Générateur de netlist de filtres numériques RIF optimisés

Ludovic Noury et Habib Mehrez
Laboratoire LIP6/ASIM
Université de Paris 6
4 Place Jussieu 75005 Paris France

E-mail: ludovic.noury@lip6.fr, habib.mehrez@lip6.fr

Résumé

L'une des opérations fondamentale du traitement du signal est le filtrage. On comprend donc l'intérêt d'un générateur de filtres numériques permettant d'obtenir simplement les filtres désirés en bénéficiant de tout le savoir faire arithmétique intégré dans le générateur, en particulier l'utilisation de plusieurs systèmes de numération. On ne s'intéresse qu'au cas des filtres à réponse impulsionnelle finie (RIF), les plus courants. La netlist du filtre est généré à partir de ses coefficients et de divers paramètres de précision. Le filtre généré n'est pas reconfigurable.

1 Introduction

Le traitement du signal a pour but d'extraire, de modifier, d'adapter un signal. L'une des opérations fondamentales est le filtrage et l'un des filtres les plus utilisés est le filtre RIF.

On comprend donc l'intérêt d'un générateur permettant d'obtenir à partir de la liste des coefficients définissant le filtre, facilement calculables à l'aide d'outils mathématiques tels que *Matlab*, une *netlist* en portes optimisée et ne plus avoir à concevoir pour chaque projet ou à chaque modification un nouveau filtre.

Intégrés dans des systèmes embarqués ces filtres se voient souvent confier une tâche unique, n'ayant pas besoin d'être reconfigurables. On décide donc afin de pouvoir intégrer le maximum d'optimisations de ne s'intéresser qu'à la génération de filtres non reconfigurables.

Enfin, on porte une attention particulière aux problèmes de précision induits par les approximations. Et aussi, à la limitation de la croissance interne afin de fournir le résultat avec la précision demandée sans le calcul de bits superflus.

2 Filtre RIF

À une séquence d'échantillons d'un signal d'entrée à temps discret $x(n)$, un filtre numérique, défini par sa réponse impulsionnelle $h(n)$, répond par une séquence d'échantillons d'un signal de sortie $y(n)$. La réponse impulsionnelle d'un filtre est définie par :

- w_p : fréquence de coupure,
- w_s : fin de la bande de transition,
- R_p : ondulation en bande passante (dB),
- A_s : ondulation hors bande passante (dB).

la réponse d'un filtre RIF est définie par l'équation suivante :

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i)$$

N est appelé longueur de la réponse impulsionnelle du filtre et correspond au nombre total de coefficients a_i. Les coefficients a_i sont obtenus par des algorithmes à partir des paramètres définissant la réponse impulsionnelle désirée précédemment cités.

3 Architecture

3.1 Structures

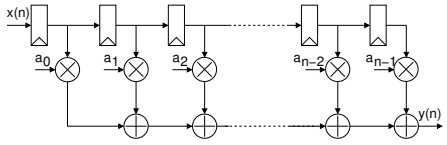

FIG. **1. Filtre RIF : structure directe**

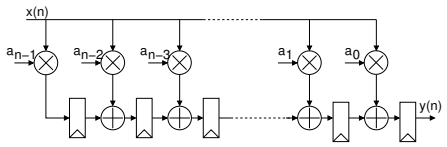

FIG. **2. Filtre RIF : structure transposée**

En matériel, deux structures sont communément utilisées. La structure directe (figure 1) obtenue par la transposition directe de l'équation et la structure transposée (figure 2) obtenue après manipulation de l'équation.

La structure indirecte a l'avantage d'avoir un chemin critique plus cours permettant généralement d'obtenir une meilleure fréquence de fonctionnement.

Mais elle présente l'inconvénient d'avoir des registres de taille croissante et un signal d'entrée distribué sur un grand nombre de multiplieurs (grande capacitance et routage complexe) donc une surface et une consommation supérieure.

On décide travailler sur la structure directe cette dernière se prêtant à plus d'optimisations et privilégiant surface et consommation faibles.

3.2 Multiplication par constante

Les filtres n'étant pas reprogrammables on décide d'utiliser des multiplieurs par constante éliminant au passage les registres de stockage de constante.

Dans un multiplieur par constante du fait qu'une des deux entrées est invariante la matrice de calcul des produits partiels est très simplifiée et l'opération se résume à l'addition de ces derniers. Augmenter les performances revient alors à diminuer le nombre de produits partiels.

Dans [1] décrit un multiplieur par constante basé l'algorithme de Booth [2] utilisé dans les multiplieurs classiques exploitant l'entrée soit fixe en utilisant des bases supérieures à 4 lors de l'encodage de la constante assurant au moins une division par deux du nombre de produits partiels. Ce multiplieur pousser l'idée encore plus loin en encodant la constante selon le même algorithme mais en base multiple. C'est la solution qui a été adoptée.

Après cette modification, le gain en surface du filtre est très important. Le gain en délai est quand à lui limité par le multiplieur par constante le plus lent, les multiplications se faisant en parallèle.

3.3 Sommation

On regroupe l'ensemble des additionneurs intermédiaires en une seule opération de sommation.

3.3.1 Arbres de Wallace Les arbres de Wallace[3] permettent d'effectuer rapidement la somme de n éléments et rendent un résultat en notation Carry-Save (redondante).

La construction de l'arbre est faite avec une variante améliorée de l'algorithme de Wallace, l'algorithme de Dadda [4]qui assure une complexité en $O(\log_{\frac{3}{2}}(n))$. La conversion en notation classique est effectué par avec un additionneur à deux termes, on utilise un CLA[1]

Les nombres que l'on a à sommer sont issus des multiplieurs c'est-à-dire de la somme de produits partiels suivit d'un CLA. On remarque donc que l'on peut aller encore plus loin dans l'optimisation en supprimant ces CLA, voir en supprimant aussi le matériel faisant la somme des produits partiels en agrandissant notre arbre.

3.3.2 Première version Dans une première version du générateur on dimensionne tous les opérateurs pour ne pas avoir de problème de dépassement de capacité. Cela se traduit par :

- Les multiplieurs par constante prennent t_x bits en entrée et rendent $t_x + t_{a_i} - 1$[2] bits en sortie.
- L'arbre doit faire la somme des N termes obtenus. Ce qui revient à un résultat sur $\max(t_x+t_{a_i}-1)+log_2N$ bits. Pour ne pas avoir de problèmes d'extension de signe sur le résultat on doit faire l'extension de signe sur $\max(t_x + t_{a_i} - 1) + log_2N$ de chaque terme. De plus l'arbre rendra alors un résultat sur $\max(t_x+t_{a_i}-1) + 2\log_2 N$ bits soit le calcul inutile de $\log_2 N$ bits qui seront toujours à 0.

Pour empêcher le calcul des bits inutiles on modifie l'arbre pour qu'il n'y ai pas de croissance de la taille.

[1] Carry lookup adder

[2] On retranche 1 pour ne pas compter le bit de signe deux fois.

3.3.3 Seconde version Nos calculs de la section précédente ne tiennent pas compte de l'application. A l'aide des coefficients des filtres et du nombre de bits sur lequel est codé l'entrée on peut calculer les minima et maximum possibles en sortie du filtre.

On obtient : $y_{max} = \sum_{a_i>0}(2^{t_x} - 1).a_i$ et $y_{min} = \sum_{a_i<0}(2^{t_x} - 1).a_i$. Ce qui permet de limiter la taille de la sortie à t_y le nombre de bits utilisé pour coder $\max(|y_min|, y_max)$ sans oublier d'ajouter un bit de signe.

On n'a alors à faire l'extension de signe seulement jusqu'à t_y bits. L'arbre de Wallace prenant alors N termes de t_y bits et rendant t_y bits.

3.3.4 Version finale Dans [5] Fadavi-Ardekani présente une méthode pour éviter de faire l'extension de signe des produits partiels dans un multiplieur. En étendant son algorithme au cas général on conclut que l'on peut éviter l'extension de signe en :

- inversant le bit de signe de chacun des termes à sommer,
- rajoutant une constante correspondant à la somme des extensions de signe que l'on effectuerait si le chaque terme était négatif (en incluant le bit de signe à 1).

Dans notre cas, on effectue néanmoins l'extension de signe du premier terme sur t_y bits. Pour s'assurer d'avoir tous les bits de signe en sortie. La sortie étant toujours sur t_y bits.

Si cette technique a l'inconvénient de rajouter un étage à l'arbre de Wallace de par l'ajout de la constante, l'économie en matériel qui en résulte justifie largement l'éventuelle augmentation du délai. Par exemple dans le cas d'un filtre passe-bas à 19 coefficients sur 16 bits et échantillons sur 10 bits on passe de 442 additionneurs complets et 8 demi-additionneurs à respectivement 387 et 15.

3.4 Gestion de l'arrondi

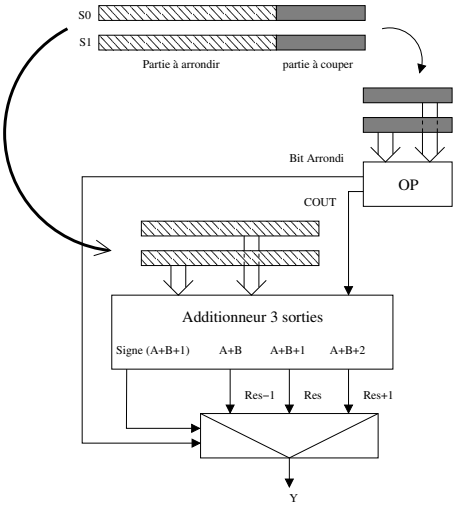

FIG. **3. Arrondi à l'aide de l'additionneur à sorties multiples**

La taille de la partie entière du résultat est déterminée par le générateur. Mais, la taille de la partie fractionnaire doit être précisé en argument du générateur de 0 à $t_a - 1$ bits.

Pour faire l'arrondi à partir du résultat en sortie on devrait regarder le bit b_a suivant la partie à garder r_g et le bit de signe b_s. Les cas suivants se présentent alors :

1. $b_s = 0, b_a = 0$: le résultat est r_g,
2. $b_s = 0, b_a = 1$: On doit arrondir à $r_g + 1$,
3. $b_s = 1, b_a = 0$: on garde r_g,
4. $b_s = 1, b_a = 1$: On doit arrondir à $r_g - 1$

On doit donc, selon la valeur de 2 bits rendre en sortie : $r_g + 1; r_g - 1; r_g$.

Or, dans [6] on décrit l'architecture d'un additionneur 3 sorties calculant à partir de deux entrée A et B : $A+B$, $A+B+1$, $A+B+2$ avec une complexité de l'ordre de celle d'un CLA mais surtout avec une surface à peine supérieur du CLA faisant juste $A + B$.

On remarque que l'on aurait pu modifier l'architecture de cet additionneur pour calculer $A+B-1$, $A+B$, $A+B+1$ mais qu'il a été plus simple de reprendre cette architecture et de se ramener au cas $A+B$, $A+B+1$, $A+B+2$ en s'arrangeant pour que l'arbre de Wallace calcule $r_g - 1$ et ce sans pénalité et profitant du fait que l'on ajoute déjà une constante et que l'on donc juste à modifier cette dernière.

Il ne reste plus qu'a connecter sur les sorties $S0$ et $S1$ de l'arbre de Wallace : sur les bits à enlever de quoi calculer la valeur de b_a et le bit de retenue, sur les bits à garder un additionneur trois sorties avec en cin le bit de retenue (en fait il n'y a pas de cin dans l'additionneur trois sorties utilisé, par manque de temps de modifier ce dernier on effectue la somme sur un bit de plus pour simuler le cin l'inconvénient et que l'on calcule un bit inutile, c'est la version représentée sur le schéma). Enfin, un multiplexeur sélectionne la sortie de l'additionneur à sortie multiple selon les valeurs de b_a et b_s. La solution obtenue est représentée sur la figure 3.

4 Validation et Résultats

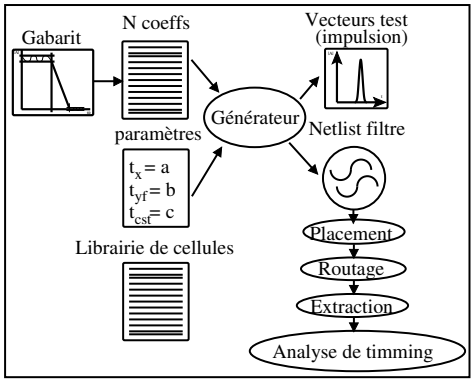

FIG. **4. Chaîne de génération**

La validation a été effectuée avec un ensemble de filtres typiques : passe-bas (avec $N = 51$ et $N = 19$), passe-bande (avec $N = 51$), passe-haut($N = 67$) et stop-bande($N = 47$) avec à chaque fois une ondulation en bande passante très faible.

Tous les filtres ont étés générés pour des échantillons sur 10 bits et l'encodage des constantes sur 16 bits. Le générateur fournit avec chaque netlist un fichier contenant les vecteurs de tests utilisés lors de la validation.

Chaque filtre a été simulé, placé et routé à l'aide de la chaîne *Alliance*[7], en technologie $0,35\mu$ et avec la librairie de cellules pré-caractérisées *sxlib* (voir figure 4). Les résultats de la simulation avec en entrée une impulsion de Dirac ont étés analysés en comparaison avec le filtre idéal correspondant et les différences maximales d'ondulation on étés notés (voir figure 5 et tableau 1).

Filtre	Fréq. (Mhz)	Transistors	Différence d'ondulation (dB)	
			hors BP[3]	en BP
Passe-bas 1	37	140701	10^{-3}	4
Passe-bas 2	44,6	56586	10^{-3}	1,4
Passe-bande	38,44	136555	10^{-2}	0,3-3,1
Passe-haut	37,93	178898	10^{-3}	7
Stop-bande	39,84	129462	10^{-3}-10^{-2}	1

TAB. **1. Résultats obtenus avec mécanisme d'arrondisseur final désactivé (BP = bande passante)**

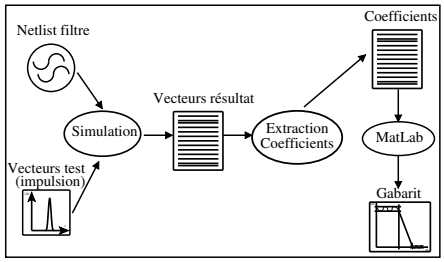

FIG. **5. Méthodologie de validation des filtres**

5 Conclusion

L'analyse des résultats montre que les filtres générés respectent les gabarits définis en entrée tout en apportant une fréquence de fonctionnement et une surface fortement améliorées comparativement à une version implantant simplement l'architecture de base.

Références

[1] Y. Dumonteix. Optimisation des chemins de données arithmétique par l'utilisation de plusieurs systèmes de numération. *Thèse de l'université Pierre et Marie Curie*, Octobre 2001.

[2] A.D. Booth A signed binary multiplication technique. *Quartely J. Mechanics and Applied Mathematics*, 4(2) : 236-240, 1951

[3] C.S. Wallace A suggestion for a fast multiplier *Prc. Techno*, 90,12(11) :14-17, 1964

[4] L. Dadda Some schemes for parallel multipliers *Alta Frequenza*, 45 :574-580, 1976

[5] Jalil Fadavi-Ardekani. MxN Booth encoded multiplier generator using optimized Wallace trees. *IEEE Transaction on very large integration (VLSI) Systems*, vol. 1., no. 2, 1993.

[6] R. Avot. Architectures matérielles pour l'arithmétique stochastique discrète. *Thèse de l'université Pierre et Marie Curie*, juin 2003.

[7] A. Greiner and F. Pecheux. A complet set of CAD Tools for teaching VLSI Design. *Third EuroChip Workshop on VLSI Design Trainning*, september 1992.