Conception optimisée d'architectures en précision finie pour les applications de traitement du signal

Optimised design of architectures in finite precision for signal processing dedicated systems

par E. MARTIN, C. NOUËT & J.M. TOURREILLES

Laboratoire d'Electronique des Systèmes Temps Réel.

Centre de recherches, rue de St Maudé, 56325 Lorient cedex

résumé et mots clés

Les nouvelles technologies sub-microniques offrent de grandes capacités en terme d'intégration de systèmes qui peuvent satisfaire les applications de traitement du signal, des images et de communications numériques (TDSI) les plus exigeantes. Pour maîtriser ces nouvelles technologies, de nouvelles méthodes de conceptions et outils de CAO voient le jour : conception système et conception comportementale. Si ces méthodes offrent une passerelle efficace entre les concepteurs en algorithmes et les concepteurs en circuits, elles posent cependant de nouveaux problèmes méthodologiques pour l'automatisation de la conception. Notre travail entre dans cette démarche et se focalise en particulier sur la transformation, sous contraintes, des types abstraits utilisés pour les déclarations de variables au niveau de spécification comportementale vers les types « vecteur de bit » que savent intégrer les outils de conception logique. Nous nous plaçons dans le cadre de l'outil de synthèse comportementale GAUT, développé au LESTER pour lequel nous présentons les différents modèles, analyses et méthodes d'optimisation définies et mises en œuvre pour la maîtrise des bruits de calculs dans les architectures temps réel en précision finie. Les implémentations d'applications de TDSI que nous avons réalisées, montrent l'efficacité et l'importance de cette démarche en terme d'optimisation des architectures.

Synthèse comportementale, architecture temps réel, traitement en précision finie, bruits de calcul.

abstract and key words

The new submicronic technologies offer real capacities in terms of integration of signal processing dedicated systems, images and digital communications. To control these new technologies, new design methods and new computer-aided design tools have appeared : the system design and the behavioral design. These methods offer an effective link between algorithm designers and circuit designers. But it creates also new methodological problems for design automation.

Our study is in keeping with this process and is more particularly focused on transformation under constraints, from the abstract types (used in the declaration of variables for the behavioral specification) to the vector of bit types (used in the logical design).

We illustrate our methodology by the use of the behavioral synthesis tool Gaut, developed in the Lester laboratory. We present the different models, analysis and methods used in a way to control computing noises in finite precision and real time architectures.

Implementation of signal processing and image applications gives the efficiency and the importance of this approach in terms of architecture optimization.

Behavioral synthesis, real time architecture, finite precision treatment, computing noises.

1. contexte

La mise en œuvre d'un algorithme de traitement du signal, sur un processeur DSP ou sur une architecture dédiée, requiert tout au long du flot de la conception de respecter les contraintes de l'application ciblée. Ces contraintes sont généralement le temps réel, latence ou débit des traitements, mais encore le coût de l'architecture dédiée ou la consommation du système mis en œuvre.

Aujourd'hui la maîtrise des nouvelles technologies sub-microniques, qui permet l'intégration de plusieurs dizaines de millions de transistors sur un même circuit, a induit l'apparition d'une nouvelle étape de conception : la synthèse comportementale [Martin 1][Philippe]. Cette étape de conception peut être assimilée à la compilation de code sur les dernières générations de DSP (comme le C6x de Texas Instrument, etc.). Elle consiste, à partir de la spécification comportementale d'un algorithme, à générer une représentation interne (arbre de syntaxe ou graphe) sur laquelle des optimisations et des transformations de code sont opérées pour aboutir soit à un code assembleur (pour DSP), soit à une spécification de niveau RTL (niveau transfert de registres ou architecture logique) pour un circuit dédié. Les optimisations de code relèvent en général des techniques de compilation (analyses lexicale et syntaxique, élimination de code mort, propagation de constantes, etc.), tandis que les transformations reposent sur des méthodes comme l'ordonnancement et l'affectation (ou assignation) de composants, en vue de satisfaire les contraintes de l'application et, en particulier, le temps réel. Quelle que soit la cible, ASIC ou DSP, il reste une transformation sémantique difficile qui consiste à passer du type abstrait des variables manipulées par l'algorithme (variables réelles, complexes, etc.) à un type logique comme le vecteur de bits, seul type admis pour la synthèse RTL. Dans un compilateur, une bibliothèque mathématique supporte la transformation d'opérations sur des types abstraits (par exemple un réel) vers un composant en précision finie (par exemple un multiplieur sur 16 bits en virgule fixe). Les outils de synthèse comportementale, comme Monet [Monet] ou Behavioral Compiler [BC], n'acceptent qu'un nombre très restreint de types abstrait de variables (par exemple un entier relatif, un entier positif). Il en est de même pour l'ensemble des outils de synthèse d'architecture développés dans les laboratoires de recherche publics et que nous citons dans le tableau suivant.

Des environnements de spécification de haut niveau (niveau système) comme CoCentric de synopsys [Synopsys] ou Certio-SPW de Cadence [Cadence] se situent en amont de la synthèse d'architecture. Ces environnements utilisent les propriétés des langages objets (C++, System C) pour introduire les techniques de polymorphisme. Ces techniques autorisent ainsi un changement rapide du type des données (de flottant à un format fixe par exemple) ce qui conduit par simulation à la détermination du format de données approprié.

La transformation d'un type abstrait (format flottant par exemple) vers un type logique (format fixe) nécessite d'une part qu'il existe des opérateurs supportant cette seconde catégorie de type et d'autre part la maîtrise, au niveau de l'application, d'une telle transformation. Celle-ci se traduit généralement par une perte de précision des traitements ou un risque de dépassement de la dynamique de codage, etc.

Nous avons développé une approche formelle qui permet la transformation automatisée de la spécification d'une application manipulant des opérations sur des types abstraits de variables vers une représentation intermédiaire où les opérations s'effectuent sur des types logiques de variables. Notre démarche repose sur un ensemble de modèles, d'analyses et de transformations prouvées. Elle a été mise en œuvre dans un environnement de synthèse d'architecture, GAUT, développé au LESTER [Martin 2][Martin 3]. Les résultats en synthèse d'architectures temps réel montrent le gain important, en terme de complexité, voire de consommation, que l'on peut atteindre par une telle démarche.

Nous allons tout d'abord justifier notre choix de précision finie et mettre en évidence la notion de bruit de calcul. Nous présentons également la stratégie de conception reposant sur des techniques d'exploration et de transformation de graphes de données. Les techniques d'optimisation font appel à l'application de fonctions Dynamique, Recadrage et Bruit de calcul que nous détaillons, dans le but d'évaluer le format de données adapté.

Nous présentons dans un dernier paragraphe les résultats obtenus sur la FFT et une transformée en ondelettes.

France		***Europe***		***USA***	
ALMA[3]	(MASI, ParisVI)	Atomium[1]	(IMEC, Louvain, *B*)	BdA[2]	(ESM, Irvine, *US*)
AMICAL[1]	(TIMA, Grenoble)	BSS[3]	(IDA, Braunschweig, *D*)	BUD[2]	(AT&T Bell, *US*)
	[JER92], [PAR92]	CADDY[1]	(FZI, Karlsruhe, *D*)	DSP Designer[1]	(HP, *US*)
CODESIS[1]	(IRISA, Rennes)	Cathedral[2]	(IMEC, Louvain, *B*)	DSS[1]	(ECECS, Cincinnati, *US*)
GAUT[1]	(LESTER, Lorient)	FIDIAS[2]	(DIA, Madrid, *ES*)	ISE[2]	(ESM, Irvine, *US*)
	& (LASTI, Lannion)	Mimola[2]	(AG, Dortmund, *D*)	Matisse[1]	(Motorola, *US*)
MACH[3]	(LIRMM, Montpellier)	MOODS[1]	(ESD, Southampton, *GB*)	Hercules&Hede[3]	(DSCAD, Stanford, *US*)
	[DUP94], [ROU94]	NEAT[1]	(ES, Eindhoven, *NL*)	HIS[3]	(IBM, *US*)
OSYS[2]	(LaMI, Evry)	PIPE[3]	(DCEIT, Budapest, *HU*)	Hyper[2]	(EECS, Berkeley, *US*)
		Synthesia[2]	(EsdLab, Stockholm, *SE*)	VSS[2]	(ESM, Irvine, *US*)
		TSE-TSE[3]	(IDA, Linköping, *SE*)		

[1] maintenu, [2] non maintenu [3] sans information

2. implantation en précision finie

Tout d'abord pourquoi intégrer une architecture en précision finie ? La mise sur le marché des DSP en virgule flottante semble être une facilité pour éviter la transformation d'opérations sur des types abstrait de type réel vers une mise en œuvre sur opérateur en précision finie. Ainsi la famille C6x, dernière génération de DSP de Texas Instrument comporte deux processeurs d'architectures équivalentes, le C6201B pour des traitements sur 16 bits en virgule fixe et le C6701 pour des traitements sur 32 bits en virgule flottante. Ce dernier n'est « que » 20 % plus lent pour une consommation guère supérieure de 5 % [TMSC6X].

Si l'on s'intéresse plus particulièrement aux opérateurs, sans tenir compte des autres composants qui forment un DSP complet, on peut noter qu'un additionneur 16 bits, par rapport à un additionneur 32 bits flottants est 6 fois plus rapide et consomme environ 10 fois moins (dans une technologie CMOS 0,8 μm) [Gailhard]. L'implantation en précision finie, particulièrement pour les architectures dédiées peut amener un gain important de performances qui peut être critique dans des applications contraintes (DVB *diffusion de la télévision numérique*, MPEG *compression de séquences vidéo*, etc.).

L'implantation en précision finie d'un algorithme de traitement du signal induit un bruit de calcul qui est du aux arrondis ou troncatures opérés ; cette implantation nécessite de plus la maîtrise de la dynamique de codage. Fréquemment utilisée, la représentation en virgule fixe cadrée à gauche permet d'éviter les problèmes de dynamique sur les multiplications (le produit de deux nombres de module inférieur à l'unité est lui-même inférieur à l'unité, donc appartient au domaine de codage), mais pas sur les additions ou soustractions.

Analyser les bruits de calcul se fait classiquement suivant deux voies : la simulation ou l'approche analytique.

La simulation effectue une comparaison de performance, selon un critère d'erreur, entre l'algorithme exécuté en précision infinie et le même algorithme exécuté en précision finie. On peut être amené à décrire le comportement de chaque opération arithmétique dans le format de codage en précision finie que l'on a retenu ou appeler l'une des fonctions mathématiques des bibliothèques Unix. Cependant, la simulation ne rend qu'une observation partielle du comportement de l'algorithme : elle dépend du format de codage que l'on a retenu et du jeu de test (signaux ou images) que l'on a utilisé pour analyser le comportement de l'algorithme. La simulation peut devenir extrêmement fastidieuse si l'on recherche le format de codage en précision finie le plus approprié pour l'application ciblée (par exemple le nombre minimum de bits pour un critère de qualité donnée pour le comportement de l'algorithme).

L'approche analytique suppose au préalable une modélisation du bruit de calcul engendré par les opérations de troncature ou d'arrondi en sortie des opérateurs. Ce modèle peut être générique par rapport au nombre de bits retenu pour le format de codage en précision finie. Le modèle basé sur les signaux discrets [Bellanger] donne une estimation assez précise du bruit engendré par les troncatures ou les arrondis, sans nécessité d'informations spécifiques sur les signaux traités ; en virgule fixe cadrée à gauche où q est le pas de quantification ($q = 2^{-b}$, b nombre de bits), la puissance du bruit d'arrondi peut être modélisée par la valeur $q^2/12$. Un modèle plus élaboré, comme celui proposé par Barnes [Barnes], affine la modélisation des bruits engendrés par l'arrondi, mais nécessite une meilleure connaissance des signaux traités ; la puissance du bruit d'arrondi est alors égale à $(q^2/12).(1 - 1/M^2)$, où M est la dynamique du signal. L'estimation analytique du bruit de calcul engendré sur un algorithme suppose par exemple que les puissances de bruits soient additives, ce qui permet de modéliser chaque opération par une fonction de propagation du bruit de calcul et une fonction de génération du bruit (s'il y a arrondi).

La modélisation que nous avons retenue, dans la suite de notre étude, associe à chaque opération deux fonctions de génération et de propagation du bruit de calcul. Considérons une opération de type *Op*, cette opération génère et propage du bruit de calcul. Les fonctions primitives sont donc les modèles en terme de bruit de calcul des opérations de l'algorithme [figure 1].

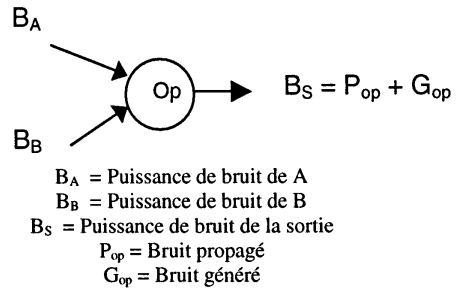


Figure 1. – Modèle des primitives de bruit de calcul

Chaque opération est modélisée par une fonction primitive opérant sur le bruit de calcul. Les fonctions de propagation et de génération sont données dans le tableau suivant pour trois exemples simples [tableau 1] :

Tableau 1. – Modèles de propagation et de génération du bruit de calcul

Opération	Propagation	Génération
addition	$B_A + B_B$	Aucun
Multiplication et multiplication par une constante α	$B_A + B_B$ ou $\alpha 2B_A$	$q^2/12$
division par une puissance de 2	$\frac{B_e}{2^d}$	$\frac{q^2}{2^{3d}} \sum_{i=1}^{2^d-1} i^2$

B_A, B_A et B_e sont les puissances de bruit de calcul aux l'entrées des opérateurs. La division par une puissance de 2 est mis en œuvre par un registre à décalage où *d* représente le nombre de décalages. Le pas de quantification est *q*, qui pour une représentation en virgule fixe cadrée à gauche sur *n* bits (hors bit de signe) vaut alors 2^{-n}. Dans notre méthode, la description des modèles nécessaires aux analyses est donnée dans la librairie de description des opérateurs, librairie qui est elle-même utilisée par l'outil de synthèse architecturale GAUT.

3. flot de conception : définition d'une méthodologie

Nous présentons en premier le flot général de conception d'une architecture dédiée à partir d'une spécification comportementale. De cette présentation émergent des problèmes d'optimisation que notre stratégie de conception tente de lever. Cette stratégie repose sur trois étapes principales pour l'optimisation de la conception d'architectures dédiées en précision finie : l'analyse de la dynamique des variables, le recadrage et l'estimation du bruit de calcul. Ces techniques sont formellement définies par des fonctions d'analyse ou de transformations locales, elles-mêmes reposant sur des modèles. Chaque étape se définit par une application valide de ces fonctions à l'ensemble de la spécification de l'application.

3.1. stratégie de conception

Dans la démarche de synthèse d'architecture, le flot « classique » de la conception optimisée comporte deux phases essentielles [Martin 2] : la première compile la spécification comportementale d'un algorithme et génère une représentation interne, sous la forme d'un graphe flot de données (DFG), qui exprime les dépendances de données entre les traitements à mettre en œuvre. Cette première étape optimise le code à l'aide de techniques connues en compilation (élimination du code mort, propagation de constantes) ou plus avancées comme c'est le cas des DSP (déroulage de boucles, mise en ligne de fonctions, etc.). Cette première étape parallélise la représentation de la spécification, ce qui permettra par la suite d'exploiter avec méthode le parallélisme en vue de satisfaire les contraintes de temps réel de l'application. La seconde étape de la conception met en œuvre différentes techniques d'optimisation comme la sélection des ressources (définir le nombre et le type de composants à mettre en œuvre dans l'architecture), l'ordonnancement des traitements (dater les exécutions des opérations), assigner les opérations sur les composants et optimiser le chemin de données (partage des registres et de bus, affectation en mémoire). Durant la sélection, le choix du format de traitement de chaque opérateur est opéré et durant l'assignation se traite l'affectation des variables aux composants (opérateurs, registres et mémoire), ce qui conduit alors à connaître précisément le type logique de chaque variable. Ces deux étapes de la conception s'achèvent par la description structurelle de l'architecture de traitement et par la spécification de la machine d'état qui gère les traitements. Ces descriptions de niveau RTL peuvent ensuite être synthétisées par un outil de synthèse logique vers un ASIC ou un FPGA. C'est au niveau de la description RTL de l'architecture que l'on peut analyser finement les bruits de calcul générés par l'exécution de l'algorithme considéré, sur l'architecture en précision finie. C'est à ce stade que l'on peut évaluer si les critères de qualité de l'algorithme sont acceptables ou non pour l'application et donc rétroagir sur la sélection du format de donnée des opérateurs à mettre en œuvre [figure 2].

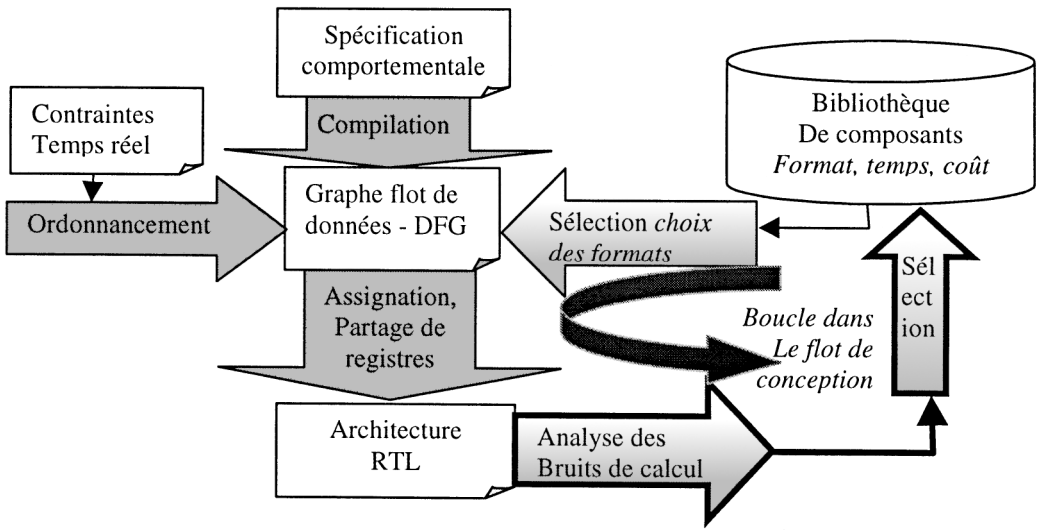


Figure 2. – Présence d'une boucle dans le flot de conception.

La présence d'une boucle dans le flot de conception entraîne une complexité telle qu'il peut devenir impossible d'automatiser le flot pour obtenir des architectures de qualité en un temps CPU raisonnable. En faisant quelques hypothèses simplificatrices, nous pouvons casser cette boucle : en effet si la largeur du chemin de données au niveau de l'architecture est uniforme, il est possible d'analyser le bruit de calcul engendré par des opérations en précision finie, indépendamment de l'étape d'assignation et donc des opérateurs implémentant ces traitements. Typiquement cette hypothèse ne peut être tenue lorsque l'on désire accumuler les résultats de produits sur une UAL à format étendu (c'est le cas de tous les DSP à virgule fixe), puisque dans ce cas il faut connaître l'assignation des variables en mémoire ou en registre pour connaître avec précision si leur format est conservé en précision étendue ou au contraire est ramené au format de base (par exemple 16 bits ou 32 bits en précision étendue).

3.2. techniques d'optimisation

Nous supposons avoir obtenu un graphe flot de données (DFG) comme représentation de l'algorithme de traitement du signal à mettre en œuvre. Ce graphe flot de données exprime les dépendances vraies entre les opérations.

$G(N, A)$ est un graphe flot de donnée où :

N est un nœud, représentant un variable ou une opération,

A est un arc orienté, représentant la dépendance entre deux nœuds.

N est annoté par l'opération réalisée (le type d'opération et son repérage dans l'algorithme) et *A* est annoté par le délai en nombre d'itérations de l'algorithme, pour la propagation de la variable (cas des algorithmes récursifs).

Il existe deux nœuds particuliers, le nœud source qui n'a aucun antécédent et le nœud puits qui n'a aucun successeur. Un DFG bien formé n'a aucune boucle (ensemble fermé de nœuds se succédant) qui ne comporte d'arc sans délais.

Exemple : *FIR d'ordre 2*

$$y = a(0) \cdot x(0) + a(1) \cdot x(1) \ ;$$
$$x(1) <= x(0) \ ;$$

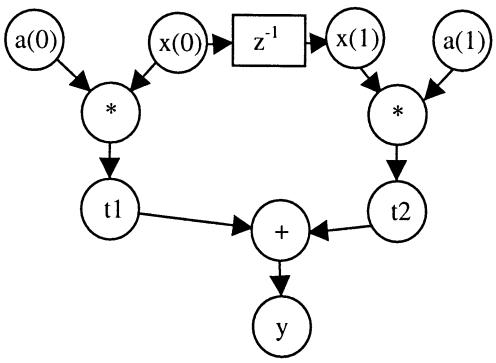


Figure 3. – Exemple de graphe flot de données.

où les $a(I)$ sont les coefficients d'un filtre, $x(i)$ les échantillons du signal, considérés sur une fenêtre glissante et y le résultat du filtrage. Le graphe obtenu est le suivant [figure 3] :

Les techniques d'optimisation consistent à analyser, par annotation, puis à transformer, par des transformations locales opérées sur un parcours régulier, le graphe flot de données. Les résultats de l'application de ces techniques permettent d'estimer globalement le bruit de calcul généré par les traitements en précision finie et d'en déduire le format de données approprié (type logique pour la représentation des variables) afin de respecter une contrainte applicative de rapport signal à bruit RSB fournie par le concepteur.

Les trois techniques que nous avons développées traitent dans un premier temps l'analyse de la dynamique des variables intermédiaires de traitement (fonction dynamique), puis le recadrage des variables (fonctions recadrage), ce qui conduit enfin à l'analyse des bruits de calcul (fonction « bruit de calcul »). Nous distinguons le bruit de calcul généré par les opérateurs du bruit de quantification (des variables d'entrée ou des coefficients) qui se traduira par un bruit d'entrée sur les variables. Nous présentons chacune de ces trois techniques dans l'ordre de leur application.

3.2.1. fonction « dynamique »

Cette fonction analyse la dynamique des différentes variables traitées par l'algorithme considéré. Cette fonction annote le graphe $G(N, A)$ issu de la compilation, en appliquant à chaque nœud du graphe une fonction primitive modélisant le comportement du nœud et en parcourant l'ensemble du graphe, des entrées primaires aux sorties primaires. Le résultat de cette analyse est un graphe flot annoté qui contient des informations relatives à la dynamique des variables.

3.2.1.1. primitives

Ces primitives modélisent les analyses élémentaires portées sur les nœuds du DFG. Pour tout nœud « opération », il existe une fonction primitive Fp que l'on décrit dans la librairie de modèles de différents opérateurs pouvant implémenter cette opération. Les fonctions Fp sont définies par la valeur de la dynamique de sortie d'un nœud en fonction des valeurs de dynamiques des entrées de ce nœud [figure 4] :

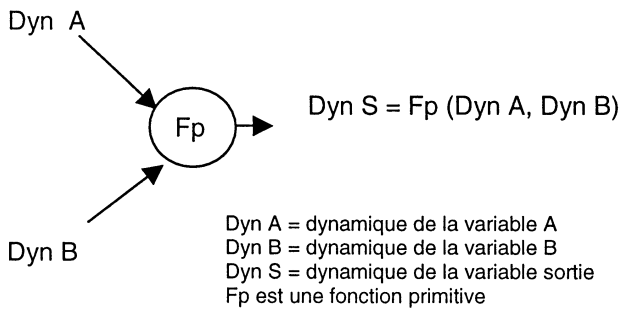


Figure 4. – Modélisation d'opérateur pour la primitive dynamique.

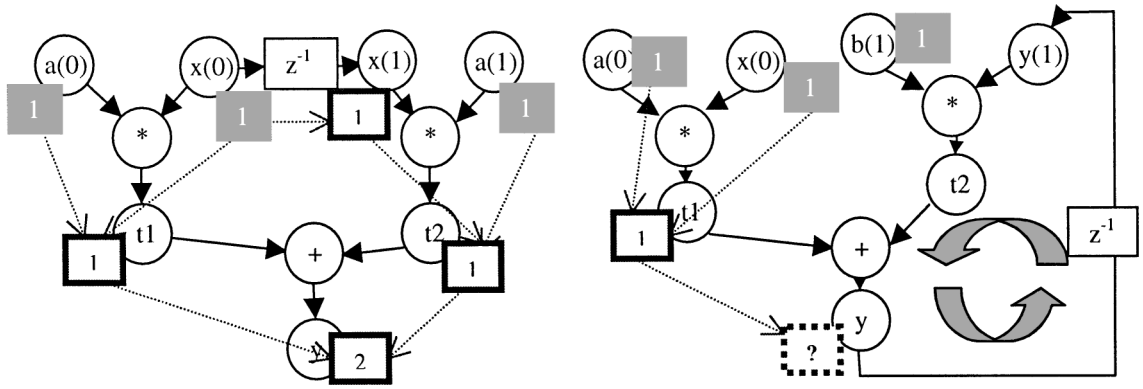


Figure 5. – Fonction dynamique : La dynamique des entrées primaires est notée en caractère plein, la propagation de la dynamique est notée en caractère entouré. On note sur le filtre IIR la présence d'un cycle dans le graphe qui ne permet pas l'analyse de la dynamique des variables.

Tableau 2. – Exemple de quelques modèles de fonctions primitives « Dynamique »

Nœud	Fonction Dynamique
Variable primaire	$Fp = 1$
Constante	$Fp = \lvert$valeur de la constante$\rvert$
Addition/soustraction *à deux entrées A et B*	$Fp = 2 \cdot \mathrm{Sup}(DynA, DynB)$
Multiplication *entre deux variables A et B*	$Fp = DynA * DynB$
Multiplication *entre une variable A et une constante C*	$Fp = DynA * \lvert C \rvert$

3.2.1.2. analyse du DFG

Pour calculer la dynamique de chaque variable traitée par l'algorithme, nous devons parcourir le DFG, des entrées primaires, variables externes ne résultant pas d'un traitement et liées au nœud source du DFG, à l'ensemble des sorties du DFG.

L'analyse échoue lorsque l'on rencontre dans le parcours avant du graphe un nœud tel que sa sortie est déjà annotée. C'est le cas des algorithmes récursifs (filtre RII), des algorithmes adaptatifs (filtre LMS, etc.), etc. [figure 5]

3.2.2. fonction « recadrage »

L'objectif de cette transformation de graphe est de garantir que la dynamique de chaque variable traitée par l'algorithme a une dynamique compatible avec le format de représentation du type logique. Par exemple pour une représentation en virgule fixe cadrée à gauche, la dynamique des variables est inférieure ou égale à 1.

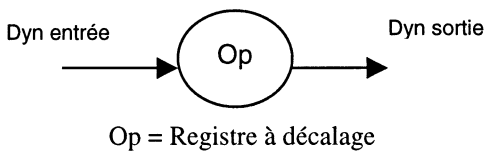
Comme pour la fonction « dynamique », nous définissons d'une part les fonctions primitives qui s'appliquent à chaque nœud, puis l'application de ces fonctions par un parcours arrière du graphe. La transformation du graphe peut être guidée vers une recadrage interne (les transformations sont internes au DFG) ou externe (les transformations ne sont portées que sur les entrées primaires du DFG).

3.2.2.1. primitives

Les primitives les mieux adaptées doivent réduire la dynamique des variables tout en conservant le comportement de l'algorithme. Nous justifierons ce deuxième point lors du développement des conditions d'application des primitives de recadrage. La primitive que nous avons retenue pour son faible coût d'implémentation est le registre à décalages [figure 6].

Cette primitive s'applique sur chaque nœud du DFG tel que la dynamique de sortie du nœud est supérieure à la dynamique de codage du type logique. L'application (conditionnée) de la primitive à un nœud du graphe conduit à la transformation locale du DFG suivante [figure 7].

$Dec\ nA$ et $Dec\ nB$ sont les décalages opérés sur les entrées *A* et *B* du nœud considéré, où pour un décalage de *n* bits à droite, $Dec\ n = 2^{-n}$. Les valeurs de $Dec\ nA$ et $Dec\ nB$ dépendent d'une part du modèle associé à l'opérateur Op et d'autre part de



Dyn-sortie = Dyn-entrée/2^n où n est le nombre de décalages

Figure 6. – Primitive « recadrage » : le registre à décalages.

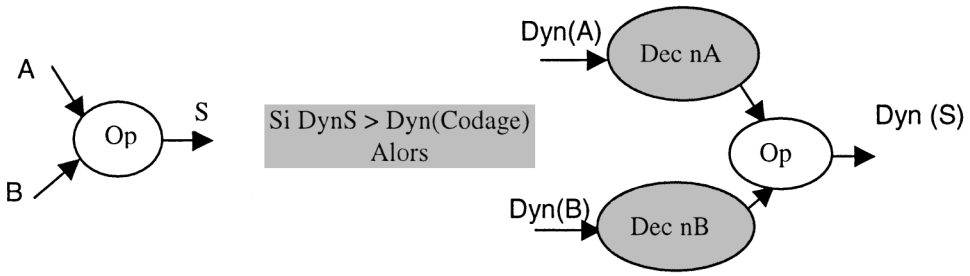


Figure 7. – Application conditionnée de la primitive de recadrage.

Tableau 3. – Modèles de recadrage

Nœud	Relations $Dec\ nA$, $DecnB$, $Dyn\ S$, Dyn Codage Si application de la fonction « recadrage »
Variable *A*	$Dec\ nA$ = [Dyn Codage / $Dyn\ S$]
Addition / soustraction *sur deux variables A et B*	$Dec\ nA$ = $DecnB$ = [Dyn Codage / $Dyn\ S$]
Multiplication *entre deux variables A et B*	$Dec\ nA$ * $DecnB$ = [Dyn Codage / $Dyn\ S$]

L'opération [.] est l'arrondi à la puissance de 2 immédiatement supérieure ou égale.

l'objectif à atteindre localement, soit que $Dyn\ S$ appartienne au domaine de codage ($Dyn\ S \leqslant Dyn$ Codage). La première contrainte est exprimée dans la bibliothèque de modèles des opérateurs. A titre d'exemple nous indiquons ci-dessous quelques modèles [tableau 3].

Notons que l'application de la fonction « recadrage » à la multiplication autorise de multiples choix puisque seul le produit $Dec\ nA$ * $Dec\ nB$ est défini. Nous verrons dans la transformation globale que pour l'équilibrage des différents chemins, le nombre de choix sera limité.

3.2.2.2. transformation globale du DFG

L'application du recadrage sur l'ensemble des opérations du graphe modifie l'algorithme considéré. Cette modification ne doit pas modifier les propriétés intrinsèques de l'algorithme. Ainsi dans le cas d'un filtre, on peut accepter une modification du gain de ce filtre sans accepter de modification de son gabarit spectral. En conséquence le recadrage ne peut affecter que les entrées primaires et plus exactement l'ensemble des chemins liant chaque entrée primaire à chaque sortie se voit affecté d'un même recadrage.

Exemple [figure 8] : Filtre RIF $y = \sum_{i=0}^{1} x(i)^* a(i)$

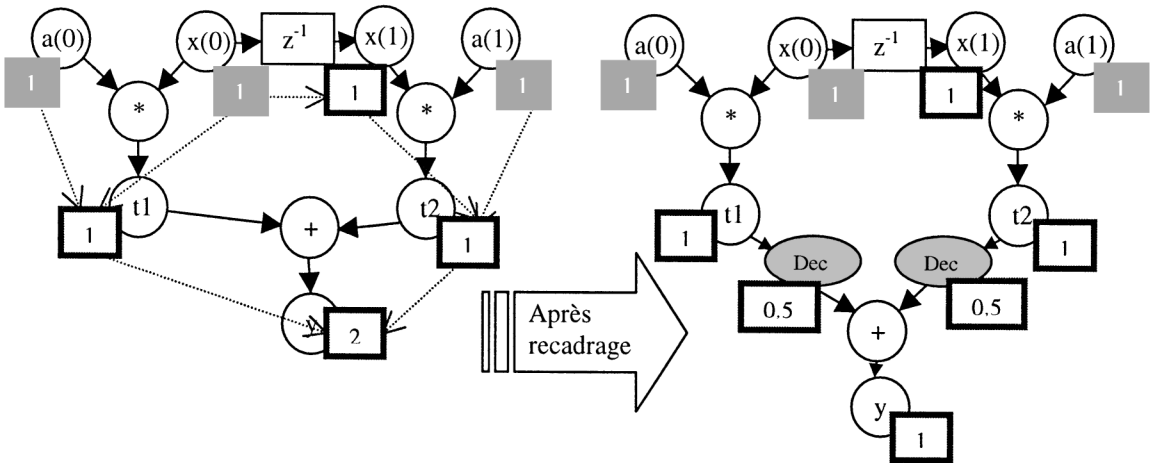


Figure 8. – Exemple de transformation globale : équilibrage des chemins entrée primaire – sortie.

La transformation globale sera traitée en deux parcours du DFG : un parcours avant, des entrées primaires aux sorties du graphe durant lequel on positionne dès que nécessaire les décalages. On recalcule la dynamique des variables et ceci jusqu'à atteindre l'ensemble des sorties. Un second parcours arrière du graphe a pour objectif d'équilibrer les chemins (à partir d'une sortie jusqu'à l'ensemble de ses entrées primaires le décalage total doit être identique) et de propager les décalages en amont pour qu'aucun chemin, vers une entrée non primaire, ne soit affecté d'un décalage.

Algorithme de transformation globale :
Pour tous les chemins des entrées primaires aux sorties
Appliquer à chaque nœud un décalage si nécessaire
Recalculer la dynamique de sortie, la propager
Pour tous les chemins des sorties aux entrées
Equilibrer en chaque nœud les décalages
Si le chemin mène à des entrées dont au moins une entrée non primaire, propager le décalage
vers une entrée primaire
Fin

La transformation globale échoue s'il n'est pas possible de propager un décalage vers une entrée primaire de manière à ne pas affecter un chemin entre sortie et entrée non primaire. Cet échec est par exemple systématique dans un algorithme adaptatif de type RII ou LMS.

La transformation globale que nous venons de présenter est appelée transformation à **recadrage interne**. En effet le graphe DFG après transformation peut comporter un ensemble de décalages placés en tout lieu sur les chemins. Une autre solution consiste à propager les différents décalages sur les entrées primaires et uniquement en ces lieux, ce qui amène une transformation globale à **recadrage externe** [figure 9].

3.2.3. fonction « bruit de calcul »

Les deux fonctions précédemment appliquées au graphe flot de données permet d'assurer qu'il n'y a pas de dépassement des valeurs des variables, dans le format de codage retenu (ici en virgule fixe cadrée à gauche). Il s'agit maintenant de nous intéresser à l'estimation des bruits de calculs engendrés d'une part par la propagation de ces bruits à travers les opérateurs, d'autre part par la génération de ces bruits au niveau de chaque opérateur. La fonction « bruit de calcul » est composée de trois parties : la première affecte à chaque opération du graphe flot de donnée ses caractéristiques en terme de génération et de propagation de bruits de calcul ; cette partie repose sur une bibliothèque de fonctions primitives associées à chaque type d'opération. La deuxième étape permet de composer les fonctions primitives suivant la composition des opérations indiquée par le graphe flot de données ; cette étape forme l'analyse globale. Enfin une évaluation du format de données peut être traitée analytiquement dans une troisième étape, par analyse du bruit de calcul sur les différentes sorties du graphe flot de données, et aboutira au choix du format de codage.

Cette fonction permet de valuer la puissance de bruit de calcul. Cette puissance est paramétrée par le nombre de bits du type logique des variables traitées par l'architecture en précision finie. La contrainte de qualité, rapport signal à bruit, imposée par l'application permet de déterminer ce format de données.

Le point de départ est le graphe flot de données issu de la transformation recadrage. Sur ce graphe nous analysons globalement le bruit généré et propagé à l'aide de fonctions primitives d'analyse. L'analyse globale s'achève lorsque toutes les sorties sont atteintes.

3.2.3.1. primitives pour l'analyse

Les fonctions primitives associées aux différents opérations du graphe flot de données sont celles que nous avons cité au para-

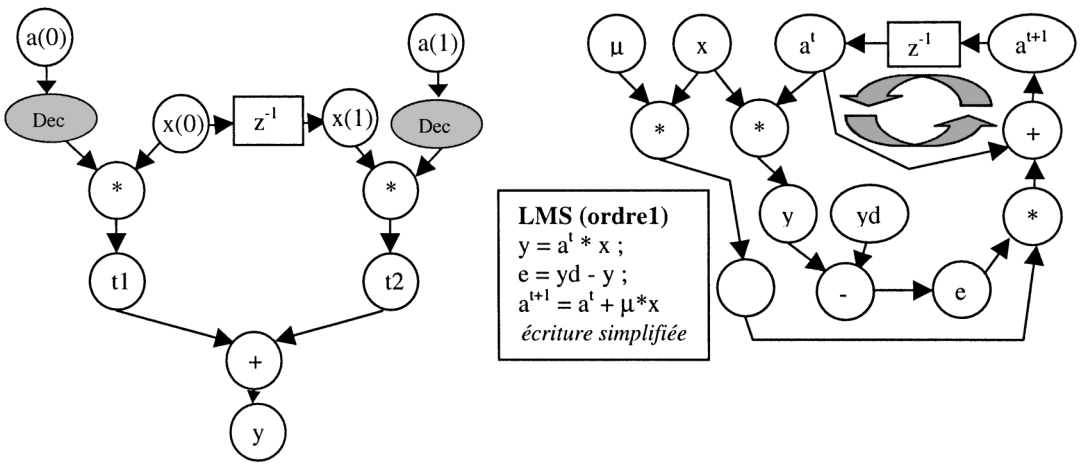


Figure 9. – *A gauche,* recadrage externe sur le filtre FIR, *à droite,* présence d'un cycle sans entrée primaire dans le graphe du LMS qui ne permet pas de traiter le recadrage.

graphe 3.2.1.1., c'est à dire pour chaque opération sa fonction de propagation et sa fonction de génération du bruit de calcul.

3.2.3.2. analyse globale

Comme pour l'analyse globale de la dynamique, cette analyse est le parcours avant du DFG. On peut prendre en compte le bruit sur les entrées primaires du DFG. Cependant l'analyse ne peut se faire sans un modèle fixé par le concepteur pour le bruit des entrées non primaires du DFG, ce qui peut être utile pour les algorithmes récursifs.

L'utilisation des modèles permet d'établir une formulation analytique de la puissance de bruit de calcul en chaque sortie du DFG. Cette expression est de la forme :

$$\text{Puissance\_Bruit\_Calcul} = A \; x \; q^2$$

où $q = 2^{-n}$ est la quantification pour un format en précision finie en virgule fixe cadrée à gauche sur n bits et A une constante dépendant de l'algorithme. La forme de cette expression est relativement simple, la puissance de bruit est additive [figure 10].

3.2.3.3. évaluation du format de données

Lorsque la puissance de bruit de calcul en sortie de l'algorithme est connue, il nous faut déterminer le format de données satisfaisant à la contrainte utilisateur. Nous faisons l'hypothèse que ce format de données est constant sur toutes les opérations du graphe flot de données. Cette hypothèse peut être levée au prix de l'augmentation de la complexité de l'optimisation, puisqu'il s'agira alors de définir conjointement les différentes zones de traitement et leurs formats de données associés. Pour pouvoir utiliser la puissance de bruit de calcul déterminée précédemment, nous formalisons la contrainte utilisateur. Tout d'abord, nous définissons le rapport signal à bruit de calcul $\text{RSB}_{\text{Calcul}}$.

$$\text{RSB}_{\text{Calcul}} = 10 \log \left(\frac{\text{Puissance\_Max}}{\text{Puissance\_Bruit\_Calcul}} \right)$$

où Puissance-Max est la puissance maximum des données codées. Sachant que les données codées sur l'architecture cible sont en virgule fixe cadrée à gauche, nous pouvons établir que Puissance_Max est inférieure ou égale à 1 (cette grandeur peut être affinée).

Soit $\text{RSB}_{\text{Contrainte}}$ l'expression de la contrainte. Pour que la contrainte soit satisfaite, il faut que l'égalité suivante soit satisfaite :

$$\text{RSB}_{\text{Calcul}} > \text{RSB}_{\text{Contrainte}}$$

Ce qui entraîne :

$$10 \log \left(\frac{\text{Puissance\_Max}}{\text{Puissance\_Bruit\_Calcul}} \right) > \text{RSB}_{\text{Contrainte}}$$

avec $\text{Puissance\_Bruit\_Calcul} = A \; x \; q^2$

$$q^2 < \frac{\text{Puissance\_Max}}{\text{A}} \times 10^{-\frac{\text{RSB}_{\text{Contrainte}}}{10}}$$

$q^2 = 2^{-2n}$ où n est le nombre de bits du codage.

Pour le codage virgule fixe cadrée à gauche, on obtient :

$$n > \frac{1}{2} \log_2 \left[A \times 10^{\frac{\text{RSB}_{\text{Contrainte}}}{10}} \right]$$

De la puissance maximum du signal utile et du coefficient A calculé dans la phase d'évaluation de la puissance de bruit de calcul, nous trouvons donc le nombre de bits en fonction de la contrainte utilisateur.

4. résultats

Les fonctions présentées dans les paragraphes précédents ont été implantées dans l'outil de synthèse architecturale GAUT. Les méthodes permettant l'application des fonctions sont basées sur

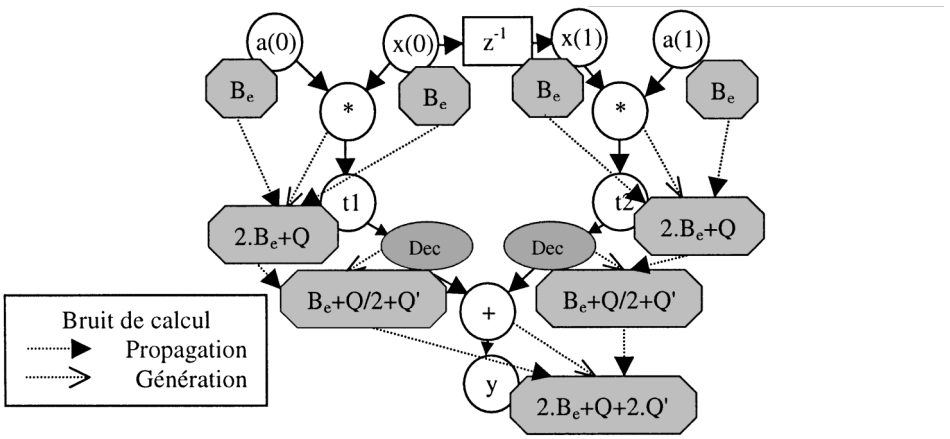


Figure 10. – Analyse de la propagation et de la génération du bruit de calcul sur le FIR d'ordre 2.

des explorations d'arbres et l'utilisation de bibliothèques de modèles. Notre méthodologie se présente sous la forme d'un module, GAUT_BC, auquel GAUT fait appel lors du déroulement du processus de synthèse. Ce module comprend les trois transformations : Dynamique, Recadrage et Bruit de calcul. La donnée d'entrée est donc le graphe flot de données et les informations de sortie sont le graphe transformé (ajout des nœuds de recadrage) et le format de codage adapté à la contrainte utilisateur.

Nous avons traité plusieurs algorithmes parmi lesquels deux transformations que sont la transformée de Fourrier rapide et la transformation en ondelettes de Debauchies [Antonini].

Pour les différents exemples, les modèles de comportement en dynamique ou en bruit de calcul des opérateurs sont entrés dans la librairie de l'outil de synthèse architectural GAUT.

Exemple : FFT sur 128 points.

L'algorithme choisi est une optimisation de la transformée de Fourier dont le but est de projeter le signal sur des signaux stationnaires sinusoidaux. Au niveau calcul, la FFT est une répétition du papillon de Cooley-Tuckey. Si $x(n)$ est un échantillon d'entrée, l'échantillon de sortie $X(k)$ est donné par :

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{-nk} \quad \text{avec} \quad W_N^{-nk} = e^{-2\pi j \frac{nk}{N}}$$

N est le nombre d'échantillons nécessaires pour calculer $X(k)$ et n est le nombre d'étapes de la FFT : $N = 2^n$

Nous considérons des valeurs d'entrée réelles.

Nous utilisons une technologie CMOS 0,7 μm pour laquelle les opérateurs de différents formats de traitement sont caractérisés en temps et en surface. La synthèse architecturale consiste à exploiter avec méthode les parallélismes de l'algorithme de manière à satisfaire au plus juste la contrainte de cadence des traitements.

Pour une cadence fixée à 57 *ns*, nous comparons trois architectures, la première utilise des opérateurs en virgule flottante sur 32 bits (format IEEE 754) et sert de « référence » puisque nous pouvons considérer que cette architecture ne génère par de bruit de calcul. Nous avons ensuite fixé deux contraintes de bruit de calcul, la première pour un rapport signal à bruit de 60 dB, la seconde pour 90 dB. Enfin pour chaque contrainte nous avons exploré les deux types de recadrage interne ou externe. Le tableau suivant donne les résultats obtenus [tableau 4].

Les cinq architectures synthétisées respectent la même cadence de traitement (57 *ns*). On remarque que la prise en compte au plus juste d'une contrainte de bruit de calcul permet de faire varier le coût en surface de l'architecture de traitement dans un rapport 1 à 14. Le gain en surface est obtenu par la réduction du coût des opérateurs, du à la réduction du nombre de bits du format des données, mais aussi par la réduction du nombre d'opérateurs (on passe par exemple de 4 multiplieurs à 2 pour les deux cas pris en référence) du à la diminution du temps de traversé des opérateurs. Enfin, concernant le coût de la mémoire, directement lié au nombre de bits du format retenu, il varie dans un rapport de 1 à 3,5 (rapport mesuré entre l'architecture en format flottant et celle obtenue pour 60 dB et un recadrage interne).

Tableau 4. – Résultats de la synthèse d'architecture de la FFT pour différentes contraintes R S/B

	format flottant	virgule fixe			
		SNR_{calcul} = 60 dB		SNR_{calcul} = 90 dB	
	IEE 754	interne	externe	interne	externe
Nombre de bits	32	9	11	14	16
Multiplieur	4	2	2	2	2
Additionneur	3		1		1
Soustracteur	4		1		1
Décaleur			1		1
Additionneur et décaleur		1		1	
Soustracteur et décaleur		1		1	
Surface $\mathrm{mm}^2 \cdot 10^{-3}$	39318	2779	5073	7145	8254

Exemple : Transformée en ondelettes (DWT). L'algorithme utilisé sert notamment dans le domaine de compression des données (image). Le processus de compression est défini de la manière suivante [figure 11]

Des transformations sont apportées à l'image numérique (transformée en ondelettes). Ce traitement est suivi d'une quantification et finalement d'un codage.

Le principe de l'algorithme de transformée en ondelettes repose sur la projection du signal sur deux fonctions mathématiques ondelettes d'une base orthogonale [Lecordier]. Le signal est donc décomposé en deux fonctions orthogonales qui peuvent être explicitées sous la forme d'une équation de filtrage de type FIR suivie d'une décimation par deux. Concrètement, le résultat de ces projections se traduit par l'obtention de sous images, représentatnt soit des coefficients, soit une image basse résolution [figure 12].

Le traitement se divise en plusieurs degrés de résolution. Le traitement de degré i est représenté sur la figure suivante [figure 13].

Un degré de résolution se fractionne en deux parties. Tout d'abord le filtrage est effectué sur les lignes et ensuite sur les colonnes. Chaque partie comporte un filtrage de type FIR passe bas L et FIR passe haut H, suivi d'une décimation par deux.

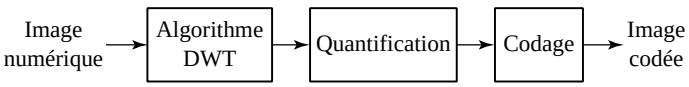


Figure 11. – Processus de compression d'image par DWT.

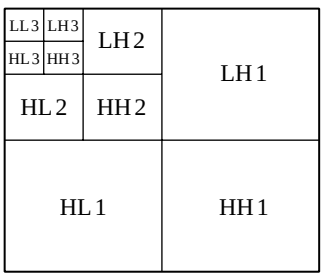


Figure 12. – Transformée de niveau 3.

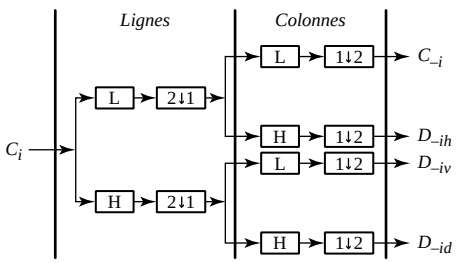


Figure 13. – Premier étage de l'algorithme de transformée en ondelettes.

Ainsi, en fin de traitement d'un degré de résolution, nous obtenons quatre sous images. L'image filtrée, par LL est l'image basse résolution, par HL représente les détails horizontaux, par LH les détails verticaux et par HH les détails obliques. La décimation entraîne le fait que la taille des images résultant d'un traitement sur un degré de résolution est divisée par deux en largeur et en hauteur.

Nous avons expertisé l'implantation de la transformée en ondelettes biorthogonale utilisant les filtres symétriques 7 – 9 de Debauchies [Antonini]. Les filtres H et L sont de longueur 7 et 9. Nous considérons que le bruit de calcul de la pahase de reconstruction du signal est négligeable (car effectué en format flottant étendu) par rapport à l'algorithme implanté en virgule fixe cadrée à gauche. La transformée s'opère sur trois niveaux de résolution et pour une image de taille 9000×9000 pixels traitée à la cadence de 9 images/s. Nous avons fixé une contrainte de bruit RSB = 50 dB et avons travaillé les deux types de recadrage [tableau 5].

Tableau 5. – Résultats de synthèse sur la DWT

	décalage externe	décalage interne
Nombre de bits	15	12
coûts en surface 10^{-3} mm^2	10985	6310
Consommation Mutl / Add / Sous	288/11/10 μW/ *calcul*	192/9/8 μW/ *calcul*
coût en consommation	1344 mW	1211 mW

Ces résultats ont également été analysés en terme de puissance consommée. Les différents recadrages permettent d'obtenir à la fois un gain en surface et en consommation.

5. améliorations et conclusions

Basée sur une connaissance très locale des traitements, la méthode de conception optimisée d'architectures en précision finie a montré sa capacité à être automatisée en donnant des résultats intéressants sur les algorithmes de traitement du signal que nous avons implémenté. La méthode est suffisamment générale pour s'intégrer dans d'autres flots de synthèse d'architecture, comme ceux des outils industriels Monet de MENTOR ou Behavioral Compiler de SYNOPSIS.

La qualité des analyses et des transformations est très liée aux modèles qui supportent les fonctions primitives. Nous pouvons cependant améliorer ces modèles indépendamment des techniques d'analyses ou de transformations globales. Une perspec-

tive que nous envisageons d'étudier à court terme, consiste à regrouper un ensemble d'opérations lors de l'analyse du DFG, suivant un modèle de structure préalablement défini, de manière à affiner les estimations ; par exemple une structure de type multiplication-accumulation pourra être traitée indépendamment ce qui autorise la prise en compte soit d'un format de représentation étendu (à l'image de ce qui est intégré dans les DSP), soit d'une meilleure connaissance sur la dynamique des coefficients d'un filtre considéré dans sa globalité : prendre en compte la norme du filtre et non plus considérer les opérations une à une, en particulier utiliser la propriété du format de codage en complément à deux qui autorise un dépassement de la dynamique sur un calcul intermédiaire à la condition que le calcul final soit dans la dynamique de codage.

BIBLIOGRAPHIE

[ANTONINI] M. Antonini, M. Barlaud, P. Mathieu & I. Debauchies « Image coding using Wavelet transform », *IEEE transaction on image processing*, Vol 1, n° 2 April 1992.

[BARNES] C.W. Barnes, B.N. Tran, S.H. Leung, « On the statistics of fixed-point roundoff error », *IEEE Transactions on acoustics, speech and signal processing*, Vol. ASSP-33, N° 3, june 1985.

[BC] Behavioral Compiler, http://www.synopsys.com/products/beh-syn/beh-comp-cs.html

[BELLANGER] M. Bellanger, « Traitement numérique du signal », *Collection technique et scientifique des Télécommunications*, CNET-ENST, 1984.

[CADENCE] www.cadence.com

[GAILHARD] S. Gailhard, « Conception d'architectures à faible consommation », *Thèse* de l'Université de Rennes, Janvier 1999.

[LECORDIER] C. Lecordier, « Rapport interne » laboratoire LESTER, 1997.

[MARTIN 1] E. Martin, A. Gilloire, P. Le Scan, « Conception assistée par ordinateur d'architectures de traitement du signal : application à l'annulation d'écho acoustique », *Annales des Télécommunications*, 49, n° 7-8, pp 447-459, 1994.

[MARTIN 2] E. Martin, O. Sentieys, H. Dubois, J.L. Philippe, « Gaut, an architecture synthesis tool for dedicated signal processors », *In proceedings of EURO-DAC 93*, pp 14-19, 1993.

[MARTIN 3] E. Martin, O. Sentieys, J.L. PHILIPPE, « Synthèse architecturale de cœur de processeurs de traitement du signal », *Techniques et Sciences Informatiques*, Vol. 13, n° 2, pp 251-279, 1994.

[MONET], http://www.mentor.com/monet/index.html.

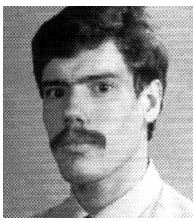
[PHILIPPE] J.L. Philippe, O. Sentieys, E. Martin, H. Dubois, « Adéquation d'un algorithme à une architecture, application à la transformée de Fourier », *Traitement du signal*, Vol. 13, n° 4, pp 3356350, 1996.

[SYNOPSYS] www.synopsys.com

[TMSC6X] « TMS320C6000 Power Consumption Summary », *application Report SPRA486A, Texas Instrument*.

LES AUTEURS

Eric MARTIN



Agrégé de génie électrique à l'ENS de Cachan en 1984, Professeur à l'UBS depuis 1994 où il dirige le laboratoire LESTER. Son domaine d'intérêt porte l'adéquation algorithme architecture et les outils de CAO en architectures dédiées aux applications de traitement du signal et de l'image.

Christophe NOUËT

Doctorat en Electronique obtenu à l'Université de Bretagne Occidentale (Brest) en 1994, Maître de conférences à l'IUT de Lorient, département Génie Industriel et Maintenance depuis la même année. Ses travaux portent sur le développement de méthodologies de synthèse de haut niveau en précision finie.

Jean-Marc TOURREILLES

Docteur en traitement du signal et télécommunications en 1999.

Il étudie en DEA à Rennes le traitement du signal pour les télécommunications et les hyperfréquences. Durant un an, il travaille dans le domaine des communications numériques dans le cadre de son service militaire tout en poursuivant des travaux dans les hyperfréquences. Ensuite, il effectue une thèse au L.E.S.T.E.R. Son projet consiste à intégrer une nouvelle contrainte dans le processus de synthèse architecturale. Plus précisément, de proposer une méthode automatique permettant la maîtrise de la dynamique et des bruits de calcul pour l'implantation d'algorithmes de traitement du signal à l'aide d'un outil de synthèse haut niveau. Actuellement, il travaille pour la société Alcatel dans le domaine de la téléphonie mobile.