Méthode d'implantation d'algorithmes de traitement du signal en précision finie

J-M. TOURREILLES, C. NOUËT, E. MARTIN

Laboratoire d'Electronique des Systèmes TEmps Réels
Université de Bretagne Sud
10 rue Jean Zay, 56100 Lorient
Tel: 02.97.87.28.34 FAX: 02.97.87.28.62
E-mail : tourreil@univ-ubs.fr / nouet@univ-ubs / emartin@univ-ubs
http://lester.univ-ubs.fr:8080/

RÉSUMÉ

Les contraintes imposées aux applications temps réel des systèmes d'électronique embarquée sont d'une part la réduction de surface et d'autre part la réduction de consommation. La conception d'architectures utilisant des opérateurs en précision finie permet d'optimiser ces contraintes. En effet, nous pouvons obtenir, à l'aide de la méthode présentée, intégrée à un outil de synthèse architecturale, des ASICs possédant un ou plusieurs chemins de traitement de données à des formats différents, optimisés pour une contrainte de rapport signal à bruit de calcul.
Cet article présente une méthode d'aide au dimensionnement des chemins de données ce qui nécessite une étude des problèmes de débordement et une analyse de la puissance de bruit de calcul. Cette méthode d'adéquation algorithme architecture sera illustrée par une application de traitement non récursif (FFT), puis par une application de traitement récursif (Filtre à Réponse Impulsionnelle Infinie). Associée à un outil de synthèse haut niveau, cette méthode nous permet de répondre au mieux aux contraintes de la conception.

ABSTRACT

The constraints for real time application of embedded systems are in the one hand the circuit area and in the other hand the power consumption. This constraint can be respected by the high level synthesis with fixed-point arithmetic. Indeed, the presented method integration to a high level synthesis tool leads to different data format architectures. Then the synthesis is done under signal to noise ratio.
This paper presents a method to help circuit design whose is based on overflow and computation noise analysis. To illustrate this relation algorithm architecture method we present two applications : a non recursive algorithm (the Fast Fourier Transform : FFT), and a recursive algorithm (an infinite impulse response filter, RII). Associated to a high level synthesis, this method allows to match the design constraints

1. Synthèse architecturale

Une des étapes de conception pour passer d'un algorithme de traitement du signal à sa réalisation pratique (ASICs) est la synthèse architecturale. Cette étape consiste, à partir d'une description comportementale de l'algorithme (VHDL haut niveau), à établir la structure parallèle du circuit réalisant l'algorithme considéré pour un énoncé de contrainte (temps, consommation, coût). En d'autres termes, l'outil de synthèse architecturale permet de choisir les opérateurs qui constitueront l'architecture, d'établir leur agencement et d'optimiser leur utilisation. Le détail du flot de conception de l'outil de synthèse GAUT [1] est illustré par la figure 2.

Actuellement, l'optimisation de l'architecture (surface) s'effectue sous contrainte de temps en utilisant une bibliothèque d'opérateurs caractérisés par leur temps de latence et leur surface. Le but de la méthode présentée est d'intégrer le format des chemins de traitement de données en tant que paramètre dans la phase de synthèse architecturale. Celle-ci se fera sous contrainte de rapport signal à bruit de calcul. Ainsi les chemins de données de l'architecture sont dimensionnés au plus juste ce qui entraîne un gain en surface et conjointement une baisse de la consommation.

1.1 Méthode globale

Lorsque les algorithmes de TDSI sont implantés sur des ASICs numériques, les données sont codées dans un format déterminé avec un nombre de bits fini. Cette contrainte nous génère plusieurs problèmes. D'une part les valeurs codées peuvent sortir du domaine de codage (overflow), d'autre part, l'utilisation de la précision finie (virgule fixe) entraîne un biais des résultats par rapport aux valeurs théoriques. Cette erreur génère un bruit de calcul qui se propage et s'accumule durant l'exécution de l'algorithme : elle sera caractérisée par une puissance de bruit de calcul et servira à conditionner le format de codage des données. D'autre part, elle sera intégrée comme une nouvelle contrainte (aspect traitement du signal) lors de la synthèse architecturale.

1.2 Démarche globale

Cette méthode part de la spécification algorithmique d'une application de traitement du signal, et suit trois

étapes. La première étape consiste à cadrer les coefficients utilisés dans le format considéré puis à trouver les facteurs d'échelle qui permettent d'éviter toute apparition d'overflow (réduction de la dynamique). La deuxième étape consiste à établir une expression analytique de la puissance de bruit de calcul en utilisant des modèles d'opérateurs (en terme de bruit). Ensuite à partir de la contrainte utilisateur, le format de données est optimisé. Lorsque le format des données est ainsi défini, l'utilisation d'une bibliothèque d'opérateurs adéquats permet, à l'aide de la synthèse architecturale, d'évaluer le coût de l'implantation.

1.3 Traitement de l'overflow

L'overflow est un phénomène qui apparaît quand une valeur codée sort du domaine de codage. Il est évident que ce phénomène doit être évité afin de s'assurer de la validité des résultats du traitement. Pour cela, nous proposons deux méthodes. Une méthode consiste à évaluer la dynamique des valeurs calculées, afin d'en déduire la plus grande valeur (Maj) pouvant exister au sein de l'algorithme. Afin de s'assurer que les valeurs resteront dans le domaine de codage, nous divisons toutes les valeurs traitées par la puissance (EchL) de deux immédiatement supérieure à Maj (décalage). EchL est le facteur d'échelle implicite associé à la sortie des calculs. L'autre méthode consiste à "éliminer" l'overflow dès qu'il est susceptible d'apparaître. Concrètement, nous diviserons par deux (décalage sur un bit) toutes les sommes effectuées par l'architecture. Cette méthode ne pourra s'appliquer qu'aux algorithmes ayant des graphes flots de données non récursifs.

1.4 Modélisation du bruit de calcul

Une fois l'overflow supprimé, nous déterminons la puissance de bruit de calculs induit par la limitation de la précision finie des formats des opérateurs. Pour cela, nous utilisons des modèles (en terme de bruit de calcul) des opérateurs utilisés [2]. Ces modèles ont la particularité de ne pas prendre en compte la valeur des données traitées, ce qui permet de ne pas faire intervenir l'aspect traitement de signal de l'algorithme. Ainsi, à l'aide de ces modèles nous déterminons une expression analytique de la puissance de bruit. Les modèles utilisés sont présentés dans la figure 1. D'autres modèles pourraient être considérés sans remettre en cause la démarche présentée.

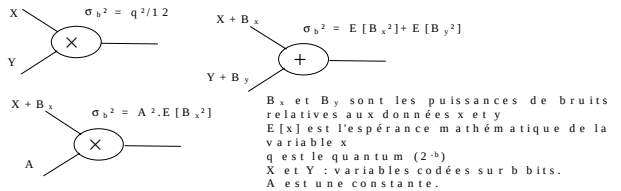


Figure 1 - Modèle de la troncature, de l'additionneur et du multiplieur

Si *q* est le quantum, la puissance de bruit de calcul peut s'écrire sous la forme $B = Aq^2$. Nous en déduisons le rapport signal à bruit de calcul dont les paramètres sont la puissance du signal de sortie non bruité et le nombre de bits codant les données. La contrainte est donnée par la formulation suivante où P est la puissance du signal de sortie non bruité.

$$SNR_{calcul} = 10\log\left(\frac{P}{Aq^2}\right)$$

La contrainte utilisateur donne l'inégalité suivante :

$$\text{contrainte} = SNR_{dB} > 10\log\left(\frac{P}{Aq^2}\right)$$

En fixant P = 1, nous obtenons l'inégalité suivante :

$$b > \frac{1}{2}\log_2\left(A \times 10^{\frac{SNR_{dB}}{10}}\right)$$

b sera donc la plus petite valeur entière vérifiant l'inégalité précédente.

1.5 Application à un outil de CAO : GAUT

Cette estimation peut se faire d'une manière automatique. L'outil de synthèse architecturale GAUT génère, à partir d'une description comportementale d'un algorithme, un graphe flot de données (GFD) décrivant le parallélisme obtenu après compilation. L'estimation de la puissance de bruit se fait à partir du graphe flot des bruits (déduit du GFD). A partir du bruit total des traitements, nous déterminons le format de codage des opérateurs satisfaisant la contrainte utilisateur (SNR bruit de calcul). Puis, à partir de la contrainte temps réel et à l'aide d'une bibliothèque d'opérateurs au format adéquat (temps de latence et surface spécifiés), la synthèse architecturale établit l'architecture de l'application.

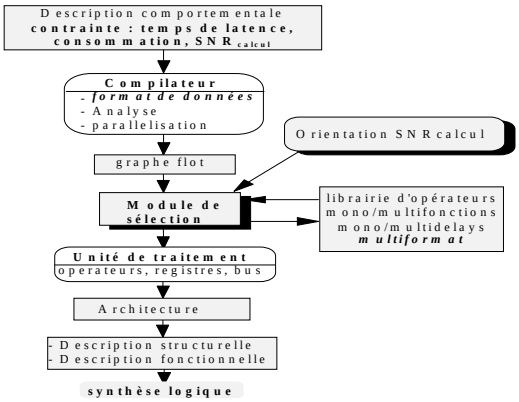


Figure 2 - Flot de conception de GAUT.

2. Applications

La méthode est illustrée par deux applications : un algorithme non récursif, la transformée de Fourier rapide (TFR), et un algorithme récursif, un filtre à réponse impulsionnelle infinie (RII) [3][4].

2.1 La FFT

2.1.1 Présentation

Au niveau calcul, la FFT est une répétition du papillon de Cooley-Tuckey. Si x(n) est un échantillon d'entrée, alors X(k), l'échantillon de sortie, est donnée par :

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

Ou 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

2.1.2 Overflow

Déterminons le facteur d'échelle *EchL*. Nous savons que

$$\left|X(k)\right| = \left|\sum_{n=0}^{N-1} x(n) W_N^{-nk}\right|$$

Nous pouvons majorer les échantillons x(n) par 1 :

$$\left|X(k)\right| < \sum_{n=0}^{N-1} \left|W_N^{-nk}\right| \quad ou \quad \left|W_N^{-nk}\right| = 1$$

$$=> \qquad Re_{x_k} < N \quad \text{et} \quad Re_{y_k} < N$$

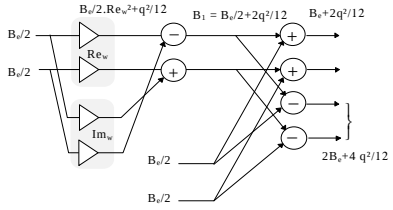
Nous pouvons affirmer que la plus grande valeur calculée est inférieure à N. Sachant que N est une puissance de deux, nous choisissons $EchL = N = 2^n$.

2.1.3 Bruit de calcul et format de données

Nous pouvons appliquer les deux méthodes de recadrage proposées (externe et interne).

2.1.3.1 Évaluation avec décalage externe

Le graphe flot des bruits est illustré par la figure 3.



- B_e est la puissance de bruit d'une donnée complexe en entrée.
- q est le quantum et $q = 2^{-b}$ où b est le nombre de bits.
- B_s est la puissance de bruit en sortie.

Figure 3 - Graphe flot des bruits

La puissance de bruit est donnée par : $B_S = B_e + 2\frac{q^2}{12}$

Les n étapes de la FFT étapes nous donnent :

$$B_n = B_{n-1} + 2\frac{q^2}{12}$$

$$\vdots \qquad \vdots$$

$$B_T = B_0 + 2n\frac{q^2}{12}$$

Le processus de décalage est appliqué en entrée ce qui génère un bruit.

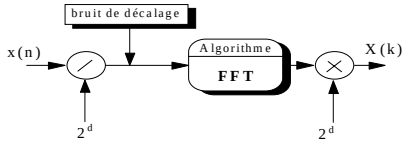


Figure 4 - Principe du décalage externe

Le bruit de décalage est donné par (avec d shifts) :

$$B_d = \frac{q^2}{2^{3d}} \sum_{i=1}^{2^d-1} i^2 = B_0$$

La puissance de bruit est donnée par :

$$B_T = \frac{q^2}{2^{3d}} \sum_{i=1}^{2^d-1} i^2 + 2n\frac{q^2}{12}$$

Si $d = 2^n$. Alors :

$$B_T = 2^{-3.2^n} q^2 \sum_{i=1}^{2^{2n}-1} i^2 + 2n\frac{q^2}{12}$$

La puissance de bruit en sortie est donnée par :

$$B_T = \left(2^{-3.2^n} \sum_{i=1}^{2^{2n}-1} i^2 + \frac{2n}{12}\right) q^2$$

Cette formulation dépend du nombre d'étapes de la FFT et du nombre de bits servant à coder les données.

2.1.3.2 Évaluation avec décalage interne

A présent nous effectuons un décalage en sortie de l'additionneur où l'overflow peut apparaître :

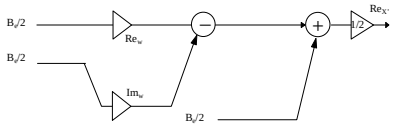


Figure 5 - graphe flot d'un demi papillon avec décalage interne

La puissance de bruit pour un décalage interne vaut :

$$B_S = \frac{B_e}{4} + 2\frac{q^2}{12}$$

Finalement la puissance de bruit de calcul vaut :

$$B_T = \frac{1}{4^n} B_0 + 2\frac{q^2}{12} \times \sum_{i=0}^{n-1} 4^{-i}$$

Dans notre cas nous choisirons $B_0 = 0$.

2.1.4 Résultats

Nous choisissons une FFT sur 128 points (N = 128). La contrainte temps imposée dans le fichier de description comportementale est de 57 microsecondes. A partir de la contrainte SNR_{calcul} spécifié par l'utilisateur, nous déterminons le nombre de bits nécessaire pour chaque cas étudiés. nous choisirons deux SNR_{calcul} : 60 dBs et 90 dBs. Ces différents algorithmes sont synthétisés par l'outil GAUT à l'aide des librairies appropriées. Les résultats sont réunis dans le tableau suivant :

	format flottant IEEE 754	virgule fixe			
		SNR_{calcul} = 60 dB		SNR_{calcul} = 90dB	
		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
add/shifter		1		1	
soust/shifter		1		1	
surface $mm^2.10^{-3}$	39318	2779	5073	7145	8254

Tableau 1 - Résultats après synthèse GAUT pour la FFT

Tout d'abord nous constatons que la surface de la FFT en flottant peut atteindre 14 fois la surface correspondant à l'implémentation en virgule fixe. Deuxièmement, nous remarquons que la surface varie suivant la contrainte SNR_{calcul} imposée. Finalement, à contrainte SNR_{calcul} fixée, nous observons que le coût de l'architecture est différent suivant le type de recadrage effectué (externe et interne).

2.2 Application à un filtre RII

A présent présentons l'application de la méthodologie à différentes structures d'un filtre récursif [3].

2.2.1 Présentation

La fonction de transfert d'un filtre peut s'écrire comme le rapport de deux polynômes. La décomposition (figure 4) du numérateur et du dénominateur nous amène à l'obtention de plusieurs formes qui donnent des structures différentes au niveau implémentation. Les propriétés algorithmiques restent les mêmes tandis que les caractéristiques au niveau implantation sont différentes pour chaque structure. Nous étudions trois de ces structures: forme canonique, forme cascade et parallèle. Voici la fonction de transfert du filtre et ses décompositions respectives.

$$H(z) = K \times \frac{\sum_{i=0}^{8} b_i z^{-i}}{1 + \sum_{i=1}^{8} a_i \times z^{-i}}$$

a_i et b_i sont les coefficients du filtre

$$G_1(z) = H_1(z).H_2(z)$$

$$G_2(z) = K.H_1(z).....H_k = K.\prod_{i=1}^{k} H_i(z)$$

$$G_3(z) = D + H_1(z) + ... + H_k = D + \sum_{i=1}^{k} H_i(z)$$

La figures 6 ci-dessous représentent l'algorithme sous forme canonique originale et après cadrage des données et traitement de l'overflow.

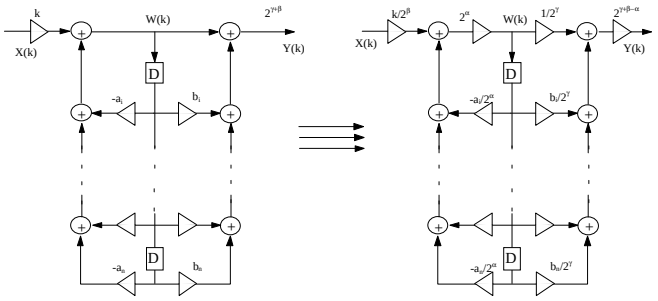


Figure 6 - Filtre avant et après cadrage et traitement de l'overflow

Ces structures sont implémentées de manière à conserver un rapport signal à bruit de calcul constant.

2.2.2 Résultats

La puissance de bruit de W_k et Y_k s'exprime de la manière suivante (filtre canonique) :

$$B_{W(k)} = 2^{2\alpha}\left[\left(\frac{k}{2^{\beta}}\right)^2 B_e + \frac{q^2}{12} + \sum_{i=1}^{2}\left(\frac{a_i^2}{2^{2\alpha}} B_{W(k-i)} + \frac{q^2}{12}\right)\right] + \frac{q^2}{12}$$

$$B_{Y(k)} = 2^{2\beta}\sum_{i=0}^{n} b_i^2 B_{W(k-i)} + \left(2^{2\gamma} + 1\right)\frac{q^2}{12}$$

Nous considérons un SNR de 90 dB. Les résultats sont présentés dans le tableau 2. Nous constatons que pour une même application (gabarit, fréquence d'échantillonnage), les trois structures mettent en oeuvre un nombre d'opérateurs différent travaillant sur un format de codage différent et aboutissent à des surfaces (coûts) différents. Le temps de latence est de 1µs, et la bibliothèque utilisée correspond à une technologie CMOS 1,2 µm.

	filtre sous forme		
	canonique	cascade	parallèle
nb de bits	20	18	22
multiplieur datapath	2	3	
multiplieur pipeline			1
additionneur	1	1	1
registre / mux / démux	9 / 3 / 3	8 / 5 / 5	5 / 5 / 2
surface $mm^2.10^{-3}$	6248	7057	4341

Tableau 2 - Résultats après synthèse GAUT pour les filtres RII

Nous constatons qu'après notre étude sur les formats de codage, le coût architectural total du filtre sous forme parallèle est inférieur aux autres structures. Ce résultat était prévisible du fait que le nombre de multiplieurs est moindre.

3. Conclusion

Cette méthodologie nous permet d'intégrer le paramètre format des chemins de données en imposant à l'application une contrainte "signal" au niveau de la synthèse. La complexité d'une architecture ne doit plus être estimée seulement en fonction du nombre d'opérateurs donnés par l'outil de synthèse mais bien selon le nombre d'opérateurs associés au format de codage. Nous avons pu constater dans cette étude que des structures ayant (environ) le même nombre d'opérateurs ne présentent pas le même coût total.

Les résultats concernant le traitement de l'overflow et l'évaluation du SNR_{calcul} sont validés par simulation. Afin d'améliorer cette méthodologie, nous pouvons étudier le problème d'une façon plus modulaire. En effet, nous pouvons scinder, soit l'algorithme soit l'architecture en modules codés sur différents formats. Cette approche est souvent mise en évidence au niveau des cellules multiplieur accumulateur. Cette approche a pour but d'améliorer l'optimisation de la surface sous contrainte SNR_{calcul}.

4. Références

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

[2] M. Bellanger, *Traitement numérique du signal*, Collection Technique et Scientifique des Télécommunications CNET-ENST, 1984.

[3] M. Kunt, *Traitement numérique des signaux*, Dunod, 1981.

[4] J-M. Tourreilles, C. Nouët, E. Martin, *Méthodologie d'implémentation d'algorithmes en précision finie pour un outil de synthèse architecturale*, Colloque CAO de circuits intégrés et systèmes, Villard-de-Lans, 1997.