Conception des unités mémoire pour des applications de traitement du signal temps réel

Memory-Unit Design for Real-Time-Digital-Signal-Processing Applications

par Daniel CHILLET*, Olivier SENTIEYS*, Jean-Luc PHILLIPE, Eric MARTIN****

* LASTI – ENSSAT – Université de Rennes 1
6, rue de kérampont, 22300 Lannion
Tél : (33) 02 96 46 66 41
eMail : chillet@enssat.fr sentieys@enssat.fr
Web : http ://www.enssat.fr/RECHERCHE/ARCHI
** LESTER – Université de Bretagne Sud
2, rue Le Coat Saint Haoen, 56100 Lorient
Tél : (33) 02 97 88 05 41
eMail : emartin@univ-ubs.fr philippe@univ-ubs.fr

résumé et mots clés

L'augmentation de complexité des algorithmes de traitement du signal et de l'image, sous contrainte de temps d'exécution, et la prise en compte de nouvelles contraintes (surface, vitesse, consommation, etc) rend l'exploration de l'espace de conception irréalisable de manière manuelle. Le développement de plates-formes de conception est alors devenu une nécessité afin d'offrir aux concepteurs des outils d'estimation et de synthèse semi-automatique permettant de concevoir les différentes parties d'une application. Dans la dernière décennie, les méthodes d'ordonnancement et d'assignation des ressources (opérateurs, registres, ...etc) ont donné naissance à de nombreux outils de synthèse de haut niveau pour la conception des parties opérative et de contrôle. Toutefois, si ces outils permettent de concevoir de manière plus ou moins automatique ces deux unités, il n'en est pas de même pour l'unité de mémorisation qui doit, la plupart du temps, être conçue explicitement. L'évolution des applications, notamment dans le domaine du traitement du signal, s'applique non seulement à la partie opérative du système, mais aussi et surtout à la mémorisation; cette dernière pouvant rapidement devenir le point critique de la mise en œuvre.
Ce papier présente une méthodologie de conception de ces unités. Partant d'une description des besoins en terme de mémorisation, nous décomposons la synthèse des mémoires en quatre étapes. Cette synthèse vient alors compléter le flot de conception d'un outil de synthèse d'architectures (par exemple de l'outil Gaut).

Synthèse systèmes, outils CAO, conception mémoire.

abstract and key words

The increase in the complexity of signal processing and image algorithms, under the execution time constraint, and taking into consideration the new constraints (area, speed, consumption etc.), makes exploration space a manually unachievable concept. The framework development for design became a necessity at that time, so as to offer estimation tools and semi-automatic synthesis to the designer, allowing the design of different parts of the application.
In the last decade, scheduling and resources allocation algorithms (operators, registers etc.) have given rise to a number of synthesis tools of a high level for the designing of operative and control parts. However, if these tools allow the designing of these two units in a more or less automatic way, then it is not the same for the memorisation unit which should, most of the time, be explicitly designed.
The evolution of applications, notably in signal processing, applies itself, not only to the operative system part, but also and above all, to the memorisation; this last point can rapidly become the critical point in the implementation.

High Level Synthesis, CAD Tools, Memory synthesis.

1. introduction

Jusqu'à présent, les études en synthèse d'architectures se sont concentrées sur la conception de la partie opérative de l'application. Partant d'une description comportementale, les outils de synthèse d'architectures fournissent une description structurelle de l'algorithme (niveau RTL) [1,2,3]. Cette description est optimisée suivant un certain nombre de critères (surface, temps de calcul, consommation, etc). Une unité de contrôle (UC) est ensuite décrite afin de piloter l'ensemble des signaux de commande de l'unité de traitement (UT). Enfin, les informations relatives aux transferts de données sont produites et seront utilisées pour générer les unités de mémorisation (UM) et de communication (UCom). L'ensemble du flot de conception d'une application peut alors être représenté par le schéma de la figure 1.

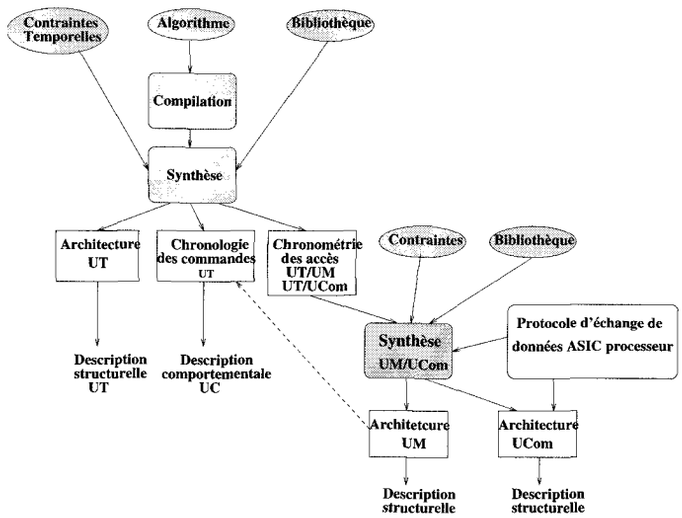

Figure 1. – Flot de conception : de l'application à l'ASIC.

Dans cette étude, nous focalisons notre attention sur la synthèse des unités mémoire. Notons que, comme le montre la figure 1, le flot de conception que nous avons développé vise à décrire des circuits spécifiques à une application (ASIC), c'est-à-dire des circuits répondant strictement aux besoins (en calcul, stockage, communication) exprimés dans la description de cette application.

Après avoir présenté les travaux réalisés dans le domaine de la synthèse haut niveau des mémoires, nous exposerons les étapes de synthèse que nous avons développées. Nous présenterons ensuite des résultats pour une application de traitement d'images temps réel ainsi que pour les différents algorithmes candidats pour une application d'annulation d'écho acoustique.

2. état de l'art

La synthèse automatique des unités de mémorisation a été assez peu abordée dans la littérature. Néanmoins, on peut distinguer quelques grands thèmes de recherche qui sont les suivants :

• Distribution et placement des données : Les méthodes présentées ici sont relatives à la distribution d'un ensemble de données dans un ensemble de bancs mémoire. La distribution est abordée pour résoudre deux problèmes. Le premier concerne la mémorisation des structures de données (généralement des tableaux) en vue d'effectuer des accès par lignes, colonnes, ou sous matrices [4,5]. Le second concerne l'augmentation de la fréquence d'accès à la mémoire [6] afin de combler l'écart entre le temps de cycle des processeurs et le temps d'accès de ces mémoires. Ces méthodes utilisent la régularité des structures de données ainsi que la régularité des traitements pour effectuer un rangement optimal dans les différents bancs mémoire. L'objectif du placement des données est d'attribuer une adresse à chacune des données d'un banc mémoire. Il est généralement réalisé durant la phase de distribution. De la qualité du placement, dépend la complexité des générateurs d'adresses associés.

• Génération des adresses : Trois solutions sont généralement proposées pour matérialiser cette fonctionnalité. Ce sont les suivantes :

– une unité de calcul des adresses, associée à un ensemble de registres de base et d'index [7];

– une unité de calcul spécifique utilisant les propriétés du TS, [8,9];

– une machine d'états générant, à chaque pas de l'algorithme, l'ensemble des adresses nécessaires [10].

La première solution impose une maîtrise du nombre de fonctions à implanter dans l'unité de calcul ainsi que la maîtrise du nombre de registres. Le placement et la distribution doivent être envisagés simultanément afin d'obtenir une solution optimale. La seconde solution consiste à proposer des générateurs sachant réaliser les manipulations de bits d'adresses que l'on retrouve fréquemment dans les algorithmes de TS. Enfin, la troisième solution s'appuie sur une machine d'états et sur de la logique combinatoire.

• Hiérarchisation de la mémoire : Il s'agit de la mise en cascade de mémoires de plus en plus rapides et de capacités moindres au fur et à mesure que l'on se « rapproche » du processeur. Dans [11], les auteurs proposent de hiérarchiser les mémoires à partir de la connaissance *a priori* des dates de production (DP) et de consommation (DC) de l'ensemble des données. Une donnée pour laquelle la valeur DC – DP est faible sera mémorisée dans une mémoire rapide. Les accès à cette donnée ne seront pas pénalisant pour l'unité de calcul mais le coût d'une telle mémoire sera important. A l'inverse, une donnée ayant une valeur DC – DP importante sera mémorisée dans une mémoire lente. Il sera alors nécessaire d'ajouter des transferts entre les différents étages de

mémoires afin de faire transiter la donnée de son lieu de stockage à l'unité de calcul.

• Transformations haut niveau : Le problème de la mémorisation peut être abordé sous deux angles différents : la synthèse *a posteriori* ou l'estimation en vue de guider la synthèse. La première approche travaille en aval de la synthèse de l'UT et est contrainte par celle-ci. La seconde approche vise à prendre en compte la mémorisation au plus tôt dans la conception en effectuant des estimations de coûts (surface, consommation, etc). Les méthodes exposées dans [12, 13] s'appuient sur des structures de données de types tableaux ou matrices et proposent des transformations au niveau du graphe flot de contrôle afin de minimiser le coût de la mémoire.

• La mémorisation dans les outils de synthèse d'architectures : Comme nous l'avons dit plus haut, il existe de nombreux outils de synthèse haut niveau. Ces outils ont plus ou moins bien intégré les méthodes présentées ci-dessus. La plupart s'appuie sur l'expérience du concepteur et vérifie simplement que l'allocation proposée est correcte du point de vue d'un certain nombre de critères (taille mémoire, nombre de mémoires). Il en est ainsi pour les outils Cathédral [14,15,16,17] et Hyper [18,19] qui utilisent alors une mémoire par tableau déclaré dans la description. Des optimisations peuvent ensuite être appliquées afin de limiter la surface, avec en contre partie une augmentation du temps de calcul de l'algorithme. L'outil Synopsys *Behavioural Compiler* [20] alloue, par défaut, des registres à toutes les variables. Il est cependant possible d'assigner des mémoires RAM ou ROM aux données lors de la déclaration des variables. L'outil Amical [21,22,23] ne permet pas la manipulation de tableaux à plus d'une dimension (vecteur). Toute déclaration de type vecteur donne lieu à la mise en place d'un bloc mémoire qui est ensuite manipulé comme les autres unités fonctionnelles.

L'approche de l'outil Alpha du Centaur est intéressante à plus d'un titre. En effet, reposant sur des architectures régulières pour lesquelles un calcul élémentaire est répliqué sur un réseau de processeurs, Alpha du Centaur travaille sur la régularité des structures de données et de traitement [24,25]. La « systolisation » de l'algorithme assure, par la suite, une simplicité de réalisation de la mémoire.

Enfin, l'outil Atomium est le seul qui soit vraiment dédié au traitement de la mémoire [26,27]. En effet, travaillant sur la description comportementale, Atomium recherche une solution mémoire optimale en minimisant une fonction de coût (par exemple la minimisation du nombre de ports de lecture et d'écriture de chaque mémoire). L'adressage des données fait aussi l'objet de toutes les attentions d'Atomium. Deux modèles de générateurs sont proposés, une évaluation de la surface permet alors de minimiser l'ensemble de la solution.

Seuls les outils Atomium, et dans une moindre mesure Alpha du Centaur, proposent une approche méthodologique de la mémorisation. Toutefois, ces approches ne sont pas directement applicables pour notre cas. En effet, rappelons que, contrairement à Atomium, nous nous situons après la synthèse de la partie opérative et que notre méthode doit donc supporter les contraintes engendrées par cette synthèse.

3. démarche de conception

Avant d'aborder la méthodologie de conception développée, expliquons et commentons le choix de la stratégie mise en œuvre dans l'outil Gaut. Ce choix s'appuie sur 3 points :

• Le premier vise à rendre la description interne la plus indépendante possible de l'application (notamment des structures de données et de contrôle de l'algorithme);

• Le second résulte de la volonté d'exploiter, si nécessaire, le parallélisme maximum de l'application;

• Enfin, le troisième point découle du choix du domaine d'application : le traitement du signal *temps réel.*

Les deux premiers points ont conduit à une modélisation de l'application sous forme de graphe flot de données (GFD), celui-ci permettant à la fois d'être indépendant des structures et exprimant, de part sa nature, l'ensemble du parallélisme. Cette modélisation implique une représentation de l'ensemble des calculs et de l'ensemble des données quelles soient scalaires ou multi-dimensionnelles. Le nombre de nœuds du graphe flot pouvant alors devenir très important et rendre sa manipulation impossible.

Le troisième point a conduit à la mise en place d'une stratégie de synthèse débutant par la partie opérative. Celle-ci supportant la contrainte de temps, les autres unités satisfaisant les besoins de cette première. Cette stratégie, optimale pour l'unité de calcul, peut s'avérer inadaptée pour les unités de mémorisation et de communication. Ceci peut alors conduire à la réalisation d'un circuit dédié très éloigné de l'optimal, ce qui est surtout vrai pour les applications à gros volume de données pour lesquelles il semble primordial de débuter la synthèse par l'unité mémoire.

3.1. modèle d'architecture

Les architectures générées par l'outil Gaut sont basées sur une unité de calcul semblable à un cœur de processeur (voir figure 2). Les opérateurs et les registres, en nombre minimum mais suffisant, forment des cellules qui sont reliées entre elles par un réseau multi-bus. La synthèse de cette unité reprend les phases classiques d'allocation, d'ordonnancement, d'assignation et d'optimisation [1]. Le point fort de l'outil Gaut réside dans la faible complexité de la phase de synthèse mise en œuvre. Celle-ci est linéaire par rapport au nombre de nœuds du graphe flot.

L'unité de traitement est piloté par un contrôleur réalisé par une machine de Moore.

Le modèle mémoire que nous avons retenu reprend les deux fonctionnalités de la mémorisation : le stockage et la production des adresses. La fonction de stockage est réalisée par une mémoire classique à laquelle on associe un générateur d'adresses.

Comme le montre la figure 2, les mémoires peuvent être interne ou externe à l'Asic en fonction de la place disponible dans celui-ci après synthèse de l'unité de traitement.

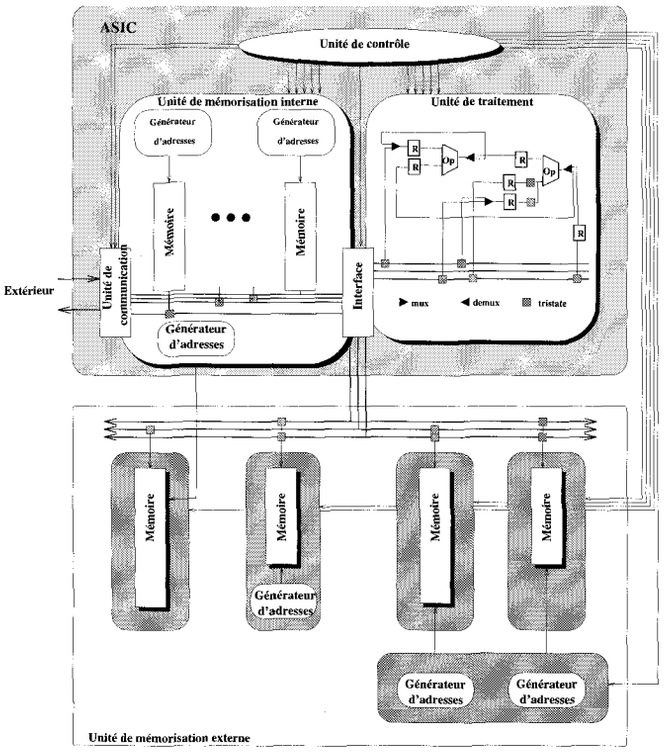

Figure 2. – Modèle des architectures synthétisées par Gaut.

3.2. stratégie de synthèse mémoire

Les phases de synthèse que nous proposons ([28]) se décomposent en quatre étapes qui sont : la sélection des composants mémoire dans une bibliothèque, la distribution des données dans les bancs mémoire, le placement des données qui est très fortement lié au choix des générateurs d'adresses et enfin l'optimisation globale de l'architecture mémoire générée.

3.2.1. la sélection des composants mémoire

La sélection consiste à rechercher, dans une bibliothèque, l'ensemble de composants mémoire le mieux adapté à la mémorisation des données [29] (voir figure 3). Cette étape s'appuie sur une évaluation du coût d'un ensemble de composants (mémoire) candidats à la résolution du problème. L'évaluation du coût prend en compte les problèmes liés au brochage du circuit. En effet, l'implantation d'un grand nombre de broches sur un Asic peut

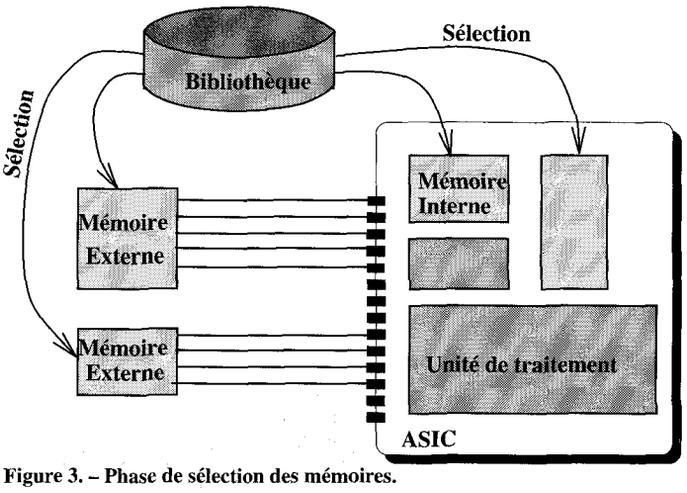

Figure 3. – Phase de sélection des mémoires.

conduire à augmenter considérablement la taille de celui-ci. Dans ce cas, le coût de l'Asic dépend du nombre de broches à implanter et non pas de la surface du corps.

Afin que la sélection soit optimale, il est important de disposer d'une large bibliothèque. Le choix des types de mémoires ainsi que leur tailles respectives est décisif vis-à-vis de la qualité de la solution finale. Les mémoires proposées dans la bibliothèque doivent être de 2 types :

– mémoires dédiées pour lesquelles le nombre de points et la position (interne ou externe à l'Asic) sont à définir;

– mémoires du commerce pour lesquelles le nombre de points mémoire est connu et la position nécessairement externe à l'Asic. Pour ces mémoires du commerce, il faut encore distinguer deux cas en fonction de la position du générateur d'adresses associé, il est soit interne, soit externe à l'Asic. Si le générateur d'adresses est interne, cela signifie que le bus d'adresses est disponible sur le brochage de l'Asic, d'où un coût important en nombre de broches. Par contre, si le générateur est externe, il suffit de quelques broches de commandes.

La faisabilité d'un Asic dépend en partie de la surface totale de celui-ci (contrainte technologique). On peut décomposer la surface d'un Asic en deux parties qui sont :

– le corps de l'Asic;

– les broches de l'Asic (nécessaires pour les échanges avec l'extérieur).

Pour une surface de corps donnée, il est possible d'implanter un certain nombre de broches sur le pourtour de l'Asic. Si l'on désire implanter plus de broches, alors la surface disponible pour le corps de l'Asic augmente. Ce supplément de surface peut, soit être utilisé pour ajouter une nouvelle fonction (opérateur, mémoire, etc), soit il s'agit d'un supplément de surface inutilisée.

3.2.2. la distribution

Cette phase consiste à attribuer un numéro de banc mémoire à chaque donnée de la séquence de transferts. Les besoins en

mémorisation sont exprimés sous forme d'un graphe de conflits qui est construit à partir de la séquence de transferts (voir la séquence de transferts sur la figure 4 et le graphe de conflits associé figure 5).

On peut assimiler la tâche de distribution au coloriage du graphe de conflits avec un nombre minimum de couleurs. La contrainte de coloriage impose que deux nœuds reliés par une arête sont de couleurs différentes. Chaque couleur représentant une mémoire sélectionnée préalablement par la phase de sélection.

Pour réaliser cette distribution de façon optimale, nous proposons de minimiser le nombre de connexions des registres de l'UT aux bancs mémoire. On profite alors des degrés de liberté de certaines données (données pour lesquelles plusieurs couleurs sont candidates) pour effectuer une distribution minimisant ce nombre.

La distribution est réalisée en considérant que toutes les mémoires disponibles sont de type RAM. Ces mémoires ont l'avantage de pouvoir remplacer tout autre type de mémoires y compris une mémoire ROM pour peu que l'on effectue une phase d'initialisation avant de débuter les calculs de l'algorithme. Ce choix nous permet d'effectuer la distribution en minimisant le coût matériel de la solution, et notamment le nombre de connexions à mettre en œuvre entre les registres de l'unité de traitement et les bancs mémoire. Une fois la distribution réalisée, il est possible, par la suite, d'envisager le regroupement des données de même type dans un même banc mémoire (par exemple le regroupement des données de type « constante » dans une même mémoire du type ROM).

[Debut ; Fin]	Adresse
[0 ; 60]	*a* *b*
[60 ; 120]	*a* *e*
[120 ; 180]	*b* *c*
[180 ; 240]	*g* *f* *c*
[240 ; 300]	*g* *b*
[300 ; 360]	*g* *d*
[360 ; 420]	*g* *h*

Figure 4. – Exemple de séquence d'accès à la mémoire fournie par Gaut.

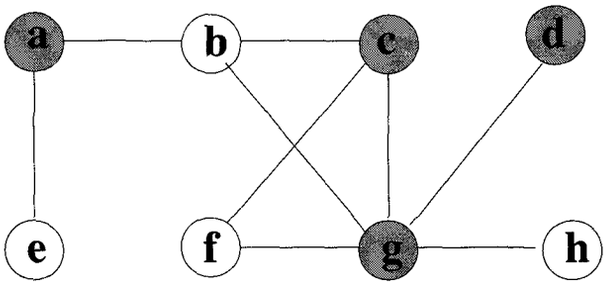

Figure 5. – Exemple de coloriage d'un graphe avec un minimum de couleurs.

3.2.3. *le placement et la génération des adresses*

Le placement des données dans les bancs consiste à affecter une adresse précise à chacune des données. Cette affectation doit se faire de façon à limiter la complexité des générateurs d'adresses associés à chaque banc mémoire.

Ces générateurs sont « construits » en fonction des besoins de la séquence à réaliser. Nous avons retenu deux modèles pour les matérialiser, ce sont les suivants :

• Machine d'états (voir figure 6).

L'avantage de cette solution est sa capacité à adresser n'importe quelle séquence de données pour peu que celle-ci soit inchangée

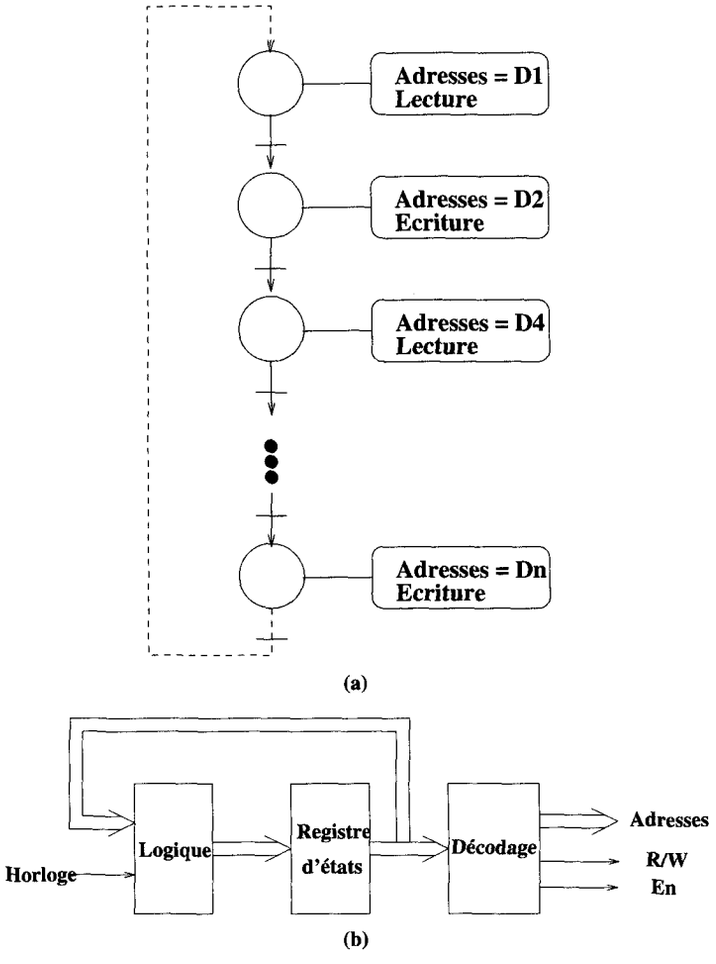

Figure 6. – Générateur sous forme de machine d'états.

d'une itération à l'autre de l'algorithme. L'inconvénient majeur de cette solution réside dans le fait que sa complexité dépend de la longueur de la séquence de transferts et que celle-ci peut être très importante. D'autre part, il est difficile d'évaluer, au niveau architectural, le coût d'un tel générateur (notamment au niveau de la logique combinatoire à mettre en œuvre). En effet, compte tenu de la puissance des outils de synthèse logique, ce type de générateur sera simplement décrit de façon comportementale et c'est seulement après placement-routage que son coût matériel sera connu. Le seul paramètre qui peut être évalué rapidement est le nombre de bascules à mettre en place pour réaliser ce générateur : il est en log_2 de la longueur de la séquence.

• Ual + registres de base et d'index (voir figure 7). L'avantage de cette solution est qu'elle permet de réaliser n'importe quelle séquence d'adresses. Cette solution est courante dans les processeurs de traitement du signal. Dans une première approche, cet avantage se transforme en inconvénient si l'on observe le coût d'une telle unité. En effet, ce type de générateur nécessite la mise en place d'une unité de calcul (qui se résume souvent à un additionneur/soustracteur ou à un simple additionneur si l'on prend la peine de « complémenter » les valeurs négatives des registres d'index) et de quelques registres. Le nombre de fonctions à implanter dans l'unité de calcul doit être minimum afin de limiter sa complexité, et le nombre de registres doit lui aussi être minimum.

En fait, il faut rappeler que nous sommes dans une situation particulière pour laquelle la séquence de transferts est connue *a priori*. Ceci implique que toutes les valeurs contenues dans les registres d'index sont déterminées après cette phase de placement. Le banc de registres d'index est alors associé à un multiplexeur à NR entrées (avec NR le nombre de registres d'index) et 1 sortie. Les valeurs des entrées de ce multiplexeur étant connues, l'ensemble peut être avantageusement remplacé par de la logique combinatoire. Dans ce cas, le coût d'un tel générateur se résume alors à un additionneur, un registre d'adresses sur ND_j bits (avec $ND_j = log_2$ du nombre de données du banc j), et à de la logique combinatoire.

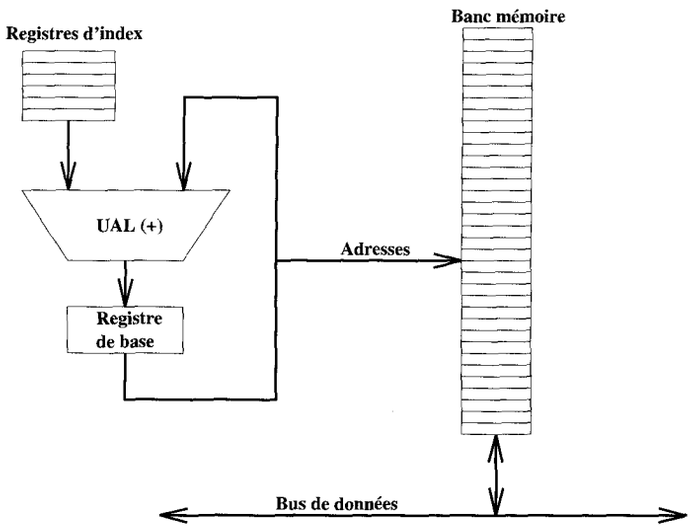

Figure 7. – Générateur sous forme d'Ual et de registres d'index.

3.2.4. les optimisations

Il s'agit ici de minimiser le coût global de l'architecture mémoire. Deux axes sont développés :

• Le placement des données dans les bancs mémoire réalise la synthèse des générateurs d'adresses en affectant un générateur par banc mémoire. Il semble intéressant d'essayer de regrouper ces générateurs lorsque cela est possible. Ces optimisations doivent envisager les regroupements des générateurs d'adresses qui ne sont pas forcément localisés au même endroit dans le système (interne ou externe à l'Asic). Lorsque les deux générateurs sont localisés au même endroit, seul le coût du générateur résultant est à prendre en considération par rapport à la somme des coûts des deux générateurs. Lorsque l'optimisation s'intéresse à la fois à des générateurs internes et externes à l'Asic, il est nécessaire d'évaluer la position du générateur résultant du regroupement. Ceci fait intervenir l'évaluation du nouveau nombre de broches.

• Dans nos différentes formulations, nous n'avons pas tenu compte des possibilités offertes par les mémoires double ports que l'on trouve aussi bien sous forme de boîtier dans le commerce que sous forme dédiée dans les bibliothèques des fondeurs. Il s'agit alors d'envisager des fusions de N mémoires simple port en une seule mémoire M ports (avec $M \leq N$). Les conditions à respecter pour assurer une fusion correcte concernent :

– le temps d'accès : il faut que les temps d'accès de la mémoire multi-ports soit plus petit ou égal au temps d'accès accordé aux mémoires simple port;

– la taille : il faut que la taille de la mémoire multi-ports soit au moins égal à la somme des tailles utiles des mémoires simple port (la taille utile correspond au nombre de points mémoire utilisé dans la mémoire).

De plus, il faut s'assurer que le coût de la mémoire multi-ports est inférieur à la somme des coûts des mémoires qu'elle remplace. Enfin, il faut s'assurer que le nombre de transferts simultanés des mémoires fusionnées est inférieur ou égal au nombre de ports de la mémoire multi-ports. Dans le pire des cas, le nombre de ports de la mémoire multi-ports doit être égal au nombre de mémoires que l'on fusionne.

4. résultats

Nous avons validé nos travaux sur un certain nombre d'applications de traitement du signal à caractère temps réel. Nous donnons dans ce paragraphe, des résultats de synthèses pour une application de traitement d'images ainsi que pour une application d'annulation d'écho acoustique.

4.1. application de traitement d'images : filtre Gaussien d'ordre 5

Il s'agit d'un filtre travaillant sur une fenêtre de taille $(2f+1)*(2*f+1)$ et calculant la valeur du pixel central en fonction de ses voisins. La valeur du pixel central est la moyenne pondérée des valeurs des pixels présents dans la fenêtre de filtrage. Le calcul du pixel central d'indice m, n est donné par la relation :

$$p_{m,n} = \frac{\sum_{i=-f}^{f} \sum_{j=-f}^{f} p_{m+i,n+j} \cdot w_{i,j}}{\sum_{k=-f}^{f} \sum_{l=-f}^{f} w_{k,l}}$$

$$\forall\, m,n = 1, \cdots, TailleImage$$

Ce filtre est appliqué à l'ensemble de l'image de taille $N * N$. Cette expression fait apparaître une division qu'il sera coûteux de mettre en œuvre dans un Asic. Nous choisissons alors de travailler sur l'expression normalisée du filtrage qui assure que la somme du dénominateur de l'expression ci-dessus est égale à 1. Le calcul de la nouvelle valeur du pixel central est donné par la relation :

$$p_{m,n} = \sum_{i=-f}^{f} \sum_{j=-f}^{f} p_{m+i,n+j} \cdot w'_{i,j}$$

$$\forall\, m,n = 1, \cdots, TailleImage$$

Les valeurs des coefficients, présentées sur la figure 8, donnent un poids plus important au pixel central et dans une moindre mesure aux pixels de la même ligne ou de la même colonne. Nous avons choisi de synthétiser un filtre de taille $5*5$ travaillant sur des images de taille $256 * 256$ codées en 8 bits et ceci à la cadence vidéo de 25 images par seconde. Pour notre étude, nous faisons l'hypothèse que les effets de bords sont éliminés par l'augmentation de la taille de la fenêtre de $2 * 2$ lignes et de $2 * 2$ colonnes. Cette hypothèse nous permet d'éviter tous les traitements conditionnels qui sont supportés par Gaut mais qui ne sont jamais résolus de façon optimale.

Notre étude suppose qu'un processeur hôte demande le calcul du filtrage d'un pixel à l'Asic dédié. Dans ce cas, les ports d'entrées de l'Asic reçoivent l'ensemble des données de la fenêtre, c'est à dire les valeurs de 25 pixels. Le filtre doit alors être appliqué 65536 fois (256^2) pour parcourir l'ensemble de l'image. Le temps de latence accordé à cet algorithme est donné par la relation ci-dessous :

$$T_{Latence} = \frac{1}{\text{Nb Images / seconde}} \cdot \frac{1}{\text{Nb Filtrage}}$$

$$T_{Latence} = \frac{1}{25} \cdot \frac{1}{256^2} = 610\, ns$$

La synthèse de l'unité de traitement s'effectue à partir de la description haut niveau (en VHDL) donnée à la figure 9.

Indice j

Poids des coefficients (10^{-2})	-2	-1	0	+1	+2
-2	1	2	4	2	1
-1	2	4	8	4	2
0	4	8	16	8	4
+1	2	4	8	4	2
+2	1	2	4	2	1

Indice i

Figure 8. – Fenêtre du filtre Gaussien et poids appliqués aux différents pixels de la fenêtre.

```
ENTITY FiltreGaussien IS
  -- Taille de l'image
  CONSTANT SF : INTEGER := 5;
  CONSTANT SFdemi : INTEGER := 3;
  -- Cadence Video (25 images/sec = 40000000 ns)
  --     40000000 ns / (256*256)
  CONSTANT latence : INTEGER := 610;
  TYPE Mat IS ARRAY (1 TO SF, 1 TO SF) OFINTEGER;
  PORT (ImageIn :IN Mat; ImageOut :OUT Integer);
END FiltreGaussien;

ARCHITECTURE comportemental OF FiltreGaussien IS
  BEGIN
    FiltreGaussien : PROCESS
      VARIABLE W : Mat;
      VARIABLE tmp : INTEGER;
      VARIABLE i,j : CONTROL;
  BEGIN
    tmp := 0;
    FOR i IN 1 TO SF LOOP
      FOR j IN 1 TO SF LOOP
        tmp := tmp + ImageIn(i, j) * W(i, j);
      END LOOP;
    END LOOP;
    ImageOut <= tmp;
    WAIT FOR latence;
  END PROCESS FiltreGaussien;
END comportemental;
```

Figure 9. – Code VHDL de l'algorithme du filtre Gaussien.

L'unité de traitement fournie par l'outil Gaut est donnée à la figure 10.

Les opérateurs retenus pour cette unité sont : deux multiplieurs (opérateurs 1 et 2) et un additionneur (opérateur 3). L'estimation de la surface (basée sur une technologie $1\,\mu m$) de l'unité de traitement fournie par l'outil Gaut est égale à :

$$S_{UT} = 1,500\ mm^2$$

A la figure 11, nous donnons les résultats de synthèse mémoire.

Nous retrouvons dans les premières lignes du tableau de la figure 11, les résultats de synthèse de l'unité de traitement. Les lignes qui suivent donnent une indication de la complexité de la mémoire. Notons que deux bancs mémoire permettent de réaliser l'ensemble de la mémorisation. Chacun de ces bancs est adressé par un générateur d'adresses reposant sur le modèle de la machine de Moore. Compte tenu de la faible complexité des deux bancs mémoire (faible nombre de points), la sélection choisit de placer ces deux bancs dans l'Asic.

Notons de plus que la surface de l'unité de traitement permet l'implantation de 23 broches alors qu'une dizaine de broches sont suffisantes pour l'ensemble du contrôle du circuit. Le nombre de broches est déterminé à partir de la distance entre chaque broche de l'Asic ($IB = 270\,\mu m$). Rappelons que la surface est donnée par la relation :

$$S \simeq \left(\frac{NBroches \cdot IB}{4}\right)^2$$

Terminons en indiquant qu'il est possible, à partir de la connaissance de la structure de la mémoire, de fournir une estimation de surface pour cette unité. Dans notre cas, cette estimation aboutit à une surface avoisinant $1,045\ mm^2$ soit plus de 40 % de la surface totale de l'Asic.

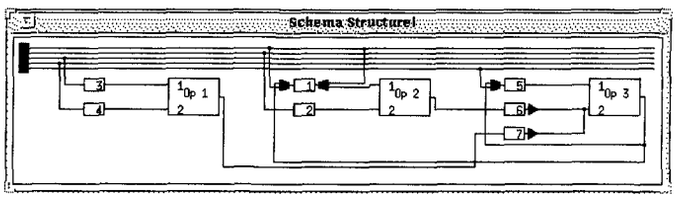

Figure 10. – Unité de traitement du filtre Gaussien.

Complexité en opérations *, +	25, 24	
Complexité en opérateurs *, +	2, 1	
Reg, Mux, Demux, Tri state	7, 2, 1, 3	
Surface UT	$1,5\ mm^2$	
Nombre de bus de données	2	
Nombre de bancs	2	
Mémoire	B1	B2
Taille des mémoires	26	26
Position des mémoires	Interne	Interne
Générateur d'adresses	M1 : 5	M2 : 5
Position des générateurs	M1 : Interne M2 : Interne	
Surface unité mémoire	$1,045\ mm^2$	
Surface Totale	$2,545\ mm^2$	
Nombre de broches implantables	23	
Nombre de broches nécessaires	< 10	

Figure 11. – Résultats de synthèses du filtre Gaussien.

Remarque : évolution des coûts en fonction de la taille de la fenêtre de filtrage

Nous donnons ici l'évolution du rapport des surfaces (mémoire, unité de traitement) pour différentes tailles de fenêtres de filtrage (voir tableau de la figure 12). Dans les trois cas, la contrainte de temps est la même ce qui conduit à une augmentation de la complexité de l'unité de traitement. Nous remarquons toutefois que l'augmentation de surface provoquée par la seule unité de mémorisation est plus importante et prend une part de plus en plus prépondérante vis à vis du coût total du circuit.

Taille des fenêtres	3.3	5.5	7.7
Opérations *, +	9, 8	25, 24	49, 48
Opérateurs *, +	1, 1	2, 1	3, 2
Reg, Mux, Demux	4, 2, 1	7, 2, 1	14, 7, 5
Nombre de bancs	1	2	3
Surface UT	$0,79\ mm^2$	$1,5\ mm^2$	$2,77\ mm^2$
Surface unité mémoire	$0,284\ mm^2$	$0,705\ mm^2$	$1,54\ mm^2$
Surface Totale	$1,07\ mm^2$	$2,20\ mm^2$	$4,31\ mm^2$
Rapport UM / Totale	27 %	32 %	36 %

Figure 12. – Évolution de la répartition de surface en fonction de la taille de la fenêtre.

4.2. application d'annulation d'écho acoustique

Dans le cadre d'un contrat portant sur l'étude du filtrage adaptatif [30], nous avons réalisé une phase de prototypage d'algorithmes pour une application d'annulation d'écho acoustique. L'objectif de cette application consiste à annuler le bouclage qui se produit entre le haut parleur et le micro dans un système de téléphonie (voir figure 13). La correction consiste à effectuer une prédiction du signal d'écho, puis à en faire la soustraction au niveau du signal capté par le micro. Cette application peut être mise en œuvre par plusieurs algorithmes de filtrage (Fir, Lms, Blms,

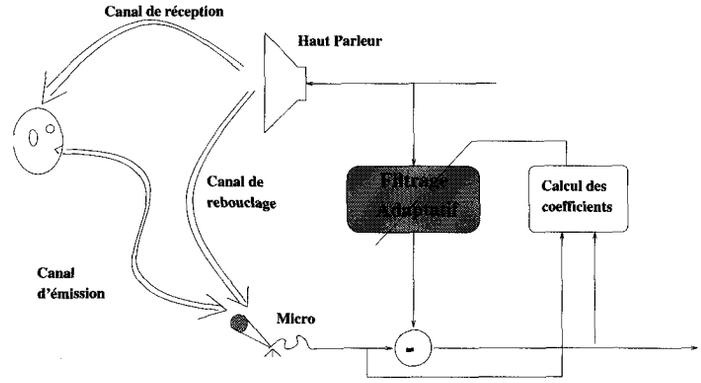

Figure 13. – Schéma de principe de l'annulation d'écho acoustique.

		Fir	Lms	Blms	Gal	Ftf
1	Complexité en opérations *	N	2N	2N	3N+5C †	8N+13
2	Complexité en opérations +	N	2N	2N	2N+4C †	8N+5
3	Complexité en opérateurs *	1	1	1	1	1
4	Complexité en opérateurs +	1	1	1	1	1
5	Complexité en registres	4	4	4	4	4
	Analyse et modification de la séquence					
7	Nb de variables déclarées	259	262	607	772	1299
8	Nb de données à mémoriser	264	385	720	648	1968
9	Nb Bus généré par Gaut	2	2	4	3	8
10	Nb Bus nécessaire	2	2	2	2	3
11	Nb Registres de lissage	0	0	2	1	8
12	Nb Bancs	2	2	4	3	4
	Distribution : taille des bancs mémoire nécessaires					
13		256	256	256	256	512
14		128	256	256	256	512
15				256	256	512
16				128		512
	Placement : type ‡ de générateurs et complexité (en nb de bascules)					
17		M : 8	C : 8	M : 8	C : 8	C : 9
18		M : 7	M : 8	M : 8	M : 8	M : 9
19				M : 8	M : 8	M : 9
20				M : 7		U : 9
21	Surface UT (mm^2)	2,12	2,53	2,54	2,64	2,41
22	Surface UM (mm^2)	3,25	4,03	7,27	6,04	13,90
23	$\frac{S_{UM}}{Surface\ Total} * 100$ en %	61	61	74	69	85
24	Temps de synthèse	$\simeq$ 2 s	$\simeq$ 7 s	$\simeq$ 12 s	$\simeq$ 10 s	$\simeq$ 28 s

† : le nombre C de cellules est compris entre 1 et N, ordre du filtre (N = C)

‡ indication du type de générateur : C : Compteur
M : Machine d'états
U : UAL et registres

Figure 14. – Complexité des filtres utilisés en annulation d'écho acoustique (128 points, 8Khz, données sur 16 bits).

Gmdf, Ftf, etc). La contrainte de temps associée à ces algorithmes est de $62,5\,\mu s$ pour une application de téléphonie radio mobile (fréquence d'échantillonnage égale à 8 Khz). De même, la taille des filtres est de 128 points.

La figure 14 fournit une première série de résultats concernant la mémorisation des données.

L'analyse de complexité, tant au niveau algorithmique qu'au niveau du nombre d'opérateurs et de registres, permet alors de comparer ces filtres. Ainsi les deux premières lignes du tableau de la figure 13 montrent que le filtre Fir est le moins complexe (mais il est le moins intéressant du fait de sa non adaptabilité au cours du temps). Notons de plus que les cinq premières lignes font apparaître une équivalence entre les filtres Lms et Blms.

Notre outil de synthèse mémoire permet de compléter les résultats d'implantation de chacun de ces filtres. En effet, alors que seul le nombre de données à mémoriser est connu après synthèse de la partie opérative, l'outil de synthèse fournit une description précise de l'organisation de la mémoire : nombre de bancs mémoire, nombre de points par banc et coût des générateurs d'adresses.

Cette première série de résultats est donnée en visant la production d'une grande série de circuits. Ceci a pour conséquence de placer l'ensemble de la mémorisation en interne à l'ASIC à l'aide de RAM.

En comparant les $7^{ème}$ et $8^{ème}$ lignes du tableau, nous remarquons que les synthèses UT des filtres Lms, Blms et Ftf ont produit des données intermédiaires de calcul, alors que les synthèses UT des filtres Fir et Gal parviennent, par une utilisation judicieuse des registres de l'UT, à diminuer le nombre de données à mémoriser.

Les $9^{ème}$ et $10^{ème}$ lignes montrent l'effet du lissage sur le nombre de bus de données à implanter. Nous observons, notamment pour le filtre Ftf, que l'application d'une fonction de lissage permet de diminuer de façon importante ce nombre. La ligne suivante indique le coût de l'interface pour parvenir à diminuer ce nombre de bus. Ce coût est donné par le nombre de registres à mettre en œuvre dans l'interface. Nous pouvons observer que ce coût

n'est pas négligeable, surtout dans le cas du filtre Ftf pour lequel le nombre total de registres passe alors de 4 à 12. Pour tous ces filtres, nous avons privilégié la minimisation du nombre de bus de données entre l'unité de traitement et l'unité mémoire au détriment d'une augmentation de la complexité de l'interface.

Le nombre de bancs mémoire nécessaires est ensuite fourni à la ligne 12 du tableau. Il est obtenu par un algorithme glouton qui peut ne pas fournir la solution optimale. Nous pouvons observer que ce nombre est différent du nombre de bus pour les filtres Blms, Gal et Ftf. En effet, pour ces trois filtres, les contraintes imposent un nombre de bancs mémoire supérieur au nombre de bus de données.

Les lignes 13 à 16 indiquent la taille des mémoires à utiliser afin de mémoriser l'ensemble des données. Compte tenu de la faible contrainte de temps (qui permet à la synthèse UT de fournir une séquence d'accès faiblement contrainte), il n'y a pas de problème particulier pour assurer les cadences de transferts. La somme de toutes les tailles est naturellement légèrement supérieure au nombre de données à mémoriser. Indiquons, pour compléter le tableau, que tous les éléments de chaque vecteur vieillissant sont distribués dans un seul banc mémoire.

Les informations concernant les complexités des générateurs d'adresses sont données aux lignes 17 à 20. Ces complexités sont données en fonction du type de générateur utilisé :

– pour une machine d'états, M : la complexité est donnée par le nombre de bascules du registre d'états;

– pour un compteur, C : la complexité est donnée par le nombre de bascules du compteur;

– pour une Ual et des registres, U : la complexité est donnée par le nombre bits de l'opérateur additionneur (donc par le nombre de bascules du registre de base à mettre en œuvre).

Nous donnons ensuite, aux lignes 21 à 23 du tableau, une évaluation de la surface des unités de traitement et de mémoire ainsi que le pourcentage de surface occupée par l'unité mémoire sur l'Asic. La surface de l'unité de traitement est fournie par Gaut, alors que notre outil de synthèse mémoire effectue une évaluation du coût de la mémorisation en utilisant une bibliothèque de composants.

Enfin, la ligne 24 du tableau donne une indication quant aux temps de calculs nécessaires pour effectuer les synthèses.

5. conclusion et perspectives

Cette étude a abordé les problèmes liés à la synthèse de l'unité de mémorisation dans la chaîne de conception d'un système. L'objectif est de rendre quasi automatique la phase de synthèse d'architectures permettant de passer du niveau fonctionnel au niveau architectural. Cette synthèse doit répondre aux besoins de l'unité de traitement, elle suit donc la synthèse de cette dernière.

Comme pour la synthèse de l'unité de traitement, la synthèse de la mémoire doit fournir une solution optimale en terme de coût. Dans cette optique, la maîtrise de la complexité des générateurs d'adresses, du nombre de bancs mémoire et du nombre de points mémoire est importante. Nous avons proposé les étapes à suivre en vue de l'obtention d'une solution correcte et optimisée. Partant d'une description comportementale des besoins en mémoire d'une application, notre méthodologie permet d'aboutir à la structure de l'unité mémoire et donc à son estimation en terme de surface.

L'approche présentée dans ce papier garantie la satisfaction de la contrainte de temps d'exécution. Associée à un outil de conception de la partie opérative, notre méthodologie permet de compléter l'évaluation du coût de l'implantation des systèmes. Alors que jusqu'à présent seul le coût et l'organisation de l'unité de traitement étaient considérés, notre outil permet, maintenant, de préciser l'organisation globale de la mémoire : nombre de bancs, nombre de points, distribution, placement, génération des adresses.

L'un des aspects qui n'a pas été abordé dans ce papier concerne la consommation. Nous étudions actuellement ce problème de façon plus générale en considérant l'ensemble de l'architecture. Nos prochains travaux s'intéresseront plus particulièrement à la consommation de l'unité mémoire. Rappelons qu'une partie importante de la surface du circuit est, en général, utilisée pour effectuer la mémorisation des données. De plus, la surface de l'unité mémoire a tendance à prendre de plus en plus de poids vis-à-vis de la surface totale, surtout pour les applications récentes qui manipulent de gros volumes de données : GSM, RNIS, de télécommunications par satellites, de stockage sur Cd et Dab, de télévision haute définition, etc.

BIBLIOGRAPHIE

[1] E. Martin, O. Sentieys, and J.-L. Philippe. Synthèse architecturale de cœur de processeurs de traitement du signal. *Techniques et Sciences Informatiques*, 1(2), 1994.

[2] J. Van. Meerbergen, J. Huisken, P. Lippens, O. McArdle, R. Segers, G. Goossens, J. Vanhoof, D. Lanneer, F. Catthoor, and H. De Man. An integrated automatic design system for complexe DSP algorithms. *Journal of VLSI Signal Processing*, pages 265–278, 1990.

[3] A.E. Casavant, M.A. D'Abreu, M. Dragomirecky, M. Dragomireky, D.A. Duff, J.R. Jasica, M.J. Hartman, K.S. Hwnag, and W.D. Smith. A synthesis environment for designing DSP systems. *IEEE Transactions on Design and Test*, April 1989.

[4] D.T. Harper III and D.A. Linebarger. A dynamic storage scheme for conflict free access. In *16th Annual International Symposium on Computer Architecture*, volume 17, pages 72–77, June 1989.

[5] J. Lenfant. A versatile mechanism to move data in array processor. *IEEE Transaction on Computer*, C34(6) :506–522, June 1985.

[6] D. T. Harper III. Increased memory perfomance during vector accesses through the use of linear address transformations. *IEEE Transaction on Computer*, 41(2) :227–230, February 1992.

[7] I. Bazon. Programmation logique avec contraintes appliquée la synthèse de l'unité de mémorisation de GAUT. Rapport de stage ingénieur, LASTI-ENSSAT, September 1993.

[8] F. Grant, P.B. Denyer, and I. Finlay. Synthesis of address generators. In *IEEE International Conference on Computer Aided Design*, pages 116–119, November 1989.

[9] R. Reynaud, E. Martin, and F. Devos. Design of a bit sliced address generator for FFT algorithms. In *Signal processing IV : Theories and Applications*, 1986.

[10] D. Chillet. Contribution la synthèse des unités de mémorisation pour GAUT. Stage de dea stir, LASTI-ENSSAT, September 1994.

[11] S. Bakshi and D.D. Gajski. A memory selection algorithm for high-performance pipelines. In *EURO-DAC Brighton*, September 1995.

[12] F. Balasa, F. Catthoor, and H. De Man. Exact evaluation of memory size for multi-dimensional signal processing systems. In *IEEE International Conference on Computer Aided Design*, 1993.

[13] I.M. Verbauwhede, C.J. Scheers, and J.M. Rabaey. Memory estimation for high level synthesis. In *ACM/IEEE Design Automation Conference*, number 31, 1994.

[14] H. De Man, J. Rabaey, P. Six, and L. Claesen. CATHEDRAL II : a silicon compiler for digital signal processing. *IEEE Transactions on Design and Test*, 3 :13–25, December 1986.

[15] H. De Man, J. Rabaey, J. Huisken, and J. Van Meerbergen. Silicon compilation of DSP systems with CATHEDRAL II. *Esprit 87 Achievements and impact*, pages 207–217, September 1987. North-Holland.

[16] D. Lanneer, S. Note, F. Depuydt, M. Pauwels, F. Catthoor, G. Goossens, and H. De Man. *Architectural Synthesis for Medium and High Throughput Signal Processing with the New CATHEDRAL Environment*. Kluwer Academic Publishers, 1991.

[17] The cathedral-2/3 silicon compiler for real time signal processing. Technical report, IMEC, Kapeldreef 75, B-3001 Leuven, BELGIUM, April 1995. ESPRIT 97 project.

[18] M. Potkonjak. *Algorithms for High Level Synthesis : Ressource Utilization Based Approach*. PhD thesis, University of California, Berkeley, jan. 1992.

[19] M. Potkonjak and J.M. Rabaey. *Exploring the Algorithmic Design Space Using High Level Synthesis*, pages 131–167. International Series in Engineering and Computer Science : VLSI, Computer and Digital Signal Processing 257. Kluwer, 1994.

[20] R. Rudell. Design of a logic synthesis system. In *Proceeding of 33 rd Design Automation Conference*, Las Vegas, USA, June 1993.

[21] I. Park. *Amical : un assistant pour la synthèse et l'exploration architecturale des circuits de commande*. PhD thesis, Université de Grenoble, July 1992.

[22] A.A. Jerraya, I. Park, and K. O'Brien. Amical : An interactive high level synthesis environment. In *European Design Automation Conference*, pages 58–62, 1993.

[23] A.A. Jerraya, M. Aichouchi, H. Ding, P. Kission, M. Rahmouni, and P. Vijaya Rghavan. Amical : interactive architectural synthesis based on VHDL. Technical report, INPG TIMA, System level synthesis group, 46, avenue Felix Viallet 38031 Grenoble Cedex, France, April 1994.

[24] C. Dezan. *Génération automatique de circuits avec Alpha du Centaur*. PhD thesis, Université de Rennes, February 1993.

[25] D.K. Wilde and O. Sie. Regular array synthesis using alpha. Publication Interne 829, IRISA, Rennes, May 1994.

[26] W. Geurts, F. Franssen, M. Van Swaaij, F. Catthoor, H. De Man, and M. Moonen. *Memory and data path mapping for image and video applications*, chapter Application-driven architecture synthesis, pages 143–166. Kluwer Academic Publisher, F.Catthoor and L.Svensson, 1993.

[27] F. Catthoor. Memory size and power optimisation for image/video and atm processing systems. In IMEC, editor, *Eurochip Course*, September 1995.

[28] D. Chillet. *Méthodologie de conception architecturale des mémoires pour circuits dédiés au traitement du signal temps réel*. PhD thesis, Enssat, Université de Rennes 1, January 1997.

[29] O. Sentieys, D. Chillet, J.-P. Diguet, and J.-L. Philippe. Memory module selection for high level synthesis. In *IEEE Workshop on VLSI Signal Processing*, 1996.

[30] E. Martin, A. Baganne, O. Sentieys, and J.-L. Philippe. Complexité d'implantation des algorithmes adaptatifs. Rapport final de contrat : Etude du filtrage adaptatif et du codage de source 93 6 B 005, LASTI-LESTER, 1995.

Manuscrit reçu le 23 juillet 1997.

LES AUTEURS

Daniel CHILLET

Ingénieur diplômé de l'ENSSAT en 1992, il a passé sa thèse de doctorat en janvier 1997 sur la définition d'une méthodologie de synthèse des unités de mémorisation pour des applications de traitement du signal temps réel. Il est actuellement Maître de Conférences à l'ENSSAT. Ses activités de recherche concernent la synthèse haut niveau de circuits dédiés pour des applications de traitement du signal. Il est notamment impliqué dans le développement de l'outil de synthèse architecturale GAUT.

Olivier SENTIEYS

Ingénieur ENSSAT en 1990, doctorat de l'Université de Rennes 1 en 1993, il est, depuis 1993, Maître de Conférences à l'ENSSAT. Ses travaux de recherche au LASTI portent sur la synthèse de haut niveau des circuits VLSI dans le cadre des applications de traitement du signal et de télécommunications. Depuis 1996, il est co-responsable scientifique des activités du groupe Signal Architecture du LASTI.

Jean-Luc PHILIPPE

Il a passé sa thèse en 1984, à l'université de Rennes 1. Professeur des Universités à l'UBS (Université de Bretagne Sud) depuis 1996, ses domaines d'intérêts portent sur l'interaction algorithme-architecture dans les domaines des applications orientées calculs (traitement du signal, télécom), et dans celles orientées contrôle (automatisme).

Eric MARTIN

Né en 1961, est professeur agrégé de l'ENS Cachan, Professeur des Universités et directeur du LESTER à l'Université de Bretagne Sud. Son domaine d'intérêt concerne l'adéquation algorithme architecture en traitement du signal et des images, et les méthodes de conception de haut niveau des circuits dédiés. Il pilote depuis 1988 le développement du projet GAUT.