Prévision de la consommation pour les algorithmes de transformée en ondelette

S. Gailhard , N. Julien , E. Martin

Laboratoire d'Electronique des Systèmes TEmps Réels - IUP
2, rue Le Coat St Haouen, 56100 Lorient
E-mail: nom@univ-ubs.fr http://lester.univ-ubs.fr:8080/

RESUME

La consommation est un critère essentiel dans les applications embarquées. Pour un algorithme de TDSI (Traitement Du Signal et Image) supportant des contraintes de temps réel, il est possible d'obtenir des gains importants en consommation sur le composant VLSI qui l'implante, en tenant compte de ce facteur dès la phase de spécification. Il faut donc posséder des estimateurs qui soient à la fois précis et rapides pour obtenir une estimation fiable de la consommation sur un large domaine d'applications. Dans cet article, nous présentons une méthode d'estimation probabiliste de la consommation au niveau comportemental permettant de réaliser l'adéquation algorithme architecture. Celle-ci est appliquée à la transformée en ondelette (DWT) utilisée dans une chaîne de compression d'images dans le cadre d'une application spatiale embarquée.

ABSTRACT

Power consumption is an essential criteria in embedded systems. Concerning an algorithm for Signal and Image Processing with real time constraint, it is possible to obtain important power saving factors on the VLSI chips when this factor is considered since the algorithmic description. Then, it is important to have a fast and accurate estimator to obtain reliable power estimation on a large number of applications. In this article, we present a methodology using a probabilistic estimation of the power dissipation at an algorithmic level in order to obtain an algorithm-architecture relation. This methodology has been applied to the Discrete Wavelet Transform (DWT) used in a image compression chain in the scope of an spatial embedded system.

1. Introduction

Les systèmes embarqués nécessitent la mise en oeuvre de circuits à faible consommation. Citons l'exemple des applications satellites pour lesquelles le stockage de l'énergie pose un réel problème, ou celui des télécommunications portables.

Lorsque l'on augmente le niveau d'abstraction d'une spécification de circuit (niveau physique, logique, RTL (Register Transfer Level) et algorithmique), le temps de calcul nécessaire à l'estimation de la consommation décroît, de même que la précision des résultats [1]. Pour comparer une large gamme d'algorithmes, il faut donc que l'estimateur se situe au plus haut niveau tout en préservant une grande précision. L'estimateur de consommation *Gaut_W* que nous proposons est basé sur l'utilisation d'une description comportementale de l'application et d'une librairie des ressources de l'architecture générique. Il permet ainsi de guider l'adéquation algorithme architecture.

Après l'avoir présenté, nous présenterons les résultats obtenus sur différents algorithmes de transformées en ondelette, parmi lesquels on cherchera à estimer le plus tôt possible la consommation du circuit VLSI susceptible d'être conçu pour leur mise en oeuvre.

2. Estimateur de consommation

La position de notre estimateur dans le flot de conception (de la spécification jusqu'au circuit VLSI) est représentée à la Figure 1.

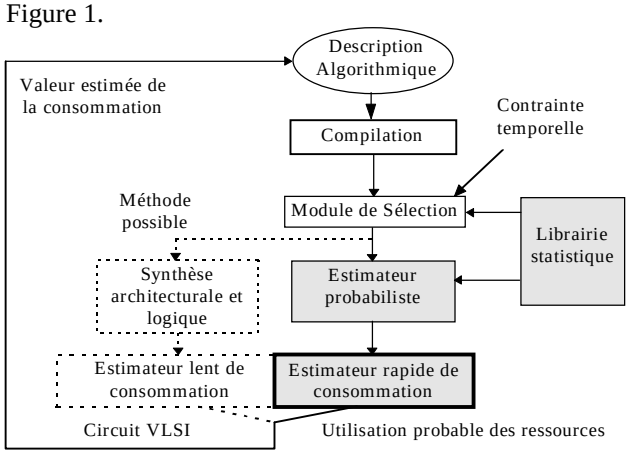

Figure 1 : Flot de l'estimateur de consommation

L'entrée de notre flot est écrit dans un langage de haut niveau permettant une description algorithmique de l'application. Nous utilisons actuellement le langage VHDL comportemental, mais notre méthode pourrait tout aussi bien supporter une entrée en langage C ou Signal. Une phase de

compilation et de transformations permet d'obtenir une représentation parallèle de l'application à parallélisme "maximum" [2]. Le module de sélection permet ensuite de choisir automatiquement les différents opérateurs de la librairie répondant aux contraintes temporelles de l'application et minimisant un coût. Deux options s'offrent alors à l'utilisateur de l'outil :

- Appliquer une synthèse architecturale puis logique, ce qui permet d'aboutir à un circuit VLSI dont on pourra estimer précisément la consommation à partir d'outils de CAO ; cette voie nécessite un temps CPU très important.
- Estimer la consommation probable du circuit VLSI susceptible d'être mis en oeuvre à l'aide d'un estimateur rapide. C'est cette voie que nous étudions.

L'estimateur probabiliste de consommation que nous proposons prend en entrée la spécification algorithmique de l'application ainsi qu'une librairie fonctionnelle comprenant les caractéristiques des composants utilisés. Après avoir présenté la méthode utilisée pour générer la librairie, nous présenterons notre estimateur de consommation *Gaut_W*.

2.1. La librairie statistique

La consommation statique d'une porte CMOS est négligeable devant sa consommation dynamique. Ainsi, la méthode statistique que nous avons développée repose sur le calcul du nombre de transitions de chaque noeud par simulation à l'aide d'un simulateur logique [3]. Un stimulus est appliqué à l entrée d'un opérateur logique, et le simulateur fournit le nombre de commutations de chacune des portes du circuit. En utilisant une librairie contenant la consommation due à la commutation des portes, on calcule la puissance moyenne dissipée par l'opérateur sur l'ensemble du stimulus :

$$P = f.V_{dd}^{2} \sum_{i} C_i . N_i = f.W \qquad (1)$$

f : Fréquence d'arrivée des données
V_{dd} : Tension d'alimentation
C_i : Capacité de la porte i sommée avec la valeur de la sortance de la porte
N_i : Nombre de transitions 0-1 de la porte i
W : Consommation moyenne de l'opérateur par accès

L'estimation de la consommation moyenne par accès W dépend de la taille du stimulus : plus sa taille augmente et plus la valeur moyenne calculée se rapproche de la valeur moyenne théorique. Pour plusieurs stimuli de taille N générés de manière aléatoire suivant une loi de probabilité spécifiée par l'utilisateur (nos exemples reposent sur une loi uniforme), on peut déterminer l'intervalle de confiance associé à la moyenne mesurée. En simulant chaque opérateur 5 fois avec des stimuli de taille 10000, on obtient 1% d'erreur pour un intervalle de confiance de 95% [4]. La simulation de la consommation d'un opérateur requiert de quelques minutes à une heure et dépend du nombre de noeuds de l'opérateur traité. Cette méthode a servi pour l'estimation de la consommation des registres, additionneurs et multiplieurs. La consommation moyenne W d'un opérateur ne comprend que sa consommation intrinsèque et non la consommation due aux interconnexions.

La consommation des mémoires a été déterminée de façon différente. Vu le nombre important de transistors intégrés dans une mémoire, la consommation statique n'est plus négligeable devant la consommation dynamique. Par conséquent, nous avons utilisé un estimateur de consommation intégré dans l'outil *Compass*. Nous avons déterminé deux valeurs pour chacune des mémoires de tailles différentes :

- la consommation statique qui correspond à une consommation indépendante du nombre d'accès à la mémoire.
- la consommation dynamique qui correspond à la consommation supplémentaire moyenne par accès.

En ce qui concerne l'estimation de la consommation de l'arbre d'horloge, nous avons utilisé un modèle simplifié : la capacité de l'arbre d'horloge suit une loi linéaire par rapport au nombre de registres qui lui est connecté. Ainsi, nous avons pu définir une valeur correspondant à la consommation de l'horloge par registre.
Les valeurs stockées dans la librairie statistique sont résumées dans le Tableau 1.

Opérateur (16 bits)	Consommation par accès (pJ)	Temps (ns)
Registre	71	3
Additionneur	190	10
Multiplieur	750	35
Horloge par registre	65	
mémoire 128×8 bits Statique: 18 mW	561	20

Tableau 1 : Bibliothèque d'opérateurs (technologie CMOS 1 µm)

2.2. Estimateur probabiliste au niveau algorithmique

Notre méthode probabiliste cible des champs d'application en traitement numérique temps réel où les ressources mises en oeuvre dans le circuit présentent des taux d'utilisation proches de 100 %.

La première étape de l'estimateur probabiliste est le calcul de la probabilité d'ordonnancement de chacune des opérations de l'application de part la connaissance de la contrainte temps réel de l'application [5]. A partir de ces probabilités d'ordonnancement, nous pouvons en déduire celles des transferts des données (transfert à travers la mémoire ou via un registre interne) et ainsi l'activité probable de chacune des ressources de l'architecture générique. De même, nous pouvons estimer le nombre probable de registres $Nb_{registres}$ (comprenant les registres stockant les variables temporaires et les registres dus aux opérateurs avec plusieurs tranches de pipeline) et en déduire la consommation du circuit d'horloge.

Connaissant la consommation moyenne des ressources mises en oeuvre (grâce à la librairie statistique donnée ci-dessus) et le nombre probable d'activités des ressources,

nous estimons alors dans une deuxième étape la consommation totale de l'application :

$$P = f_{appli} \cdot \sum_i W_i . N_i + f_{hor\log e} . W_{hor\log e} . Nb_{registres} \quad (2)$$

f_{appli}	Fréquence de fonctionnement de l'application
$f_{horloge}$	Fréquence de fonctionnement de l'horloge interne
W_i	Consommation de la ressource i
N_i	Nombre probable d'accès à la ressource i
$W_{horloge}$	Consommation de l'horloge par registre
$Nb_{registres}$	Nombre probable de registres du circuit

3. Application

3.1 Chaîne de compression

La compression des images peut être obtenue par une transformation des données en les projetant sur une base de fonctions orthogonales et en réalisant ensuite une quantification suivie d'un codage (Figure 2).

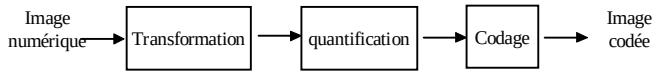

Figure 2 : Chaîne de compression d'images

3.2 Algorithme de transformée en ondelette

Il existe de nombreuses transformées dont les plus connues sont la transformée de Fourrier (FFT), les transformées en cosinus (DCT) et enfin la transformée en ondelette. Cette dernière possède plusieurs propriétés intéressantes dont l'existence d'un algorithme rapide. L'algorithme de Daubechies que nous utilisons [6] repose uniquement sur l'utilisation de filtres passe haut "g" et passe bas "h".

:

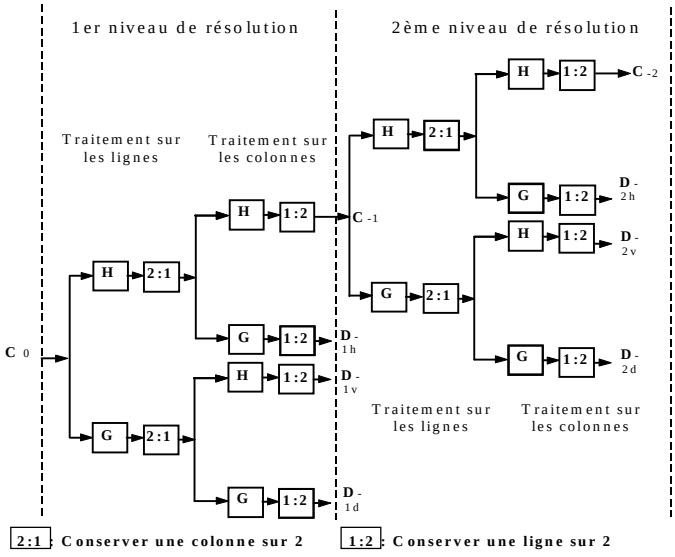

Figure 3 : DWT à 2 niveaux de résolution

Plusieurs étapes sont alors nécessaires à chaque niveau de résolutions :

- L'image initiale C0 subit tout d'abord 2 traitements en parallèle sur des filtres g et h lignes. Puis, on ne conserve qu'une colonne sur 2 pour chacun des 2 résultats.
- Les 2 résultats de ce premier traitement subissent alors 2 traitements en parallèle sur des filtres g et h colonnes dont on ne conserve qu'une ligne sur 2.

Ces traitements aboutissent à quatre images distinctes dont la taille est 4 fois inférieure à l'image initiale. Seule l'image ayant subit deux filtres h est conservée et servira d'image d'entrée pour le niveau supérieur. Les trois autres images sont gardées telles quelles (Figure 4).

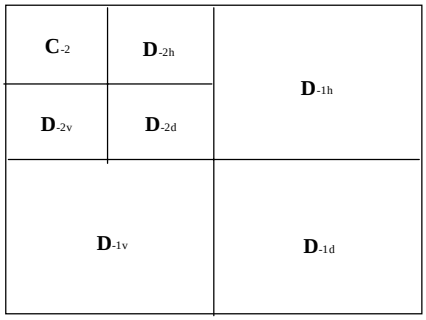

Figure 4 : Image résultante après une DWT

Plusieurs types de filtres ont été proposés dans [6]. Les filtres g et h sont des FIR (Finite Impulse Response) de tailles différentes :

- 9-3 : h filtre du 9ème ordre et g filtre du 3ème ordre
- 9-7 : h filtre du 9ème ordre et g filtre du 7ème ordre

L'algorithme présenté ci-dessus peut être utilisé avec plusieurs niveaux de résolution.

3.3 Spécification de l'application

Nous travaillons sur des images de taille 512×512 évoluant à la fréquence de 30 images par seconde, ce qui implique une contrainte de temps réel de 1/30=33 ms par image. Pour permettre de satisfaire cette contrainte de temps, nous avons spécifié notre algorithme de DWT uniquement sur un bloc de l'image de taille *n×n.* Ainsi, nous pouvons utiliser plusieurs PE (Processeurs Elémentaires) réalisant chacun des calculs élémentaires (DWT avec 1 niveau de résolution sur un bloc). L'image doit subir un traitement ligne puis un traitement colonne. Par conséquent, les différents PE peuvent être utilisés en parallèle pour le traitement des différents blocs constituant *n* lignes, tandis qu'ils ne doivent être utilisés qu'en mode pipeline pour le traitement des différents blocs constituant *n* colonnes.

Nous allons maintenant évaluer le nombre de calculs élémentaires nécessaires et leur temps de fonctionnement maximal en fonction du degrés *d* de résolution, du nombre N_{PE} de processeurs élémentaires et de la taille *n×n* du bloc (*n<512).*

Premier degré de résolution : *d* = 1

Le nombre de bloc à traiter vaut N_c. Nous en déduisons donc le temps par calcul élémentaire $T_c(d)$ sachant que tous les calculs réalisés sur les N_{PE} PE doivent être réalisés en 33 ms.

$$N_c(1) = (512/n)^2$$
$$T_c(d) = (33.33/N_c).N_{PE}(ms) \quad (3)$$

***d*ème degré de résolution :**
Le nombre de calculs élémentaires supplémentaires à effectuer à chaque niveau de résolution est divisé par quatre par rapport au nombre de calculs élémentaires effectués au niveau de résolution précédent :

$$N_C(d) = (512/n)^2 \left(\underbrace{1 + 1/4 + \ldots + (1/4)^{d-1}}_{d} \right) = (512/n)^2 \left(\frac{1-(1/4)^d}{1-1/4} \right) \quad (4)$$

3.4 Résultats

Nous avons appliqué différents algorithmes de DWT à l'outil présenté au paragraphe 2.2 en faisant varier le niveau de résolution *d*, le nombre N_{PE} de processeurs utilisés et la taille des filtres h et g pour une taille de bloc de *8*8*. Les résultats de notre outil sont décrits à la Figure 5. Pour chaque type de filtre et pour chaque résolution, nous donnons les résultats de l'estimation de consommation totale de l'application, le nombre de multiplieurs présents dans un PE ainsi que la contrainte temporelle T_c (ns) imposée pour chaque calcul élémentaire.

Niveau de résolution	Nombre de PE	Taille des filtres	Consom. de l'application (mW)	Nb de Mult. par PE
d = 1 :	N_{PE} = *2*	Filtres 9/3	560	3
	T_c = 16113	Filtres 9/7	718	4
	N_{PE} = *4*	Filtres 9/3	604	2
	T_c = 32277	Filtres 9/7	764	2
	N_{PE} = *8*	Filtres 9/3	688	1
	T_c = 64453	Filtres 9/7	856	1
d = 3	N_{PE} = *2*	Filtres 9/3	736	4
	T_c = 12277	Filtres 9/7	942	6
	N_{PE} = *4*	Filtres 9/3	784	2
	T_c = 24554	Filtres 9/7	1004	3
	N_{PE} = *8*	Filtres 9/3	856	1
	T_c = 49107	Filtres 9/7	1080	2

Figure 5 : Résultats sur différents types de DWT

Les temps de calcul nécessaires à l'estimation de la consommation varient entre 2 et 10 minutes sur une station *Sparc Station 5*.

La Figure 5 montre que la consommation de l'application est effectivement dépendante des paramètres N_{pe} ,*d* et la taille du filtre utilisé. On aperçoit que :

- augmenter le nombre de processeurs élémentaires dans le circuit implique une augmentation de la consommation totale de l'application par exemple de l'ordre de 20 % pour le passage de 2 à 8 PE. Toutefois, la complexité en surface des PE est moins importante lorsque leur nombre alloué est plus important (nombre de multiplieurs présents dans chaque PE). Les résultats de l'estimateur de consommation guideront donc l'utilisateur pour trouver un compromis entre nombre de processeurs, complexité de chacun des processeurs élémentaires et consommation totale de l'application.
- l'augmentation de la consommation due à la variation du nombre de calculs (taille des filtres différent) augmente moins vite que cette dernière : il y a 20% de calculs en plus entre le filtre 9/7 et 9/3 tandis que la consommation n'augmente que de 15%.

Toutefois, l'estimateur probabiliste ne prend pas encore en compte la consommation des multiplexeurs, démultiplexeurs, l'unité de contrôle ainsi que la consommation du générateur d'adresses. Mais, il a été montré dans [7] que cette consommation était négligeable par rapport à la consommation des opérateurs, registres et mémoires pour les petits circuits, ce qui est notre cas.

Une autre étude d'un point de vue algorithmique permettrait de fournir à l'utilisateur différents résultats sur l'image en fonction de ces mêmes paramètres. L'utilisateur pourrait alors décider au plus tôt dans la phase de conception le meilleur compromis entre consommation, complexité des PE et performances des algorithmes mis en oeuvre.

4. Conclusion

L'estimateur de consommation proposé dans cet article permet à l'utilisateur une estimation de la consommation au plus tôt dans la phase de conception, dès la spécification algorithmique. Ceci permet à l'utilisateur de choisir très rapidement les meilleurs spécifications algorithmes qui répondent au mieux à ses exigences.

Les résultats obtenus sur la DWT montrent qu'une certaine augmentation en complexité en calcul n'a pas toujours le même impact sur la consommation. Ceci implique l'utilité d'un estimateur au niveau algorithmique plutôt qu'une simple comparaison d'algorithmes par rapport à leur complexité en calculs. Une autre perspective d'utilisation d'un estimateur de haut niveau est de fournir une fonction coût évaluée rapidement permettant ainsi son utilisation dans le module de sélection afin d'intégrer le facteur consommation dès la première étape de la synthèse architecturale.

4. Références

[1] P. E. Landman, "*Low-power Architectural design methodologies*", PhD, Berkeley, Août 94.

[2] E. Martin, O. Sentieys, H. Dubois, J.-L. Philippe, "*GAUT, an architectural synthesis tool for dedicated signal processors*", Proceedings EURO-DAC 93.

[3] A. Ghosh, S. Devadas, K. Keutzer, *"Estimation of average switching activity in combinatorial and sequential circuits*", IEEE DAC 1992, pp. 253-259.

[4] T. Valentin, "*Estimation de la consommation des circuits VLSI*", Rapport de DEA, Lester, Juillet 97

[5] J.-Ph. Diguet, O. Sentieys, J. L. Philippe, E. Martin, "*Probabilistic resource estimation for pipeline architecure*", IEEE Workshop on VLSI S.P.,Sakai, Japan, Oct. 95.

[6] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, "*Image coding using wavelet transform*", IEEE trans. on Image Processing, Vol. 1, No 2, Avril 92, pp. 205-220

[7] S. Gailhard, O. Ingremeau, J.-Ph. Diguet, N. Julien, E. Martin, "*Une méthode probabiliste pour estimer la consommation à un niveau algorithmique*", Colloque CAO de circuits intégrés et systèmes, Grenoble, Jan. 97, pp. 124-127.