Environnement de Synthèse d'Applications de Traitement du Signal

Michel AUGUIN, Cécile BELLEUDY,
Fernand BOERI, Alain GIULIERI,
Guy GOGNIAT

Laboratoire I3S - Université de Nice/Sophia-Antipolis, URA 1376 du CNRS
41 Bd Napoléon III - 06041 Nice cedex

RÉSUMÉ

Les systèmes embarqués interviennent dans un nombre toujours croissant d'applications. Les fonctions a y intégrer sont de plus en plus complexes. Au niveau technologique, les possibilités sont de plus en plus riches. La conception et le prototypage d'un système embarqué nécessitent des méthodes capables de gérer cette complexité. L'environnement CAPSYS permet d'explorer efficacement plusieurs solutions architecturales basées sur une structure de processeur spécialisé microcodé qui exploitent divers degré de parallélisme et réalisent ainsi un compromis matériel/microcode.

ABSTRACT

Embedded systems find their way more and more into a wide range of applications. Complexity of integrated functionalities is increasing continuously. Technology provides more and more efficient possibilities. Hence, designing and prototyping an embedded system require methodologies able to deal with this complexity. The CAPSYS framework permits to explore efficiently various designs, based on a microcoded specific processor architecture model, that consider several degrees of parallelism. Hence, hardware/microcode trade-offs can be easily produced.

1. Introduction

Aujourd'hui les progrès de la technologie autorisent la conception de composants spécialisés (ASIC) de plusieurs millions de transistors, soit l'équivalent d'un processeur de type RISC. En parallèle, il y a un besoin d'intégration de plus en plus poussé d'applications de traitement du signal avec des contraintes de coût et performances très sévères (GSM, MPEG par exemple). Le problème de l'implémentation d'une telle application devient de plus en plus complexe. Aussi, la définition de la "meilleure" architecture est la conséquence de nombreux choix[3]. Dans une approche descendante, on est amené à opérer des partitionnements raffinés successivement qui ont pour objectif de répartir les différents traitements sur des composants soit de type logiciel (c'est à dire des fonctions programmables sur des processeurs de type DSP par exemple) soit de type matériel (fonctions intégrées dans un composant spécifique : ASIC ou FPGA). Les méthodes de conception qui s'intéressent à automatiser ce processus de conception sont des méthodes de conception mixte ou "*codesign*" [2]. Les techniques de synthèse dites de haut niveau [8]ont pour objectif de définir une structure matérielle optimisée ("data path" et son contrôleur) de niveau RTL à partir d'une spécification comportementale. Ces techniques opèrent généralement par ordonnancement et allocation des opérations arithmétiques et logiques de la spécification pour obtenir une structure matérielle. Les méthodes de synthèse de haut niveau interviennent dans la réalisation des parties matérielles définies par les méthodes de *codesign.*

2. Synthèse de processeurs spécialisés micro-codés

L'approche que nous proposons (CAPSYS) est intermédiaire entre ces deux techniques. L'objectif est de synthétiser à partir d'une spécification de l'application cible une structure matérielle spécialisée, "programmée" suivant un microcode horizontal VLIW [9]. L'environnement de synthèse CAPSYS est illustré sur la figure 1 •. CAPSYS produit des architectures (structure matérielle et microcode de contrôle) composées d'unités fonctionnelles (UF) dont les descriptions sont rassemblées dans une bibliothèque. La partie matérielle est construite automatiquement à partir d'unités arithmétiques et logiques, d'unités de contrôle et de gestion de données (mémoires de données, unités de calcul d'adresses) mais également d'unités spécifiques définies par le concepteur, par exemple issues d'un outil de synthèse de haut niveau (HLS). La phase de synthèse de CAPSYS qui opère sur le résultat de la compilation de la spécification d'entrée, peut être considérée comme un ordonnanceur/allocateur générique capable d'utiliser des UF spécifiques [1]. Ce point per-

met par exemple de tirer profit aussi bien de l'expérience ou de l'intuition du concepteur que de l'existence d'unités spécialisées adaptées. Le concepteur a ainsi la possibilité de déclarer dans le programme définissant la spécification d'entrée des fonctions externes qui auront, après synthèse, une réalisation matérielle. Pour chaque fonction externe, la bibliothèque d'UF doit donc contenir au moins une unité capable de réaliser cette fonction. Ces UF peuvent être multi-fonctions, pipeline et multi-sorties. La spécification d'entrée peut contenir la déclaration de tableaux dynamiques, des fonctions récursives, des traitements vectoriels (points originaux de CAPSYS) et utiliser les instructions de contrôle classiques (boucles, schémas conditionnels).

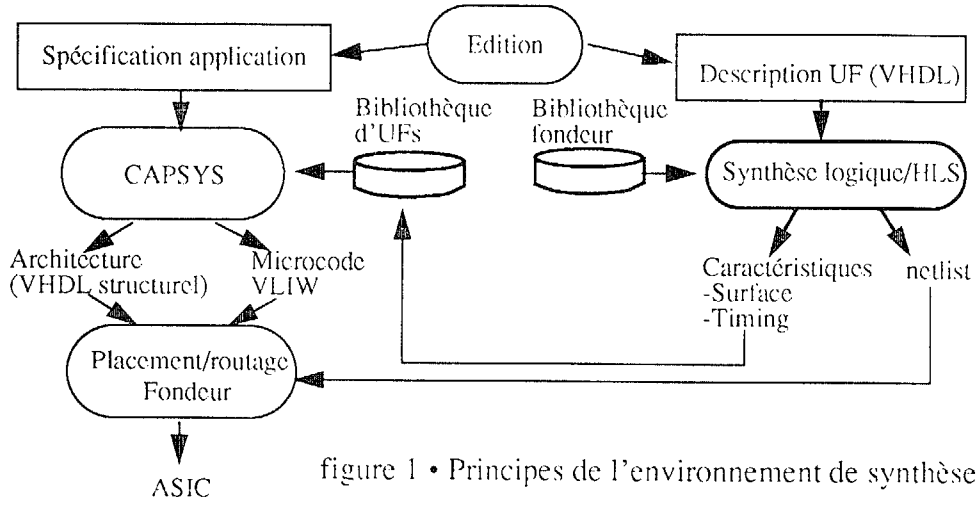

figure 1 • Principes de l'environnement de synthèse CAPSYS

3. Principes de synthèse dans CAPSYS

A l'issue d'une phase classique de compilation du programme de l'application cible on obtient un graphe de contrôle (CDFG) sur lequel opère le processus de synthèse. Le CDFG est composé de microtâches qui sont les différentes actions élémentaires que devront réaliser les unités de l'architecture à construire. Ces microtâches sont aussi bien relatives aux opérations arithmétiques et logiques qu'à la gestion de la mémoire de données. En particulier, cette mémoire de données est organisée en pile, ce qui autorise la prise en compte de la récursivité et des tableaux dynamiques dans les programmes d'entrée. La synthèse opère à partir de ce CDFG et se déroule macroscopiquement en trois étapes. Une phase d'allocation affecte à chacune des microtâches du CDFG une liste de tous les modèles d'unités fonctionnelles (UF) de la bibliothèque qui réalisent cette microtâche. Ensuite, l'étape de sélection et d'ordonnancement produit des modèles d'architecture. Un modèle d'architecture est constitué d'une liste d'unités fonctionnelles et d'un code objet de contrôle des unités, induit par l'application cible. L'allocation des UF de la bibliothèque aux microtâches du CDFG permet de produire plusieurs modèles d'architecture dans la phase d'ordonnancement de microtâches. Dans le cas de traitements scalaires, l'algorithme modifié du *List-Scheduling*[6] est utilisé, pour les traitements vectoriels nous avons adapté l'algorithme issu de [7]. La production de plusieurs modèles d'architecture est basée sur les différentes utilisations du parallélisme à grain fin contenu dans l'application. Enfin la dernière phase de synthèse construit l'interconnexion entre les UF, le séquenceur micro-codé et le système mémoire tout en respectant les contraintes d'ordonnancement définies par la phase précédente. Le paragraphe suivant précise certains de ces aspects.

4. Description des fonctionnalités

La figure 2 • présente l'environnement de base de CAPSYS. Il permet de définir le contenu de la bibliothèque des UF, les contraintes, d'effectuer la synthèse,

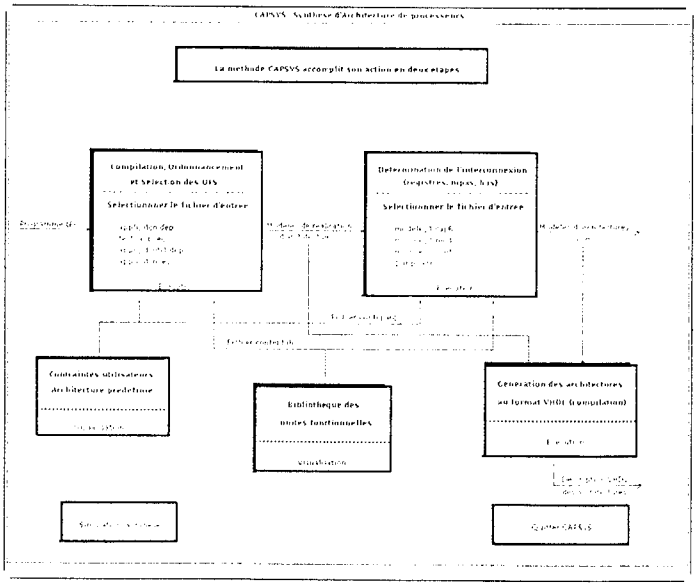

figure 2 • Environnement de base de CAPSYS

de produire une description VHDL structurelle de l'architecture et de se connecter à un outil de simulation ou de synthèse logique/comportementale VHDL. Le programme de l'application est compilé en une représentation interne: le CDFG (figure 3 •). Les nœuds de ce graphe sont les actions élémentaires (microtâches) que doivent réaliser les unités fonctionnelles (UF) qui constitueront l'architecture synthétisée. Les descriptions des UF sont regroupées dans une bibliothèque. Les UF doivent avoir un temps de réponse fixe en fonction des données mais peuvent être pipeline et multifonctions.

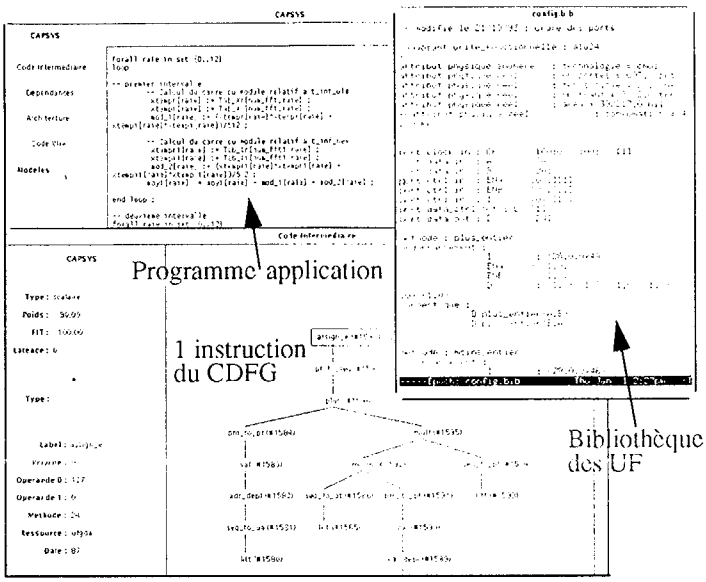

figure 3 • Structures utilisées dans CAPSYS

Pour chaque UF, l'utilisateur fournit une liste de caractéristiques qui peuvent servir à constituer des contraintes vis à vis du processus de synthèse. Ensuite, pour chaque fonction (méthode) réalisée par cette UF la bibliothèque contient d'une part, un schéma d'exécution qui indique la suite des valeurs des contrôles à appliquer à l'UF pour exécuter la fonction et d'autre part, les ports de données en entrée et en sortie concernés par l'évaluation de cette fonction. Toutes les microtâches associées aux nœuds du CDFG doivent avoir au moins une réalisation possible parmi les UF de la bibliothèque. Ces microtâches sont de deux types: il s'agit soit de microtâches primitives dans CAPSYS soit d'une microtâche générique. Les premières définissent l'architecture générique de CAPSYS et concernent les opérations de base de type arithmétiques et logiques mais aussi les modes d'adressage et les lectures/écritures mémoire. Par l'intermédiaire de la microtâche générique, l'utilisateur peut utiliser des fonctions matérielles spécifique dans le programme de l'application cible. Par exemple un opérateur de renversement de bits pour remettre en ordre les composantes obtenues à l'issue d'une FFT.

L'utilisateur spécifie des contraintes de réalisation qui permettent également de limiter l'espace de conception du processus de synthèse. On peut ainsi limiter le nombre d'instances d'unités fonctionnelles, déclarer l'existence a priori d'unités ainsi que des chemins de données avec ou sans registre de pipeline.

Le processus de synthèse de CAPSYS est récursif c'est à dire qu'un ensemble d'architectures respectant les contraintes spécifiées est construit. Chaque architecture est définie par une structure (les UF et unités d'interconnexion : registres, multiplexeurs, liens) et par un microcode de type VLIW qui est construit pendant la phase d'ordonnancement (figure 4 •).

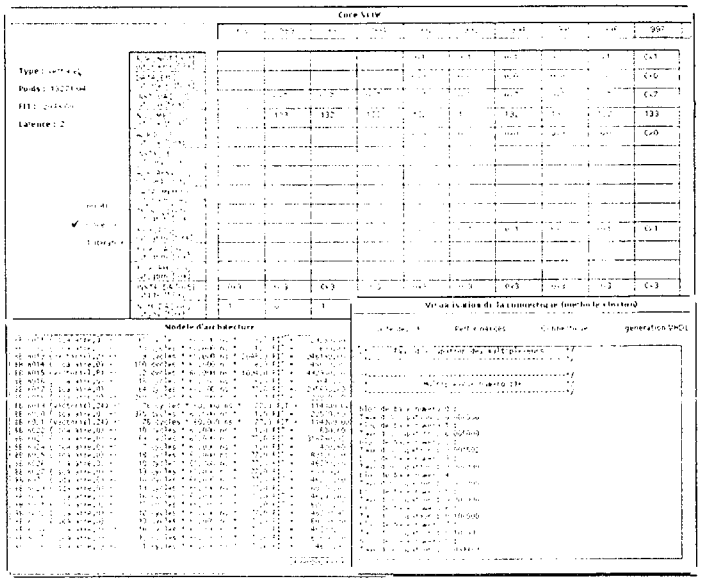

figure 4 • Résultats de synthèse.

Pour illustrer les fonctionnalités de CAPSYS, la synthèse de la partie traitement du signal d'un système sonar simplifié et celle d'un filtre médian en traitement des images sont présentées dans le paragraphe suivant.

5. Exemples de synthèse

Dans l'autodirecteur de torpille, les traitements à réaliser toutes les 5ms font intervenir pour chaque voie une FFT sur 256 points ainsi qu'un filtrage/détection

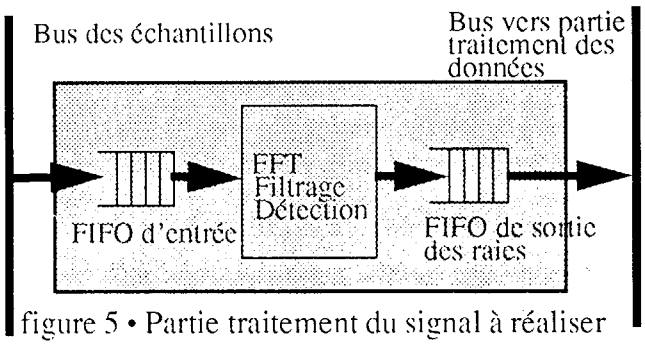

figure 5 • Partie traitement du signal à réaliser

sur chaque raie des 40 derniers spectres calculés. Les échantillons sont placés dans une FIFO d'entrée et le

	FUs	Cycle (ns)	Nb portes	Temps exécution
S1	3*ALUs, 2*pMuls (mult. pipelines), 1*Bit reverse	45	28.000	10 ms
S2	3*ALUs, 1*pMul, 1*Bit reverse	45	22.000	4.7 ms
S3	1*ALUs, 1*pMuls, 1*Bit reverse	60	19.000	4.8 ms
S4	2*ALUs, 2*pMuls, 1*Bit reverse, 2*RAMs, 1*ROM	60	86.000	2.4 ms

Table 1. Résultats de synthèse sur l'application sonar.

système fournit les résultats vers une partie de traitement de données aval au travers d'une FIFO de sortie. Le système à réaliser correspond à la partie grisée de la figure 5 •. Dans un premier temps, tous les traitements représentés par cette application ont été programmés sans aucune optimisation avec les deux files d'attente décrites sous forme de fonctions matérielles spécifiques. Cette première version de l'application conduit à une architecture microcodée (avec environ 2200 lignes de microcode) composée d'une mémoire de données de 48kmots, d'une mémoire de constantes, de 2 ALU, de 2 multiplieurs et des 2 FIFO avec une performance de 82ms pour un temps de cycle de 60ns. Ce premier résultat est très éloigné de la limite des 5ms. Une forte optimisation est nécessaire. En utilisant des multiplieurs pipelines, une unité câblée de renversement de bits et en vectorisant la FFT, on divise par un facteur 8 le temps d'exécution (architecture S1 de la Table 1.). En tirant profit des coefficients particuliers des premières passes de la FFT, on optimise l'utilisation du parallélisme pipeline ce qui permet d'obtenir un temps d'exécution inférieur à 5ms (architecture S2). Pour l'architecture S3 on remplace l'algorithme classique de FFT radix-2 par un algorithme à structure régulière [5](qui autorise une meilleure vectorisation) et on vectorise la partie filtrage/détection. On maintient ainsi le temps d'exécution en dessous de 5ms tout en diminuant le nombre de ressources nécessaires. L'architecture S4 est obtenue en considérant des mémoires spécifiques pour les coefficients et des résultats intermédiaires ce qui permet d'augmenter le débit des données dans l'architecture. Il est ainsi possible de traiter deux voies dans le temps de 5ms.

Dans le cas du filtre médian, pour chaque position de la fenêtre de filtrage, il est nécessaire d'effectuer un tri pour déterminer la valeur médiane retenue pour l'image résultat. Dans [4], un algorithme vectoriel a été proposé. Mais des schémas conditionnels dans la boucle limitent fortement les performances. Aussi, une unité fonctionnelle spécialisée qui réalise le traitement complet pour une itération de la boucle vectorielle a été conçue par un outil de synthèse logique VHDL. Avec cette unité et en profitant de la vectorisation, un gain en performance de 40 est ainsi obtenu. L'architecture du processeur obtenu avec cette unité spécifique est constituée de 18700 portes au lieu de 25600 ce qui signifie que l'introduction d'une unité spécifique évite le coût du contrôle et des UF associés aux schémas conditionnels de l'algorithme initial tout en produisant une importante accélération.

6. Conclusion

L'environnement de synthèse CAPSYS permet au concepteur d'effectuer un prototypage rapide d'une application en traitement du signal en étudiant diverses décompositions de la spécification. Ces décompositions sont le résultat de différentes répartitions des fonctions à réaliser sous forme d'unités matérielles et/ou de microcode.

7. Bibliographie

[1]Menez G., Auguin M., Boéri F., Carrière C. A partitioning algorithm for system level synthesis. *ICCAD92*, pages 482-487. Santa-Clara, California, november 8-12, 1992.

[2]Steinhausen U., Camposano R., Gunther H., Ploger P., Theibinger M. System synthesis using hardware/software co-design. *Proceedings Int. Workshop on Hardware-Software Co-Design*. Cambridge, Mass., October 7-8, 1993.

[3]Wilberg J. et al. Hierarchical multiprocessor system for video signal processing. *Proceedings SPIE*. Nov., 1992.

[4]Duclos, P. *Etude du parallélisme en traitement des images, réalisation sur une architecture mixte SIMD/SPMD*. Thèse, Univ. Nice Sophia/Antipolis, octobre 1988.

[5]Stone H. S. Parallel processing with the perfect shuffle. *IEEE Trans. on Computers*. C-20153-161, Feb., 1971.

[6]Landskov D., Davidson S., Shriver B. Local microcode compaction techniques. *Journal Computing Survey*. 12(3), 1980.

[7]Eisenbeis, C. Optimization of horizontal microcode generation for loop structures. *Proceedings International Conf. on Supercomputing*. Saint Malo, july, 1988.

[8]McFarland M.C., Parker A.C., Camposano R. The high level synthesis of digital systems. *Proceedings of IEEE*. 78(2):301-318, february, 1990.

[9]Fisher J.A., O'Donnel J.J. VLIW machines : Multiprocessor we can actually program. *Proceedings IEEE Compcon*, pages 299-305. february, 1984.