Description, évaluation, et utilisation d'un générateur d'adresses général adapté à la FFT.

J. Ph. HALLAY * °, Ph. ELLEAUME *
C.A. WAMBERGUE *, F. DEVOS °, R. REYNAUD °

* THOMSON-CSF SDC, 18 av. du Maréchal Juin 92363 MEUDON-LA-FORET CEDEX
° INSTITUT D'ELECTRONIQUE FONDAMENTALE, Faculté d'Orsay Paris-XI 91405 ORSAY CEDEX

RESUME

Une bonne solution économique aujourd'hui pour construire des machines SIMD de traitement de signal est, d'une part, de concevoir soi-même un processeur élémentaire optimisé pour les applications visées et, d'autre part, d'utiliser les circuits les plus adaptés du commerce pour réaliser tout ce qui concerne le séquencement et la génération d'adresses. On tend ainsi vers des puissances de calcul telles qu'un processeur élémentaire effectue un papillon FFT radix 2 complexe en 100 ns [5]. Se pose alors le problème de générer trois adresses non triviales pendant ce laps de temps, avec un système qui soit également performant pour les autres algorithmes traités.

On montre d'abord que les générateurs d'adresses généraux et performants du commerce ne conviennent pas tels quels, à cause de la longueur du talon servant à leur initialisation pendant lequel l'unité de traitement reste inactive.

On propose alors une structure simple à base d'un générateur d'adresses et d'un système original de post-génération qui permet d'approcher de très près la puissance de traitement maximale théorique. Cette solution entraîne de plus une réduction importante de la taille des programmes et une simplification de leur écriture, en éliminant les cas d'exception ce qui facilite l'inclusion dans un compilateur de traitement de signal. Cette structure conserve non seulement toutes les fonctionnalités du générateur d'adresses, mais se transpose aussi à des algorithmes autres que la FFT, pour lesquels une solution classique conviendrait mal.

SUMMARY

A fairly economical solution today to build SIMD signal processing machines is to design an elementary processor optimised for the desired application, and then to use off-the-shelf components to build all that concerns the sequencing (time control,address generation...). Thus one can get processing powers such that an elementary processor can achieve a complex radix 2 FFT butterfly in 100 ns. Then appears the problem of generating three non trivial addresses during this period of time, with a system which should also be performant for other DSP algorithms.

It is shown that the use of performant off-the-shelf general purpose address generators is not a good solution because they imply a rather long initialization requirement.

A simple post-generation system to be added to an address generator is then proposed to allow intensive use of the operating units. The exhibited solution supplies also an important shrinking of the program's length and a non negligible programming simplification that would ease the inclusion in a signal processing compiler. This structure however is not FFT algorithm specific: it can be used in a general purpose signal processing machine because, not only all of the address generator's features can be used, but all of the above results can be transposed to algorithms other than FFT, for which a classical solution would not suit. A parallel use of the described structure, that leads to more regular program coding, is then presented.

I. Introduction

Les concepteurs de machines de traitement du signal à forte puissance de calcul ont la possibilité de personnaliser l'unité de traitement de leur machine pour que celle-ci réponde de façon optimale aux problèmes posés, et de l'utiliser avec un séquenceur de programme et des générateurs d'adresses intégrés du commerce. Ces derniers en effet sont de véritables unités autonomes prévues pour résoudre une vaste gamme de problèmes.

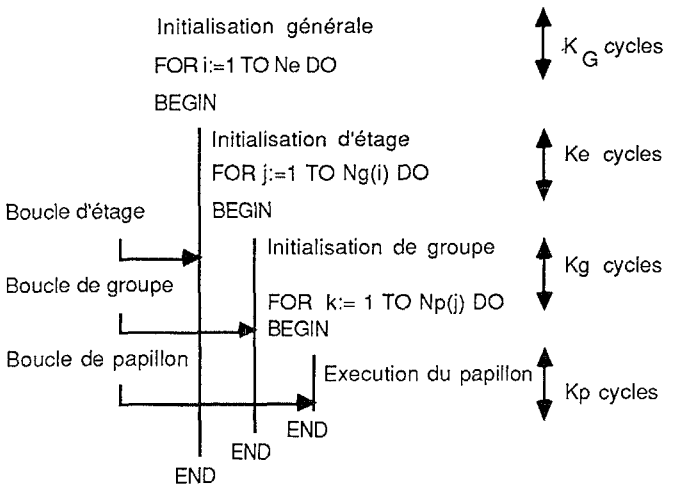
Cependant, par leur conception à vocation "générale", ils doivent être initialisés avant tout calcul, certains algorithmes nécessitant même plusieurs initialisations au sein du même traitement. Ces initialisations se font à l'aide d'instructions spécifiques ne produisant aucun traitement effectif sur les données; elles peuvent donc, dans certains cas, alourdir sensiblement le temps total d'exécution et diminuer ainsi la puissance de calcul de la machine par rapport à celle de l'unité de traitement, ce qui réduit en partie les gains d'optimisation du concepteur.

En nous appuyant sur l'algorithme courant qu'est la transformée de Fourier rapide radix 2 à décimation en temps (Fast Fourier Transform: FFT [11]) illustré figure 1, nous allons dans cette communication évaluer cette perte et présenter un dispositif général de post-génération, situé en sortie du générateur d'adresses (GA), qui en utilisant le surplus de bits fourni par le GA comparé aux besoins de la mémoire utilisée, permet de combler les faiblesses des circuits standards sans pour autant se priver des nombreuses fonctionnalités qu'ils fournissent.

II. Influence des organes de séquencement sur le rythme d'utilisation de l'unité de traitement (UT)

Celle-ci va être mesurée à partir du temps d'exécution d'un programme courant à trois boucles implanté sur une machine générique non spécialisée pour la FFT. On verra que les deux boucles intérieures déterminent la quasi totalité de ce temps, ce qui permettra d'étendre les résultats obtenus aux programmes basés sur une structure à deux boucles. Les programmes en ligne ou à une boucle sont eux bien moins intéressants car ils entraînent un accroissement important de la taille du code sans

rapport avec le gain en temps d'exécution. Leur cas ne sera donc pas traité explicitement ici.



La structure ci-dessus est purement formelle: on a regroupé au début de chaque boucle tous les cycles d'initialisation dont seul le nombre ici nous intéresse.

Ce programme exécutera une FFT de taille 2^{Ne} en un temps T (exprimé en nombre de cycles) valant:

$$T=K_G+ Ne \times (Ke +2^{Ne-1} \times Kp) + Kg \times (2^{Ne}-1)$$

Cette formule dépend des différents paramètres K_G, Ke, Kp, Kg dont la valeur de chacun traduit l'adaptabilité d'une ou plusieurs parties de la machine à l'algorithme de FFT. En effet:

--> Ke, K_G et Kg représentent des temps consacrés à initialiser les GA et le séquenceur de programme. Kg peut en plus contenir des cycles consacrés à remplir ou à vider des niveaux de pipe avant ou après le calcul des papillons et peut donc également dépendre de l'UT. Nous considérerons ici que ces niveaux, s'ils existent, sont invisibles à l'utilisateur; nous n'avons donc pas à les considérer. Cette hypothèse est licite car nous nous plaçons dans le cas d'une UT optimisée pour quelques traitements bien définis, qui dispose donc d'instructions adaptées pour ceux-ci.

--> Kp dépend seulement de l'architecture de l'UT puisqu'il représente le temps mis par celle-ci pour effectuer un papillon.

T est un temps d'exécution observé. Il peut être comparé au le temps d'exécution qui serait obtenu si l'UT était utilisée à 100% :

$$T_{ut}= Ne \times 2^{Ne-1} \times Kp$$

(remarquons que T_{ut} ne considère pas seulement les opérateurs de calcul que contient l'UT mais tient également compte de son architecture.)

On peut alors, à partir de T_{ut} et de T, définir un coefficient d'utilisation de l'UT par ß = Tut / T, dont l'évolution en fonction de Ne à Kg et Kp constants, est donnée figure 2.

Les deux courbes montrent que les performances de la machine sont bridées d'autant plus que l'ordre de la transformée est faible, et que Kg < Kp.

Kg = Kp correspond au cas d'un processeur de signal monolithique courant [7] alors que Kg = 4 x Kp décrit la situation où nous nous plaçons: celle d'une UT performante pour cet algorithme associée à des éléments standards.

III. Intérêt d'une PROM de transcodage d'adresses.

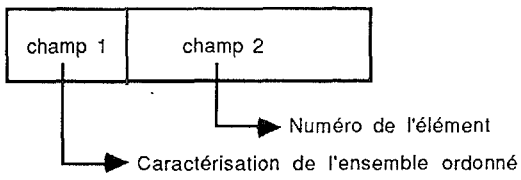
Une amélioration de ß à T_{ut} constant, sera obtenue en diminuant les valeurs de K_G , Ke et surtout Kg. Ceci revient :

-à réduire le nombre de boucles différentes ce qui réduira le temps consacré à la programmation du séquenceur de programme.

-à générer une séquence d'adresses régulière nécessitant une programmation simple du GA.

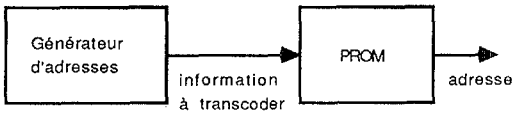
Ces deux objectifs peuvent être atteints en désignant une donnée à traiter non plus par la valeur de l'adresse de la case mémoire où elle se trouve, mais par le rang de celle-ci dans un ensemble convenablement ordonné. La structure séquentielle des machines auxquelles nous nous intéressons fait que le choix de la relation d'ordre que constitue la position temporelle de l'occurrence de chacun des éléments du référentiel lors du déroulement de l'algorithme s'impose pour la simplicité qui en résulte. On peut en effet, dans le cas d'une exécution normale, passer d'un élément au suivant par une simple incrémentation, cela indépendamment de la valeur de celui-ci.

Quelle que soit la relation d'ordre choisie, l'information en sortie du GA est composée de deux champs de la forme :



- figure 3.1 -

Le transcodage "rang de l'adresse -->adresse " se fait en ajoutant une PROM en sortie du GA.



- figure 3.2 -

Pour une application donnée, on peut construire plusieurs types de mots sur le modèle présenté ci-dessus. Ils se distinguent par:

- le référentiel dans lequel on se place. Il a deux caractéristiques importantes, et interdépendantes qui sont: - son cardinal qui influe sur la taille du mot par le nombre de bits nécessaires pour spécifier un élément - et le domaine de l'application qu'il recouvre dont la taille du mot dépend par le nombre total d'ensembles à spécifier pour traiter le problème complètement. On peut trouver ici, en exploitant des propriétés propres à l'algorithme, une possibilité d'ajuster la largeur du mot au nombre de bits d'entrée de la PROM.

- la nature des entités que l'on désigne à l'intérieur du référentiel choisi: simples éléments, couples... Dans le cas où l'on spécifie un groupe d'éléments, un compteur généré en local en entrée de la PROM viendra différencier les adresses du sous-ensemble désigné. Par cette approche un GA unique peut être à l'origine de plusieurs adresses dans un cycle.

IV. Evaluation de la solution proposée sur la FFT. Compromis largeur du mot d'entrée de la PROM - temps d'exécution.

Nous allons introduire une PROM de transcodage sur le GA consacré aux adresses des données.

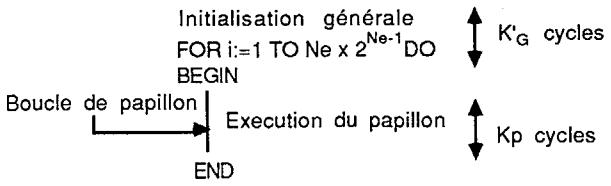
Dans le cas où la machine exécute un papillon en un cycle il est intéressant de grouper les adresses en couples, et on spécifiera alors un numéro de papillon, dans les autres cas on désignera les adresses une à une.

Quelle que soit la solution retenue, on aura le choix entre deux référentiels qui donneront des résultats légèrement différents:

- on peut choisir l'ensemble de toutes les adresses générées lors d'une FFT complète. Spécifier le référentiel revient à ici à indiquer le nombre de points de la transformée.

Le nombre total de bits nécessaires pour effectuer un ensemble de FFT comportant 2^L tailles différentes dont la plus importante

est d'ordre 2^M , est égal à $M + L + \log_2(M)$ (alors que la méthode traditionnelle ne nécessite que M bits d'adresse).
Le programme correspondant pour une FFT d'ordre 2^{Ne} devient:



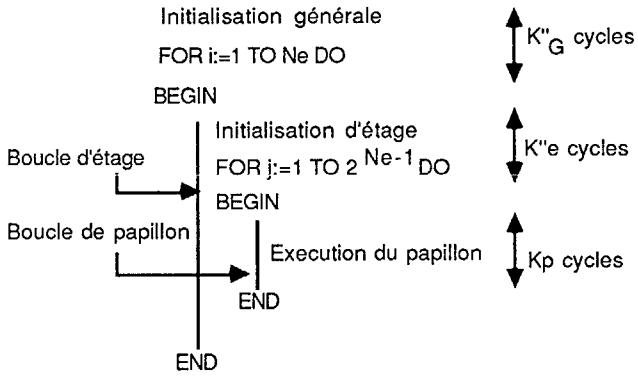
On constate l'extrême simplification de sa structure et sa faible occupation mémoire. Son temps d'exécution vaut, en cycles:

$$T' = K'_G + T_{ut}$$

avec K'_G pouvant varier de 5 à 20 suivant les machines.

- on peut également choisir comme référentiel l'ensemble des adresses générées lors d'un étage, ceci indépendamment du nombre de points grâce à la structure particulière des enchaînements propres à l'algorithme à décimation en temps. On désigne alors le référentiel par le numéro de l'étage en cours.

Le nombre de bits nécessaires pour effectuer l'ensemble des transformées précédentes devient $M + \log_2(M)$.
Le programme correspondant est, toujours pour une FFT d'ordre 2^{Ne} :



Il est exécuté en un nombre de cycles valant:

$$T'' = K''_G + Ne \times K''e + Tut$$

où K''_G est du même ordre de grandeur que K'_G alors que K"e peut être estimé à trois cycles.

Cette dernière solution comparée à la précédente est plus avantageuse en nombre de bits d'adresses, mais a par contre un code programme et un temps d'exécution qui sont plus longs.

On a représenté sur les figures 3 et 4 les coefficients d'utilisation de l'UT découlant de ces deux solutions, en fonction de Ne , et pour différentes valeurs de K'_G et K''_G . Il apparaît que la première solution est plus avantageuse pour les transformées portant sur un nombre de points peu important: elle y donne une meilleure utilisation de l'UT, alors que le nombre total de bits d'adresses n'est pas trop important. La seconde solution est, elle, plus intéressante pour les FFT d'ordre plus élevé, car elle permet de gagner des bits d'adresses pour une utilisation de l'UT équivalente à celle observée avec la première solution.

Il existe des enchaînements autres que celui exposé figure 1, qui permettent également de réaliser une FFT à l'aide du papillon radix 2 à décimation en temps. L'exploitation de leurs propriétés spécifiques dans le choix du référentiel et des groupes d'éléments que l'on spécifie peut s'avérer plus intéressant pour certaines machines.

On trouve parmi ceux-ci les algorithmes à constante géométrique [3] pour lesquels la séquence d'adresse de lecture et d'écriture de toutes les étapes sont identiques, ce qui permet de ne pas différencier les étapes, pour une taille donnée.

V. Extension de l'utilisation de la PROM.

Il est possible d'étendre l'utilisation de la PROM dans deux directions différentes: sa généralisation à d'autres algorithmes d'une part, et son introduction dans la génération de bits d'instructions d'autre part.

La solution fournie en effet n'est pas spécifique à l'algorithme FFT qui a été un support d'introduction pour lequel nous avons montré que plusieurs approches étaient envisageables. Elle peut être appliquée chaque fois que les générateurs d'adresses standards sont à l'origine d'un alourdissement excessif du temps de calcul.

V.1. Utilisation d'adressages plus larges pour disposer de plusieurs transcodages

Un intérêt de la méthode proposée est que plusieurs modes de transcodage peuvent cohabiter sur une même machine, chacun étant adapté à une classe d'algorithmes: il suffit de disposer d'un nombre de bits suffisant en sortie du GA pour pouvoir décrire complètement les informations exposées figure 3.1

Cela permet en particulier, et nous touchons là une autre particularité intéressante, d'associer à plusieurs transcodages spécifiques le transcodage identité qui rend, au niveau du second champ de la figure 3.1, la PROM invisible à l'utilisateur. Celui-ci peut alors utiliser pleinement toutes les fonctionnalités du générateur d'adresse pour ce champ.

Les améliorations citées plus haut ont été obtenues grâce à l'adjonction d'un organe porteur d'informations a-priori, relatives à la structure temporelle de l'algorithme du point de vue de l'adressage: la PROM. Le rôle du programmeur n'est alors plus de décrire l'enchaînement successif des adresses à l'aide d'un GA non spécialisé, c'est à dire qui ne contient d'emblée aucune indication relative à l'algorithme effectué. Il doit donner seulement un complément d'information à celle contenue dans la PROM: le rang, qui est simple à exprimer.

Dans la pratique, la réunion de ces deux types d'informations, le contenu de la PROM et l'ordre dans lequel elle est lue, donne des renseignements complets sur l'algorithme traité et son état d'avancement qui peuvent être redondants avec certaines indications données au niveau de l'écriture des instructions.

V.2. Utilisation de mots de données plus larges dans l'optique de générer des bits de microcode.

Dans un programme FFT, par exemple, on est dans certains cas amené à distinguer les papillons de la première étape dont les entrées proviennent de l'extérieur du processeur, à ceux des autres étapes pour lesquels les données se situent dans une mémoire locale. A cette distinction, correspondent deux parties distinctes dans le programme, car les papillons de la première étape sont réalisés à l'aide instructions légèrement différentes de celles utilisées pour ceux des étapes suivantes. Or le numéro de l'étape est également connu au niveau de la PROM. Il est donc possible d'écrire un programme d'un seul bloc en réservant un bit de sortie de la PROM pour être interprété comme bit d'instruction indiquant ici la provenance des données: il suffit d'avoir prévu le champ adéquat lors de la définition du codage des instructions, ce qui n'est pas une contrainte supplémentaire, car l'introduction de champs dans les instructions permet de simplifier leur décodage.

Plus généralement, il est possible de profiter des connaissances situées au niveau de la PROM en reliant certains bits de celle-ci à un ou plusieurs champs d'instruction. On peut, de cette manière, simplifier l'écriture des programmes en éliminant, entre autres, les effets de bord propres à certains algorithmes ou à certaines configurations matérielles, les problèmes liés à la parité du nombre de données à traiter, ce qui réduit les temps et les coûts de mise au point d'applications.

Il apparaît ici un second compromis, comparable au précédent (§ IV), entre le nombre d'instructions nécessaires pour décrire un algorithme et la largeur de celles-ci.

VI.Conclusion.

Une PROM de transcodage constitue un système de post-génération qui, adjoint à un générateur d'adresses général du commerce, permet d'utiliser celui-ci efficacement dans les

algorithmes pour lesquels il était mal adapté. Il est à la fois plus simple et moins coûteux que les solutions où l'on développe des éléments spécialisés. Les avantages obtenus sont de quatre ordres:

1)Gain en puissance de calcul obtenu par l'utilisation maximale de l'unité de traitement.

2) Simplification de l'utilisation par une programmation de haut niveau.

3) Gain en taille de code (dans les meilleurs cas une FFT sur 2^{Ne} points est programmée en moins de dix instructions).

4) Réduction éventuelle de la complexité et du coût du matériel par une diminution du nombre total de générateurs d'adresses.

De plus ce système peut être rendu transparent sur un certain champ, ce qui permet de disposer sur ce champ de toutes les fonctionnalités offertes par le générateur d'adresses.

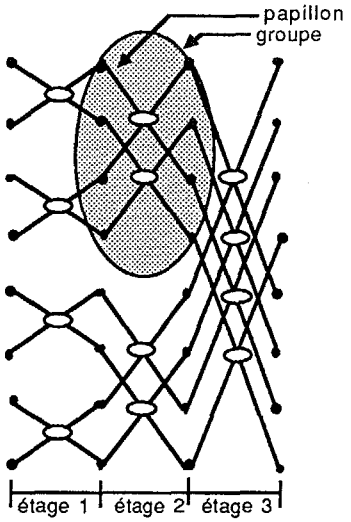
On assiste en contrepartie à une augmentation du nombre de bits fournis par le générateur d'adresses, par rapport à une solution directe sans PROM. Cet accroissement peut être limité grâce à la flexibilité de la solution:

*au niveau du matériel, par le choix entre un transcodage total ou partiel du bus adresse, et par la possibilité d'ajouter un compteur à l'entrée de la PROM;

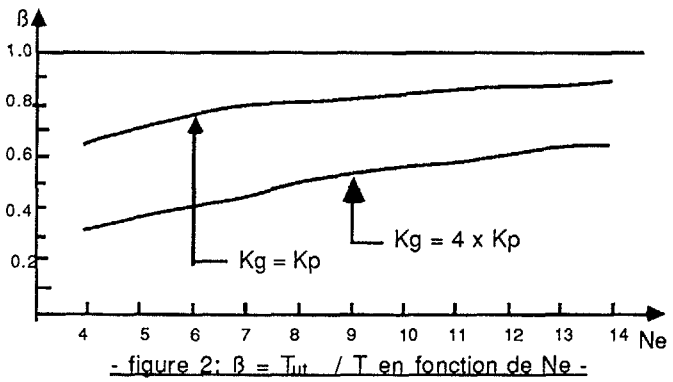
*au niveau de la programmation de la PROM où plusieurs transcodages sont possibles pour un algorithme donné.

L'intégration de la solution proposée ne peut se faire telle quelle car la PROM est trop volumineuse. On peut par contre, si le transcodage après une passe de réduction n'est pas trop compliqué, envisager d'utiliser une logique à base de PLA. Le problème de l'intégration d'un GA spécifique FFT a, quant à lui, été abordé dans [4].

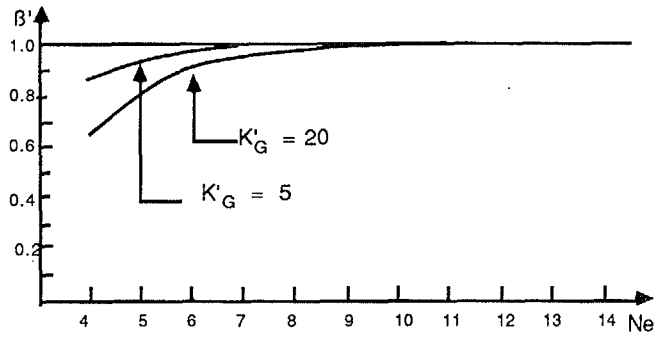
Nous donnons sur la figure 5 un exemple d'implantation pratique, où sont rassemblées les diverses possibilités matérielles qu'offre la solution exposée.



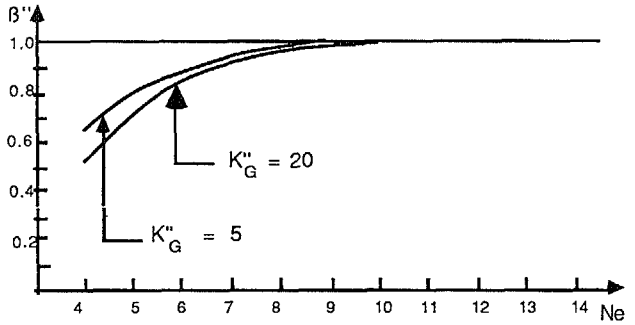
- figure 1-



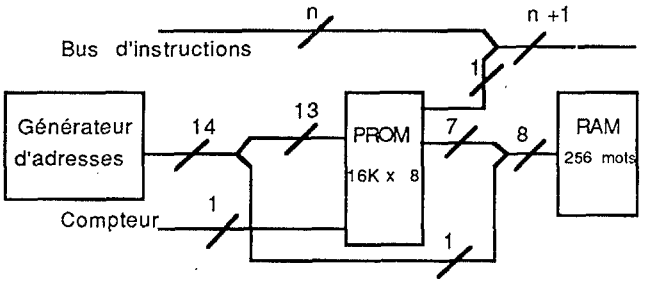
- figure 2: ß = T_{ut} / T en fonction de Ne -



- figure 3: ß' = T_{ut} / T' en fonction de Ne -



- figure 4: ß'' = T_{ut} / T'' en fonction de Ne -



- figure 5 : exemple d'implantation pratique -

Bibliographie

[1] M. BELLANGER "*Traitement numérique du signal*" Masson, 1984

[2] E. MARTIN "*Les architectures de traitement de signal à la demande. Une approche de conception automatisable*". Thèse 3e cycle, Université Paris 11, 1986

[3] K. LAMB "*Cmos building blocks shrink and speed up FFT systems*" Electronic Design .August 6, 1987 pp.101-106.

[4] R. REYNAUD, E. MARTIN, F. DEVOS "*Design of a bit-sliced address generator for FFT algorithms*" Proceedings of EUSIPCO-88 vol. 1, pp.335-338

[5] D. BURSKI "*One chip FFT processor risp through signal data*" Electronic Design. September 8, 1988 pp. 127-129.

[6] E. MARTIN, R. REYNAUD, P. ELLEAUME, C.A. WAMBERGUE, F. DEVOS "*Une architecture optimisée de processeur de filtrage numérique intégré: évaluation graphique de son rendement* " Traitement du signal, vol 1 n°1, 1986.

[7] ANALOG DEVICE "*ADSP 2100 application handbook*", vol.1

[8] J.HINDIN "*Digital signal processing moves into high gear*" Computer Design October 15, 1984 pp.61-74