Cette couche est le niveau de description de l'application, et plus particulièrement des tâches de l'application (*ApplicativeTask*). Cette description est indépendante de l'architecture matérielle. Une tâche applicative est spécifiée via des paramètres classiques du domaine du temps réel, tels que son identifiant, sa priorité, sa période, son échéance, sa date de validité (ou *ReadyTime*), son type qui spécifie si la tâche est stricte ou relative et un dernier paramètre indiquant son caractère préemptif ou non. Ce niveau permet aussi de spécifier les dépendances entre tâches (dépendances de données *DataAccess*, ou de contrôle).
Enfin, dans ce niveau, le concepteur peut être en mesure de spécifier la ou les qualités de services (QoS[3]) associée(s) à chaque tâche. La QoS peut être définie selon plusieurs paramètres (respect du temps réel, consommation, ...) et doit être adaptée, par le concepteur logiciel, à chaque application.

La couche architecture

La couche *architecture* supporte la modélisation des entités matérielles du RSoC présentes physiquement dans la plate-forme (niveau 0). Elle englobe les cibles de traitement (*ProcessingTarget*) qui devront supporter l'exécution des tâches, les cibles de stockage (*MemoryTarget*) des données manipulées et enfin les cibles d'interconnexion (*InterconnectTarget*) permettant l'échange de ces données.

- Les cibles de traitement (*ProcessingTarget*) sont déclinées en trois sous-ensembles représentant respectivement les cibles de type processeur (*SoftwareTarget*), les cibles matérielles (*HardwareTarget*) et les cibles reconfigurables (*ReconfigurableTarget*). L'architecture du RSoC contient au minimum un processeur et des supports d'exécution matériels de type IP et ARD. Le processeur supporte l'exécution de l'OS général du RSoC. Il est physiquement relié à tous les éléments de la plate-forme.
  Un Accélérateur Reconfigurable Dynamiquement est un ensemble de ressources dont la structure et le nombre peuvent être très variables d'une architecture à l'autre. Cet accélérateur peut supporter l'exécution de plusieurs tâches simultanément.
- Le modèle de ressource mémoire permet la description d'une organisation mémoire hiérarchique. Les notions de hiérarchie mémoire, d'optimisation de ces hiérarchies, de caches, etc, constituent un enjeu essentiel dans les SoC et les RSoC. Les travaux présentés dans le chapitre 1 couvrent la définition de l'organisation mémoire, et devront proposer des services de gestion de cette ressource. Les études en cours devraient conduire à une amélioration de la gestion de cette partie du RSoC.
- Les ressources de communication permettent de définir le support physique des échanges entre les différents éléments de la plate-forme. La modélisation proposée à ce niveau concerne les interconnexions entre les processeurs, les accélérateurs spécialisés (blocs IP) et les accélérateurs reconfigurables. Ce type d'interconnexions est statique et optimisé à la conception. La tendance actuelle semble montrer que des solutions basées sur les NoC[4] sont un bon compromis entre les performances attendues et la flexibilité nécessaire pour les RSoC.

La couche Intergicielle (ou *Middleware*)

La couche *intergicielle* contient toutes les entités manipulables par le système d'exploitation, et notamment les tâches. Ce niveau décrit en particulier les différentes tâches de configuration ainsi que les protocoles de communication pouvant être utilisés. Enfin, c'est à ce niveau que sont spécifiées les interactions entre une application, une implémentation sur une plate-forme reconfigurable et le système d'exploitation qui gère toutes ces entités.

[3] QoS : Quality of Services
[4] NoC : Network on Chip

Cette couche s'articule autour d'un élément central qui est la "tâche" au sens du système d'exploitation (*Task*). Une tâche est caractérisée par un certain nombre de paramètres classiques (durée, périodicité, ...) et peut se décliner de deux façons différentes : *SoftwareTask* et *HardwareTask*. Cette modélisation permet de considérer la définition de tâches logicielles et/ou matérielles pour toutes les tâches de l'application.
Certaines tâches matérielles particulières peuvent être des supports d'exécution de tâches logicielles à l'instar des processeurs classiques. C'est le cas de cœurs de processeurs "mous" configurés dans l'ARD (comme c'est le cas dans certains FPGA embarquant des cœurs de processeur de type : NIOS pour Altera, μBlaze pour Xilinx) ou de chemins de données spécialisés exécutant des instructions. Le modèle général proposé les regroupe sous le terme de machines virtuelles (*VirtualMachine*). Une machine virtuelle peut prendre en charge l'exécution de tâches fournies par le système d'exploitation général.
Enfin, chaque tâche, qu'elle soit logicielle ou matérielle, nécessite une phase de configuration (*Configuration-Context*) pour s'exécuter sur une cible. Lorsque la tâche considérée s'exécute sur un processeur, ce contexte est similaire à celui d'une tâche logicielle classique : valeur du pointeur de programme, pointeur de pile, etc. Dans le cas d'une tâche matérielle, le contexte est plus complexe et contient l'ensemble de la séquence de configuration d'une cible matérielle (par exemple un *bitstream* pour une zone de type FPGA). On considère ici que la technologie de la zone reconfigurable permet la réalisation de reconfigurations partielles en ligne de n'importe quelle zone de celui-ci.
Enfin, c'est dans cette couche qu'est situé le système d'exploitation. L'OS est modélisé au travers de ses services, chaque service pouvant alors être décliné en une implémentation particulière (tâche *OSTask*). Un certain nombre de services de l'OS relatifs à la gestion de la plate-forme ont été identifiés. Les services spécifiques au caractère reconfigurable de l'architecture sont décrits dans la section suivante.

2.3 Evolutions de l'OS pour un RSoC

Le rôle de l'OS consiste à orchestrer l'ensemble des traitements sur la plate-forme RSoC. Pour prendre en charge efficacement la présence d'un ARD dans le système, certains services doivent subir une évolution, parfois en profondeur. Dans les paragraphes suivants, nous donnons quelques éléments de réflexion pour les services principaux de l'OS pour un RSoC.

2.3.1 Services aux tâches

Les services aux tâches sont sans aucun doute les plus nombreux dans un OS. Ils sont généralement plus ou moins imbriqués et il est assez délicat de les étudier indépendamment les uns des autres. Nous tentons de lister chacun d'eux dans les paragraphes suivants en essayant, autant que faire se peut, de nous attacher aux fonctionnalités principales.

Service de création de tâches

Dans un système classique, le service de création de tâches consiste principalement à réserver l'espace mémoire nécessaire à l'exécution d'une tâche et à insérer la tâche dans la liste des tâches à ordonnancer. En règle générale, lors de l'initialisation d'un système, l'ensemble des tâches est créé, quitte à laisser une grande partie de celles-ci en suspend jusqu'à l'arrivée d'un événement. Dans le contexte d'une plate-forme de type RSoC, il n'est pas envisageable de "réserver l'espace nécessaire" au sein de l'ARD à l'initialisation du système. L'objectif de la mise à disposition de l'ARD est justement de pouvoir remplacer les tâches au fur et à mesure de l'exécution de l'application. Ainsi la notion de création doit ici être redéfinie. La création d'une tâche à exécuter sur l'ARD doit aussi s'accompagner de phases de placement et de configuration de l'ARD. Cette configuration intervenant alors au moment où la tâche doit s'exécuter. Le reste du temps, la tâche est présente dans la table des tâches de l'OS mais pas physiquement présente dans l'ARD. De même, en règle générale, le support de communication est mis en place lors de l'initialisation d'un système. Dans le cas de l'ARD, il n'est pas possible de construire celui-ci à l'avance compte tenu de la méconnaissance de la position spatiale de la tâche avant son placement.

Service de sélection de tâches/ressources

Toutes les tâches, matérielles et logicielles, sont gérées par l'OS, celui-ci pouvant être amené à gérer plusieurs instances possibles pour une même tâche applicative. En effet, notre modèle de description de l'application repose sur la possibilité de spécifier des tâches applicatives au travers de tâches logicielles et/ou matérielles. Le choix d'une tâche matérielle ou logicielle, pour l'exécution d'un traitement, doit être réalisé dynamiquement par l'OS en fonction de l'état du RSoC. Par exemple, si le processeur central du RSoC peut supporter la charge d'une nouvelle tâche logicielle tout en respectant l'ensemble des contraintes, alors la tâche pourra être exécutée par celui-ci. Par contre, si le processeur ne peut pas accepter une nouvelle tâche sans provoquer la violation des contraintes d'une ou plusieurs tâches, alors il faut envisager le placement (éventuellement le déplacement, i.e. la migration) d'une ou plusieurs tâches vers un (ou plusieurs) ARD ou bloc IP.
Quelque soit la stratégie retenue, on voit ici qu'il est nécessaire de mettre à disposition de l'OS des métriques. En effet, pour être en mesure de prendre des décisions quant aux placements des tâches, l'OS doit pouvoir connaître l'état de la zone reconfigurable. Deux techniques peuvent alors être envisagées. La première consiste à équiper l'OS d'estimateurs lui permettant d'évaluer l'état de la zone reconfigurable au cours du temps. La seconde consiste à doter l'ARD "d'intelligence" et de l'équiper de mécanismes lui permettant de prendre en charge une partie des services de l'OS. Dans ce cas, l'OS est alors distribué et un dialogue doit être mis en place entre les services implémentés sur le processeur et l'ARD. Au sein du projet OverSoC, nous avons retenu la seconde solution qui nous permet d'envisager une exploration des services de l'OS non seulement en terme de fonctionnalité, mais aussi en terme de performances des implémentations sur les différentes cibles d'exécution. Les travaux que nous développons dans la section 2.4 se placent dans ce contexte.

Service d'assignation et de lancement des tâches

Le service d'assignation consiste à "placer" les tâches sur les différentes ressources d'exécution. Pour un processeur, cela consiste principalement à charger le code de la tâche. Pour un ARD, le placement consiste à rechercher un espace libre dans la zone reconfigurable, puis se poursuit par "l'envoi" de la configuration. Avant de pouvoir débuter l'exécution de la tâche, il est primordial de s'assurer que la tâche dispose des moyens de communication et de stockage de données nécessaires. Une fois cette vérification effectuée, l'allocation de l'espace physique peut alors être réalisée. Il est ensuite nécessaire de construire le support de communication afin d'assurer les échanges de données de la tâche précédemment placée. Notons que la phase de configuration peut être optionnelle si la tâche est déjà présente au sein de l'ARD. Finalement, une tâche logicielle peut, elle aussi, être affectée à l'ARD à condition qu'une machine virtuelle soit préalablement configurée. Cette configuration peut elle aussi être optionnelle dans le cas ou l'ARD disposerait déjà d'une instance d'une machine virtuelle.

2.3.2 Service de gestion de la qualité de services (QoS)

La gestion de la qualité de services de l'application/des tâches est délicate puisqu'il s'agit d'optimiser, *on-line*, l'exécution du système. La présence d'un ARD augmente le nombre de degrés de liberté sur lesquels l'OS peut agir. Toutefois, il est fort probable que l'exécution d'une tâche sur un ARD ne puisse pas offrir toute la flexibilité d'une exécution sur un processeur. La préemption semble en effet une caractéristique qui ne puisse pas facilement être proposée par une exécution matérielle. De la disponibilité de la préemption découle la possibilité d'assurer la migration de tâches au sein de l'ARD.

2.3.3 Service de gestion des ressources

La connaissance de l'occupation des ressources est nécessaire pour tout OS. Dans le cas d'un RSoC, l'assignation dynamique d'une tâche sur une zone géographique d'un ARD rend la notion de ressources non seulement dynamique d'un point de vue temporel mais aussi d'un point de vue spatial (ou géographique). Pour réaliser efficacement cette gestion, des informations concernant la quantité, la forme, la position des ressources "consommées" par les tâches sont absolument nécessaires. Finalement, la gestion des ressources de la zone reconfigurable peut nécessiter des mécanismes particuliers, c'est le cas par exemple de la migration de tâches qui se complexifie

lorsque la cible d'exécution n'est plus un simple processeur. De même, la dynamicité du placement au sein de cette ressource va provoquer un fractionnement qu'il peut être important de "récupérer".

La migration de tâches au sein d'un système peut être intéressante pour plusieurs raisons, par exemple pour assurer la répartition des charges de calcul sur tous les éléments d'exécution. Mais cette fonctionnalité peut aussi permettre d'optimiser l'exécution selon certaines contraintes (temps d'exécution, énergie, qualité de services, etc). Pour assurer ce service, les tâches ainsi que leurs supports d'exécution doivent pouvoir supporter une éventuelle préemption par l'OS (caractéristique précisée dans la classe *ProcessingTarget*). Par ailleurs, une tâche peut être autorisée à migrer d'une cible, matérielle ou logicielle, vers une autre cible équivalente (*IntraMigration*) ou d'un type différent (*InterMigration*). Dans le cas d'une inter-migration, cela suppose l'existence de plusieurs instances différentes (matérielle et/ou logicielle) d'une même tâche applicative. Pour une tâche matérielle, la migration s'effectue après sauvegarde du contexte de la tâche. Le contexte correspond à un point de reprise (*BreakPoint*), antérieur à la date d'interruption, le plus proche parmi les points de reprise spécifiés au sein de cette tâche. À chaque bloc compris entre deux points de reprise peut correspondre une représentation différente suivant la cible d'exécution.
Il est à noter que le problème principal dans l'*InterMigration*, reste l'identification de l'équivalence d'états entre la représentation logicielle et la représentation matérielle de la tâche à faire migrer ; les informations sur l'état doivent être transmises pour pouvoir relancer la tâche à partir de son point d'arrêt [57]. Un service de conversion de contextes doit être mis en œuvre pour permettre la migration entre cibles d'exécution hétérogènes. Nous supposerons que cette conversion peut être réalisée ou que le concepteur a donné les équivalences entre les différentes représentations disponibles.

Le morcellement inévitable de l'ARD, découlant des différentes instanciations de tâches au cours du temps, va régulièrement conduire à l'impossibilité de trouver une zone dans l'ARD de taille suffisante pour y allouer une nouvelle tâche à exécuter.
Un service de compactage (ou encore de "ramasse miettes") doit alors être proposé. L'objectif d'un tel service consiste à récupérer la forme rectangulaire la plus grande possible dans l'ARD. Le déclenchement de ce service doit reposer sur une mesure "de fractionnement" et une évaluation du temps d'exécution du ramasse miettes.
Notons que la fonction de compactage peut aussi prendre en charge la définition d'une nouvelle forme lors du déplacement d'une tâche. C'est la possibilité de modifier la forme des tâches matérielles afin d'optimiser l'utilisation des ressources. On dira alors que la tâche matérielle est "malléable" ou géométriquement souple.
Comme pour le service logiciel de ramasse miettes disponible dans certains langages, on retrouve ici la problématique de la non prédictibilité de l'exécution d'une application. Des mécanismes de déclenchement ou d'exécution parallèle du ramasse miettes peuvent alors être étudiés.

Finalement, le service de gestion des ressources doit couvrir les aspects mémorisation. En effet, pour s'exécuter, chaque tâche doit disposer d'une certaine quantité de mémoire. Pour satisfaire ces besoins, deux points de vue peuvent être envisagés :
- Le premier cas envisageable est celui pour lequel la mémoire nécessaire pour l'exécution de la tâche est incluse dans la description de la tâche elle-même. On suppose donc dans ce cas, que le concepteur a "embarqué" la mémoire dans la description de la tâche et que les ressources matérielles synthétisées contiennent cette mémoire ;
- Le second cas envisageable consiste à prévoir, en même temps que l'allocation des ressources de calcul, l'allocation mémoire pour l'exécution de la tâche. Dans ce cas, la phase de placement est plus complexe puisqu'elle doit s'assurer que suffisamment de mémoire est disponible pour chaque tâche.

Pour les travaux envisagés dans le projet OverSoc, nous nous plaçons dans le second cas, ce qui nous amène à compléter la description des tâches par une information concernant la quantité mémoire nécessaire à l'exécution de chaque tâche.

2.3.4 Service de communication

Ce service doit pouvoir offrir toutes les possibilités de communication classiques pour les tâches. Les communications entre tâches peuvent mettre en jeu plusieurs média. Entre les différents éléments du RSoC, les com-

munications peuvent être assurées par un support de type NoC. Un certain nombre de travaux ont abordé cette problématique et proposent de générer des réseaux en fonction des besoins applicatifs, et d'allouer de la bande passante dédiée pour les échanges entre tâches [65, 66]. Si des solutions de communication globales existent, les communications internes à l'ARD sont plus délicates à mettre en œuvre. En effet, la dynamicité du placement des tâches ne permet pas de déterminer à l'avance (*off line*) les interconnexions entre tâches. Une solution offrant de la flexibilité et de la dynamicité doit alors être envisagée. Une structure flexible ainsi que le contrôle de cette structure doivent être définis. Des services spécifiques doivent être associés au support physique défini pour que l'accès au média de communication soit le plus transparent possible pour les tâches. Nous avons entamé des travaux sur cette problématique, nous développons ce point dans la section 2.4.3.

2.3.5 Synthèse

Sans prétendre avoir effectué une étude exhaustive des services d'un OS, il apparaît clairement que certains services doivent être modifiés afin de prendre en compte efficacement la présence d'une zone reconfigurable au sein d'un SoC. Le projet OverSoC a permis de couvrir l'ensemble des problématiques présentées brièvement ci-dessus. En proposant une méthodologie d'exploration de la totalité de la plate-forme, y compris l'exploration du système d'exploitation, l'outil développé durant le projet OverSoC offre un potentiel d'optimisation intéressant. Une capture d'écran de cet outil est présentée à la figure 2.4, elle donne un aperçu de la façon dont l'OS est composé [A3]. Des résultats graphiques sont ensuite accessibles pour guider le concepteur face aux différents choix qui s'offrent à lui.

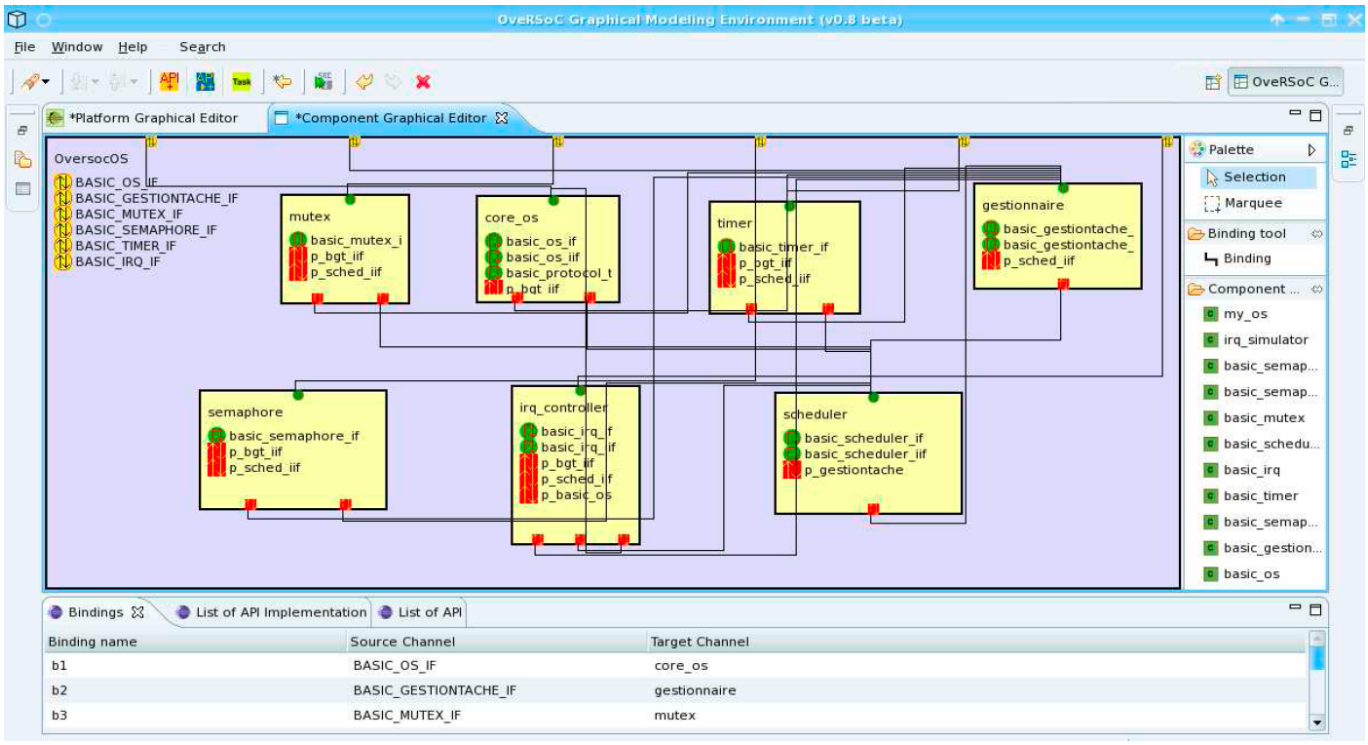


FIG. *2.4 : Capture d'écran de l'outil DOGME développé durant le projet OverSoC. Cet outil permet de construire un operating system à partir de briques/services de base. Ces différents services peuvent être associés à différentes ressources d'exécution et ainsi permettre de juger de l'opportunité de distribuer l'OS au sein de la plate-forme.*

Au sein du projet OverSoc, nous nous sommes focalisés sur deux points importants que nous présentons dans les sections suivantes. Le premier point concerne la modélisation de l'ARD. Afin d'étudier l'implémentation des services de l'OS sur cette cible d'exécution, nous avons tout d'abord réalisé une modélisation de cette cible. Cette modélisation est hiérarchique et permet d'aborder la complexité du problème de façon progressive. Nous

focalisons ensuite notre attention sur la problématique des interconnexions au sein de la zone reconfigurable et nous présentons nos travaux de description d'une architecture assurant flexibilité et performance.

2.4 Travaux spécifiques concernant l'ARD

Les travaux que nous exposons dans les sections suivantes sont menés depuis plusieurs années au sein de l'équipe CAIRN et ont été en grande partie intégrés dans le projet ANR OverSoc. Au sein de ce projet, nous avons notamment réalisé la modélisation de la zone reconfigurable dynamiquement présente au sein des RSoC. Cette modélisation a été envisagée au travers de la définition de plusieurs niveaux permettant une approche incrémentale. Cette approche est en adéquation avec la méthodologie de raffinement/exploration mise en place dans le projet OverSoc. La modélisation est basée sur le langage graphique UML et est compatible avec la modélisation globale qui définit l'ensemble de la plate-forme RSoC. Cette modélisation se décline ensuite en divers niveaux de descriptions SystemC. Nous décrivons la modélisation proposée puis nous détaillons les niveaux de descriptions que nous avons réalisés.

Nous nous sommes ensuite focalisés sur la problématique de la communication au sein d'un RSoC et plus particulièrement des interconnexions au sein de la zone reconfigurable. Au niveau global, les communications au sein d'un RSoC ont été assez largement étudiées et ont trouvé une réponse intéressante au travers du déploiement de réseaux sur puce, NoC [67]. Les solutions proposées apportent flexibilité et adaptabilité qui sont les deux paramètres importants recherchés afin de pouvoir maîtriser la complexité grandissante des RSoCs [68, 69].
Au niveau local, c'est-à-dire au sein de la zone reconfigurable, le problème majeur relève de la faculté à pouvoir accueillir et exécuter plusieurs tâches au cours du temps selon les besoins de l'application. La dynamique applicative ne permet pas de prévoir à l'avance (*off-line*) la position des tâches au sein de la zone reconfigurable. Cette méconnaissance de la position géographique des tâches au cours du temps implique que la zone reconfigurable soit dotée de mécanismes lui permettant d'acheminer les données aux tâches de façon efficace. Nous avons étudié les différentes solutions proposées dans la littérature, en comparant notamment les solutions de type bus et NoC. Compte tenu du type de traitements que devra supporter la zone reconfigurable (plutôt des calculs intensifs), nous avons conclu qu'une solution "intermédiaire" devait être étudiée. Les solutions présentées dans [65, 66] répondent assez bien à nos attentes : allocation de bande passante en fonction des besoins applicatifs, mais ces solutions n'offrent pas la dynamicité nécessaire au sein d'un RSoC, c'est-à-dire la construction du réseau en cours d'exécution.
Les sections suivantes présentent les travaux que nous savons menés concernant la modélisation de la zone reconfigurable.

2.4.1 Modélisation de l'Accélérateur Reconfigurable Dynamiquement

Nos travaux de modélisation sont issus des réflexions menées durant le travail de projet technologique de Karim HAROUAT (élève ingénieur de troisième année ENSSAT 2005-2006). Une première version de modélisation a fait l'objet d'une présentation et d'une discussion lors des premières réunions du projet OverSoC. Une seconde version de cette modélisation a alors été proposée et a servi de base pour le travail de description SystemC.

Comme nous l'avons dit, notre modélisation propose une découpe en niveaux. En effet, dans la méthodologie OverSoC, il est envisagé de pouvoir obtenir des estimations de performances au fur et à mesure de l'exploration du système, or suivant les applications, il est probable que le concepteur ne souhaitera pas forcément explorer les modules/composants de façon homogène. Il est alors nécessaire de disposer de plusieurs niveaux de modélisation (et donc de description) permettant de raffiner les modules plus ou moins indépendamment les uns des autres. Le point de départ de l'exploration est un niveau très abstrait dans lequel les modules ne proposent que très peu de métriques et pour lequel la description est assez "pauvre". Ce niveau (niveau 0) correspond au haut de la figure 2.5.

Dans la modélisation, l'identification d'une interface commune à tous les niveaux de description a été abordée. Il nous est apparu intéressant de proposer une interface clairement identifiée, afin de faciliter l'inter-opérabilité

des différents composants de la plate-forme RSoC. Les avantages de ce choix sont clairement de pouvoir envisager l'intégration de modules développés hors du projet OverSoC, par exemple à un niveau de description plus fin. De plus, ce choix permet de proposer une vue externe similaire quelque soit le niveau de raffinement. Dans

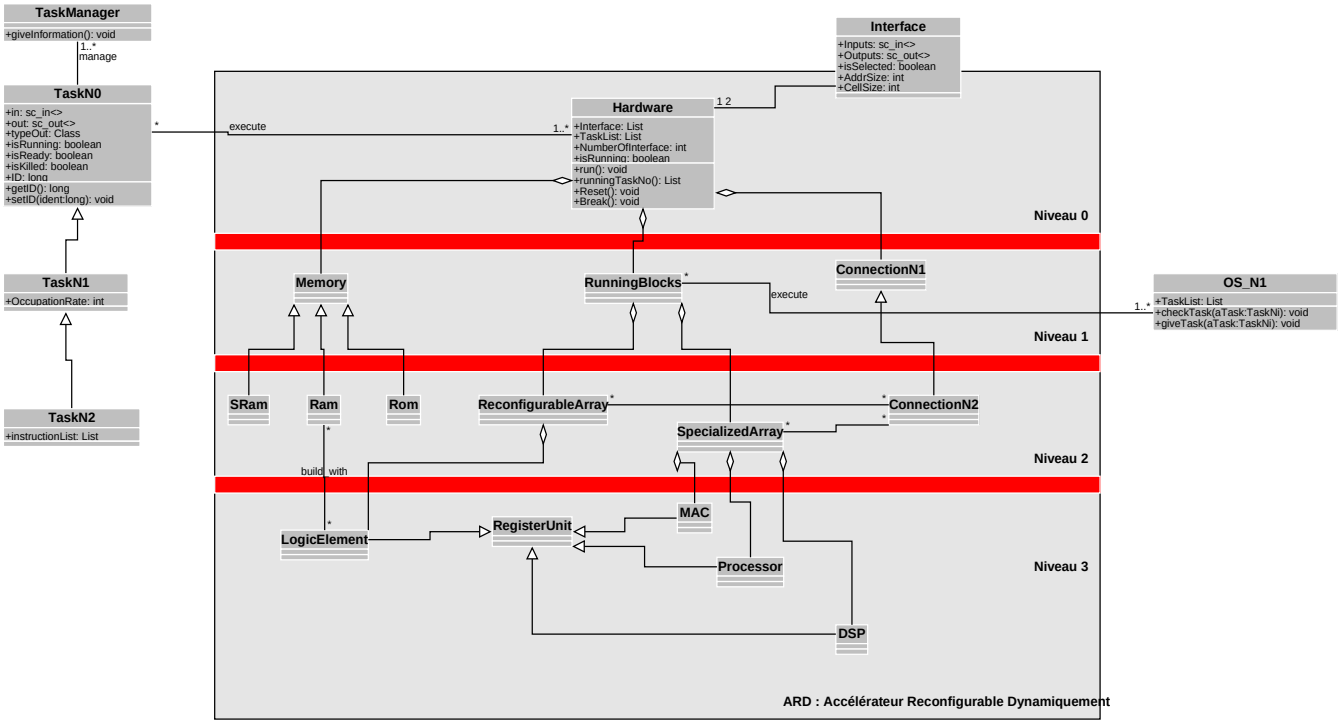


FIG. *2.5 : Diagramme de classes de la zone reconfigurable. Ce diagramme de classes fait apparaître une découpe en niveaux correspondant à des niveaux de détails de plus en plus fins. L'objectif de cette modélisation est de proposer une graduation dans la description de la zone reconfigurable afin de faciliter l'exploration architecturale d'un RSoC. D'un niveau purement* fonctionnel *(niveau 0) à un niveau complètement* structurel*, cette modélisation a permis de développer tout ou partie des descriptions SystemC de cette ressource reconfigurable.*

cette modélisation, l'interface est volontairement placée à la frontière de l'ARD et du reste du RSoC. De plus, la modélisation des tâches apparaît sur le côté gauche de la figure en dehors du périmètre de l'ARD afin qu'il n'y ait pas de confusion entre la modélisation de la ressource matérielle de l'ARD et les "objets" logiciels manipulés par cette ressource.

Description du niveau 0 de l'ARD

À ce niveau de description, l'ARD est simplement vu comme un bloc décrit au niveau comportemental. Ce bloc, est dans un premier temps démuni de toute "intelligence" et est aux ordres d'un système d'exploitation extérieur. Dans ce mode de fonctionnement, l'OS prend donc en charge le chargement de la configuration dans l'ARD. Une fois configurée, la tâche s'exécute et l'OS doit alors acheminer les données vers la tâche aux instants souhaités. Dans le contexte d'OverSoC, cette solution est peu satisfaisante puisqu'elle suppose que l'ARD est un bloc de type IP un peu particulier et qu'il n'implémente aucun service de l'OS. La stratégie mise en place dans le projet consiste à explorer l'implémentation des services, et notamment l'implémentation des services spécifiques à l'ARD au sein de l'ARD lui-même. Nous avons donc opté pour une modélisation de l'ARD incluant une première brique d'OS comme le montre la figure 2.6. À ce niveau de description, l'ARD est équipé d'un bloc lui permettant de contrôler le chargement des tâches. Ce bloc est aux ordres d'un système d'exploitation extérieur et ne prend en charge que le chargement de la description. Les évolutions que nous présenterons par la suite sont des extensions de fonctionnalités de ce contrôleur. La figure 2.6 présente l'organisation que nous utilisons pour cette solution.
À partir de ce schéma, le scénario de fonctionnement envisagé est le suivant :

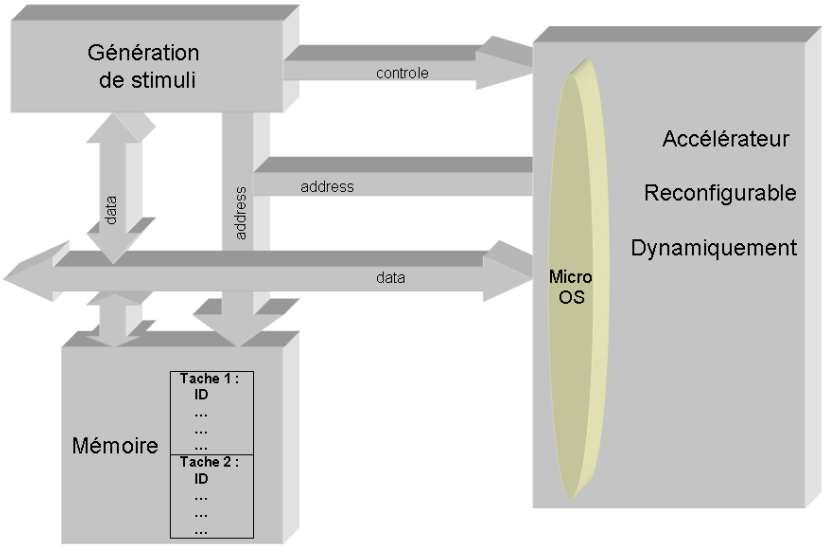


FIG. *2.6 : Description de la zone reconfigurable et test de celle-ci. Pour parvenir à tester notre modèle d'ARD, une mémoire a été mise en place afin de contenir les descriptions des tâches à exécuter. Un générateur de stimuli (modélisant l'OS) envoie des ordres/commandes à la zone reconfigurable (chargement de tâche, arrêt de tâche, etc) qui réalise alors l'action demandée sans aucune vérification. Ce mode de fonctionnement correspond à une solution dans laquelle le système d'exploitation s'exécute sur le processeur général et dispose de l'ensemble des informations, notamment d'utilisation de la zone reconfigurable. Seul le chargement des tâches est déporté dans l'ARD.*

1. l'ARD reçoit un ordre de chargement d'une tâche accompagné d'un pointeur sur la zone mémoire contenant la description de la tâche. L'ordre provient de l'OS global au système (ici le générateur de stimuli) et deux cas sont possibles :
   - l'OS partage sa mémoire avec l'ARD qui peut alors accéder à l'unique description de la tâche ;
   - l'OS a préalablement copié la description de la tâche dans une mémoire spécifique pour l'ARD et qui est ensuite utilisée uniquement par celui-ci ;
2. l'ARD accède à la zone mémoire stockant la description de la tâche et effectue alors le chargement de celle-ci ;
3. on peut imaginer ensuite, qu'en fonction de la disponibilité des ressources de calcul, la tâche soit mise en état *Sleeping* puis *Ready* et enfin *Running*. À ce niveau de description, on imagine que l'OS externe pilote les changements d'états des tâches au travers d'ordres envoyés par le biais du port *contrôle*.

Pour simplifier le développement de cette solution, nous avons décrit un bloc de génération de stimuli qui envoie des ordres/commandes de type *LOAD_TASK* accompagnés de l'adresse mémoire où se trouve la description des tâches à charger. La liste des commandes reconnues par le micro OS présent dans l'ARD est donnée dans le tableau 2.1. La description d'une tâche est réalisée en SystemC au travers des éléments suivants : Notons que la spécification que nous donnons ci-dessus a été complétée dans la suite des travaux afin d'ajouter des informations nécessaires pour les autres niveaux de description.

D'autre part, le modèle de tâches que nous avons adopté supporte les états classiques des tâches (*Running, Waiting, Ready*), par contre les transitions entre états sont restreintes au cas du reconfigurable, c'est-à-dire en posant comme hypothèse que la préemption des tâches matérielles n'est pas possible. La figure 2.7 montre ces états et les transitions possibles entre ces états.

Commandes	Significations
LOAD_TASK	cette commande permet de charger une tâche au sein de l'ARD. Cette commande équivaut à la fonction *create*. Pour l'instant, à la réception de cet ordre, l'ARD crée une tâche et la place dans l'état *Sleeping* ;
KILL_TASK	cette commande permet d'extraire une tâche de l'ARD, il s'agit donc d'une opération de libération des ressources de calcul. Du point de vue de l'ARD, cette commande équivaut bien à une opération de type *kill* ;
RESUME_TASK	cette commande déclenche directement la fonctionnalité *resume* présentée dans sur la figure 2.7. Il s'agit du passage en mode *Ready* de la tâche lorsque cela est possible ;
SUSPEND_TASK	cette commande déclenche directement la fonctionnalité *resume* présentée à la figure 2.7. Quoi qu'il arrive, la tâche est placée dans cet état et elle libère donc la ressource de calcul si nécessaire ;
START_TASK	cette commande déclenche directement la fonctionnalité *resume* présentée à la figure 2.7. Si la tâche en question est prête (état *Ready*) alors la tâche est déclenchée, elle démarre donc son exécution ;
STOP_TASK	cette commande déclenche directement la fonctionnalité *resume* présentée à la figure 2.7. Lorsqu'une tâche est en mode *Running*, alors celle-ci est stoppée et mise en attente. Elle reste éligible pour une nouvelle exécution ;

TAB. *2.1 : Liste des commandes pour le contrôle de la zone reconfigurable. Ces commandes sont envoyées à la zone reconfigurable via le* bus *de contrôle représenté à la figure 2.6. Cette commande s'accompagne d'un numéro d'identifiant de la tâche. Le* micro OS *réceptionne ces commandes, les décode et s'assure de leur exécution.*

```
int<16> identifiant
int<8>  priorite
int<8>  duree
int<8>  nbRessources
```

Listing 2.1 : Description SystemC des tâches logicielles et/ou matérielles. Pour chaque tâche, on retrouve les caractéristiques classiques nécessaires à la gestion par un operating system*. Toutefois les caractéristiques des tâches se voient complétées par des informations sur les ressources utiles pour leur exécution. Par exemple, pour une tâche matérielle, la quantité de ressources matérielles nécessaires devra être précisée.*

Simulations

À partir de ce premier niveau de description, nous avons réalisé des simulations. Ces simulations se déroulent par envois successifs de commandes à l'ARD. Le format utilisé est constitué de la commande suivie du numéro de tâche concernée : (COMMANDE, VALEUR), un exemple est donné dans le listing 2.2. Notons que le numéro de tâche indiqué ici correspond au numéro de tâche vue de l'OS général et qu'il peut être différent du numéro de tâche géré par l'ARD.
Cette séquence d'ordres ou de commandes déclenche les opérations de chargement, de modification des états des tâches et de destruction des tâches. Nous donnons dans le listing 2.3 , la trace de simulation générée par ce niveau de description.
Comme on peut le voir sur cette trace d'exécution, le générateur de stimuli (i.e. l'OS) commence par écrire la description de la tâche dans la mémoire. Dans notre cas, la description de la tâche est stockée à l'adresse 10 (soit 0000000000001010 en binaire sur 16 $bits$). Cette description est constituée du numéro de la tâche, ici 120 (soit 01111000 en binaire sur 8 $bits$) et de deux valeurs 35 et 45 inutilisées ici. Ensuite, un ordre de chargement de tâche est envoyé à la ressource reconfigurable (`load`), accompagné de l'adresse où est stockée la description de la tâche. La zone reconfigurable charge la description de la tâche, ici simplement le numéro de celle-ci. Le générateur

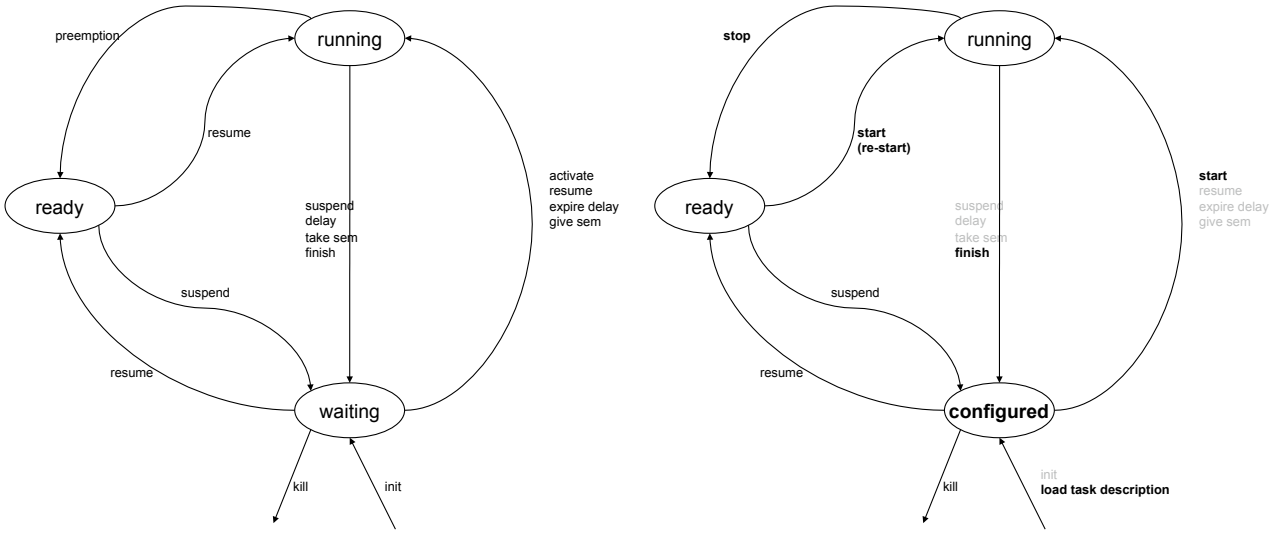


FIG. *2.7 : Diagrammes représentant les différents états que peuvent prendre les tâches. a) Dans un système classique de type processeur, les tâches évoluent entre trois états qui permettent de contrôler leur exécution ainsi que les interactions qu'elles peuvent avoir avec l'environnement et les autres tâches du système. b) Dans une ressource reconfigurable, les possibilités d'évolutions des états des tâches sont, en général, moins nombreuses compte tenu des caractéristiques de fonctionnement de cette zone. En effet, le coût d'une reconfiguration est plus important qu'un "simple" changement de contexte au sein d'un processeur, et cela conduit généralement les concepteurs à ne pas envisager de préemption au sein de cette ressource. Nous considérons, dans un premier temps que la préemption n'est pas possible au sein de cette ressource.*

```
- le chargement de la mémoire avec la description de la
  tâche :
  - (WRITE_MEM, Adresse = 10, Valeur = 120)
  - (WRITE_MEM, Adresse = 11, Valeur = 35)
  - (WRITE_MEM, Adresse = 12, Valeur = 45)
- l'envoi d'un ordre de chargement de tâche à l'ARD :
  - (LOAD_TASK, Valeur = 10)
  - (RESUME_TASK, Valeur = 120)
  - (START_TASK, Valeur = 120)
  - (STOP_TASK, Valeur = 120)
  - (SUSPEND_TASK, Valeur = 120)
  - (RESUME_TASK, Valeur = 120)
  - (START_TASK, Valeur = 120)
  - (SUSPEND_TASK, Valeur = 120)
  - (KILL_TASK, Valeur = 120)
```

Listing 2.2 : Exemple de commandes envoyées à l'ARD afin de lancer l'exécution de tâches dynamiquement. Des commandes de chargement, démarrage, etc ont été définies. Pour chaque commande, l'identifiant de la tâche concernée est précisé afin que l'ARD puisse agir en conséquence.

de stimuli pilote ensuite l'exécution de la tâche en envoyant des commandes de démarrage (`start`), d'arrêt, de terminaison de la tâche (`stop`).

Dans ce niveau de description, aucune initiative n'est prise par le *micro OS* embarqué dans la zone reconfigurable. Ce niveau peut alors permettre au concepteur de vérifier la fonctionnalité de son application.

```
Note: VCD trace timescale unit is set by user to 1e-9 sec.
==> DEBUG at 0 s from <ZoneReconfigurable> : debut d'execution
==> DEBUG at 100 ns from <Stimulus> : 0000000000001010 01111000
==> DEBUG at 100 ns from <Memoire> : ecriture memoire
==> DEBUG at 200 ns from <Stimulus> : 0000000000001011 00100011
==> DEBUG at 200 ns from <Memoire> : ecriture memoire
==> DEBUG at 300 ns from <Stimulus> : 0000000000001100 00101101
==> DEBUG at 300 ns from <Memoire> : ecriture memoire
==> DEBUG at 400 ns from <Stimulus> : 0000000000000000 00000000
==> DEBUG at 400 ns from <Memoire> : haute impedance memoire
==> DEBUG at 400 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 400 ns from <ZoneReconfigurable> : load task description
==> DEBUG at 400 ns from <Memoire> : lecture memoire
    Tache  <120 ; 0> CONFIGURED
==> DEBUG at 500 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 500 ns from <ZoneReconfigurable> : start task
    Tache  <120 ; 0> RUNNING
==> DEBUG at 600 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 600 ns from <ZoneReconfigurable> : stop task
    Tache  <120 ; 0> READY
==> DEBUG at 700 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 700 ns from <ZoneReconfigurable> : suspend task
    Tache  <120 ; 0> CONFIGURED
==> DEBUG at 800 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 800 ns from <ZoneReconfigurable> : resume task
    Tache  <120 ; 0> READY
==> DEBUG at 900 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 900 ns from <ZoneReconfigurable> : start task
    Tache  <120 ; 0> RUNNING
==> DEBUG at 1000 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 1000 ns from <ZoneReconfigurable> : suspend task
    Tache  <120 ; 0> CONFIGURED
==> DEBUG at 1100 ns from <ZoneReconfigurable> : Gestion des evenements de control
==> DEBUG at 1100 ns from <ZoneReconfigurable> : kill task
==> DEBUG at 2000 ns from <Stimulus> : Fin
SystemC: simulation stopped by user.
```

Listing 2.3 : Trace d'exécution obtenue lors de la simulation d'un système composé d'une zone reconfigurable et d'une mémoire contenant le descriptif des tâches à exécuter. Un process *est instancié afin de produire les stimuli pour la zone reconfigurable.*

Description plus précise de l'ARD

Ce premier niveau de description a permis de mettre en place une première structure de base ainsi qu'un environnement de simulation. Nous avons ensuite poursuivi nos travaux en faisant évoluer la description de l'ARD vers une description plus réaliste. Nous avons alors inclus les éléments suivants dans notre modèle :

– gestion de la notion de temps d'exécution des tâches sur l'ARD ;
– gestion simplifiée de la notion de ressources occupées par des tâches ;
– gestion simplifiée de la notion de mémoire nécessaire pour l'exécution des tâches ;
– gestion simplifiée de la notion de bande passante nécessaire pour les échanges des tâches.

Dans ce cas, le micro OS est doté de mécanismes de contrôle qui se chargent, entre autres, de vérifier la possibilité de configurer une tâche affectée par l'OS. Les vérifications réalisées à ce niveau sont très simples, elles peuvent s'exprimer par les règles suivantes :

$$R_{ARD} \geq \sum_{T_i \in ARD} Rt_i \tag{2.1}$$

$$M_{ARD} \geq \sum_{T_i \in ARD} Mt_i \tag{2.2}$$

$$B_{ARD} \geq \sum_{T_i \in ARD} Bt_i \tag{2.3}$$

avec R_{ARD}, M_{ARD} et B_{ARD} respectivement le nombre de ressources, la quantité mémoire et la bande passante totale de la zone reconfigurable. Et Rt_i, Mt_i et Bt_i respectivement le nombre de ressources, la quantité mémoire et la bande passante nécessaires pour la tâche i. Les vérifications de ces règles sont effectuées sur toutes les tâches présentes dans l'ARD au moment de la réception de l'ordre de chargement d'une nouvelle tâche.
Ces règles sont nécessaires, mais en aucun cas suffisantes pour assurer qu'une tâche peut être assignée à la zone reconfigurable. Concernant le nombre de ressources, une règle prenant un compte la forme de l'instanciation de la tâche sur l'ARD est un élément important. L'expression ci-dessous exprime cette règle :

$$(W_{max}, H_{max}) \geq (W_i, H_i) \tag{2.4}$$

Avec (W_{max}, H_{max}) la taille de la plus grande surface rectangulaire disponible dans la zone reconfigurable. Et (W_i, H_i) la taille de la tâche i.
La règle permettant de s'assurer qu'il reste suffisamment de mémoire sur la zone reconfigurable souffre des mêmes problèmes de précision et demanderait à être affinée en fonction notamment de la distribution de la mémoire au sein de cette zone.
Finalement, la règle permettant de vérifier que l'ARD dispose d'une bande passante suffisante pour supporter les échanges des tâches est très grossière. Cette règle pourrait être précisée en exprimant les bandes passantes internes et externes de cette ressource et en considérant, en fonction du placement des tâches, que celles-ci nécessitent des échanges internes ou des échanges externes à l'ARD.

2.4.2 Modélisation d'une plate-forme minimale intégrant ARD et OS

Les modélisations précédentes n'intègrent pas de véritable OS, et de ce fait ne permettent pas de simuler facilement l'exécution d'une application. Aussi, nous avons étudié la modélisation d'une plate-forme minimale contenant un processeur et une zone reconfigurable. De plus, cette modélisation a permis d'étudier l'inter-opérabilité des modèles décrits au sein du projet OverSoc par des équipes différentes. Dans ce système, le générateur de stimuli est remplacé par un OS s'exécutant sur un processeur (voir la figure 2.8).
Nous avons testé ce système soumis à l'exécution d'une application, découpée en plusieurs tâches dont certaines s'exécutent sur le processeur et d'autres sur l'ARD. Les tests réalisés sur cette plate-forme ont permis de montrer que les deux modèles développés de part et d'autre, sont inter-opérables.
La principale difficulté rencontrée pour le test de cette plate-forme a concerné la description de l'application et la capacité à explorer différents découpages possibles de cette dernière. Nous avons alors défini une description XML de la plate-forme contenant à la fois une description de l'architecture ainsi qu'une description de l'application sous forme de tâches. Les tâches sont décrites par leurs caractéristiques classiques, mais aussi par leurs dépendances les unes par rapport aux autres. Le modèle d'exécution SystemC a été modifié afin de prendre en compte cette description.
Le format XML[5] retenu pour modéliser la plate-forme est donné dans le listing de la figure 2.4.
L'utilisation de cette description permet de découpler la phase de spécification de l'architecture et de l'application, des phases de compilation et de simulation. L'outil OverSoc permet donc au concepteur de spécifier son objectif d'exploration, l'outil l'aidant alors à spécifier les paramètres nécessaires (dimensionnement de la plate-forme, distribution des services, ...) pour l'obtention des résultats.

[5]XML : Extensible Markup Language

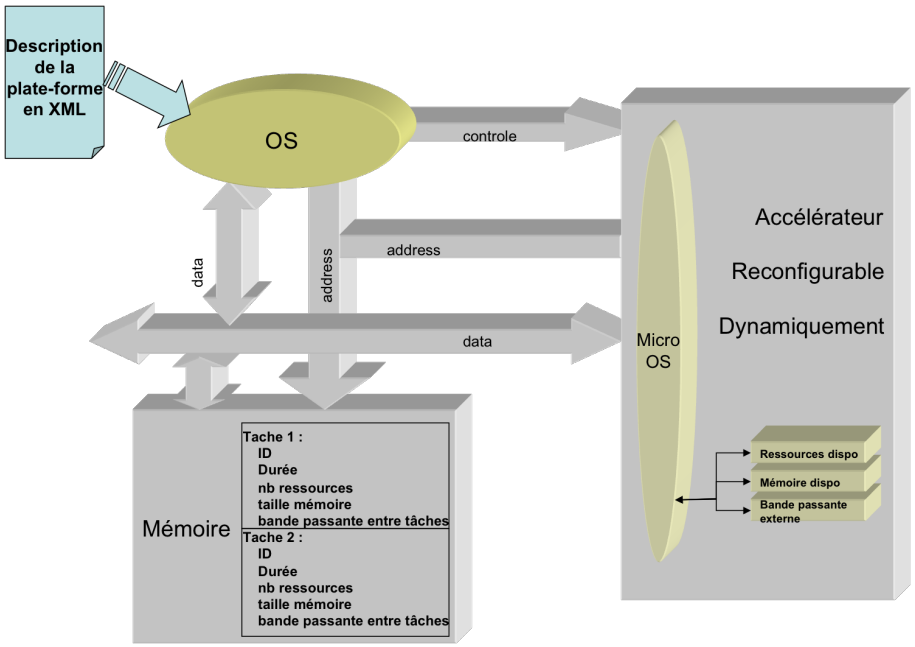


FIG. *2.8 : Cette modélisation intègre l'OS et l'ARD. L'OS prend en charge la gestion des tâches dont la description est donnée en XML. Les tâches logicielles sont créées et exécutées sur la ressource processeur, non représentée mais implicitement présente pour l'exécution de l'OS. Les tâches matérielles sont créées au sein de l'OS, puis chargées et exécutées dans l'ARD.*

Evolutions envisagées

L'objectif de nos travaux consiste à décrire de façon plus précise la ressource reconfigurable. Nous envisageons donc d'affiner ce modèle en incluant au fur et à mesure des informations structurelles de la zone reconfigurable. Ce raffinement s'accompagnera d'un accroissement de la complexité du "contrôleur" inclus dans la zone reconfigurable, celui-ci prenant en charge petit à petit l'exécution de certains services du système d'exploitation. L'OS sera alors progressivement déporté/distribué vers cette ressource de traitement.
Nos futurs travaux s'attacheront à modéliser la phase de placement des tâches au sein de l'ARD. Des algorithmes de type *first fit* ou *best fit* seront en premier lieu implémentés. Puis nous étudierons et instancierons des algorithmes plus efficaces, notamment des algorithmes de type MER (Maximum Empty Rectangle) [70].
Finalement, des solutions concernant la préemption et la migration des tâches matérielles seront envisagées afin de proposer une gestion plus efficace de la ressource reconfigurable.

2.4.3 Etude de l'interconnexion au sein d'un ARD

Comme nous l'avons mentionné plus haut, le placement des tâches au sein de la zone reconfigurable est non prédictible et dépend de l'état de "remplissage" au cours du temps. Une contradiction majeure apparaît alors entre la non prédictibilité du placement des tâches et le besoin de satisfaire les contraintes d'exécution de ces tâches. En effet, rappelons que dans le modèle proposé, le rôle de la zone reconfigurable consiste à prendre en charge l'exécution de tâches de calcul intensif. Il s'agit alors, en règle générale, de tâches dont la latence et/ou la cadence des traitements sont contraintes. La méconnaissance du placement d'une tâche au sein de la zone reconfigurable pose un problème important en terme d'acheminement des données vers/depuis les tâches.
La problématique des communications entre tâches ne se pose pas seulement au sein de la zone reconfigurable, mais couvre l'ensemble du SoC. En effet, les différents éléments du SoC vont exprimer des besoins en communication, et les besoins pour ces échanges globaux sont de natures très variées. Ils découlent des différentes tâches applicatives qui doivent respecter des qualités de services assez hétérogènes. En effet, certaines tâches peuvent très bien se satisfaire de temps de réponse souples, alors que d'autres ne peuvent supporter la moindre violation de temps de réponse. Pour assurer cette hétérogénéité des échanges de données, l'introduction de NoC au sein des

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <Platform>
<Application>
    <Name> AppliTest </Name>
    <Task>
        <Name> Main </Name>
        <Priority> 10 </Priority>
        <Resource> 0 </Resource>
        <Memory> 200 </Memory>
        <Type> SOFT_TARGET  </Type>
        <ExecTime> 1000 </ExecTime>
        <Next> T1 </Next>
        <Next> T2 </Next>
        <Next> T3 </Next>
    </Task>
    <Task>
        <Name> T1 </Name>
        <Priority> 20 </Priority>
        <Resource> 500 </Resource>
        <Memory> 400 </Memory>
        <Type> SOFT_TARGET </Type>
        <ExecTime> 2000 </ExecTime>
    </Task>
    <Task>
        <Name> T2 </Name>
        <Priority> 30 </Priority>
        <Resource> 600 </Resource>
        <Memory> 500 </Memory>
        <Type> HARD_TARGET </Type>
        <ExecTime> 1000 </ExecTime>
        <Previous> Main </Previous>
    </Task>
    <Task>
        <Name> T3 </Name>
          ...
          ...
</Application>

<Architecture>
    <Name> Test1 </Name>
    <Processor>
        <Name> Proc </Name>
    </Processor>
    <ARD>
        <Name> Ard1 </Name>
        <Resources> 1234 </Resources>
        <Memory> 9876 </Memory>
        <ArdWidth> 10 </ArdWidth>
        <ArdLenght> 20 </ArdLenght>
    </ARD>
</Architecture>
</Platform>
```

Listing 2.4 : Fichier XML décrivant l'application sous forme de tâches ainsi que l'architecture. Les tâches applicatives sont décrites par leurs caractéristiques (périodes, priorité, besoin en mémoire, etc) et les dépendances entre tâches sont spécifiées au travers des tags XML `next` *et* `previous`*. L'architecture est décrite par ses ressources. Dans le cas illustré, les informations concernant la mémoire disponible, le nombre de ressources d'exécution ainsi que la largeur et la hauteur de la zone reconfigurable sont spécifiées.*

SoC semble une solution intéressante [71].

Les solutions proposées, par exemple dans [72, 73], permettent d'assurer une flexibilité importante et les algorithmes de routage de type *Best effort* sont "satisfaisants" pour une partie de l'application. Malheureusement, pour les parties les plus critiques, ce type de réseaux ne fournit aucune garantie. Pour tenter de contrer ce problème, certains auteurs ont proposé de doter le système d'exploitation de capacité de réservation de la bande passante sur le réseau [65, 66]. Il s'agit par exemple de construire le réseau à partir de la connaissance des échanges réalisés par les tâches de l'application. Une solution mixant du *best effort* et du *guarantee throughput* permet de définir la topologie, de dimensionner le réseau et dans certains cas de produire une description simulable et/ou synthétisable du NoC.

Si ces solutions semblent prometteuses pour gérer efficacement les communications au niveau système, elles ne s'intéressent pas aux échanges à réaliser au sein de la zone reconfigurable du SoC. Une solution consistant à prolonger le NoC au sein de la zone reconfigurable pourrait être une première approche et simplifierait l'interfaçage entre ces deux mondes. Toutefois, compte tenu du type de tâches que devra exécuter la zone reconfigurable, nous pensons qu'il est peu opportun d'englober les échanges propres à cette zone au niveau du réseau du SoC. Dans

ce contexte, nous avons entamé des travaux visant à proposer une infrastructure de communication propre à la zone reconfigurable. Notre étude a porté sur la définition d'une architecture flexible et performante ainsi que sur la définition du contrôle de celle-ci.

Dans les paragraphes suivants, nous présentons notre infrastructure de communication en illustrant notamment par des premiers résultats de prototypage, puis nous présentons les fonctionnalités du contrôleur qui devra prendre en charge la configuration, *on-line*, de cette structure.

Infrastructure de communication au sein de l'ARD

Il convient dans un premier temps de spécifier les caractéristiques que nous ciblons pour cette fonctionnalité. Notre point de départ est un ensemble de tâches dont au moins une description d'implémentation est disponible. À partir de ces descriptions, des contraintes d'exécution, des contraintes de charge de calcul, etc, le système d'exploitation doit pouvoir *placer et ordonnancer*, *on-line*, ces tâches sur la zone reconfigurable. L'infrastructure doit donc offrir suffisamment de flexibilité à l'OS pour que celui-ci puisse "construire" un média de communication entre les tâches applicatives. Cette infrastructure doit aussi permettre la mise en place de chemins de communication performants entre tâches lorsque la bande passante et/ou le débit le nécessite.
Il apparaît que deux caractéristiques principales doivent être mises en place :
– la dynamicité de la topologie : cette caractéristique est nécessaire pour que le système puisse être adapté, *on-line*, aux besoins des tâches applicatives à exécuter ;
– la possibilité de créer des liens privilégiés : ce point est primordial si l'on souhaite conserver le bénéfice de l'accélération de la zone reconfigurable et ne pas ralentir les tâches par des congestions de l'infrastructure d'échange des données.
L'infrastructure doit, autant que faire se peut, être la plus transparente possible pour le développeur de l'application. En effet, il est important de découpler les phases de spécification, d'implémentation et d'optimisation des tâches sur la zone reconfigurable, des phases de spécification et d'implémentation de l'infrastructure de communication.

Compte tenu des caractéristiques et des contraintes énoncées ci-dessus, nous avons entamé des travaux sur la définition d'une structure adaptée aux communications sur un circuit de type FPGA supportant la reconfiguration dynamique. Ces travaux ont donné naissance à une structure basée sur un *FatTree*, mais dont l'un des objectifs a été de limiter le coût matériel de l'implémentation. La proposition se base alors sur un *FatTree* dont les feuilles de l'arbre permettent la connexion des tâches applicatives et dont le sommet de l'arbre accepte aussi des éléments. Pour conserver les bonnes propriétés du *FatTree*, les tâches connectées au réseau DRAFT[6] doivent respecter quelques contraintes. Le réseau a été prototypé sur circuit FPGA Virtex 5 de Xilinx. Une description complète du réseau est donné dans [A2, A16]. Ce réseau est l'une des briques du projet FosFor qui vise à développer un OS distribué assurant la gestion de zones reconfigurables.

Définition des communications entre tâches

Dans le cadre de ce projet, le réseau DRAFT a été livré et servira de base pour la réalisation du démonstrateur final. Nous poursuivons actuellement nos travaux en couvrant les aspects gestion du réseau. En effet, si actuellement le réseau est parfaitement bien spécifié, sa gestion notamment au travers du système d'exploitation, n'est pas complètement définie. Pour parvenir à définir cette gestion, il est important de considérer une architecture générique mettant en œuvre ce réseau et d'identifier les différents scénario de communication possible. Cette architecture a été définie dans le cadre du projet FosFor et est présentée dans la figure 2.9.
Sur la base de cette structure, nous avons défini les différents scénarii de communication entre tâches. Ceux-ci seront toujours initialisés par deux tâches, l'une émettrice, l'autre réceptrice et le code typique d'un échange de ce type est donné dans le listing de la figure 2.5.

[6]DRAFT : Dynamic Reconfiguration Adapted Fat-Tree

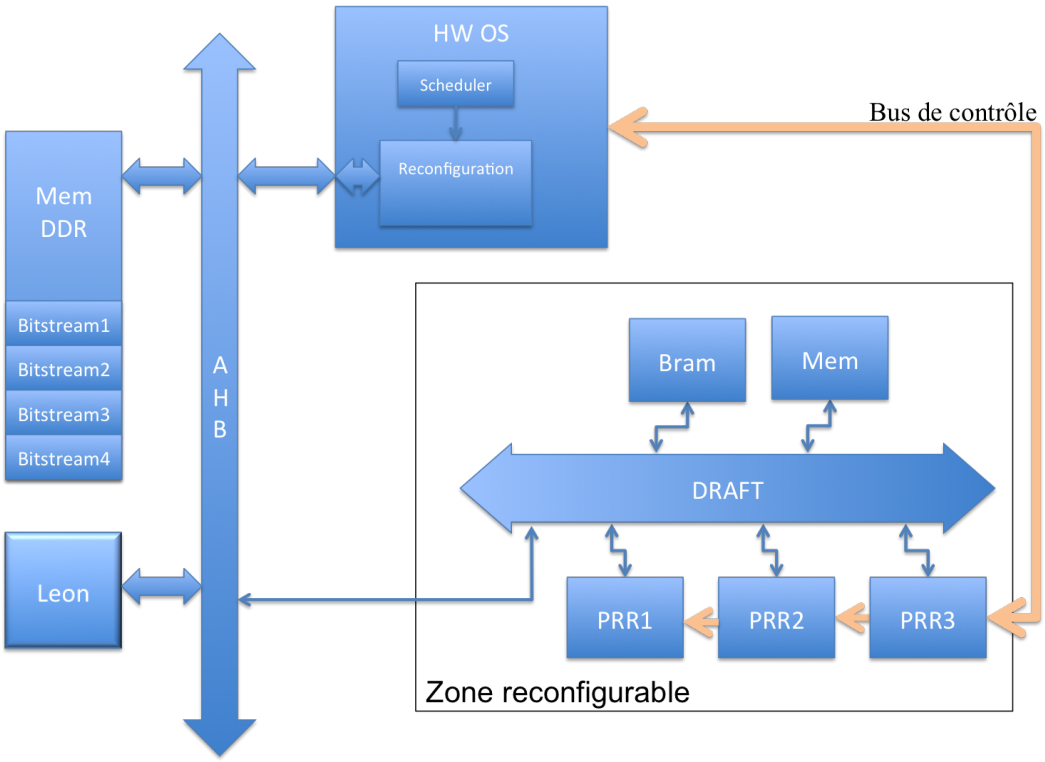


FIG. 2.9 : *Architecture définie dans le projet FosFor. Cette architecture repose sur la mise en place d'une zone reconfigurable dans laquelle des tâches pourront être instanciées au sein des PRR. Cette zone reconfigurable accueille le réseau DRAFT proposé par notre équipe. Ce réseau sera la base des communications entre les tâches matérielles ainsi qu'entre tâches matérielle et logicielle. Le réseau transporte des messages en routant ceux-ci à partir des informations fournies en entête. DRAFT est basé sur un réseau de type FatTree dont le sommet de l'arbre peut accepter des connexions d'éléments de type mémoire par exemple.*

```
Void T1() {
        ...
        IdCh = openChannel(''ChannelName", ''w'');
        ...
        PtrSrc[0] = data0 ;
        PtrSrc[1] = data1 ;
        PtrSrc[2] = data2 ;
        ...
        SendData(IdCh, &PtrSrc[0], 3)
        ...
        close(IdCh) ;
        ...
}
```

a)

```
Void T2() {
        ...
        IdCh = openChannel(''ChannelName", ''r'');
        ...
        ReceiveData(IdCh, &PtrDest[0])
        ...
        data0 = PtrDest [0] ;
        data1 = PtrDest [1] ;
        data2  = PtrDest [2] ;
        ...
        close(IdCh) ;
        ...
}
```

b)

Listing 2.5 : Codes classiques d'échanges de données entre deux tâches. a) La tâche T_1 ouvre le canal de communication en écriture, prépare les données à émettre, puis appelle la fonction `send`*. b) De son côté, la tâche T_2 ouvre le canal de communication en lecture, puis effectue la réception des données. Une fois l'échange terminé, les deux tâches ferment le canal de communication.*

La réalisation de l'échange entre les tâches T_1 et T_2 présentées dans le listing de la figure 2.5 peut suivre plusieurs scénarii et cela dépend notamment de la présence ou non des tâches lors des appels aux fonctions `send` et `receive`. Les diagrammes de séquences présentés dans la figure 2.10 illustrent deux cas. Le diagramme 2.10.a présente le cas où les deux tâches appellent les fonctions `send` et `receive` au même instant (d'un point de vue de l'OS). Dans ce cas, l'OS place les deux tâches directement en communication et l'échange peut alors être

réalisé directement entre les deux tâches matérielles au travers de DRAFT. Le diagramme 2.10.b présente le cas où la tâche émettrice T_1 n'est pas présente au sein de la zone reconfigurable au moment où la tâche réceptriceT_2 effectue l'appel à la fonction `receive`. Dans ce cas, la tâche réceptrice T_2 est placée en attente (tâche suspendue) jusqu'au moment où la tâche émettrice T_1 effectue l'appel à la fonction `send`.

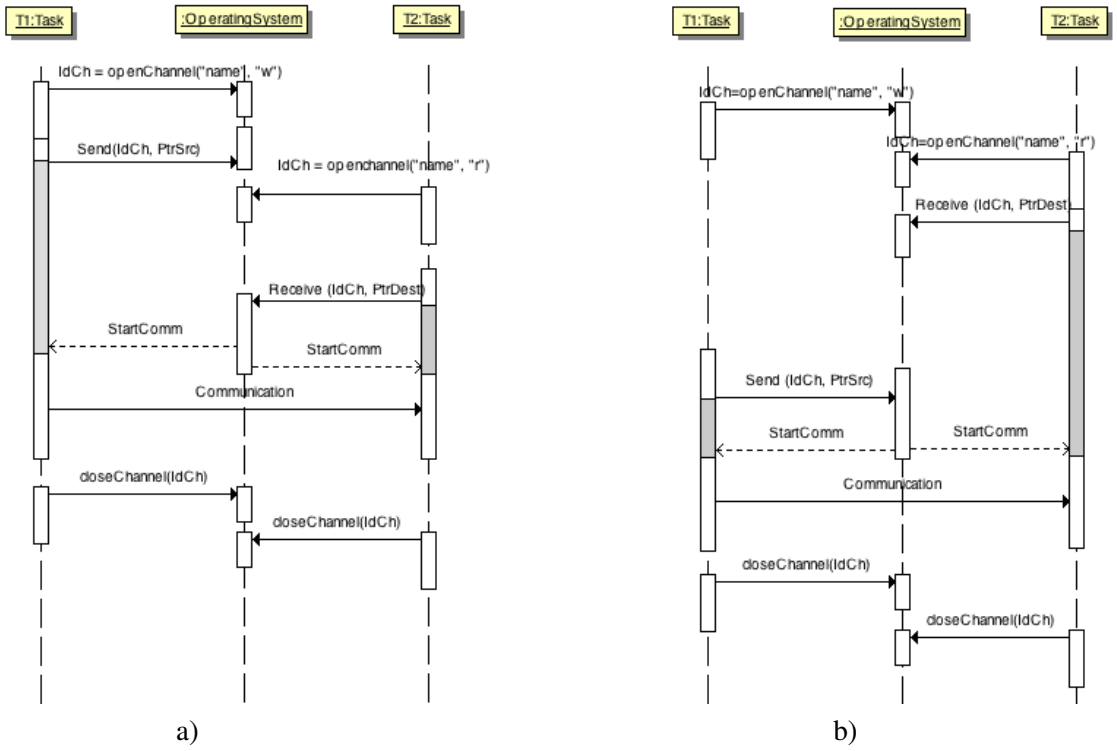


FIG. *2.10 : Diagrammes de séquence d'un échange entre deux tâches matérielles T_1 et T_2. a) Dans le premier cas, les deux tâches appellent les fonctions* `send` *et* `receive` *au même moment et la communication peut être assurée sans difficulté ; b) Dans le second cas, la tâche T_2 est suspendue à la suite de l'appel à la fonction* `receive`*, cette suspension est maintenue jusqu'à l'appel, par la tâche T_1 de la fonction* `send`*.*

D'autres scénarii d'échanges ont été proposés et formalisés. Il s'agit par exemple des cas de figures pour lesquels la tâche réceptrice n'est pas présente au moment où la tâche émettrice effectue l'appel à la fonction `send`. Dans ce cas, nous proposons que les données à échanger soient stockées dans l'une des mémoires du système. Compte tenu de l'architecture proposée dans la figure 2.9, plusieurs possibilités sont offertes à l'OS, nous les détaillons ci-dessous.

– La première solution pour le stockage des données consiste à allouer de l'espace mémoire dans la mémoire partagée et globale. La tâche matérielle émettrice effectue l'appel système `send`. Le système d'exploitation lui renvoie l'adresse réseau du port de DRAFT connecté vers le bus AHB global. La tâche émettrice envoie ensuite ces données en précisant cette adresse réseau, sans avoir connaissance de l'absence de la tâche destinatrice (voir figure 2.11.a). Lorsque la tâche émettrice effectuera l'appel à la fonction `receive`, alors le système d'exploitation devra déclencher une lecture en mémoire des données.
– La seconde solution consiste à allouer de l'espace mémoire dans une mémoire partagée mais localement présente dans la zone reconfigurable. Le fonctionnement est identique au précédent, c'est-à-dire que la tâche émettrice reçoit une adresse réseau et effectue son envoi de message. Lorsque la tâche réceptrice effectuera l'appel à la fonction `receive`, alors le système d'exploitation déclenchera une lecture des données en mémoire locale.
– La troisième solution consiste à exploiter les mémoires incluses dans les zones PRR[7] pour configurer temporairement ces PRR en ressource de stockage. Cette solution n'est envisageable que si chaque PRR dispose

[7]PRR : Partial Reconfigurable Region

d'une configuration (bitstream) assurant cette fonctionnalité. Il est ici utile de rappeler que les PRRs sont définis pour les besoins des tâches applicatives, mais qu'il est possible d'envisager pour chaque PRR de générer une tâche supplémentaire dont le comportement consiste à reproduire une mémoire. Ainsi, pour un PRR donné P_1, défini pour recevoir un ensemble de tâches $\varepsilon_1 = \{T_1, T_2, \ldots, T_n\}$, la quantité de mémoire allouée dans le PRR P_1 est au moins égale au maximum des quantités mémoire utile pour chaque tâche, c'est-à-dire $maxmem = max(mem_{T_1}, mem_{T_2}, \ldots, mem_{T_n})$ (avec mem_{T_i} la mémoire nécessaire pour la tâche T_i). L'ensemble des tâches configurables sur le PRR P_1 est alors augmenté de un, en lui ajoutant une "tâche mémoire" Tm_1 disposant d'un espace mémoire de taille $maxmem$. Sous cette hypothèse, il est possible de doter le système d'exploitation d'un mécanisme lui permettant d'allouer un PRR libre (non configuré pour une tâche à un instant donné), afin d'assurer le stockage temporaire des données. Dans ces conditions, la tâche émettrice n'a pas conscience de l'absence de la tâche réceptrice et le système d'exploitation doit veiller à maintenir le PRR de stockage en place jusqu'à ce que la tâche réceptrice effectue sa lecture de données (voir figure 2.11.b).

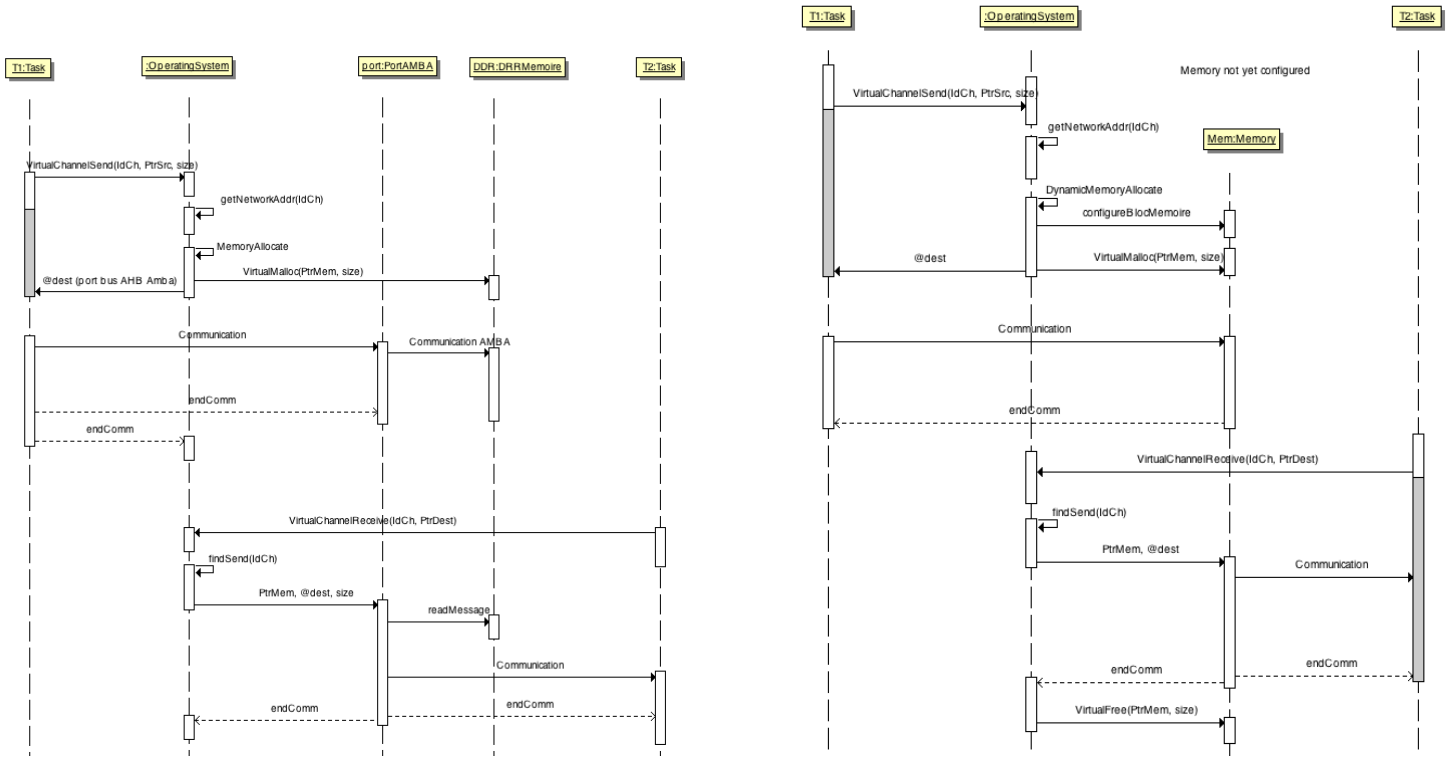


FIG. *2.11 : Diagrammes de séquences pour une communication entre deux tâches (dont l'une au moins est matérielle) avec un stockage temporaire dans une mémoire. a) Dans ce diagramme de séquences, le stockage s'effectue dans la mémoire partagée et globale. Le système d'exploitation "trompe" la tâche émettrice en lui fournissant une adresse réseau correspondant au port AHB ; b) Pour ce diagramme de séquences, les données sont stockées dans la mémoire contenue dans un PRR. Le système d'exploitation "trompe" la tâche émettrice en lui fournissant l'adresse réseau d'un PRR préalablement configuré en espace de stockage temporaire.*

Pour ce qui concerne le projet FosFor, l'ensemble de ces cas de figures ne sera pas implémenté dans le démonstrateur. Seules les solutions de stockage dans les mémoires partagées globale ou locale seront déployées.

2.5 Reconfiguration dynamique et consommation

Les sections précédentes montrent comment la reconfiguration dynamique peut être exploitée afin d'obtenir une utilisation plus efficace de la ressource reconfigurable. Toutefois, comme nous l'avons mentionné auparavant, cette

reconfiguration dynamique a un coût, notamment temporel qui est loin d'être négligeable. Elle présente aussi une consommation d'énergie non négligeable et c'est souvent une contrainte mise en avant pour ordonnancer des tâches non préemptibles sur ce type de ressource.
Indépendamment de la préemption des tâches, la mise en œuvre de la reconfiguration dynamique doit être caractérisée dans le but d'offrir à l'*operating system* les moyens de décider, `on-line`, quelles tâches doivent être assignées sur quelles ressources d'exécution.
Au sein du projet Open-People, nous avons proposé de travailler sur la caractérisation des circuits reconfigurables. L'objectif poursuivi consiste à construire des modèles de consommation de ces circuits et de proposer ensuite des techniques d'optimisation au niveau système de l'ordonnancement spatio-temporel des tâches. Dans ce sens, nous avons entamé des travaux de thèse en octobre 2009. De plus, nous avons démarré des campagnes de mesures sur circuits reconfigurables très faible consommation. Ces travaux nous permettront dans quelques mois de fournir les premiers modèles de consommation à partir de mesures réelles sur cibles matérielles. Les cibles que nous avons actuellement retenues sont d'une part un circuit Igloo de la société Actel et un circuit Virtex V5 de la société Xilinx.
Notons que ces travaux ont un point commun avec ceux menés dans le projet OverSoC dans le sens où un objectif de modélisation est clairement identifié. Dans le projet Open-People, la modélisation concerne les aspects consommation et couvrira l'ensemble des éléments constituant un MPSoC. Le langage de modélisation d'ores et déjà retenu pour le projet Open-People est AADL[8].
Nos travaux actuels progressent sur les deux aspects que sont la construction de modèles de consommation ainsi que l'analyse des techniques de modélisation des architectures reconfigurables avec le langage AADL. Ces travaux sont réalisés par Robin BONAMY qui a débuté ses travaux de thèse en octobre 2009 et Ferhat ABBAS, stagiaire de master depuis février 2010. Notons de plus que ces travaux s'accompagnent d'une étude conjointe entre la société InPixal (sous-traitante du laboratoire LabSticc au sein du projet Open-People) et notre équipe de recherche. Il s'agit dans cette étude de travailler à la modélisation AADL d'une application de InPixal dans l'objectif d'effectuer un transfert de compétences au sein de cette société. Il s'agit aussi pour notre équipe de monter en compétences sur les concepts de modélisation au travers de la maîtrise du langage de modélisation AADL.

2.6 Conclusion et perspectives

Les travaux que nous menons sur le thème des RSoC reconfigurables s'attachent essentiellement à des aspects modélisation. Ces travaux sont en grande partie intégrés dans les projets ANR OverSoC et FosFor. Au sein de ces deux projets, nous avons étudié et poursuivons l'étude de la modélisation de la zone reconfigurable et du service de communication au sein de cette zone. Nous avons montré qu'il est possible de raffiner l'un des éléments de la plate-forme RSoC sans que cela nécessite un raffinement de tous les éléments. Cette approche est intéressante puisqu'elle permet au concepteur de se concentrer sur un élément particulier afin d'en étudier les différentes implémentations. Pour mener à bien ce travail, nous avons établi une modélisation UML d'un RSoC. Cette modélisation a été un support d'échange et de cohérence notamment pour le projet OverSoC. Nous pouvons noter que la méthodologie et l'outil OverSoC se sont appuyés sur cette modélisation UML qui offre une approche par niveaux très clairement identifiés.

Concernant les travaux liés aux communications, nous avons établi une première solution dont nous avons montré qu'elle exhibe des caractéristiques intéressantes d'un point de vue implémentation. Cette proposition de topologie a été accompagnée d'une proposition de gestion au travers du service de communication du système d'exploitation. La suite de ces travaux concerne le développement d'un démonstrateur permettant de mettre en œuvre les différents scénarii de communications proposés.
Ces différents travaux ont été menés avec l'aide d'élèves ingénieur de l'Enssat au travers de la réalisation de projets technologiques que j'ai encadrés. La modélisation et la définition de la structure d'interconnexion a été définie dans le cadre de stages de master dont j'ai assuré l'encadrement. Ces travaux sont poursuivis en thèse depuis oc-

[8] L'acronyme AADL a eu une première signification qui était Avionics Architecture Description, mais est devenu Architecture Analysis and Design Language au travers de la normalisation par la SAE (Society of Automotive Engineers)

tobre 2008 par Ludovic DEVAUX, ils sont pleinement intégrés dans le projet ANR FosFor.

La modélisation du RSoC que nous avons réalisée est pour l'instant de niveau comportemental. Il est évident que ce niveau ne peut être suffisant dès lors que l'on souhaite affiner l'exploration de la plate-forme. En effet, l'étude de la distribution des services sur la zone reconfigurable du RSoC doit pouvoir prendre en compte des informations plus précises. On pourra noter par exemple la prise en compte des formes de tâches au sein de la zone reconfigurable afin de proposer des algorithmes de placement efficaces. La planification des phases de reconfiguration doit aussi être étudiée afin de rendre celles-ci transparentes vis-à-vis de l'exécution des tâches.
Compte tenu de la complexité du contrôle que nous envisageons d'étudier, nous pensons qu'une partie de ce contrôle devra être implémentée directement sur la cible reconfigurable. L'étude de l'implémentation matérielle du contrôleur (qui peut être vu comme un ensemble de services de l'OS) devra alors être menée.

Chapitre 3

Ordonnancement de tâches au sein d'un SoC multi-processeurs hétérogène

La conception de SoC s'appuie généralement sur un modèle architectural contenant un nombre croissant de ressources de calcul. L'un des objectifs est alors de répartir la charge de calcul de l'application sur ces ressources afin notamment de répondre aux contraintes applicatives : temps de réponse, consommation, etc. Conjointement à l'architecture, l'application doit être décomposée en tâches dont la granularité doit être la plus proche possible de la granularité offerte par l'architecture. Par la suite, ces tâches doivent être placées/instanciées sur les différentes unités disponibles, sachant qu'une tâche peut être décrite (et donc exécutable) pour plusieurs d'entre elles. Les cibles d'exécution sont généralement de natures et de caractéristiques (temporelle notamment) différentes, on qualifie alors le système global de "machine" hétérogène. La complexité globale de fonctionnement de ces architectures repose en grande partie sur l'hypothèse de disponibilité d'un système d'exploitation permettant de gérer la répartition spatiale et temporelle des tâches au sein du circuit. On parle alors de placement des tâches sur les cibles d'exécution et d'ordonnancement de ces tâches sur les différentes cibles. L'ordonnancement consiste à définir les instants durant lesquels les tâches s'exécutent. Dans le contexte des architectures hétérogènes, l'ordonnancement est une problématique difficile, pour laquelle l'obtention d'une solution optimale n'est pas assurée.
La complexité du problème ainsi que les contraintes temporelles (liées aux aspects temps réel des applications) ont conduit à des travaux visant à étudier l'implantation de services d'un *operating system* sous forme matérielle. Des solutions mixant logiciel et matériel ont été proposées pour implémenter des noyaux d'OS [74, 75]. Certains travaux se sont concentrés sur la partie ordonnancement en proposant des implémentations matérielles [76]. Enfin, d'autres travaux ont étudié l'implémentation de la "presque" totalité de l'OS sous forme matérielle [77, 78, 79].

Parmi les nombreuses solutions proposées pour tenter de résoudre la problématique de l'ordonnancement, l'une d'elles repose sur les réseaux de neurones [80]. Le modèle sous-jacent est basé sur la proposition de Hopfield [81] à partir de laquelle les auteurs de [82] proposent une règle de construction permettant de définir les poids de connexions entre neurones ainsi que les poids des entrées. Dans [83], les auteurs proposent une modélisation par réseaux de neurones du problème d'ordonnancement pour architectures homogènes. Cette proposition a été étendue vers les architectures hétérogènes dans [84, 1]. Les principales limitations de ces dernières propositions concernent le nombre important de neurones pour la modélisation du problème ainsi que la nécessité de réinitialiser un grand nombre de fois le réseau pour s'assurer de sa convergence vers une solution correcte.

Pour prendre en compte la nature multi-cibles de l'instanciation des tâches, nous avons proposé des évolutions pour cette modélisation. La première évolution est basée sur l'utilisation de neurones inhibiteurs. Cette modélisation ne respecte plus les règles de construction classiques définies par Hopfield, en particulier la symétrie de la matrice de connexions entre neurones, mais nous montrons que ce modèle permet une limitation importante du nombre de neurones nécessaires pour représenter le problème à résoudre, et donc une diminution du temps de convergence.

En plus de la nature multi-cibles de l'instanciation des tâches, nous avons proposé une technique pour prendre en charge l'hétérogénéité des cibles d'exécution. Notre proposition permet la prise en compte d'une fonction de

coût d'exécution sur les différentes cibles d'exécution. Notre solution repose principalement sur l'introduction d'une notion de mémoire au sein même du neurone. L'idée générale consiste alors à accorder à une tâche des probabilités d'instanciation sur les cibles d'exécution inversement proportionnelles aux coûts d'exécution de cette tâche sur ces cibles. La notion de coût peut inclure une notion de consommation énergétique par exemple, et le réseau de neurones convergera en priorité vers une solution limitant l'énergie globale du système.
La présence d'une zone reconfigurable au sein des SoC complexifie la problématique de l'ordonnancement. En effet, cette zone peut éventuellement accueillir simultanément plusieurs tâches pour peu qu'il existe suffisamment de place sur la zone reconfigurable. Pour parvenir à modéliser le problème de l'ordonnancement sur cette cible d'exécution, nous proposons une construction du réseau de neurones tenant compte de cette contrainte. Nous montrons que cette structure permet bien de gérer les différentes configurations possibles de placement sur la zone reconfigurable.
Enfin, ayant pour objectif d'étudier une implémentation matérielle du service d'ordonnancement, nous avons étudier une architecture permettant d'envisager une implémentation efficace. Il s'agit ici de fournir l'architecture du neurone et du réseau global. Nous présentons ces premiers travaux en montrant notamment comment nous pouvons exploiter la structure particulière de notre réseau afin de limiter au maximum la complexité matérielle de celui-ci.

Finalement, nous terminons ce chapitre par une présentation des travaux actuels que nous menons sur la problématique du placement des tâches sur la zone reconfigurable par l'utilisation de réseaux de neurones. Plusieurs pistes ont été explorées avant d'aboutir à une solution qui nous semble actuellement la plus prometteuse. Nous faisons une brève présentation du principe de fonctionnement sachant que cette piste doit encore être affinée.

3.1 Méthodologie pour la résolution de l'ordonnancement de tâches par réseaux de neurones

La problématique générale de l'ordonnancement de tâches au sein d'un SoC relève directement d'un problème d'optimisation. Il s'agit en effet de déterminer les cycles d'ordonnancement ainsi que les cibles d'exécution pour chaque tâche et cela sous contraintes. Globalement le problème peut être énoncé de la façon suivante :

Problématique : Soit un ensemble de tâches $\mathcal{T}$ à ordonnancer sur un ensemble de ressources $\mathcal{R}$ dans l'intervalle de temps $[0; I]$. Il s'agit de définir conjointement, et pour chaque tâche, l'intervalle de temps durant lequel la tâche est instanciée, ainsi que la ressource prenant en charge cette instance.

Typiquement, dans le contexte des systèmes sur puce, une cible d'exécution classique (cœur de processeur ou de DSP, bloc IP) ne peut prendre en charge qu'une seule tâche par cycle d'ordonnancement. En effet, les cœurs de processeurs embarqués actuellement dans les SoC n'exécutent qu'un seul *thread* à chaque cycle. Nous limiterons, dans un premier temps, notre étude à ce modèle d'exécution bien que notre proposition devrait permettre de prendre en charge une exécution *multi-threads* et/ou l'exécution de plusieurs tâches sur une même cible, comme cela est possible sur une zone reconfigurable par exemple.
D'une façon générale, pour une problématique d'optimisation, la méthodologie développée dans [85] consiste à résoudre successivement les trois problèmes suivants :

1. Rechercher une représentation du problème sous forme de réseau de neurones pour lequel les états des neurones représentent les variables du problème à résoudre ;
2. Déterminer une fonction (dite fonction d'énergie) telle que son expression soit minimale (égale à 0) lorsque les états des neurones correspondent à une solution recherchée ;
3. Calculer les poids des interconnexions et des entrées des neurones à partir de la fonction précédente.

Une fois le réseau défini, il a été démontré que sous certaines conditions, l'évolution de celui-ci va naturellement conduire à la minimisation de la fonction d'énergie [86]. On dit alors que le réseau converge vers un état stable, et cet état est à énergie minimale, égale à 0.

De la première étape dépend la facilité à définir une fonction d'énergie, mais aussi la facilité à exploiter les solutions produites. La complexité du problème d'optimisation à résoudre rend souvent la définition de la fonction d'énergie délicate, il est alors nécessaire de raisonner par association/combinaison/addition de "règles" pour aboutir à une solution. Ces deux étapes sont donc très importantes, mais ne doivent pas occulter l'objectif final de nos travaux qui concerne l'implémentation de ce service d'ordonnancement des tâches. Nous avons continuellement conservé à l'esprit cet objectif afin de remettre en cause nos propositions et de proposer des solutions adéquates, notamment en essayant de limiter au maximum le nombre de neurones.

3.1.1 Représentation initiale du problème

Pour le problème d'ordonnancement mono-processeur, la modélisation classique proposée est schématisée sur la figure 3.1.a. Cette modélisation a été proposée dans [83]. Sur cette figure, chaque point (ou neurone) représente un

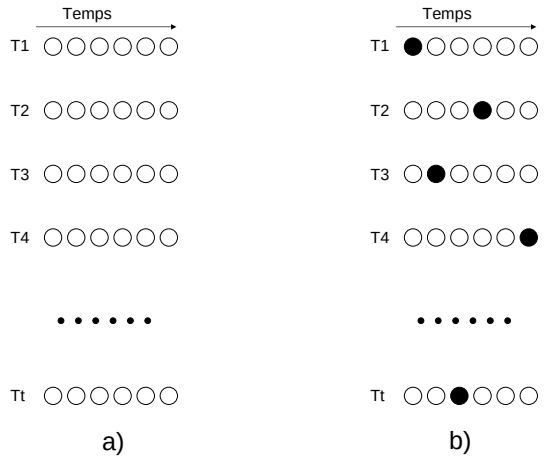


FIG. *3.1 : a) Représentation initiale du problème d'ordonnancement pour systèmes mono-processeur. Chaque tâche se voit attribuer une ligne de neurones. Le nombre de neurones de la ligne correspond au nombre de cycles d'ordonnancement, dans l'exemple 6 cycles d'ordonnancement. b) Représentation d'une solution possible résultant de la convergence du réseau. L'activation d'un neurone particulier, par exemple le 4^{ieme} neurone de la tâche 2, indique que la tâche 2 reçoit la ressource de calcul au temps de cycle 4 et s'exécute durant ce temps de cycle.*

cycle d'ordonnancement. L'état du neurone (actif ou inactif) représente l'état de la tâche ("running" ou "suspended"). Un neurone actif indique que la tâche doit être placée dans l'état "running" alors qu'un état inactif indique que la tâche ne possède pas la ressource d'exécution (voir figure 3.1.b). Le problème consiste alors à trouver une solution pour laquelle un neurone est actif ou inactif en fonction des contraintes d'exécution des tâches.

Dans [83], les auteurs proposent une modélisation des tâches au travers de leur période, d'une date au plus tôt ainsi que de leur charge de calcul. La modélisation des tâches est donc la suivante :

$$T_i = \{C_i, D_i, P_i\} \tag{3.1}$$

avec C_i représentant la charge de calcul de la i^{ieme} tâche sur la ressource d'exécution, D_i la date au plus tôt de la

tâche et P_i la période de la tâche.

Notons qu'il est proposé d'ajouter une tâche supplémentaire, dite tâche fictive T_f, permettant d'assurer qu'à chaque cycle d'ordonnancement une tâche possède la ressource de calcul. Lorsque toutes les tâches applicatives sont ordonnancées et qu'il reste des cycles d'ordonnancement non utilisés, alors la ressource exécute la tâche fictive T_f.

Dans ses travaux, Cardeira a montré que ce modèle permet d'assurer l'ordonnancement d'un ensemble de tâches et que la structure proposée assure systématiquement la convergence vers un état stable et représentant bien une solution correcte. Néanmoins, cette modélisation est limitée à une architecture disposant d'une seule cible d'exécution, elle doit donc être étendue pour tenir compte des différentes cibles potentielles présentes dans les SoC actuels. Le modèle des tâches est alors étendu en spécifiant les charges des tâches sur chacune des cibles. La modélisation des tâches est donc la suivante :

$$T_i = \{(C_{i,1}, C_{i,2}, \dots C_{i,r}), D_i, P_i\} \tag{3.2}$$

avec $C_{i,p}$ représentant les charges de calcul de la i^{ieme} tâche sur la p^{ieme} cible d'exécution et r le nombre de plans de cibles d'exécution dans l'architecture (on appelle "plan" un ensemble de cibles de même type).

Pour prendre en compte ce nouveau modèle de tâches, la représentation initiale est étendue (voir figure 3.2). Il

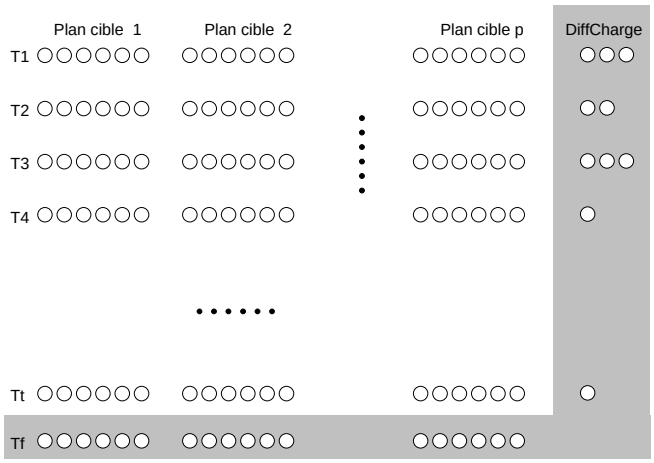


FIG. *3.2 : Extension de la modélisation pour prendre en compte l'hétérogénéité des cibles d'exécution ainsi que l'utilisation non maximale des cibles d'exécution. Pour chaque couple (tâche, plan cible) un ensemble de neurones est mis en place. L'objectif consiste ensuite à s'assurer que seuls les neurones d'un plan cible peuvent devenir actifs. Les neurones des autres plans cibles convergeront vers un état inactif. De plus, des neurones "cachés" sont ajoutés afin de prendre en compte la différence de charge de la tâche sur les plans cibles. Finalement, une tâche fictive supplémentaire est ajoutée afin de modéliser l'inactivité des cibles d'exécution durant des cycles d'ordonnancement.*

s'agit ici de placer autant de plans de neurones qu'il existe de types de ressources d'exécution. Dans ce cas de figure, l'instanciation d'une tâche sur un plan ou sur un autre n'activera pas le même nombre de neurones. Prenons le cas d'une tâche dont les charges seraient les suivantes : $(C_{1,1} = 4; C_{1,2} = 1; C_{1,3} = 2)$. La figure 3.3 présente différents cas d'instanciation de cette tâche sur 3 ressources de calcul.
La modélisation proposée à la figure 3.2 fait apparaître une zone de neurones appelée $DiffCharge$. L'objectif de cette zone est de s'assurer que le nombre de neurones actifs pour la tâche soit toujours égal au maximum

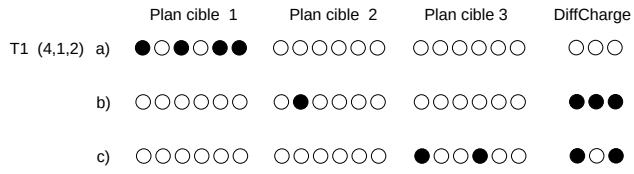


FIG. *3.3 : Exemple d'instanciations d'une tâche sur* 3 *ressources d'exécution ; a) la tâche est instanciée sur la ressource d'exécution* 1 *et occupe dans ce cas* 4 *cycles d'exécution ; b) la tâche est instanciée sur la ressource d'exécution* 2 *et occupe seulement* 1 *cycle d'exécution ; c) finalement, la tâche est instanciée sur la ressource* 3 *et occupe* 2 *cycles d'exécution ; Au travers de cet exemple simple, il apparaît clairement que le nombre de neurones actifs est différent selon la ressource prenant en charge l'exécution de la tâche.*

possible. Par exemple, dans notre cas, la charge maximale de la tâche est égale à 4, la ligne de neurones doit alors toujours converger vers un nombre de neurones actifs égal à 4. Lorsque la tâche est instanciée sur la ressource d'exécution 2, alors le nombre de neurones actifs sur la ligne est égal à $4 : 1$ neurone actif parmi les neurones de la ressource 2 et 3 neurones actifs parmi les neurones de la zone $DiffCharge$. La zone $DiffCharge$ assure donc la variabilité du nombre de neurones actifs en permettant "d'absorber" la différence entre les nombres de cycles maximum et minimum pour l'exécution de chaque tâche. Le nombre de neurones de cette zone est égal à $max(C_{1,p})_{p=1...3} - min(C_{1,p})_{p=1...3}$. Dans le cas de la figure 3.3, le nombre de neurones à ajouter dans la zone $DiffCharge$ est égal à $max(4, 1, 2) - min(4, 1, 2) = 4 - 1 = 3$.

Notons à nouveau la présence d'une tâche fictive permettant de modéliser l'oisiveté des cibles d'exécution lorsqu'aucune tâche ne leur a été affectée durant un ou plusieurs cycles d'exécution (ce qui revient à modéliser le fait que l'architecture n'est pas utilisée à 100% de sa capacité).

3.1.2 Définition de la fonction d'énergie

La définition d'une fonction d'énergie peut parfois être délicate et il est souvent plus simple de raisonner sur des combinaisons de neurones représentant une solution. Dans [82], les auteurs ont montré qu'il existe une combinaison particulière de neurones qui peut ensuite être exprimée par une fonction d'énergie puis mise en forme pour construire le réseau. Il s'agit de la règle dite $k - de - N$ qui spécifie que les solutions recherchées sont telles que k neurones sur N disponibles sont actifs (les autres étant inactifs). Cette règle peut s'exprimer sous la forme de la fonction d'énergie suivante :

$$E_{k-de-N} = \left(k - \sum_{i=1}^{N} x_i \right)^2 \tag{3.3}$$

avec x_i l'état du neurone i (l'état peut prendre les valeurs 0 ou 1). La fonction d'énergie est bien minimale (et égale à zéro) lorsque k neurones sont actifs et que tous les autres sont inactifs.

3.1.3 Calcul des poids des connexions et des entrées des neurones

Pour construire le réseau de neurones, la fonction d'énergie doit exhiber les poids des connexions et des entrées du réseau. Pour cela, elle doit être mise sous une forme particulière. La forme généralement utilisée est la suivante :

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij} \cdot x_i \cdot x_j - \sum_{i=1}^{N} I_i \cdot x_i + \beta \tag{3.4}$$

avec T_{ij} le poids de la connexion entre les neurones i et j, x_i l'état du neurone i, I_i la valeur appliquée à l'entrée du neurone i et β une quantité indépendante de T_{ij} et de I_i.
Après quelques manipulations, l'expression 3.3 peut être mise sous la forme suivante :

$$E_{k-de-N} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N}(-2) \cdot x_i \cdot x_j - \sum_{i=1}^{N}(2k-1) \cdot x_i + k^2 \tag{3.5}$$

De cette expression, on en déduit que les poids des connexions et les poids des entrées des neurones sont égaux à :

$$\begin{aligned} T_{ij} &= -2 \cdot \overline{\delta_{i,j}} \\ I_i &= 2k-1 \\ &\forall\, i = 1 \ldots N \\ &\forall\, j = 1 \ldots N \end{aligned} \tag{3.6}$$

avec $\overline{\delta_{i,j}}$ le complément du symbole de Kronecker qui prend la valeur 0 si $i = j$, et 1 sinon.

Cette règle peut être appliquée directement pour assurer l'exécution d'une et une seule tâche à chaque cycle d'ordonnancement. En effet, en appliquant la règle $k - de - N$ pour $k = 1$ sur l'ensemble des colonnes du réseau de neurones, on modélise l'exécution d'une et une seule tâche sur une cible d'exécution à un cycle donné. De plus, s'il n'y a pas de tâche à exécuter pour ce cycle particulier, alors la tâche fictive sera "ordonnancée" indiquant alors que la cible d'exécution est en attente.

L'additivité des règles de construction a été démontrée dans [80]. Cela implique qu'il est possible d'appliquer plusieurs règles $k - de - N$ sur un ensemble de neurones en ayant l'assurance que la fonction d'énergie sera bien minimale lorsque toutes les règles seront respectées. Il est toutefois mentionné que l'additivité des règles va créer des minima locaux dans la fonction d'énergie globale et poser des problèmes de convergence du réseau. En effet, celui-ci peut alors se stabiliser dans un état qui n'est pas une solution correcte et nécessiter soit un apport d'énergie pour "extraire" le réseau de ce minimum, soit une réinitialisation du réseau et une nouvelle évolution de celui-ci vers un état stable.

Sous l'hypothèse de la détection de minima locaux et d'une solution pour en sortir, l'additivité des règles est un outil simple permettant d'appliquer une règle sur chaque ligne du réseau afin de modéliser l'ordonnancement d'une instance de chaque tâche. Ainsi les règles suivantes peuvent être appliquées :
– une règle $k-de-N$ est appliquée sur chaque ensemble de neurones représentant un plan cible j, avec $k = C_{i,j}$;
– une règle $k - de - N$ est appliquée sur chaque ligne i étendue aux neurones de la zone $DiffCharge$ avec $k = max\,(C_{i,1}, C_{i,2}, \ldots C_{i,r})$,
– une règle $k - de - N$ est appliquée sur chaque colonne de neurones, représentant les cycles d'exécution.

Finalement, sur l'exemple précédent, la ligne de neurones de la tâche 1 se voit appliquer les règles représentées à la figure 3.4 (les règles $k - de - N$ sur les colonnes ne sont pas représentées).

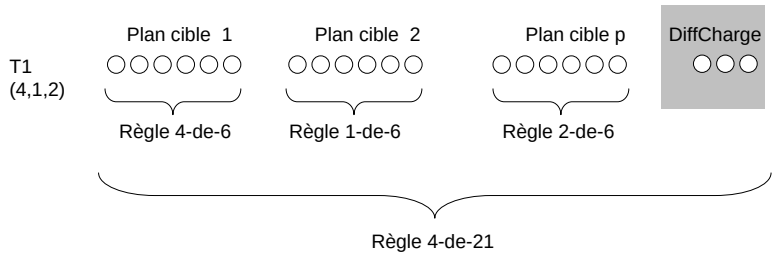


FIG. *3.4 : Illustration de l'additivité des règles $k-de-N$. Sur l'ensemble des neurones de chaque plan cible, la règle $k-de-N$ est appliquée, avec k égal à la charge de la tâche sur la cible d'exécution. Une nouvelle règle $k-de-N$ est ensuite appliquée sur l'ensemble des neurones modélisant le problème étendu de neurones dits "cachés". Dans l'exemple, compte tenu des charges indiquées $(4,1,2)$, les règles suivantes sont alors appliquées : règle $4-de-6$ sur le premier plan cible, règle $1-de-6$ sur le second plan et règle $2-de-6$ sur le troisième plan. De plus, la règle $max(4,1,2)-de-(6+6+6+max(4,1,2)-min(4,1,2))$, soit la règle $4-de-21$ sur l'ensemble des neurones d'une ligne est appliquée.*

3.1.4 Résultats et limitations

Des simulations ont été réalisées afin de tester la validité des solutions produites. Nous donnons dans cette section deux résumés de résultats dont une description plus complète peut être trouvée dans [1].

Le premier exemple concerne le cas d'une architecture disposant de deux cibles d'exécution ayant à exécuter un ensemble de tâches variant de 2 à 7. On suppose dans cet exemple que l'ordonnancement est à effectuer sur 20 cycles. Le tableau 3.1.a donne les informations de charges des tâches (WCET) sur les deux cibles potentielles. Le

	Charges	
Tâches	$C_{i,1}$	$C_{i,2}$
T_1	1	2
T_2	2	1
T_3	4	2
T_4	3	5
T_5	4	6
T_6	3	2
T_7	2	3

a)

Nombre moyen de ré-initialisations						
Nombre de tâches	2	3	4	5	6	7
Nb neurones	135	192	255	312	364	416
Nb utile neurones	80	120	160	200	240	280
Surcoût neurones cachés	68%	60%	59%	56%	52%	48%
Moyenne ré-init	49,9	179,9	667,2	1985	3781	6069

b)

TAB. *3.1 : Résultats de convergence d'un réseau de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de 2 ressources d'exécution et pour un ordonnancement à réaliser sur 20 cycles. Les tests réalisés intègrent les tâches au fur et à mesure. On constate que l'augmentation du nombre de neurones cachés est relativement important. Notons que chaque réinitialisation induit un nombre d'évaluations de l'ordre d'une dizaine afin d'aboutir soit à une solution correcte (convergence vers la nullité de la fonction d'énergie) soit à une solution incorrecte (stabilisation dans un minimum local, fonction d'énergie non nulle).*

tableau 3.1.b fournit les informations concernant le nombre de neurones pour modéliser le problème ainsi que le nombre de réinitialisations nécessaires pour que le réseau converge vers un état stable.

Deux remarques s'imposent à la vue de ces résultats. La première concerne le nombre important de neurones pour parvenir à modéliser le problème. Notons que cela provient en partie du nombre important de neurones cachés nécessaires pour pouvoir appliquer les différentes règles (voir la ligne "Surcoût neurones cachés" du tableau 3.1.b).

Nombre moyen de ré-initialisations						
Nombre de tâches	2	3	4	5	6	7
Nb neurones	180	240	306	364	416	468
Nb utile neurones	80	120	160	200	240	280
Surcoût en neurones cachés	125%	100%	91%	82%	73%	67%
Moyenne ré-init	54,5	337,9	1492	9191	31783	28546

TAB. *3.2 : Résultats de convergence d'un réseau de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de 2 plans d'exécution disposant chacun de 2 ressources d'exécution et pour un ordonnancement à réaliser sur 20 cycles. Notons que le nombre de neurones utiles à la représentation de la solution reste inchangé. Les neurones cachés supplémentaires proviennent de la nécessité d'augmenter le nombre de tâches fictives afin d'assurer la non-utilisation de tous les cycles d'ordonnancement de la part des tâches de l'application. L'évolution la plus importante à noter concerne le nombre de réinitialisations nécessaires pour aboutir à une solution correcte. Il atteint dans ce cas des valeurs très importantes.*

La deuxième remarque concerne le nombre non négligeable de réinitialisations nécessaires pour assurer la convergence du réseau. Ces réinitialisations cachent le problème délicat de la détection de stabilisation sur un état qui n'est pas une solution valide, c'est-à-dire sur un minimum local, et donc de la nécessité de stopper le réseau afin de le réinitialiser. Notons qu'il faut environ une dizaine d'évaluations par initialisation pour parvenir à la convergence ou à la stabilisation sur un minimum local. Le nombre global de neurones évalués pour parvenir à une solution est alors en moyenne égal à 10 fois le nombre de réinitialisations, ce qui est relativement élevé et difficilement acceptable en vue d'une implémentation matérielle efficace.

Le second exemple concerne une extension de l'architecture précédente dont on double le nombre de ressources de calcul. Les tâches sont identiques à l'exemple précédent et le nombre de cycles d'ordonnancement est égal à 20. Le tableau 3.2 fournit les informations sur le réseau ainsi que sur le nombre de réinitialisations nécessaires pour aboutir à la convergence.
Les mêmes remarques concernant les nombres importants de neurones et de réinitialisations peuvent être faites sur ces résultats et cela bien que l'application soit relativement peu complexe.

Au regard de ces résultats, on constate que deux points sont à améliorer :
– Le premier concerne la diminution du nombre de neurones pour parvenir à modéliser le problème d'ordonnancement. L'objectif d'implémentation de ce service au sein d'un système sur puce nous impose de limiter autant que possible le coût de la solution.
– Le second concerne la limitation du nombre de réinitialisations afin de limiter le temps de convergence du réseau (non fourni ici mais proportionnel au nombre de réinitialisations). Ce point est d'autant plus critique qu'il ne faut pas oublier que pour chaque initialisation du réseau, environ une dizaine de neurones sont évalués.

Compte tenu des résultats présentés et des objectifs poursuivis, nous allons nous intéresser, dans les sections suivantes, à la réduction du nombre de neurones pour modéliser le problème. Les neurones cachés seront la principale cible de cette réduction.

3.2 Réduction du nombre de neurones pour modéliser le problème d'ordonnancement

Pour limiter le nombre de neurones nécessaires à la modélisation, nous avons cherché à définir une nouvelle structure/organisation du réseau. L'objectif étant de s'attacher dans un premier temps à la réduction du nombre de neurones cachés. Nos travaux nous ont conduit vers l'utilisation de neurones "inhibiteurs". L'idée générale repose sur la mise en place de neurones qui "captent" l'instanciation de la tâche sur un plan de cibles d'exécution,

et qui bloquent alors l'instanciation sur un autre plan [A28, A29]. Le principe de fonctionnement d'un neurone inhibiteur est relativement simple et s'appuie sur une connectique particulière d'un neurone spécifique (le neurone inhibiteur) avec les neurones à inhiber. La figure 3.5 représente un ensemble de neurones $\{n_1, n_2, \ldots n_5\}$ dont l'activation est inhibée dès lors que le neurone inhibiteur Nh est actif. Le poids de la connexion entre le neurone Nh et les neurones n_i est tel que l'énergie reçue par chaque neurone n_i ne sera jamais suffisante pour déclencher son activation. Pour assurer ce fonctionnement, il faut fixer la valeur de poids p_i de la façon suivante :$p_i < -I_i$ (avec I_i l'entrée du neurone n_i). On peut aussi fixer $p_i \rightarrow -\infty$.

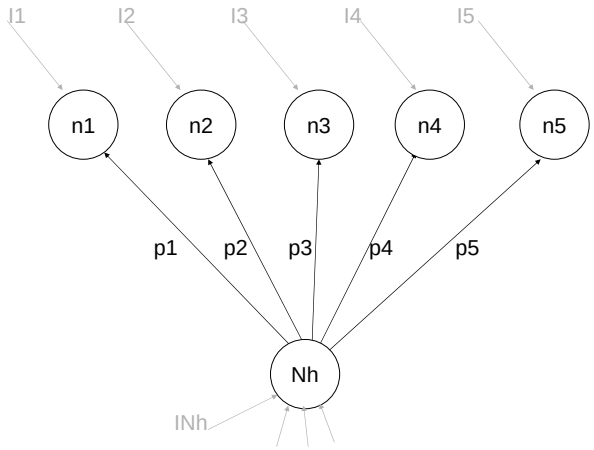


FIG. *3.5 : Principe de mise en œuvre d'un neurone inhibiteur. Le neurone inhibiteur est connecté aux autres neurones* $\{n1, n2 \ldots n5\}$ *par une connexion dont le poids* p_i *est négatif et de très forte valeur absolue. L'activation du neurone inhibiteur va provoquer l'annulation de toutes les énergies présentes aux entrées des neurones* $n1, n2, \ldots n5$*. Si le neurone inhibiteur est actif, alors lors des prochaines évaluations des neurones* n_i*, ces derniers ne pourront que converger vers un état inactif.*

En associant un neurone inhibiteur à chaque plan cible, il est alors possible de s'assurer de la convergence du réseau vers une solution n'ayant qu'un seul ensemble de neurones actifs. La modélisation globale peut alors être représentée par le schéma de la figure 3.6.
Dans ce schéma, les neurones placés dans les zones grisées (Nh_p) sont les neurones inhibiteurs du plan p. En se focalisant sur les neurones de la tâche i, la connectique est définie de la manière suivante : le neurone inhibiteur $Nh_{i,p}$ est connecté à l'ensemble des neurones du plan p de telle façon qu'il devienne actif lorsque le nombre de neurones du plan p a atteint la valeur souhaitée (charge de la tâche i sur le plan p : $C_{i,p}$). Une fois l'activation d'un neurone inhibiteur obtenue, la connectique avec tous les autres neurones de tous les autres plans est définie telle que lesdits neurones convergent vers un état inactif. La structure complète est donnée sur la figure 3.7.
Contrairement aux modélisations proposées précédemment, cette dernière proposition est basée sur une connectique non symétrique. Cette non symétrie est tout à fait visible pour toutes les connexions qui concernent les neurones inhibiteurs. En effet, un neurone inhibiteur $Nh_{i,p}$ dépend de l'état des neurones correspondant à la tâche i et au plan p. Par contre, il n'existe pas de connexion du neurone $Nh_{i,p}$ vers les neurones de la tâche i du plan p. Cette remarque est importante puisqu'elle remet en cause les conditions de stabilité et de convergence définies par Hopfield (notamment la symétrie de la matrice de connexions). Toutefois, comme nous l'expliquons ci-dessous, la convergence du réseau est bien assurée.
La convergence du réseau peut être simplement expliquée de la façon suivante.

1. À l'état initial, le système est dans un état tel que tous les neurones inhibiteurs sont inactifs ;
2. À partir de cet état, chaque plan p de neurones va avoir tendance à converger vers un état tel que $C_{i,p}$ neurones deviennent actifs (règle `k-de-N` sur les neurones du plan p, avec $k = C_{i,p}$) ;

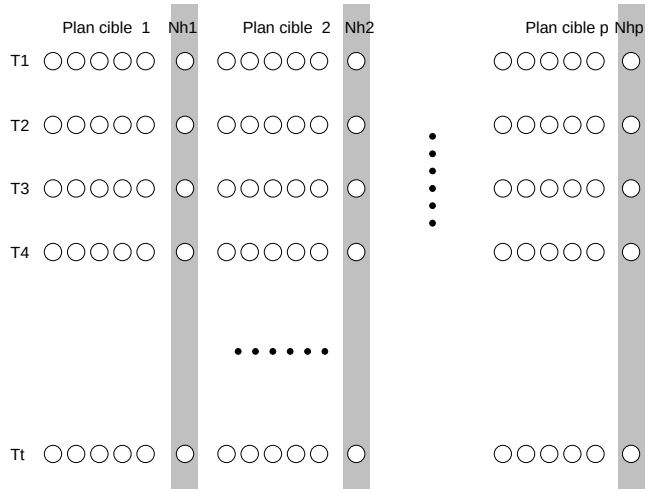


FIG. *3.6 : Modélisation avec neurones inhibiteurs. Chaque couple (tâche, plan) se voit attribuer un neurone inhibiteur. Le rôle du neurone inhibiteur consiste à s'assurer que seuls les neurones d'un plan cible peuvent devenir actifs. L'obtention de la charge souhaitée sur l'un des plans cibles va provoquer l'activation du neurone inhibiteur associé et par voie de conséquence bloquer l'activation de tous les autres neurones de tous les autres plans.*

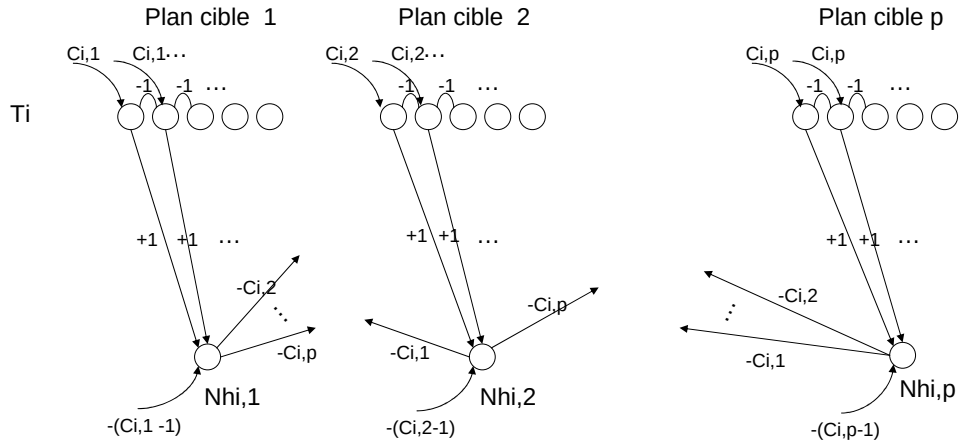


FIG. *3.7 : Modélisation de la connectique avec neurones inhibiteurs. Pour des raisons de lisibilité, les neurones inhibiteurs ont été décalés vers le bas. Un neurone inhibiteur est associé à chaque plan cible. Pour la tâche i, tous les neurones du plan cible j sont connectés au neurone inhibiteur $Nh_{i,j}$ avec un poids égal à 1. Le neurone inhibiteur $Nh_{i,j}$ recevant une énergie négative correspondant à la charge de la tâche i sur le plan j, il ne pourra être actif que lorsque $C_{i,j}$ neurones seront actifs dans le plan j. Lorsque le neurone $Nh_{i,j}$ est actif, il empêche/inhibe toute possibilité d'activation des neurones des autres plans de part sa connectique avec un poids négatif très important.*

3. À partir du moment où l'un des plans dispose de ses $C_{i,p}$ neurones actifs, si le neurone inhibiteur $Nh_{i,p}$ est évalué, alors il deviendra actif, c'est d'ailleurs la seule combinaison qui peut amener à l'activation de l'un des neurones inhibiteurs ;

	Charges	
Tâches	$C_{i,1}$	$C_{i,2}$
T_1	1	2
T_2	2	1
T_3	4	2
T_4	3	5
T_5	4	6
T_6	3	2
T_7	2	3

a)

Nombre moyen d'évaluations de neurones						
Nombre de tâches	2	3	4	5	6	7
Nb neurones	84	126	168	210	252	294
Nb utile neurones	80	120	160	200	240	280
Moyenne évaluations	339	539	1025	1170	1583	1951
Surcoût en neurones cachés	5%)					

b)

Réduction nb neurones	37%	34%	34%	32,7%	30,8%	30%
Réduction nb évaluations	32%	70%	84,6%	98,4%	95,8%	96,8%

c)

TAB. *3.3 : Résultats de convergence d'un réseau de neurones modélisant le problème d'ordonnancement de tâches sur une architecture disposant de 2 ressources d'exécution et pour un ordonnancement à réaliser sur 20 cycles. On constate que le surcoût, en nombre de neurones cachés, est relativement faible et on constate surtout que le nombre d'évaluations est fortement réduit. Cette réduction est d'ailleurs plus importante lorsque la complexité du système augmente.*

4. Lorsque un seul neurone inhibiteur est actif, le poids de ses connexions avec tous les autres neurones des autres plans annule le poids de l'entrée de ces neurones ;
5. Dans ce cas, lorsque les neurones des autres plans seront évalués, ils resteront/passeront dans un état inactif ;
6. Finalement, le système va converger vers un état tel que les neurones du plan dont le neurone inhibiteur est actif, seront dans l'état actif, et les neurones de tous les autres plans seront dans l'état inactif.

Résultats

Pour illustrer notre proposition, nous reprenons les mêmes configurations de tâches et d'architectures que précédemment. Les tableaux 3.3.a et 3.3.b précisent les charges ainsi que les résultats obtenus avec cette nouvelle modélisation.
Notons que la différence entre le nombre de neurones utiles et le nombre de neurones total pour la modélisation du problème est beaucoup plus faible. En fait, il faut simplement $NbTaches * NbPlans$ neurones inhibiteurs pour parvenir à assurer la convergence. Globalement, notre proposition permet de réduire le nombre de neurones d'un facteur variant de $1,6$ à $1,43$ (voir tableau 3.3.c).
Remarquons encore que les résultats donnés ici comptabilisent le nombre d'évaluations des neurones et non pas le nombre de réinitialisations globales du réseau (résultats du tableau 3.1.b). En effet, comme nous l'avons dit précédemment, notre proposition permet de s'affranchir complètement du problème de la réinitialisation du réseau, puisque la convergence est atteinte quelque soit l'état initial de celui-ci. La réduction du nombre moyen d'évaluations des neurones est donnée à la dernière ligne du tableau 3.3.c en posant comme hypothèse que le nombre moyen d'évaluations de neurones est égal à 10 pour chaque réinitialisation du tableau 3.1. On constate alors que l'augmentation de complexité de l'application, se traduisant par un nombre de tâches plus important, est beaucoup mieux supportée par une modélisation du problème avec des neurones inhibiteurs.
La réduction du nombre d'évaluations est très importante et laisse envisager la capacité à pouvoir traiter des applications disposant d'un nombre de tâches beaucoup plus important.

3.3 Extension de la modélisation pour la prise en compte d'une fonction de coût d'instanciation

La disponibilité de ressources de calcul hétérogènes ainsi que la capacité, pour le système d'exploitation, à ordonnancer une tâche sur plusieurs cibles posent la question du coût du placement de ladite tâche sur ces cibles. On imagine assez bien que les coûts (en performance ou en énergie par exemple) ne seront pas les mêmes pour les

exécutions d'une tâche sur un cœur de processeur de type GPP et sur un cœur de processeur de type DSP ou encore sur des cibles spécialisées type IP (Intellectual Properties).
Dans les modélisations précédentes, et pour une tâche qui aurait une charge identique sur deux cibles d'exécution, la probabilité d'être ordonnancée sur l'une ou l'autre des cibles est égale à $\frac{1}{2}$. L'objectif est ici de donner plus de "poids" (d'augmenter la probabilité) à l'ordonnancement d'une tâche sur une cible plutôt que sur une autre.
On définit la notion de coût comme suit : soit une tâche pouvant s'exécuter sur deux types de ressources r_1 et r_2 avec un coût sur chacune des ressources égal à $Cout_1$ $Cout_2$ et soient In_1 et In_2 les nombres d'instanciations de la tâche sur les ressources r_1 et r_2. Le rapport des nombres d'instanciations $\frac{In_1}{In_2}$ doit être égal à l'inverse du rapport des coûts $\frac{Cout_2}{Cout_1}$.
Pour implémenter ce comportement, nous avons développé une solution consistant simplement à "retarder" l'activation d'un neurone, en le faisant passer par différents paliers. On peut représenter cette évolution par une fonction d'évolution interne, déclenchant, lorsqu'elle arrive à son plus haut niveau, l'activation du neurone, voir figure 3.8. Cette modélisation implique que le neurone soit équipé d'une notion de mémoire et que son état, notamment son état interne, dépende des états précédents. Dans l'exemple de la figure 3.8, pour atteindre l'état interne $3/6$, le neurone doit d'abord passer par les états internes $1/6$ puis $2/6$. Le calcul interne réalisé par les neurones sur lesquels on applique la fonction de coût est alors le suivant :

$$\begin{aligned} x_i = 1 \qquad & \text{si } Etat_i == Cout_i \\ & \text{avec } Etat_i = max\left(Cout_i, Etat_i + I_i + \sum w_{j,i} \cdot x_j\right) \end{aligned} \tag{3.7}$$

avec $w_{j,i}$ le poids de la connexion entre le neurone j et le neurone i, et $Cout_i$ le coût que l'on souhaite appliquer au neurone. Cette expression fait apparaître à la fois l'état interne du neurone ainsi que l'accumulation à saturation réalisée sur cet état interne.

Compte tenu des objectifs définis ci-dessus, deux implémentations nous ont paru intéressantes à explorer. Ces deux implémentations correspondent à deux façons de reporter cette notion de coût sur le réseau de neurones :
– une première implémentation consiste à reporter le coût sur chacun des neurones des plans d'exécution ;
– une seconde implémentation consiste à reporter le coût sur les neurones inhibiteurs de chaque plan d'exécution.
Quelque soit la solution retenue, le report de la notion de coût doit respecter au mieux le coût des instanciations des tâches sur les plans d'exécution. Dans le cas de la première implémentation, le report de la notion de coût sur tous les neurones des plans n'est pas trivial puisque la probabilité d'instanciation doit être déclinée sur un grand nombre de neurones. La seconde implémentation, quant à elle, propose un report des coûts sur les seuls neurones inhibiteurs, dont le nombre est beaucoup plus faible.

Résultats

Dans un souci de simplicité, nous avons réalisé deux séries de simulations en reportant simplement le coût d'une tâche :

1. sur l'ensemble des neurones des plans d'exécution ;
2. sur les neurones inhibiteurs des plans d'exécution.

Si l'instanciation d'une tâche i sur la ressource d'exécution j "coûte" $Cout_{i,j}$, alors les coûts sont reportés de la façon suivante :

1. le coût $Cout_{i,j}$ est reporté directement sur tous les neurones des tâches du plan j ;
2. le coût $Cout_{i,j}$ est reporté seulement sur les neurones inhibiteurs du plan d'exécution j.

Nous donnons, dans les tableaux 3.4 et 3.5, les résultats des simulations réalisées pour un système comportant 2 ressources d'exécution et ayant à exécuter une seule tâche. Le coût d'exécution sur la cible 1 est maintenu à la valeur unitaire, et nous faisons varier le coût d'instanciation de la tâche sur la cible d'exécution 2.

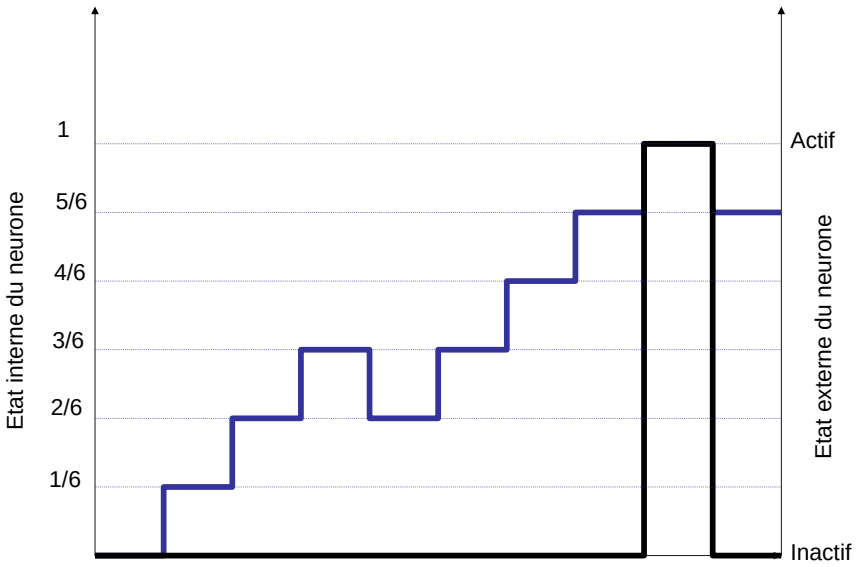


FIG. *3.8 : Exemple d'évolution de l'état interne d'un neurone ayant un "coût" égal à* 6*, et devant donc franchir 5 paliers avant de pouvoir devenir actif. Le principe de fonctionnement s'appuie sur une fonction d'accumulation de l'énergie au sein de chaque neurone. Lorsqu'un neurone est évalué, il ne peut devenir actif que s'il est passé par les* $N-1$ *paliers précédents. Le nombre de paliers définit donc la "difficulté" pour un neurone à devenir actif. Si l'activation d'un neurone particulier est coûteuse alors il suffit de le doter d'un nombre de paliers importants.*

La première remarque que l'on peut faire en observant ces résultats consiste à dire que la fonction de coût a un effet important sur les occurrences des instanciations des tâches, et que l'effet va dans le sens souhaité, c'est-à-dire que lorsqu'une tâche a un coût d'instanciation important sur une cible d'exécution, alors le nombre d'instanciations sur cette cible est réduit.

Par contre, ces simulations montrent qu'un rapport de coûts entre deux plans égal à C, engendre un rapport d'instanciations supérieur à C dans les deux cas de figure. On peut d'ailleurs noter que ce rapport d'instanciations est plus important dans le second cas (tableau 3.5). Cette différence entre l'objectif visé et les résultats obtenus devrait nous amener à revoir la répartition des coûts afin de respecter le rapport des coûts de tâches sur les différentes cibles. Toutefois, en imaginant que la notion de coût représente la consommation énergétique des tâches, il n'est pas aberrant de privilégier plus fortement les instanciations de tâches les moins consommatrices d'énergie, cela va même plutôt dans le bon sens.

Notons pour terminer que la proposition faite pour modéliser une notion de coût pose un problème de rapidité de convergence du système. En effet, plus les valeurs des coûts seront élevées, plus le nombre d'évaluations des neurones (pour que ceux-ci changent d'états et passent d'un état inactif à un état actif) sera important. Globalement, un coût $Cout_i$ appliqué à un neurone, va "ralentir" l'activation de celui-ci d'un facteur $Cout_i$.

Cette proposition devra alors certainement faire l'objet d'une nouvelle étude afin de proposer une solution plus satisfaisante tant du point de vue de la rapidité de convergence que du point de vue de la conservation du rapport de coûts lors des instanciations.

	Coût de la tâche		Nb instanciations		Rapports
	sur cible 1	sur cible 2	sur cible 1	sur cible 2	d'instanciations
Conf 1	1	1	5015	4985	1,01
Conf 2	1	2	6746	3254	2,07
Conf 3	1	5	9305	695	12,9
Conf 4	1	10	9948	52	191

TAB. *3.4 : Nombres moyens d'instanciations d'une tâche sur deux cibles d'exécution en prenant en compte la notion de coût d'exécution de la tâche sur les cibles d'exécution (10000 simulations sont réalisées). Le coût est appliqué directement sur les neurones des tâches.*

	Coût de la tâche		Nb instanciations		Rapports
	sur cible 1	sur cible 2	sur cible 1	sur cible 2	d'instanciations
Conf 1	1	1	4986	5014	0,99
Conf 2	1	2	7442	2558	2,91
Conf 3	1	5	9673	327	29,6
Conf 4	1	10	9985	15	665

TAB. *3.5 : Nombres moyens d'instanciations d'une tâche sur deux cibles d'exécution en prenant en compte la notion de coût d'exécution de la tâche sur les cibles d'exécution (10000 simulations sont réalisées). Le coût est appliqué sur les neurones inhibiteurs.*

3.4 Gestion spécifique de la zone reconfigurable

Lorsqu'une tâche est ordonnancée sur un processeur à un cycle particulier, le processeur est alors pleinement occupé (sur un modèle de processeur non SMT en particulier). A contrario, l'ordonnancement d'une tâche sur une zone reconfigurable ne monopolise pas forcément complètement celle-ci. En effet, l'instanciation d'une tâche sur une zone reconfigurable n'occupe en général qu'une partie de la surface de ladite zone. Il est alors possible d'instancier et d'ordonnancer d'autres tâches pour peu que la surface restante soit suffisante (en quantité, et aussi en forme).
En supposant qu'un mécanisme de compactage soit disponible et se charge de libérer la surface maximale sur la zone reconfigurable, nous proposons de gérer l'instanciation et l'ordonnancement des tâches sur la zone reconfigurable par un réseau de neurones construit sur la base d'une règle $k - de - N$ spécifique. Il n'est en effet pas possible d'utiliser une règle $k - de - N$ classique puisque la valeur de k indiquant le nombre tâches que l'on peut affecter à la ressource dépend des occupations des tâches instanciées, et ne peut donc pas être une constante.

Nous proposons alors de construire une nouvelle règle basée sur l'occupation surfacique de chaque tâche sur la zone reconfigurable. Nous étendons tout d'abord la modélisation des tâches de façon à prendre en compte l'occupation de surface sur la zone reconfigurable. Pour un ensemble de tâches, les caractéristiques sont alors étendues comme indiqué dans la modélisation suivante :

$$T_i = \{C_i, D_i, P_i, A_i\} \tag{3.8}$$

avec A_i la surface occupée par la tâche i sur la zone reconfigurable.

Alors, la fonction d'énergie peut être écrite de la façon suivante :

$$E = \left(\sum_{i=0}^{N} x_i \cdot A_i \ - TA \right)^2 \tag{3.9}$$

avec TA la surface totale disponible au sein de la zone reconfigurable (Total Area).

La minimisation de cette fonction d'énergie peut conduire à l'activation d'un nombre variable de neurones. Cette minimisation ne va cependant pas forcément aboutir à l'annulation de la fonction d'énergie puisque la somme des surfaces des tâches instanciées à un instant donné, ne va pas forcément occuper toute la surface de la zone reconfigurable.

Pour illustrer notre propos, prenons le cas d'un système composé de 4 tâches dont les caractéristiques (surfaciques et temporelles) sont les suivantes :
– tâche 1 : occupe $A_1 = 10\ us$; durée $E_1 = 40\ ut$;
– tâche 2 : occupe $A_2 = 20\ us$; durée $E_2 = 20\ ut$;
– tâche 3 : occupe $A_3 = 10\ us$; durée $E_3 = 30\ ut$;
– tâche 4 : occupe $A_4 = 40\ us$; durée $E_4 = 20\ ut$;
avec us l'unités de surface de la zone reconfigurable, et ut l'unité de temps d'exécution ;
Dans une première approche, si la surface globale de la zone reconfigurable est de $50\ us$, alors les configurations possibles sont au nombre de 10. Elles sont données dans le tableau 3.6.

Tâches	Configurations									
	1	2	3	4	5	6	7	8	9	10
Tâche 1	X				X	X	X		X	
Tâche 2		X			X		X	X		
Tâche 3			X			X	X	X		X
Tâche 4				X					X	X
Surface occupée	10	20	10	40	30	20	40	30	50	50
Type de configuration							CM		CM	CM
Nb de tâches	1	1	1	1	2	2	3	2	2	2

TAB. *3.6 : Liste des configurations possibles pour l'instanciation et l'exécution de tâches au sein d'une zone reconfigurable. Toutes les configurations notées CM représentent des configurations maximales : en d'autres termes, les autres configurations ne sont que des sous-ensembles des configurations maximales. Comme le montre la ligne de surface occupée, aucune des configurations n'a de surface supérieure à* 50.

Pour tenir compte du coût de l'instanciation d'une tâche sur la zone reconfigurable, nous adaptons les poids des connexions entre neurones ainsi que les valeurs des poids d'entrées. L'activation d'un neurone, indiquant l'instanciation d'une tâche sur la zone reconfigurable, induit un coût équivalent à la surface occupée par cette tâche. Ce coût est porté par la connexion entre le neurone de ladite tâche et tous les autres neurones de ce cycle d'ordonnancement. Dans le cas de la figure 3.9.b, on peut prendre l'exemple de la tâche 4 qui une fois activée intervient à hauteur du coût 40 sur l'activation des autres tâches. Dans ce cas, le neurone de la tâche 2 ne pourra pas passer à l'état actif, ce qui traduit bien l'incompatibilité d'instanciation simultanée de ces deux tâches sur la zone reconfigurable.
Le poids de l'entrée d'une tâche doit définir la surface maximale occupée pour que la tâche en question puisse être instanciée sur la zone reconfigurable. Globalement, les caractéristiques du réseau sont :

$$\begin{aligned} T_{ij} &= -A_i \cdot \overline{\delta_{i,j}} && \forall i, \forall j \\ I_i &= TA - A_i + 1 && \forall\ i \end{aligned} \tag{3.10}$$

avec TA la surface totale disponible dans la zone reconfigurable (Total Area).
Le cas général de construction du réseau pour un cycle d'ordonnancement est alors donné à la figure 3.9.a.
A partir de cette structure, nous proposons de réaliser l'ordonnancement temporel des tâches sur la zone reconfigurable. Le principe de fonctionnement repose sur une adaptation du réseau au fur et à mesure de l'ordonnancement

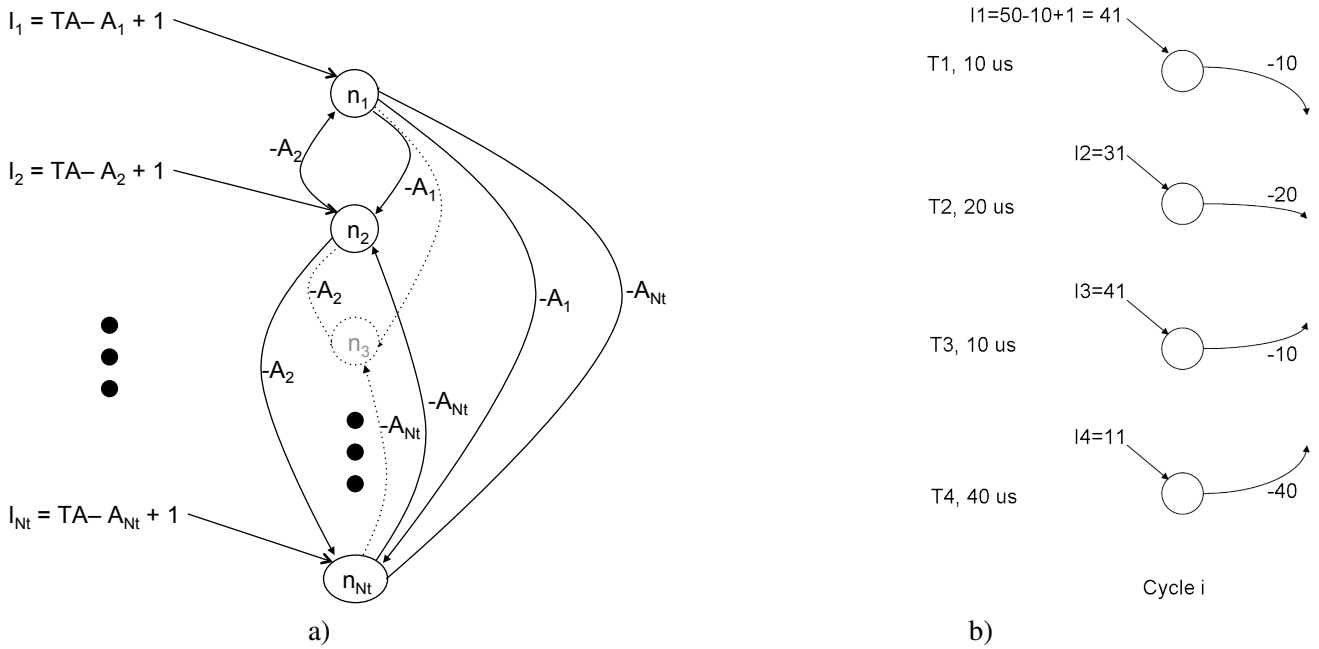


FIG. *3.9 : a) Modélisation générale de l'ordonnancement de tâches s'exécutant sur une zone reconfigurable en tenant compte de l'occupation des tâches ainsi que de la surface totale disponible ; b) Modélisation de l'ordonnancement de tâches s'exécutant sur une zone reconfigurable en tenant compte de l'occupation des tâches ainsi que de la surface totale disponible. La modélisation prend complètement en compte les informations d'occupation de chaque tâche sur la zone reconfigurable (connexion du neurone de la tâche i avec les autres neurones avec un poids égal à $-A_i$). De même, l'énergie apportée à chaque neurone (entrée du neurone) est directement liée à la surface totale de la zone reconfigurable et de l'occupation de la tâche sur cette zone.*

des tâches. Le modèle que nous proposons est donc un réseau de neurones reconfigurable, (RANN, Reconfigurable Artificial Neural Network).

Le modèle de tâches supportées gère les dépendances entre tâches mais exclut la préemption afin de limiter les coûts de chargement de *bitstream* (coût énergétique et temporel).

Globalement, le management du réseau est réalisé par un contrôleur dont le rôle consiste à adapter les poids des entrées des neurones du réseau proposé. Cette gestion des entrées des neurones est formalisée ci-dessous.

• Soit $\mathcal{D}$ la matrice définissant les dépendances entre tâches. La matrice est de taille $(N_t \times N_t)$, avec $d_{i,j}$ une variable binaire égale à 1 si la tâche T_i précède la tâche T_j, et égale à 0 sinon. On peut noter que $d_{i,i}$ est égale à 0.

• Soit $\mathcal{F}_s$ un vecteur de valeur binaire de taille N_t, dépendant du cycle d'ordonnancement s. Ce vecteur est composé des éléments $f_{s,i}$ égaux à 1 si la tâche T_i a terminé son exécution avant le cycle d'ordonnancement s, égale à 0 sinon. Par exemple, si la tâche T_1 (qui est caractérisée par son temps d'exécution $E_1 = 30\ [tu]$) débute son exécution au temps 0, alors la variable binaire $f_{s,1}$ vaut : $f_{s,1} = 0\ \ \forall\ s < 30$, et $f_{s,1} = 1\ \ \forall\ s \geq 30$.

• Soit $\mathcal{X}_s$ le vecteur de taille N_t indiquant le nombre de cycles d'ordonnancement obtenus par chaque tâche à l'instant s. A chaque cycle d'ordonnancement, les valeurs $x_{s,i}$ sont incrémentées pour tous les neurones n_i actifs au cycle courant, ce qui peut s'écrire de la façon suivante :

$$x_{s,i} = x_{s-1,i} + 1, \qquad \forall\ i \mid n_i \text{ est actif au cycle } s. \tag{3.11}$$

Dans ce cas, la variable binaire $f_{s,i}$ est évaluée (au cycle s) de la façon suivante :

$$f_{s,i} = \begin{cases} 1 & \text{si } x_{s,i} \geq E_i, \\ 0 & \text{sinon,} \end{cases} \qquad \forall\, i = 1, \ldots, N_t. \tag{3.12}$$

A partir de ces variables, on définit une variable de contrôle $c_{s,i}$ pour l'entrée de chaque neurone n_i. Cette variable est définie par la relation suivante :

$$\begin{aligned} c_{s,i} &= \left(f_{s,1} \vee \overline{d_{1,i}}\right) \wedge \left(f_{s,2} \vee \overline{d_{2,i}}\right) \wedge \cdots \wedge \left(f_{s,N_t} \vee \overline{d_{N_t,i}}\right) \\ &= \bigwedge_{j=1}^{N_t} \left(f_{s,j} \vee \overline{d_{j,i}}\right) \qquad \forall\, i = 1, 2, \ldots, N_t \end{aligned} \tag{3.13}$$

avec $\wedge$ l'opération logique *and* et $\vee$ l'opération logique *or*.
Si une dépendance existe entre les tâches T_i et T_j (la tâche T_i précède la tâche T_j), alors cette expression force l'exécution de la tâche T_i avant d'autoriser l'exécution de la tâche T_j.
Nous pouvons alors définir le vecteur de contrôle des entrées des neurones $\mathcal{C}_s$ par

$$\begin{aligned} \mathcal{C}_s^* &= \begin{pmatrix} f_{s,1} & f_{s,2} & \cdots & f_{s,N_t} \end{pmatrix} \odot \begin{pmatrix} \overline{d_{1,1}} & \overline{d_{1,2}} & \cdots & \overline{d_{1,N_t}} \\ \overline{d_{2,1}} & \overline{d_{2,2}} & \cdots & \overline{d_{2,N_t}} \\ \cdots & \cdots & \cdots & \cdots \\ \overline{d_{N_t,1}} & \overline{d_{N_t,2}} & \cdots & \overline{d_{N_t,N_t}} \end{pmatrix} \\ &= \mathcal{F}_s^* \odot \overline{\mathcal{D}} \end{aligned} \tag{3.14}$$

avec $\mathcal{F}_s^*$ la transposée du vecteur $\mathcal{F}_s$, $\overline{\mathcal{D}}$ la matrice complémentaire (au sens où les valeurs $d_{i,j}$ sont binairement complémentées) de la matrice $\mathcal{D}$ et $\odot$ l'opérateur matriciel exprimant l'opération *maxterm*.
Pour implémenter ce contrôle, nous avons proposé de placer une fonction `and` devant chaque entrée de neurone comme illustré à la figure 3.10.

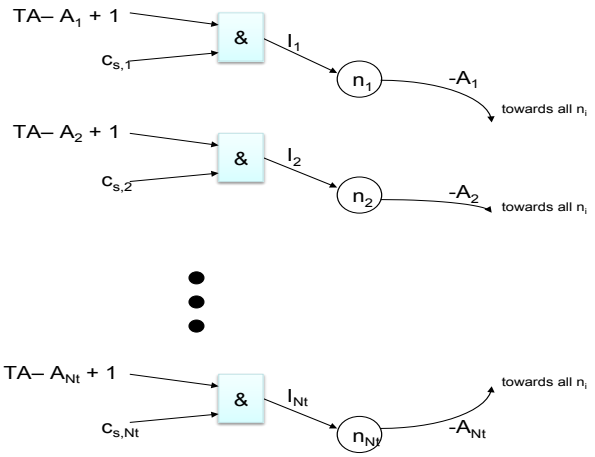


FIG. *3.10 : Structure du RANN avec ajout d'une fonction* `and` *à chaque entrée de neurone afin de contrôler les dépendances de données. Au travers de cette fonction, l'énergie apportée au neurone est maîtrisée et il est alors possible de contrôler sa possible activation.*

Le fonctionnement de notre proposition, est illustré par la série de cycles, dits cycles de reconfiguration (RST Reconfigurable Scheduling Tick), représentés sur la figure 3.11. Ces cycles montrent une évolution possible du réseau de 4 neurones au cours des cycles d'ordonnancement sur une zone reconfigurable.
L'adaptation du réseau RANN s'effectue cycle d'ordonnancement après cycle d'ordonnancement, de la façon suivante

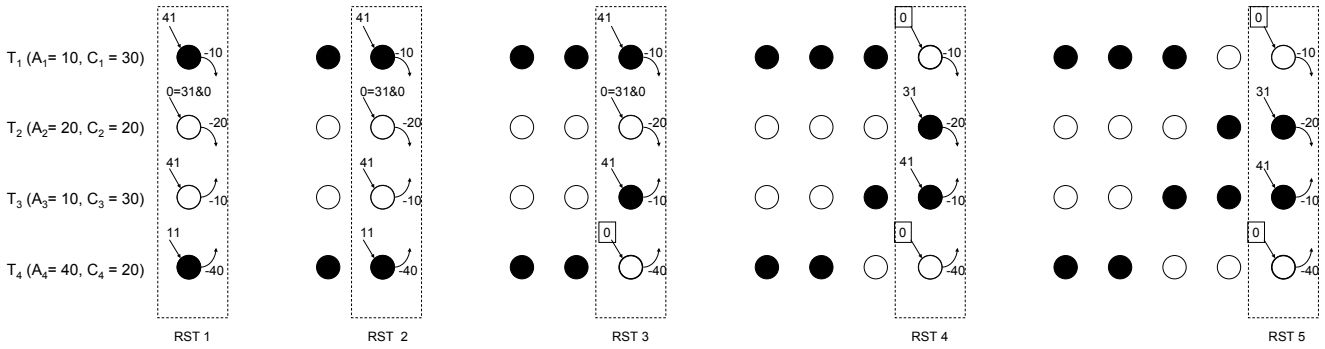


FIG. *3.11 : Exemple d'évolution du réseau de neurones RANN pour plusieurs cycles d'ordonnancement. Dans cet exemple, quatre tâches sont considérées. Chaque cycle d'ordonnancement s'assure que la surface occupée par les tâches est inférieure ou égale à la surface totale disponible dans la zone reconfigurable. Le contrôle du réseau permet de contrôler la non préemption des tâches ainsi que les dépendances entre les tâches.*

RST 1 : Au premier cycle d'ordonnancement, compte tenu des dépendances entre les tâches, la tâche T_2 ne peut pas être ordonnancée. Comme on le voit, l'entrée du neurone n_2 de la tâche T_2 est effectivement forcée à 0, à cause de la variable binaire $c_{s,1}$ égale à zéro. Dans ces conditions, le neurone n_2 ne peut pas devenir actif. Si on suppose que le neurone n_1 correspondant à la tâche T_1 est évalué, l'énergie reçue par ce neurone est alors égale à $I_1 + \sum_{j=1}^{Nt} x_j \cdot W_{1,j} = I_1 = 41$). Alors ce neurone bascule dans l'état actif. Imaginons ensuite que le neurone n_4 soit évalué. L'énergie reçue par ce neurone est alors égale à $I_4 - A_1 = 11 - 10 = 1$, ce qui permet une activation du neurone n_4. Ensuite, quelque soit le neurone évalué, le RANN ne peut changer d'état, il se trouve d'ailleurs dans un cas de configuration maximale.

RST 2 : L'évaluation d'un nouveau cycle d'ordonnancement s'effectue sans ré-initialiser l'état des neurones. Dans ces conditions, comme aucune des deux tâches ordonnancées au cycle précédent (T_2 et T_4) n'est achevée, alors les entrées des neurones restent inchangées. Le RANN ne peut alors évoluer et ce cycle de convergence conserve l'état du réseau.

RST 3 : A ce cycle, la tâche T_4 a terminé son exécution. La variable binaire $f_{s,4}$ prend alors la valeur 1 indiquant que la tâche T_4 est terminée et ne peut pas être ré-ordonnancée. Malgré la terminaison de la tâche T_4, la tâche T_2 ne peut pas être ordonnancée compte tenu de la dépendance avec la tâche T_1. Si on suppose alors que le neurone n_3 correspondant à la tâche T_3 est évalué, alors l'énergie reçue par ce neurone est suffisante pour modifier son état. Cette énergie est égale à $41 - 10 = 31$ et le neurone change d'état. Ensuite, quelque soit les neurones évalués, le RANN reste stable.

RST 4 : A ce nouveau cycle, la tâche T_1 a terminé son exécution. La variable binaire $f_{s,1}$ prend la valeur 1 indiquant la fin d'exécution de la tâche T_1 et empêchant sa ré-exécution. Dans ce cas, les dépendances de T_2 sont levées et celle-ci peut alors être ordonnancée. La variable binaire $c_{s,2}$ est maintenant égale à 1, et l'entrée du neurone n_2 est fixée à $I_2 = TA - A_2 + 1$. L'énergie reçue par le neurone n_2 est alors suffisante (égale à $31 - 10 = 21$) pour activer le neurone. Les autres neurones restent inchangés et le réseau RANN est stable.

RST 5 : Pour les mêmes raisons que dans le cas du cycle RST 2, le réseau RANN reste stable. Toutes les tâches ont alors été ordonnancées.

Notons que ce modèle est particulièrement bien adapté à la gestion de la zone reconfigurable, puisque l'on effectue une "reconfiguration" régulière du réseau en vue de s'adapter aux tâches à exécuter. D'autre part, cette solution est peu coûteuse puisqu'elle ne nécessite qu'un très faible nombre de neurones. Ce nombre de neurones est égal au nombre de tâches à gérer à un instant donné. Toutefois, cette structure nécessite un contrôle particulier afin de fournir, cycle après cycle, des ordonnancements valides. Nous avons pour l'instant supposé que ce contrôle serait

pris en charge par le cœur de processeur présent au sein du RSoC. Une structure matérielle sera très certainement à développer pour assurer efficacement ce contrôle.

Résultats

Pour illustrer notre proposition, nous avons réalisé des simulations à partir de l'exemple de la figure 3.9. Le tableau 3.7 fournit les occurrences de chaque configuration pour les tâches spécifiées ci-dessus. Ce tableau montre que certaines configurations n'apparaissent jamais ou très peu, cela s'explique par le fait que chaque simulation laisse évoluer le réseau vers les configurations *maximales*.

	Configurations									
	1	2	3	4	5	6	7 CM	8	9 CM	10 CM
Occurrences pour 1000 iterations	0	0	0	0	1	1	566	2	209	220
Occurrences pour 10000 iterations	0	0	0	0	11	0	5840	15	2032	2102

TAB. *3.7 : Occurrences de chaque configuration possible pour l'instanciation de* 4 *tâches sur une zone reconfigurable. Les configurations maximales sont privilégiées compte tenu du nombre d'évaluations de neurones réalisées avant de stopper la convergence. En augmentant le nombre d'évaluations des neurones, toutes les convergences auraient abouti aux configurations maximales.*

De plus, ce tableau fait apparaître un déséquilibre d'occurrences des configurations. En effet, la configuration 7 a plus d'occurrences que les configurations 9 et 10. Ce déséquilibre s'explique au travers de l'expression des probabilités d'occurrences des configurations en posant comme hypothèse l'équiprobabilité des tirages des neurones. Ce déséquilibre dans les occurrences des configurations peut être problématique puisqu'il intervient indépendamment de la notion de priorité des tâches. En effet, en imaginant que toutes les tâches soient de priorité équivalente, la tâche 4 a une probabilité d'ordonnancement plus faible. Ici, elle est ordonnancée 429 fois pour 1000 itérations. Notons que ce déséquilibre est maintenu pour un nombre de simulations plus importants.
Pour palier à ce déséquilibre, cette solution doit alors être accompagnée d'une gestion de la notion de coût (ou de priorité) au travers, par exemple, de la proposition faite à la section 3.3.

Concernant la rapidité de convergence du réseau, nous comparons dans le tableau 3.8 les résultats de notre structure aux résultats non-optimaux que donnerait une modélisation classique. Il est important de noter qu'une modélisation du problème d'ordonnancement à l'aide d'une structure de réseaux de neurones classiques n'est pas en mesure de gérer un nombre variable de tâches au cours du temps. Pour établir ce tableau, nous considérons alors un nombre fixe de tâches actives à chaque cycle. Les caractéristiques des tâches sont tirées aléatoirement, tout en s'assurant que toutes les tâches ne puissent pas être instanciées simultanément sur la ressource reconfigurable.
Les résultats présentés dans ce tableau montrent que cette structure de réseau de neurones dispose de propriétés intéressantes pour une implémentation efficace. Dans la section 3.6, nous présentons les premiers résultats d'implémentation de ce type de structure.

3.5 Outillage informatique pour la simulation et la synthèse

L'ensemble des propositions que nous avons exposées dans ce chapitre a fait l'objet de validations sur la base d'un simulateur de réseaux de neurones développé en interne. Cet outil, développé en Java, a été conçu pour être le plus générique possible et accepter tout type de réseaux sous condition de pouvoir en faire une description complète et non ambiguë. Le réseau est donc décrit de façon très fine en spécifiant, soit neurone par neurone soit par ensemble de neurones, les valeurs des entrées et des poids de connexions. Cette finesse de description, importante pour assurer le plus de flexibilité possible, rend la description d'un problème d'ordonnancement de

Nb de tâches N_T	Modélisation classique		Notre proposition	
	Nb de neurones $N_T \times N_C$	Nb de cycles pour converger	Nb de neurones N_T	Nb de cycles pour converger
10	1000	$\simeq 10^3$	10	$\simeq 10$
20	2000	$\simeq 2 \cdot 10^3$	20	$\simeq 20$
40	4000	$\simeq 4 \cdot 10^3$	40	$\simeq 40$
60	6000	$\simeq 6 \cdot 10^3$	60	$\simeq 60$
80	8000	$\simeq 8 \cdot 10^3$	80	$\simeq 80$
100	10000	$\simeq 10 \cdot 10^3$	100	$\simeq 100$

TAB. *3.8 : Résultats de convergence pour un ensemble d'applications possédant un nombre de tâches variant de 10 à 100. Le nombre de neurones nécessaire pour la modélisation classique est donné par le produit du nombre de cycles N_C par le nombre de tâches N_T. Pour notre proposition, le nombre de neurones est simplement égal au nombre de tâches de l'application. Concernant le nombre de cycles pour obtenir la convergence, la modélisation classique nécessite un très grand nombre de cycles en partie dû au grand nombre de neurones. Notre proposition converge très rapidement, cette convergence rapide est en partie due à la contrainte de surface totale nécessaire qui conduit très vite le système à ne plus pouvoir activer de neurones.*

tâches assez lourd, aussi pour simplifier la construction du réseau, nous avons développé un générateur de réseaux de neurones prenant en entrée une description de haut niveau des tâches et de leurs caractéristiques. De plus, nous avons développé un générateur de réseaux de neurones synthétisables. La cible architecturale est de type FPGA, et plus précisément les circuits de la famille Virtex 4 du constructeur Xilinx. Le schéma de la figure 3.12 montre les trois outils que nous avons développés et qui nous ont permis d'obtenir les résultats de simulations ainsi que les descriptions synthétisables des réseaux.

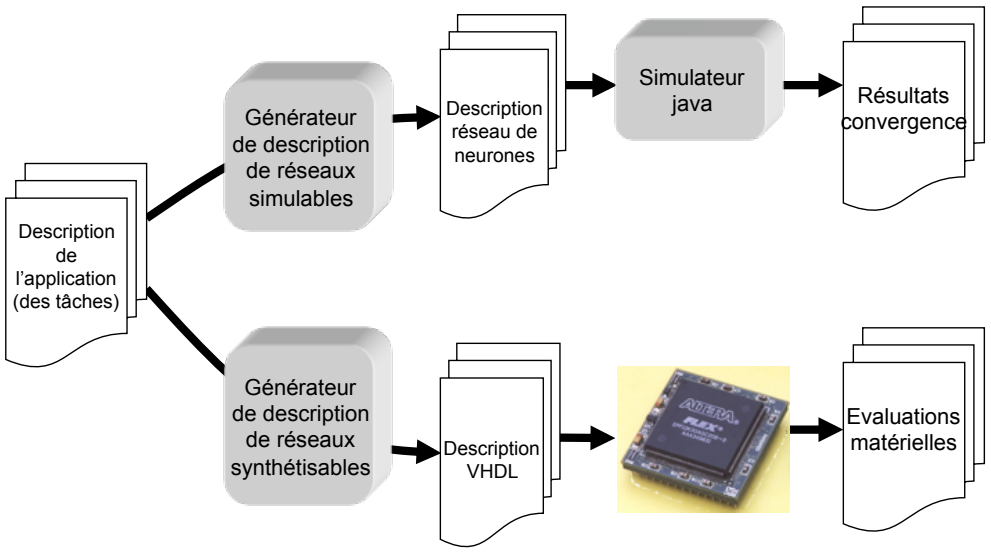


FIG. *3.12 : Chaîne d'outils mis en place pour assurer à la fois la description de réseaux de neurones simulables par un simulateur logiciel, et la description de réseaux de neurones synthétisables sur cible FPGA.*

3.6 Implémentation matérielle

Comme nous l'indiquions en introduction de ce chapitre, notre objectif final concerne la définition d'une architecture matérielle implémentant un réseau de neurones pour le problème de l'ordonnancement. Pour ces travaux d'implémentation, nous avons retenu deux modélisations de réseaux de neurones. La première structure retenue est celle basée sur l'utilisation de neurones inhibiteurs. La seconde repose sur la structure reconfigurable. Afin de ne pas surcharger le document, nous ne détaillons que la première structure dans la section suivante.
Comme nous l'avons indiqué, la structure de réseau s'appuyant sur des neurones inhibiteurs est intéressante par l'assurance de convergence vers une solution correcte dans tous les cas de figure. La principale difficulté pour implémenter un réseau de neurones réside dans le nombre de connexions à mettre en place. En effet, en toute rigueur, tout neurone doit être connecté à tous les autres. Toutefois, cette connectivité globale n'est jamais nécessaire dans le problème qui nous intéresse puisque les connexions concernent des ensembles de neurones sur lesquels une règle de convergence est appliquée. Malgré tout, le nombre de connexions peut être difficile à gérer pour une réalisation matérielle.
Pour limiter ce nombre de connexions, il est possible de s'appuyer sur la notion d'ensemble de neurones dont les rôles sont similaires. En effet, l'application des règles de convergence nous amène à définir des poids ainsi que des connexions généralement identiques à tout un ensemble. Or le traitement de base d'un neurone est un calcul du type :

$$Etat_i = I_i + \sum_{j=1}^{N} x_j \cdot T_{i,j} \tag{3.15}$$

Ce calcul peut s'exprimer alors de la façon suivante :

$$Etat_i = I_i + \sum_{\omega_j \in \epsilon} \left(\sum_{x_j \in \omega_j} x_j \cdot T_{\omega_j} \right) \tag{3.16}$$

avec ϵ l'ensemble total des neurones, ω_j un sous-ensemble de neurones sur lequel est appliqué une règle de convergence (avec une connectivité identique entre chaque neurone du sous-ensemble) et T_{ω_j} le poids d'une connexion relatif à cet ensemble. Alors l'expression peut s'écrire :

$$Etat_i = I_i + \sum_{\omega_j \in \epsilon} \left(\sum_{x_j \in \omega_j} x_j \right) \cdot T_{\omega_j} \tag{3.17}$$

Où le terme $\sum_{x_j \in \omega_j} x_j$ représente simplement le nombre de neurones actifs dans l'ensemble considéré. Il est alors possible de simplifier l'expression en considérant une seule fois le calcul du nombre de neurones actifs d'un ensemble. En exprimant ce nombre comme étant NbA_{ω_j}, il vient alors l'expression :

$$NbA_{\omega_j} = \sum_{\omega_j} x_j \tag{3.18}$$

$$Etat_i = I_i + \sum_{\omega_j \in \epsilon} \left(NbA_{\omega_j} - x_i \right) \cdot T_{\omega_j} \tag{3.19}$$

Le terme $\sum_{\omega_j \in \epsilon} \left(NbA_{\omega_j} - x_i \right)$ permet de déterminer le nombre de neurones actifs autres que le neurone i en cours d'évaluation.
D'un point de vue implémentation matérielle, cette expression se traduit par la mise en place d'un élément commun à tous les neurones impliqués dans une même règle. Le rôle de cet élément est simplement de déterminer combien de neurones sont actifs dans l'ensemble en question.
Le schéma synoptique de l'implémentation matérielle que nous proposons est donné à la figure 3.13.
Cette figure illustre une implémentation matérielle pour un problème d'ordonnancement composé de T tâches, de C cycles d'ordonnancement et d'une seule cible d'exécution. Dans ce cas de figure, chaque neurone est alors

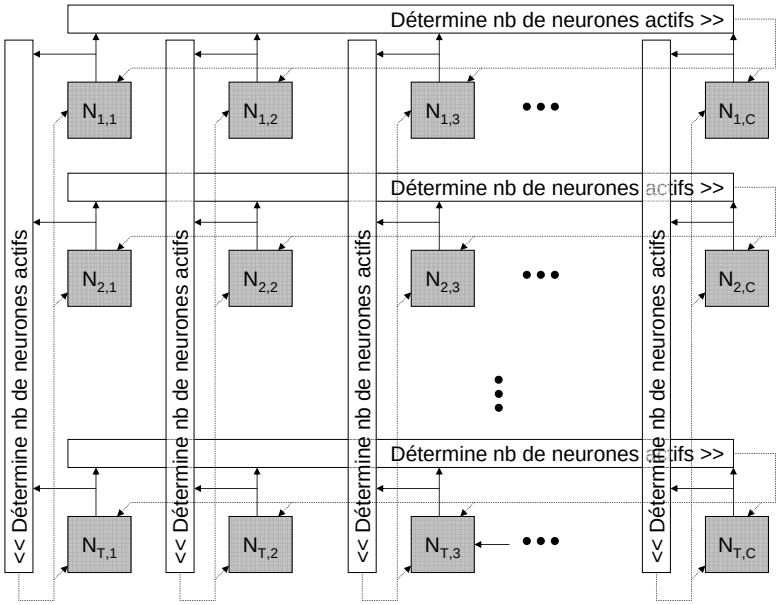


FIG. *3.13 : Implémentation matérielle proposée pour modéliser une structure de réseau de neurones. La représentation concerne un ensemble de T tâches à ordonnancer sur C cycles d'ordonnancement. Chacun des blocs $N_{i,j}$ correspond à un neurone. Chaque neurone $N_{i,j}$ reçoit le nombre de neurones actifs sur sa propre ligne, ainsi que le nombre de neurones actifs sur sa propre colonne. À partir de ces éléments, de son état et de son entrée (stockée en interne) il est en mesure de déterminer s'il doit s'activer ou pas lorsque le contrôleur global lui demande de s'évaluer (ligne de demande d'évaluation non représentée ici).*

réalisé comme illustré à la figure 3.14. Sur cette figure, on constate que le nombre d'entrées de chaque neurone est limité au nombre de règles auxquelles il est soumis, dans notre exemple simplement 2 règles sont appliquées à chaque neurone. Compte tenu des regroupements effectués, le calcul de chaque neurone est donc simplement donné par l'expression suivante :

$$Etat_{ij} \quad = \quad I_{ij} + (NbA_{\omega_1} - x_{ij}) \cdot T_{\omega_1} + (NbA_{\omega_2} - x_{ij}) \cdot T_{\omega_2} \tag{3.20}$$

Bien que les entrées I_{ij} de tous les neurones d'une même ligne soient identiques, nous mémorisons cette information au sein de chaque neurone afin de limiter la connectique globale. De même, les valeurs T_{ω_1} et T_{ω_2} sont stockées au sein de chaque neurone. Dans ce cas, la connectique globale est limitée au strict nécessaire à savoir, pour un neurone particulier, la connaissance du nombre de neurones actifs dans les ensembles dont il fait lui-même partie.

Notons que la figure 3.14 fait apparaître plusieurs signaux de contrôle non représentés à la figure 3.13. Les signaux (`Reset, Set`) permettent l'initialisation des neurones. Le signal `Active` permet de déclencher l'activation du neurone, c'est-à-dire son évaluation interne (calcul du nouvel état).
L'implémentation des neurones inhibiteurs est réalisée par un comparateur et en utilisant les entrées `Reset` des neurones. Un comparateur (ou neurone inhibiteur) est placé à la sortie de chaque bloc comptabilisant le nombre de neurones actifs dans une ligne. Lorsque le nombre de neurones est supérieur ou égal à la charge de la tâche, alors la sortie du comparateur passe à l'état 1 et force tous les autres neurones de tous les autres plans à 0, par une connexion à leur signal d'entrée `Reset`. Le schéma simplifié est donné à la figure 3.15.
Nous sommes actuellement en phase de description VHDL des éléments de cette structure de réseaux de neurones. Pour ces implémentations, nous ciblons une architecture matérielle de type FPGA et plus particulièrement des

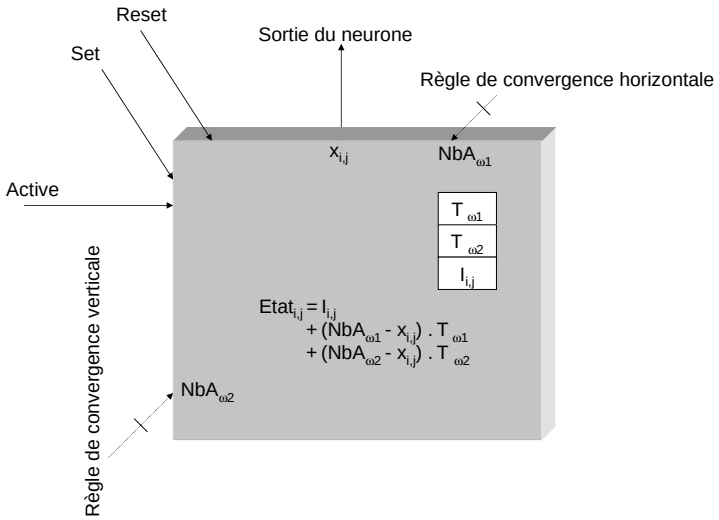


FIG. *3.14 : Implémentation matérielle d'un neurone. En plus des entrées spécifiées précédemment, le neurone est équipé d'entrées de remise à* 0*, de mise à* 1 *et de demande d'activation. Les poids des connexions sont stockés directement dans le neurone afin de limiter les besoins en accès externe.*

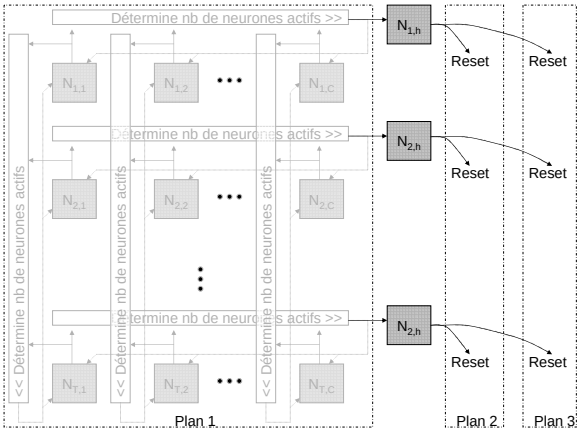


FIG. *3.15 : Implémentation matérielle d'un neurone. En plus des entrées spécifiées précédemment, le neurone est équipé d'entrées de remise à* 0*, de mise à* 1 *et de demande d'activation. Les poids des connexions sont stockés directement dans le neurone afin de limiter les besoins en accès externe. Finalement le neurone doit simplement connaître le nombre de neurones actifs dans les ensembles où il est lui-même impliqué, ce qui limite beaucoup la connectique.*

circuits Xilinx de la famille Virtex 4. Des résultats, à la fois de coût d'implémentation et de temps de convergence devraient bientôt être disponibles.

3.7 Placement de tâches sur une zone reconfigurable

Les sections précédentes se sont intéressées à l'ordonnancement "classique" des tâches, c'est-à-dire à l'ordonnancement temporel. Or lorsque l'on s'intéresse à des tâches matérielles, l'ordonnancement temporel ne suffit pas puisqu'il est nécessaire de spécifier l'emplacement "géographique" de la tâche sur la zone reconfigurable. Comme nous l'avons indiqué dans le chapitre 2, des propositions ont été faites sur ce point, notamment par une gestion des rectangles vides et la recherche du meilleur emplacement pour une nouvelle tâche. Les travaux que nous avons entamés en octobre 2008 avec Antoine EICHE concernent la définition d'une structure de réseau de neurones en vue de réaliser le placement spatial des tâches sur la ressource reconfigurable. Précisons que la structure de réseau reconfigurable RANN que nous avons présentée dans la section 3.4 permet de gérer le cas d'un placement en colonnes des tâches sur la zone reconfigurable. Toutefois, il est clair qu'un placement en colonnes est loin d'être optimal et réaliste. En effet, ce type de placement suppose que les tâches sont configurables à n'importe quel endroit de la zone reconfigurable et qu'elles prennent forcément des colonnes entières de ressources dans la zone reconfigurable (colonnes de CLB dans le FPGA).
La proposition que nous avons établie est basée sur une modélisation plus réaliste des tâches dans le sens où celles-ci sont considérées comme configurables seulement à certains endroits de la zone reconfigurable et qu'elles occupent des zones rectangulaires au sein de cette ressource.
Ces hypothèses sont simplificatrices, mais elles sont aussi beaucoup plus réalistes puisqu'actuellement, sur les circuits reconfigurables supportant la reconfiguration dynamique et partielle, notamment pour les circuits de la famille Virtex 4, 5 et 6, le concepteur doit définir manuellement des PRRs (Partial Reconfigurable Region) et pour chaque PRR le concepteur doit effectuer une synthèse de l'ensemble des tâches qu'il souhaite configurer dans ledit PRR.
Globalement, nous proposons donc poser le problème du placement des tâches au sein de la zone reconfigurable de la façon suivante.
• Soit RR une région reconfigurable ;
• Soit $\varepsilon = \{PRR_1, PRR_2, \ldots, PRR_{N_{prr}}\}$ l'ensemble des PRRs de RR, avec N_{prr} le nombre total de PRRs de RR ;
• Soit $\omega_{T_i} = \{\lambda_{i,j}\}$ l'ensemble des instances possibles de la tâche T_i, avec $\lambda_{i,j}$ une variable binaire égale 1 si la tâche T_i est configurable dans le PRR_j, égale à 0 sinon ;
La recherche d'une solution à ce problème de placement doit répondre aux contraintes suivantes :

$$\begin{aligned} \sum_{j=1}^{n} \lambda_{i,j} \leq 1 \qquad & \forall\, i = 1, \ldots, N_T \\ \sum_{i=1}^{N_T} \lambda_{i,j} = 1 \qquad & \forall\, j = 1, \ldots, N_{prr} \end{aligned} \tag{3.21}$$

La première sommation assure que chaque PRR recevra au plus une seule tâche pour un placement donné, alors que la seconde sommation assure que chaque tâche est bien configurée une seule fois. À partir des définitions données ci-dessus, nous construisons un réseau de neurones ayant autant de neurones que d'instances de tâches, soit

$$nbNeurones = \sum_{i=1}^{N_T} Card(\omega_{T_i}) \tag{3.22}$$

Avec N_T le nombre de tâches à instancier dans la RR. Pour chaque couple de tâches (T_i, T_j) tel qu'il existe un PRR_k pour lequel $\lambda_{i,k} = \lambda_{j,k} = 1$, alors une connexion est créée entre les neurones représentant les instances de T_i et de T_j dans le PRR_k. Le poids positionné sur cette connexion est tel que l'activation de l'un des neurones empêche l'activation de l'autre neurone. On peut voir cette construction comme l'application d'une règle $0 - ou - k - de - N$ sur l'ensemble des neurones représentant les instances possibles sur le PRR_i, avec $k = 1$. En appliquant une nouvelle fois la règle $k - de - N$ sur l'ensemble des instances de la tâche T_i, il est possible de

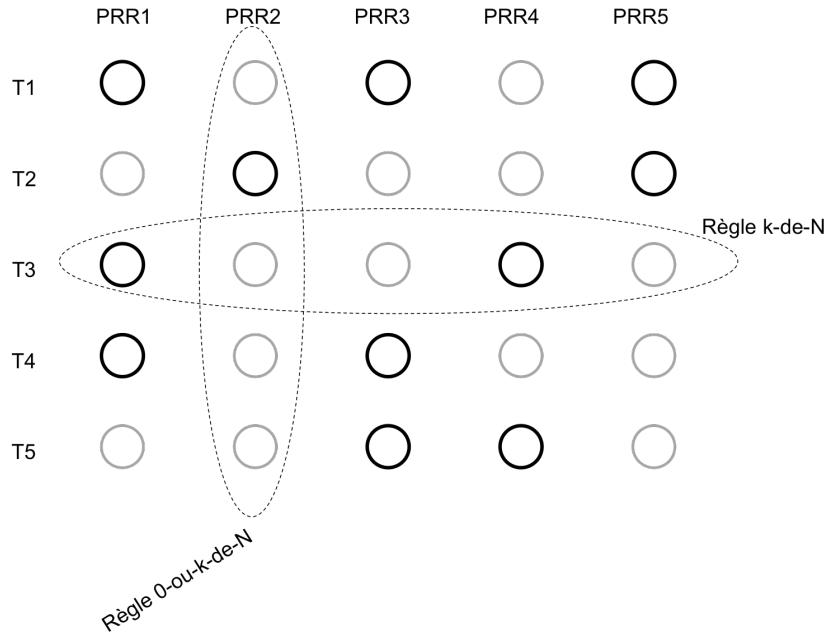


FIG. *3.16 : Représentation du réseau de neurones mis en place pour rechercher une solution au placement de tâches sur une zone reconfigurable. Le réseau dispose d'autant de lignes que de tâches à placer, et d'autant de colonnes que de PRRs disponibles dans la zone reconfigurable. Les cercles noirs indiquent qu'il existe une instance de tâche pour le PRR, alors qu'un cercle gris indique que la tâche ne peut pas être instanciée dans le PRR en question. Seuls les cercles noirs nécessitent la mise en place d'un neurone. En appliquant une règle* $k-de-N$ *sur les lignes (avec* $k=1$*) et une règle* $0-ou-k-de-N$ *sur les colonnes (avec* $k=1$*), le réseau converge vers une solution respectant les conditions énoncées dans 3.21.*

contraindre le réseau à configurer au moins une fois la tâche T_i dans un PRR. Ces deux règles correspondent aux sommations définies dans 3.21.

Nous avons réalisé la description de ce réseau, mais il est relativement aisé de comprendre qu'aucune optimisation n'est prise en compte dans cette structure. En effet, construit de cette façon, le réseau assure une convergence, mais aucun critère d'optimisation n'est mis en avant. Nos travaux actuels concernent la définition du critère d'optimisation. Nous avons pour cela ajouté des informations de positionnement des instances des tâches. Nous recherchons alors à compacter les tâches (en groupe, ou sur les bords de la zone reconfigurable) afin de conserver le maximum de rectangles vides ou, en tout cas, les rectangles vides les plus grands possibles de façon à pouvoir accepter les prochaines tâches plus facilement. La principale difficulté dans ces travaux concerne l'ajout de règles de convergence supplémentaires. En effet, l'application de plusieurs règles fait apparaître des minima locaux dans le processus d'optimisation et la convergence du réseau peut alors être stoppée dans un minimum ne représentant pas une solution correcte. En pondérant chaque règle de construction du réseau et par adaptation de ces pondérations il est toutefois possible de parvenir à faire converger le réseau.

La suite de ces travaux devrait nous amener à proposer une structure matérielle assurant l'ordonnancement spatio-temporel d'un ensemble de tâches et une méthodologie pour définir l'ensemble des poids des connexions entre les neurones. Un travail d'implémentation devra être réalisé et un contrôleur de réseau de neurones devra être défini. Ces travaux feront très probablement l'objet d'un projet technologique d'élève ingénieur ENSSAT durant l'année universitaire 2010-2011.

3.8 Conclusion et perspectives

Tout au long de ce chapitre, notre souci principal a consisté à limiter la complexité des réseaux de neurones permettant de modéliser le problème d'ordonnancement. Cette recherche nous a conduit à remettre en cause le modèle de Hopfield. Rappelons qu'en respectant le modèle de Hopfield, le réseau peut présenter des minima locaux qu'il faut détecter afin de s'en extraire et de pouvoir converger vers une solution valide.
Si la présence de minima locaux n'est qu'un moindre souci dans une simulation logicielle, elle est beaucoup plus problématique lorsque l'on envisage une implémentation matérielle. La détection des minima ainsi que l'extraction de ces minima rendent le contrôle du réseau beaucoup plus délicat. Il nous est alors apparu intéressant de proposer une structure de réseau minimisant aussi le contrôle de celui-ci.
Nous avons proposé plusieurs évolutions permettant tout d'abord de prendre en compte l'hétérogénéité de l'architecture. Nous avons ensuite proposé une structure basée sur la mise en place de neurones inhibiteurs. Cette structure possède une évolution de complexité limitée et est une réponse intéressante pour les deux objectifs que nous ciblons. Nous avons poursuivi nos investigations pour prendre en compte la gestion d'une zone reconfigurable au sein d'un SoC, ce qui nous amené à proposer une structure spécifique pour la gestion efficace d'une telle zone. Nous poursuivons actuellement des travaux pour prendre en compte le placement des tâches au sein de la zone reconfigurable. La structure du réseau de neurones proposé permet une prise en compte plus réaliste des implémentations possibles des tâches sur la zone reconfigurable, tout en assurant une certaine compacité des solutions générées.
Finalement, nous avons entamé des travaux de description VDHL des structures proposées. Dans un proche avenir nous devrions être en mesure de fournir les premiers résultats de synthèse de ces différents éléments et de montrer que nos solutions présentent de très bonnes caractéristiques pour une implémentation dans un RSoC.

Troisième partie

Conclusion et perspectives

Cette partie conclut tout d'abord sur mes activités d'enseignement puis sur mes activités de recherche. Une deuxième partie est consacrée aux perspectives de recherche en prenant en compte l'évolution constatée des systèmes sur puce.

Conclusion et réflexions concernant l'enseignement

Depuis ma nomination au poste de Maître de Conférences à l'ENSSAT en septembre 1997, j'ai assuré, en plus de mes enseignements statutaires à l'ENSSAT, un certain nombre d'enseignements dans d'autres établissements. Au travers de ces différentes expériences, j'ai pu côtoyer un public d'apprenants variés : étudiants "classiques", étudiants "CNAM" et ingénieurs en formation. Conscient que ces différentes expériences sont nécessaires pour bien appréhender le métier d'enseignant, j'ai favorisé autant que faire se peut ce type d'expériences pour quelques doctorants dont j'ai assuré, ou non, l'encadrement de thèse. Ainsi, j'ai permis à plusieurs doctorants de prendre en charge des travaux dirigés, travaux pratiques et projets pour des modules d'"Architectures évoluées des processeurs" et de "Temps réel". J'ai aussi proposé à de nombreuses reprises et à plusieurs doctorants, de prendre en charge une partie de mon cours magistral. À chaque fois, ces enseignements ont été préparés avec les doctorants, en prenant le temps nécessaire pour détailler les objectifs du cours et vérifier qu'ils sont atteints.

Parmi les projets d'enseignement proposés aux doctorants, l'un des plus délicats est très certainement celui que nous réalisons avec les étudiants de niveau BAC+4 en projet de "Temps réel". La difficulté de ce projet concerne la mise en œuvre de la méthodologie de conception SART. L'enseignement de la méthodologie et la pratique de celle-ci sont un exercice difficile. Pour un doctorant assurant un tel projet pour la première fois, la difficulté réside notamment dans l'absence de solution unique et dans la position délicate de client qu'il faut "jouer" vis-à-vis des étudiants. Les doctorants qui acceptent de prendre en charge ce type de projet doivent, à mon sens, faire preuve de recul vis-à-vis du sujet et de leurs connaissances éventuelles d'une solution afin de placer l'étudiant dans une position de décideur et de défenseur de sa solution. Au-delà des aspects techniques inévitables pour lesquels il faut aider les étudiants, ce projet est des plus formateurs, pour les étudiants, mais aussi pour l'encadrant.

Au cours de ces quelques dix années d'enseignement, j'ai eu l'occasion de mettre en pratique des enseignements basés sur plus d'autonomie des étudiants. Notamment, j'ai participé au développement d'une plate-forme d'enseignement autour du langage VHDL pour le CNED. Cette plate-forme était composée d'une partie "apprentissage" basée sur un polycopié de cours adapté et d'un outil permettant de pratiquer les thèmes abordés. Cette plate-forme a été proposée à des stagiaires qui devaient s'initier à ce langage avant d'effectuer des développements. En règle générale, cette plate-forme a joué son rôle de prise en main du langage et a permis aux stagiaires de se former en autonomie puis de réaliser leur travail de stage par la suite.

Ce principe d'enseignement à distance, ou en tout cas en non présentiel est déjà bien présent, notamment au travers de la mise à disposition de contenus numériques sur les plate-formes des Universités Numériques Thématiques. S'il est clair que ces offres vont très certainement s'étoffer dans les années à venir, il semble évident que les enseignements proposés dans ce cadre doivent être repensés, dans le contenu mais aussi et surtout dans la forme. En effet, les enseignements traditionnels, sous forme de cours magistraux se prêtent très mal à ce nouveau type de support où l'apprenant est seul devant sa machine et où son attention doit être captée pour éviter qu'il ne passe à autre chose. Le format idéal, notamment dans sa durée, doit être revu, et ramené dans un intervalle de 15 à 20 minutes d'apprentissage par séance, puisqu'il a été démontré qu'au delà de ce temps, la lassitude s'installe et que l'efficacité diminue nettement. De même, la simple mise à disposition d'un support de cours sur un site web n'est pas une solution suffisante et doit s'accompagner d'autres outils. Les outils que nous avons développés dans le cadre du cours d'architectures des processeurs constituent un premier pas pour tenter de répondre à cette problématique. Nous poursuivrons cet effort de développement et tenterons d'étoffer les moyens pédagogiques afin de placer l'étudiant en position d'acteur de sa formation. Mon implication dans la cellule TICE de l'ENSSAT est un moyen de suivre l'évolution des offres pédagogiques et de proposer, à la communauté des enseignants de l'école, des évolutions en vue de nous adapter à notre public d'étudiants.

Finalement, en un peu plus de dix ans d'enseignement, en particulier à l'ENSSAT, j'ai pu observer l'évolution des étudiants, de moins en moins disposés à s'approprier le contenu d'un cours lorsque celui-ci est trop "magistral" et qu'ils ont du mal à projeter les compétences abordées dans leur futur métier. Pour favoriser l'intérêt des étudiants, il faut renforcer leur positionnement en tant qu'acteurs de leur formation et non plus simplement en position de spectateurs des cours qui s'enchaînent. Pour parvenir à cela, une possibilité pourrait consister à démarrer les enseignements par la présentation d'un système qui capte l'intérêt des étudiants. Actuellement, un point de départ pourrait être un PDA (Personnal Digital Assistant) pour lequel on amènerait progressivement l'étudiant d'une position d'utilisateur à une position de concepteur, tout au moins d'une partie du système. Par décomposition progressive du système, il serait possible d'exhiber plusieurs domaines et de montrer les mécanismes et phénomènes mis en œuvre. Cette façon de procéder permettrait de conserver l'enseignement des parties plus théoriques mais dans ce cas l'étudiant pourrait trouver d'emblée une motivation et un intérêt à la formalisation.

Conclusion sur mes activités de recherche

Depuis que j'ai débuté mes activités de recherche, j'ai eu l'occasion de voir une évolution importante dans le domaine de la conception des systèmes intégrés. Nous sommes passés de techniques de conception d'ASIC en nous concentrant presque uniquement sur les aspects traitements, et en ne considérant que la surface de silicium comme paramètre à optimiser, à des techniques de conception de SoCs. Les travaux de l'équipe CAIRN ont pris en compte cette évolution en intégrant de nouvelles contraintes, comme par exemple la minimisation de la contrainte de consommation énergétique. Cette contrainte nous a alors amenés à considérer d'autres architectures, notamment les architectures reconfigurables. Actuellement, l'équipe est impliquée dans plusieurs projets mettant en œuvre le concept de reconfiguration dynamique.

Dans ce contexte, j'ai abordé plusieurs problématiques, regroupées dans ce document. Tout d'abord, j'ai abordé la problématique de la définition de hiérarchies mémoires efficaces pour un système sur puce. Notre objectif a été de répondre aux contraintes de performances tout en montrant qu'il existe un potentiel d'optimisation énergétique important sur cette partie du système. Nous avons proposé une structure dont l'objectif principal est de localiser des mémoires au plus près des unités de calculs pour obtenir de bonnes performances et limiter le coût énergétique du système. Nous avons aussi montré qu'il existe un potentiel d'optimisation énergétique intéressant sous l'hypothèse que l'on sache adapter dynamiquement la tension d'alimentation des mémoires au fil de l'eau.

La maîtrise de la conception d'un système sur puce, qu'il soit reconfigurable ou non, est une problématique délicate, et pour tenter de fournir une solution à ce problème, nous avons proposé une modélisation de la partie reconfigurable d'un RSoC. Le modèle proposé repose sur le langage UML et la modélisation a été l'un des supports importants pour le développement du modèle de simulation SystemC d'un système complet contenant une zone reconfigurable. Ce modèle de simulation a été intégré dans l'outil développé durant le projet OverSoC.

D'une façon plus générale, ces travaux nous ont amenés à proposer une réflexion sur les services du système d'exploitation. Le service de communication est l'un des services les plus fortement impactés par la mise à disposition de ce type de ressource d'exécution. Nous avons alors entamé une réflexion sur ce service et les travaux de thèse menés dans le cadre du projet FosFor nous ont conduits à proposer une topologie particulière de réseau sur puce et à développer les mécanismes de gestion de ce réseau.

Le service de création et de gestion des tâches est lui aussi impacté par l'apparition d'une ressource reconfigurable au sein des SoC. En effet, la présence de cette ressource reconfigurable au sein du système rend le système hétérogène. Dans ce contexte, nous avons proposé des solutions d'ordonnancement basé sur l'utilisation des réseaux de neurones. Nous avons étendu les propositions classiques de réseaux de neurones en prenant en compte l'hétérogénéité des ressources d'exécution. Nous poursuivons actuellement nos travaux pour une modélisation plus réaliste de l'ordonnancement des tâches au sein de la zone reconfigurable en prenant en compte le paramètre "géographique". En effet, pour exécuter une tâche matérielle sur la zone reconfigurable, l'ordonnanceur doit alors gérer la position "géographique" de la tâche. Pour être complet, nous conduirons des travaux d'implémentation sur cible FPGA de la solution proposée afin d'effectuer une validation matérielle.

Perspectives de recherche

Les évolutions très importantes auxquelles les concepteurs ont été confrontés, aussi bien du point de vue technologique que du point de vue des besoins applicatifs, ont conduit aux développements des systèmes très complexes que sont les systèmes sur puce. L'intégration des capacités de reconfiguration au sein de ces SoCs permet actuellement d'envisager des systèmes qui s'adaptent à leur environnement. Néanmoins, la gestion de ce type de plate-forme peut sans aucun doute progresser afin de fournir des systèmes proposant une qualité de services conditionnée à l'utilisation. Dans ce cadre, j'entends proposer, sous contraintes de ressources et en restant dans le spectre large des activités de l'équipe, des travaux dans les axes présentés dans les sections suivantes.

Maîtrise de la consommation des systèmes

Si la technologie propose des solutions pour mieux répondre aux contraintes, comme la minimisation de la consommation énergétique en fournissant des mécanismes de gestion de la tension d'alimentation, il est toutefois nécessaire de proposer un support en terme d'*operating system* afin d'élever le niveau d'abstraction de ces mécanismes. C'est dans ce cadre que vont se poursuivre les travaux que nous avons entamés au sein du projet Open-People concernant les architectures reconfigurables. En effet, ces deux aspects seront abordés et permettront de proposer aux concepteurs à la fois des modèles de consommation des composants, mais aussi et surtout des techniques d'optimisation en lien très étroit avec le système d'exploitation. La construction des modèles de consommation des architectures reconfigurables nécessitera de cibler précisément chaque partie de ce type d'architecture afin d'extraire les paramètres du modèle de consommation. On s'attachera à solliciter chaque partie de l'architecture en configurant des tâches spécifiques, par exemple une tâche de contrôle pour solliciter uniquement les CLBs de l'architecture, une tâche de traitement de type filtrage pour solliciter spécifiquement les blocs DSP, etc. De plus, pour pouvoir proposer des optimisations au niveau du système d'exploitation, nous construirons un modèle de consommation pour les mécanismes de reconfiguration dynamique et partielle. Pour définir ce modèle de consommation, un grand nombre de paramètres sera sans doute à analyser, on pourra citer par exemple la taille du bitstream, la position de la reconfiguration, le type d'éléments configurés, etc. Le cadre de travail que nous avons défini pour les 3 années de thèse de Robin Bonamy permettra d'exploiter pleinement les compétences complémentaires des équipes CAIRN et du LEAT pour bâtir les modèles et les exploiter au niveau de l'application. Les techniques d'optimisation qui seront proposées se heurteront peut être à des impossibilités inhérentes aux architectures reconfigurables dynamiquement et partiellement disponibles à l'heure actuelle ou à venir. Si tel est le cas, un travail visant à définir l'architecture reconfigurable "idéale" d'un point de vue du système d'exploitation et de l'application sera envisagé. L'expérience accumulée au sein de l'équipe CAIRN concernant la gestion des architectures reconfigurables pourrait effectivement amener un point de vue différent quant à l'organisation de telles architectures, en prenant en compte dès la conception les problématiques de préemption et de migration des tâches matérielles, tout en tenant compte des classiques contraintes de performances et de consommation.

La modélisation, une approche incontournable

D'une façon générale, il apparaît assez évident qu'il existe un lien étroit entre les avancées architecturales et les services à mettre en place afin que les applications puissent en bénéficier. Les progrès technologiques et architecturaux doivent s'accompagner d'efforts aux niveaux supérieurs (notamment au niveau système) sous peine de ne pas être exploitables. Finalement, cela signifie que la conception d'un système passe par une maîtrise de l'ensemble des éléments qui le composent, depuis l'application, jusqu'à l'architecture. Compte tenu de la complexité croissante de ces systèmes, il est inconcevable d'imaginer que cette maîtrise puisse se passer d'outils logiciels. Ces outils doivent d'ores et déjà intégrer des méthodologies de conception, s'appuyant sur des modèles, et permettre l'exploration et la sélection de solutions au plus tôt dans le cycle de développement. Il est indéniable que ce champ de recherche peut et doit encore progresser, et cela sera d'autant plus vrai que les progrès architecturaux et technologiques se poursuivront à un rythme soutenu. La modélisation à un haut niveau, au travers de langages de modélisation comme AADL, MARTE, CAL, ou d'autres, a déjà permis d'apporter une meilleure maîtrise du processus de conception, il semble évident que cela va se poursuivre. Les travaux que nous menons autour des architectures reconfigurables, et qui se situent à un assez bas niveau, devront être "remontés" dans les couches

hautes de la modélisation afin que les outils et méthodes puissent les exploiter. Ces travaux, que nous avons menés dans le cadre des projets OverSoC et FosFor, seront poursuivis par une démarche de modélisation à un plus haut niveau. De plus, dans le cadre du projet Open-People, j'ai entamé des travaux de modélisation au travers du langage AADL, cette voie me semble une réponse incontournable pour la conception des futurs systèmes. Je suis convaincu que seules les approches formelles peuvent apporter suffisamment d'abstraction pour que les concepteurs puissent continuer à "comprendre" ce qu'ils conçoivent. Je souhaite poursuivre mon investissement dans cet axe de recherche. Des travaux autour de la hiérarchie mémoire que nous avons proposée pourraient être une bonne base de départ et nous permettraient de proposer un flot d'exploration pour cette architecture.

Vers des architectures multi-cœurs

D'autre part, toujours sur cette thématique de la modélisation, la généralisation des architectures multi-cœurs engendre une demande importante d'outils et de méthodes pour maîtriser le développement des systèmes. L'évolution rapide vers des systèmes *many-core* va très probablement complexifier la problématique et le passage à l'échelle est loin d'être simple. Les problématiques de développement logiciel par exemple se trouvent très largement impactées par cette évolution et seule une approche par modélisation à un haut niveau me semble pouvoir proposer des outils et méthodes permettant aux concepteurs de maîtriser le développement et le déploiement d'applications sur ces systèmes. L'exploration de l'ensemble du système ne pourra raisonnablement être réalisée et il sera nécessaire de disposer de méthodes formalisées assez éloignées des approches classiques. Ces nouvelles approches méthodologiques devront d'ailleurs être rapidement insérées dans le cursus d'enseignement des élèves ingénieurs puisqu'ils seront les prochains utilisateurs de ces concepts pour concevoir les systèmes numériques de demain. Je pense qu'il y a donc sur ce point une synergie intéressante à mettre en place entre les activités de recherche que nous menons et les enseignements relevant des aspects méthodologie, notamment en développement de systèmes temps réel. Dans le cadre pédagogique qui est en place à l'ENSSAT, je poursuivrai la sensibilisation des étudiants à ces nouveaux concepts afin qu'ils soient mieux armés pour leur futur métier.

Pour conserver de bonnes performances au sein des systèmes embarqués, cette évolution vers les architectures multi-cœurs ne pourra pas se passer de l'utilisation d'unités spécifiques : blocs IP, zone reconfigurable, etc. Pour ce domaine, il semble alors évident que l'hétérogénéité restera la base des architectures. Ces systèmes, qui auront des interactions de plus en plus importantes avec l'environnement, devront disposer de mécanismes d'optimisation en ligne, et de nouvelles techniques devront sans aucun doute être développées pour répondre aux exigences applicatives. Des techniques d'optimisation issues du domaine du vivant (algorithmes génétiques du type réseaux de neurones, colonies de fourmis, essaim d'abeilles, etc) disposent de propriétés intéressantes qui pourraient permettre une gestion adaptée de ces systèmes. Notre expérience de l'optimisation du problème d'ordonnancement par réseaux de neurones pourrait être un point de départ pour l'étude de ces techniques et leur extension vers une prise en compte plus large des contraintes des systèmes embarqués.

De plus, l'apparition de circuits dits 3D, pour lesquels plusieurs plans de transistors sont empilés, conduira sans doute les concepteurs à proposer des systèmes avec plusieurs plans de processeurs (actuellement, les systèmes envisagés sont plutôt construits en empilant un plan multi-cœurs avec un plan de mémoires). Dans ce contexte, les travaux d'ordonnancement spatio-temporel que nous menons actuellement pourraient être ré-utilisés en ajoutant une dimension supplémentaire. La gestion de cette nouvelle dimension se traduira par l'ajout de nouvelles règles de construction du réseau de neurones et rendra probablement la convergence plus délicate à obtenir, mais je crois qu'il est intéressant d'explorer cette piste en avance de phase par rapport à la diffusion de ces systèmes.

Notons que si ces systèmes peuvent être une réponse intéressante à la fois pour les concepteurs de cœurs de processeurs matériels, de plans mémoire, etc, l'empilement des couches de silicium va engendrer une difficulté pour la dissipation thermique. Une gestion plus fine de l'activité des différentes parties du système sera alors nécessaire, et cela sera à réaliser en ligne. Des techniques d'optimisation dynamique devront être développées, et là aussi, les algorithmes génétiques pourraient apporter une solution intéressante.

Architectures à forte variabilité technologique

Si l'évolution technologique apporte sans cesse un accroissement des performances, elle engendre aussi une variabilité de plus en plus importante au sein du système. Cette variabilité peut avoir plus ou moins d'impact, en allant de la non prédictibilité des performances du système à l'apparition de fautes dans le système. Dans ce contexte, il est clair qu'il sera nécessaire de définir des architectures capables de s'adapter à cette caractéristique des futurs circuits/systèmes. Les caractéristiques intrinsèques des algorithmes d'optimisation dynamique sont une opportunité intéressante pour imaginer de nouveaux mécanismes de contrôle au sein de systèmes soumis à cette variabilité. Le vivant sait parfaitement "fonctionner" même en cas de défaut de l'un des éléments le constituant, la "redondance" est l'un des mécanismes qui permet à l'ensemble de poursuivre son activité malgré des fautes élémentaires.

Les compétences acquises lors de nos travaux sur l'ordonnancement de tâches par réseaux de neurones pourraient très certainement être exploitées pour contrer ces effets néfastes en profitant du caractère adaptatif de ces structures. Dans certaines conditions, les modèles de réseaux de neurones que nous avons développés supportent un défaut (issu d'une variabilité technologique ou d'un vieillissement amenant à un collage par exemple) au sein même du réseau. En effet, le défaut sur un neurone peut perturber l'évolution du réseau, mais la convergence peut encore être atteinte. Pour pouvoir être généralisable, un modèle de réseaux de neurones générique supportant les défauts est à bâtir. L'une des difficultés pour parvenir à valider une telle proposition sera de disposer de moyens fiables pour vérifier le bon fonctionnement en présence de ces défauts de variabilité. Un effort de modélisation de la variabilité devra sans aucun doute être réalisé afin de pouvoir disposer de modèles de simulation ayant la capacité à montrer l'intérêt de nos propositions. Ces travaux pourraient débuter par un travail de stage de master sur la définition de ce modèle de variabilité et se poursuivre par un travail de thèse pour définir le modèle de réseaux de neurones en charge de la gestion globale du système.

Un devoir pour les futures générations

Enfin, je pense qu'un défi important repose depuis quelques années sur les *épaules* de la recherche en France et plus largement dans le monde, il s'agit de la prise en considération que nous sommes les initiateurs de systèmes toujours plus complexes, rendant toujours plus de services, et entraînant dans ce sillage toujours plus de consommation énergétique. Sans vouloir remettre en cause l'évolution technologique, ni en appeler à un modèle économique de décroissance, il me semble que nous ne devons pas perdre de vue qu'il est de notre responsabilité de concevoir des systèmes de moins en moins gourmands en énergie. Si des travaux importants ont déjà été faits sur ce point, les prévisions de consommation énergétique pour les années à venir poussent les industriels à une mobilisation importante pour limiter leur empreinte énergétique, on citera par exemple l'initiative *GreenTouch* pilotée par Alcatel, ou encore la réelle prise en compte de cette problématique par les fournisseurs de services Internet (sites de recherche par exemple). Dans ces systèmes, un grand nombre d'équipements est un jeu, on citera par exemple les routeurs internet, les éléments de stockage, les stations de base, les serveurs de calcul, etc et la multiplication de ces éléments à très grand échelle pose alors la question de l'optimisation de chacun pour gagner de l'énergie à tous les niveaux. Les architectures reconfigurables, associées à des mécanismes de mise en veille intelligente des équipements, permettraient de mettre en place des stratégies d'optimisations importantes. Notons toutefois qu'aussi intelligentes qu'elles soient, les optimisations proposées au niveau circuit devront être exploitables par l'application. Le lien avec le système d'exploitation est donc très important et devrait se renforcer dans les architectures à venir. Ce challenge ne pourra donc être relevé qu'avec l'ensemble de la communauté des concepteurs de ces systèmes, des technologues, aux développeurs d'application, en passant par les architectes du matériel et du système d'exploitation.

Dans ce contexte, notre rôle, en tant qu'enseignant chercheur de ce large domaine, est d'accompagner, d'encourager et de participer à cet effort de réduction de l'énergie des systèmes numériques, notamment en sensibilisant plus fortement nos futurs ingénieurs/doctorants à cette problématique, pour des raisons économiques mais aussi et plus simplement pour des raisons environnementales.

Bibliographie

[1] Imène Benkermi. *Modèle et algorithme d'ordonnancement pour architectures reconfigurables dynamiquement*. PhD thesis, ENSSAT-Université de Rennes, Irisa, Janvier 2007.

[2] J.G. Cousin. *Méthodologie de conception de cœurs de processeurs spécifiques : mise en œuvre sous contraintes, estimation de la consommation*. PhD thesis, ENSSAT-Université de Rennes 1, Septembre 1999.

[3] R. Yu. *Estimation de haut niveau du placement et des interconnexions dans les circuits VLSI submicroniques*. PhD thesis, ENSSAT-Université de Rennes, 7 mai 2002.

[4] Ekué Kinvi-Boh. *Conception de circuits en logique ternaire : de la caractérisation au niveau transistor à la modélisation architecturale*. PhD thesis, ENSSAT-Université de Rennes, 2006.

[5] J.O. Dedou. *Synthèse de haut niveau d'architectures asynchrones en traitement numérique du signal*. PhD thesis, ENSSAT-Université de Rennes, Octobre 2000.

[6] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, fourth edition, 2007.

[7] Daniel Wiklund. Data and memory optimization techniques for embedded systems. Dept. of Electrical Engineering, Linkoping University, Sweden, 2001.

[8] John Mellor Crummey, David Whalley, and Ken Kennedy. Improving memory hierarchy performance for irregular applications using data and computation reorderings. In *Int. J. Parallel Program.*, volume 29, pages 217–247, Norwell, MA, USA, 2001. Kluwer Academic Publishers.

[9] Antoine Fraboulet, Guillaume Huard, and Anne Mignotte. Optimisation de la consommation et de la place mémoire par transformations de boucles. Colloque CAO de circuits intégrés et systèmes, Aix en Provence, May 1999.

[10] Russell Tessier, Vaughn Betz, David Neto, and Thiagaraja Gopalsamy. Power aware ram mapping for fpga embedded memory blocks. In *FPGA'06 : Proceedings of the internation symposium on Field programmable gate arrays*, pages 189–198, New York, NY, USA, 2006. ACM Press.

[11] Wen-Tsong Shiue, Sathishkumar Udayanarayanan, and C. Chakrabarti. Data memory design and exploration for low-power embedded systems. *ACM Transactions on Design Automation of Electronic Systems.*, 6(4) :553–568, 2001.

[12] Poletti Francesco, Paul Marchal, David Atienza, Luca Benini, Francky Catthoor, and Jose M. Mendias. An integrated hardware/software approach for run-time scratchpad management. In *DAC '04 : Proceedings of the 41st annual conference on Design automation*, pages 238–243, New York, NY, USA, 2004. ACM Press.

[13] Paul M. Petersen and David A. Padua. Static and dynamic evaluation of data dependence analysis techniques. *IEEE Trans. Parallel Distrib. Syst.*, 7(11) :1121–1132, 1996.

[14] Amir H. Farrahi, Gustavo E. Téllez, and Majid Sarrafzadeh. Memory segmentation to exploit sleep mode operation. In *DAC '95 : Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 36–41, New York, NY, USA, 1995. ACM Press.

[15] Mmalpha. http ://www.irisa.fr/cosi/Alpha.

[16] Powerescape. http ://www.powerescape.com/technology/papers.

[17] Synfora. http ://www.synfora.com.

[18] Cédric Bastoul. *Amélioration de la localité dans les programmes à contrôle statique*. PhD thesis, Université de Paris 6, December 7 2004.

[19] Antoine Fraboulet. *Optimisation de la mémoire et de la consommation des systèmes multimédia embarqués*. PhD thesis, Institut National des Sciences Appliquées de Lyon, nov 23 2001.

[20] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.*, 18(4) :424–453, 1996.

[21] Bjorn Franke, Michael O'Boyle, John Thomson, and Grigori Fursin. Probabilistic source-level optimisation of embedded programs. In *LCTES '05 : Proceedings of the 2005 ACM Sigplan/Sigbed Conference on Languages, compilers, and tools for embedded systems*, pages 78–86, New York, NY, USA, 2005. ACM Press.

[22] Prasad Kulkarni, Wankang Zhao, Hwashin Moon, Kyunghwan Cho, David Whalley, Jack Davidson, Mark Bailey, Yunheung Paek, and Kyle Gallivan. Finding effective optimization phase sequences. In *LCTES '03 : Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 12–23, New York, NY, USA, 2003. ACM Press.

[23] Preeti R. Panda, Francky Catthoor, Nikil D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 6(2) :149–206, 2001.

[24] Dan Nam Truong. *Optimisations logicielles de la localité, le placement précis des données en mémoire*. PhD thesis, Université de Rennes 1, Institut de Formation Superieure en Informatique et Communication, 1998.

[25] Xavier Vera, Josep Llosa, and Antonio Gonzalez. Near optimal padding for removing conflict misses, 2002.

[26] L. Benini, D. Bruni, A. Maciiand, and E. Maci. Memory energy minimization by data compression : algorithms, architectures and implementation. *IEEE Trans. VLSI Syst*, 12 :255–268, March 3 2004.

[27] A. Beszedes, Rudolf Ferenc, Tibor Gyimothy, André Dolenc, and Konsta Karsisto. Survey of code-size reduction methods. *ACM Comput. Surv.*, 35(3) :223–267, 2003.

[28] Barry Shackleford, Mitsuhiro Yasuda, Etsuko Okushi, Hisao Koizumi, Hiroyuki Tomiyama, and Hiroto Yasuura. Memory cpu size optimization for embedded system designs. In *DAC '97 : Proceedings of the 34th annual conference on Design automation*, pages 246–251, New York, NY, USA, 1997. ACM Press.

[29] M Loghi, O Golubeva, E Macii, and M Poncino. Architectural leakage power minimization of scratchpad memories by application-driven sub-banking. *Computers, IEEE Transactions on*, PP(99) :1 –1, 2010.

[30] Mahmut Kandemir, J. Ramanujam, M.J. Irwin, N. Vijaykrishnan, I. Kadayif, and A Parikh. A compiler based approach for dynamically managing scratch-pad memories in embedded systems. *IEEE Transactions on : Computer-Aided Design of Integrated Circuits and Systems*, 23(2) :243– 260, Feb 2004.

[31] Maha Idrissi Aouad and Olivier Zendra. A survey of scratch-pad memory management techniques for low-power and-energy. In *ICOOOLPS*, 2007.

[32] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu. Banked scratch-pad memory management for reducing leakage energy consumption. In *Proc. of ICCAD 04*, pages 120–124, nov 2004.

[33] Songping Mai, Chun Zhang, Yixin Zhao, Jun Chao, and Zhihua Wang. An application-specific memory partitioning method for low power. pages 221 –224, oct. 2007.

[34] Tadayuki Matsumura, Tohru Ishihara, and Hiroto Yasuura. An optimization technique for low-energy embedded memory systems. *IPSJ Transactions on System LSI Design Methodology*, 2 :239–249, 2009.

[35] Afrin Naz, Mehran Rezaei, Krishna Kavi, and Philip Sweany. Improving data cache performance with integrated use of split caches, victim cache and stream buffers. *SIGARCH Comput. Archit. News*, 33(3) :41–48, 2005.

[36] Chuanjun Zhang and Frank Vahid. Cache configuration exploration on prototyping platforms. In *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP'03)*, page 164. IEEE Computer Society, 2003.

[37] Chuanjun Zhang and Frank Vahid. Using a victim buffer in an application-specific memory hierarchy. In *DATE'04 : Proceedings of the conference on Design, automation and test in Europe*, pages 220–225, CNIT La Defese, Paris, France, feb 16-20 2004. IEEE Computer Society.

[38] R. Saied and C. Chakrabarti. Scheduling for minimizing the number of memory accesses in low-power applications. In *Proc. of the VLSI Signal Processing Workshop*, pages 169–178, San Francisco, CA , USA, oct 1996.

[39] Jaewon Seo, Taewhan Kim, and Preeti R. Panda. An integrated algorithm for memory allocation and assignment in high-level synthesis. In *DAC '02 : Proceedings of the 39th conference on Design automation*, pages 608–611, New Orleans, Louisiana, USA, jun 10 - 14 2002. ACM Press.

[40] S. Wuytack, F. Catthoor, G. De Jong, and H. De Man. Minimizing the required memory bandwidth in vlsi system realizations. *IEEE Trans. on VLSI Systems*, 7(4) :433–441, dec 1999.

[41] Chun-Gi Lyuh and Taewhan Kim. Memory access scheduling and binding considering energy minimization in multi-bank memory systems. In *DAC '04 : Proceedings of the 41st annual conference on Design automation*, pages 81–86, San Diego, CA, USA, jun 7-11 2004. ACM Press.

[42] Gwénolé Corre. *Synthèse des unités mémoires*. PhD thesis, Université de Bretagne Sud, February 2005.

[43] G. Talavera, M. Jayapala, J. Carrabina, and F. Catthoor. Address generation optimization for embedded high-performance processors : A survey. *Journal of Signal Processing Systems*, 53(3) :271–284, 2008.

[44] Francky Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandercappelle. *Custom Memory Management Methodology*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[45] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2) :34–44, /1997.

[46] Richard Fromm, Stylianos Perissakis, Neal Cardwell, Christoforos E. Kozyrakis, Bruce McGaughy, David A. Patterson, Thomas E. Anderson, and Katherine A. Yelick. The energy efficiency of IRAM architectures. In *ISCA*, pages 327–337, 1997.

[47] Joseph Gebis, Sam Williams, David Patterson, and Christos Kozyrakis. Viram1 : A media-oriented vector processor with embedded dram. In *DAC04*, pages 7–11, San Diego, California, USA, Jun 2004.

[48] Saman Amarasinghe. The future. Lecture 18, MIT, 2007.

[49] R. David. *Architectures enfouies reconfigurables dynamiquement pour les télécommunications mobiles 3G : modèle et outils*. PhD thesis, ENSSAT-Université de Rennes, 2003.

[50] Raphaël David, Dominique Lavenier, and Sebastien Pillement. Du microprocesseur au circuit fpga, une analyse sous l'angle de la reconfiguration. *Technique et Science Informatiques*, 24(4) :395–422, 2005.

[51] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. Pact xpp — a self-reconfigurable data processing architecture. *The Journal of Supercomputing*, 26(2) :167–184, 2003.

[52] R. David, S. Pillement, and O. Sentieys. Energy-Efficient Reconfigurable Processsors. In C. Piguet, editor, *Low Power Electronics Design*, Computer Engineering, Vol 1, chapter 20. CRC Press, August 2004.

[53] Stephane Chevobbe. *Unité de commande pour systèmes parallèles contrôleur basé sur l'implémentation dynamique de réseaux de Petri*. PhD thesis, Université de Rennes I, IRISA - CEA List, October 2005.

[54] Jean-Marc Masgonty, Stefan Cserveny, and Christian Piguet. Low-power sram and rom memories. In *PATMOS'01 : Proceedings of the 11th International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 7.4.1 – 7.4.8, Yverdon-les-Bains, Switzerland, sep 26-28 2001.

[55] Kurt Keutzer, Sharad Malik, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli. System-Level Design : Orthogonalization of Concerns and Platform-Based Design. *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, 19(12) :1523–1543, December 2000.

[56] Oliver Diessel and Grant Wigley. Opportunities for Operating Systems Research in Reconfigurable Computing. Technical Report ACRC-99-018, Advance Computing Research Center, School of Computer and Information Science, Univ. of South Australia, August 1999.

[57] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an Operating System for a Heterogeneous Reconfigurable SoC. In *Reconfigurable Architectures Workshop*, Nice, France, April 2003.

[58] Herbert Walder and Marco Platzner. Reconfigurable Hardware Operating Systems : From Design Concepts to Realizations. In *Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*, pages 284–287. CSREA Press, June 2003.

[59] M. Ullmann, M. H¸bne, B. Grimm, and J. Becker. On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities. In *Field Programmable Logic and Application : 14th International Conference, FPL*, pages 454–463. Springer-Verlag Heidelberg, August 2004.

[60] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J.C. PrÈvotet, and F. Verdier. System-level modelling for reconfigurable socs. *DCIS'05, XX Conference on Design of Circuits and Integrated Systems Lisboa - Portugal*, November 23-25 2005.

[61] ATMEL. FPSLIC (AVR with FPGA). `http://www.atmel.com/products/FPSLIC/`.

[62] H. Schueler. Smart media processing with XPP, white paper. `http://www.pactcorp.com/xneu/px_SMeXPP.html`, April 2003.

[63] Raphaël David, Daniel Chillet, Sebastien Pillement, and Olivier Sentieys. Dart : A dynamically reconfigurable architecture dealing with next generation telecommunications constraints. *9th IEEE Reconfigurable Architecture Workshop RAW*, April 2002.

[64] G. Nicolescu and J. Mosterman. *Model-Based Design for Embedded Systems*. CRC Press, 2009.

[65] S. Evain, J. P. Diguet, M. El Khodary, and D. Houzet. Automated Derivation of NoC Communication Specifications From Application Constraints. pages 238–243, Octobre 2006.

[66] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago Gonzalez Pestana, Andrei Radulescu, and Edwin Rijpkema. A Design Flow for Application-specific Networks on Chip with Guaranteed Performance to Accelerate SoC Design and Verification. pages 1182–1187, Washington, DC, USA, 2005. IEEE Computer Society.

[67] Luca Benini and Giovanni DeMicheli. Networks on Chips : A New Paradigm for System on Chip Design. *Computer*, pages 70–78, january 2002.

[68] Liwei Ma and Yihe Sun. On-Chip Network Design Automation with Source Routing Switches. *Tsinghua Science & Technology*, 12(1) :77–85, February 2007.

[69] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on chip : An Architecture for Billion Transistor Era. 2000.

[70] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *In IEEE Design and Test -Special Issue on Reconfigurable Computing*, 17(1) :68–83, Jan.-Mar. 2000.

[71] E. Salminen, A. Kulmala, and T. D. Hamalainen. Survey of network-on-chip proposals. *OCP-IP White Paper,* `http ://www.ocpip.org/whitepapers.php`, 2008.

[72] V. Nollet, T. Marescaux, P. Avasare, and J-Y. Mignolet. Centralized Run-time Resource Management in a Network-on-chip Containing Reconfigurable Hardware Tiles. pages 234–239, 2005.

[73] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen. DyNoC : A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices. pages 153–158, 2005.

[74] Andrew Morton and Wayne M. Loucks. A hardware/software kernel for system on chip designs. In *Proceedings of the ACM Symposium on Applied Computing*, pages 869–875, 2004.

[75] T.Samuelsson, M.Akerholm, P.Nygren, J.Starner, and L.Lindh. A comparison of multiprocessor real-time operating systems implemented in hardware and software. In *International workshop on advanced real-time operating system services*, Porto, portugal, july 2003.

[76] P. Kuacharoen, M. Shalan, and V. Mooney III. A configurable hardware scheduler for real-time systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pages 96–101, Las Vegas, USA, June 2003.

[77] Paul Kohout, Brinda Ganesh, and Bruce Jacob. Hardware support for real-time operating systems. In *CODES+ISSS '03 : Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 45–51, New York, NY, USA, october 2003. ACM Press.

[78] Mohamed Shalan and III Vincent J. Mooney. Hardware support for real-time embedded multiprocessor system-on-a-chip memory management. In *CODES '02 : Proceedings of the tenth international symposium on Hardware/software codesign*, pages 79–84, New York, NY, USA, may 2002. ACM Press.

[79] N. Ventroux, S. Chevobbe, F. Blanc, and T. Collette. An auto-adaptative reconfigurable architecture for the control. In *Proceedings of the 9th Asia-Pacific Computer Systems Architecture Conference (ACSAC'04)*, Beijing, China, September 2004.

[80] C. Cardeira, M. Silva, and Z. Mammeri. Handling precedence constraints with neural network based real-time scheduling algorithms. *ecrts, 9th Euromicro Workshop on Real Time Systems (euromicro-rts '97)*, page 207, 1997.

[81] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79 :2554–8, 1982.

[82] G. Tagliarini, J. Fury Christ, and W. E. Page. Optimization using neural networks. *IEEE Trans. Comput.*, 40(12) :1347–58, December 1991.

[83] C. Cardeira and Z. Mammeri. Preemptive and non-preemptive real-time scheduling based on neural networks. *Proceedings DDCS'95*, pages 67–72, September 1995. the 13^{th} IFAC Workshop on Distributed Computer Control Systems, Toulouse France.

[84] I. Benkermi, S. Pillement, and O. Sentieys. *SympAAA'2003*, 14-17 Octobre 2003.

[85] J. J. Hopfield and D. W. Tank. Computation of decisions in optimization problems. *Biological Cybernetics*, 52 :141–52, 1985.

[86] S. Grossberg. *Studies of Mind and Brain : Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*, volume 70. Reidel Press Bonston, 1982.