UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE DE LA MAÎTRISE
EN GÉNIE ÉLECTRIQUE

PAR
KAMAL SAKKAY

CONCEPTION D'UNE ARCHITECTURE PARALLÈLE POUR DES ALGORITHMES BASÉS SUR LE FILTRE DE KALMAN DÉDIÉ À LA CORRECTION DE SIGNAUX SPECTROMÉTRIQUES

Août 1998

Résumé

Ce travail vise principalement à apporter une contribution pour l'amélioration des systèmes de mesure par le biais d'une proposition d'architectures en technologie VLSI. Ces architectures seront complètement, mais pas exclusivement, dédiées à la mise en oeuvre d'algorithmes sophistiqués pour la reconstitution de mesurande et plus spécifiquement pour la correction des données spectrométriques.

Ce mémoire présente l'étude et le développement d'architectures spécifiques pour l'implantation d'une classe d'algorithmes basés sur le filtre de Kalman (KALSPL) et l'algorithme des moindres carrés moyens (LMS). La complexité des algorithmes de reconstitution de mesurande implique une demande particulière pour les circuits à usage spécifique 'ASIC' pour améliorer les performances ou rencontrer les exigences des applications en traitement de signal.

Une bonne connaissance de l'algorithme à implanter permet de faire une bonne conception d'architecture qui lui sera dédiée. Notre approche est basée sur l'analyse des algorithmes visés et l'étude d'architectures parallèles susceptibles pour leur mise en oeuvre. Cette analyse nous a permit d'introduire des transformations sur la structure des deux algorithmes pour les adapter à l'implantation en technologie VLSI. En effet, l'étude algorithmique a permis de bien comprendre les algorithmes à implanter et d'établir leur flot

de données en vue d'une implantation d'une architecture dédiée à la correction de données de mesure spectrométrique utilisant les filtres de Kalman-Spline et LMS.

Grâce à la nature récursive des deux algorithmes, les architectures systoliques sont très bien adaptées pour leur implantation. Toutes les opérations nécessaires dans les deux algorithmes sont efficacement réparties sur l'ensemble des PEs (Processeurs Élémentaires) du réseau systolique linéaire avec des contraintes E/S (ENTRÉE-SORTIE) très réalistes. La simplicité et la régularité d'une telle architecture en font une attraction pour l'implantation en technologie VLSI. Ainsi, les transformations introduites nous ont permis de réaliser une bonne conception d'architecture systolique dédiée à l'intégration du filtre de KALSPL et LMS. Les étapes de conception de l'architecture et du design sont explicitement adressées.

Une transformation dans la formulation des équations décrivant les algorithmes a permit de réaliser des réductions substantielles dans le temps d'exécution de l'architecture. Le défi dans ce projet est de réussir à implanter les deux algorithmes sur un même circuit sans introduire des changement radicaux sur l'architecture de base. Des données synthétiques sont utilisées pour valider la conception et pour évaluer les performances par rapport au temps de calcul et la précision comparativement au DSP 56000. On montre par un exemple que le nombre de cycles exigés par notre architecture est 100 fois moins que celui du DSP. L'architecture est modélisée en utilisant les outils d'aide à la conception et les langages de description du matériel, les résultats de synthèse de l'architecture donnent une bonne idée sur les coûts et les performances d'une telle architecture.

La principale limitation de l'architecture systolique est son manque de flexibilité à implanter d'autres algorithmes de la même classe. Pour rendre le processeur flexible à la mise en œuvre d'autres algorithmes de reconstitution de mesurande nous proposons au chapitre 4 une architecture basée sur l'approche VLIW. Bien que l'idée fondamentale et les techniques des machines VLIW soient très simple, leur conception continue à poser des problèmes divers au niveau de la modélisation d'un fichier de registre complètement connecté, la répartition et la gestion efficace des ressources et même la grandeur de la mémoire programme. Au chapitre 4 nous discutons des étapes de conception de ce type d'architecture et nous développons une structure hautement parallèle dédiée à la mise en œuvre d'algorithmes spécialisés pour la reconstitution de mesurande. Nous proposons une architecture VLIW avec des connexions limitées rendant l'utilisation de processeurs basés sur ce type d'architecture une réalité courante.

Remerciements

Je voudrais remercier mes deux directeurs, Daniel Massicotte et Andrzej Barwicz, professeurs au département de génie électrique à l'Université du Québec à Trois-Rivières (UQTR), d'avoir accepter de diriger ce travail et participer activement dans toutes les phases de développement du sujet. Je les remercie également pour leur soutien financier, intellectuel et moral.

Je remercie également Dr. Mohamed Ben Slima, chercheur post-doctoral au laboratoire des systèmes de mesure pour sa précieuse aide, son sérieux et son soutien.

Je tiens à remercier aussi Monsieur Ahmed Chériti, professeur au département de génie électrique à l'UQTR pour son aide et son soutien.. Mes remerciements vont également à tous mes collègues au laboratoire des micro-systèmes et du laboratoire des systèmes de mesure.

Je voudrais aussi exprimer ma reconnaissance vers les organismes qui ont aidé au développement de ce travail soit, la société canadienne de micro-électronique (CMC), et le CRSNG, pour leur support financier, matériel et même intellectuel.

Ma sincère gratitude va à Koudoushia Burtally, (tiKOKO), de m'avoir soutenu durant toute la phase de rédaction, correction et même développement de ce mémoire en faisant preuve de courage, compassion et patience.

Finalement, je désire exprimer ma plus profonde gratitude à mes parents qui ont tout fait pour me permettre d'atteindre mes objectifs de carrière; mon père qui a toujours travaillé pour le bonheur de toute la famille, et ma mère qui m'a toujours soutenu sur tous les plans. Un gros merci à Haj Mokhtar et Hajja Zuhra car sans vous jamais je n'aurais atteint un tel grade. Ma sincère reconnaissance va également à mes sœurs, Fatima, Naima, et leurs enfants : Soufian, Mounir, Karim, Khadija, Ilhame, Asmaa et Soumaia. Mes frères, Abder-rahim, Mohamed, Adil, et leurs enfants : Khaoula, Abdel-mouniîm, Asaâd et Marwane sont aussi remerciés. Je témoigne aussi de la gratitude à tous mes oncles, tantes, cousines, cousins et amis pour l'amour, support et compréhension.

Table des matières

Liste des figures

page

Liste des tableaux

page

Liste des Symboles

â	amplitude des pics dans un spectrogramme;
A/A	conversion analogique-analogique;
A/N	conversion analogique-numérique;
ACFAS	association canadienne française pour l'avancement des sciences;
AGU	unité de génération d' adresse;
ASIC	application specific integrated circuit;
b	vecteur de commande dans la représentation d'état;
CAD	Conception assistée par ordinateur;
CAO	Conception assistée par ordinateur;
CCGEI98	congrès canadien en génie électrique et informatique 1998;
CBN	nombre de bits de calcul;
CMC	canadian microelectronics corporation, société canadienne de microélectronique;
CMOS	complementary metal oxide semi-conductor;
CP	coût d'une puce;
CT	coût d'une tranche de silicium;
DBN	nombre de bits de données;

d(.)	détection de la dérivée;
D_d	latence dans le réseau systolique;
DB	bloc de décision;
dim(.)	dimension du vecteur ou de matrice;
DMX	démultiplexeur;
DSP	processeur ou algorithme de traitement de signal ou (digital signal processors or processing);
E	espérance mathématique;
e_n	erreur;
E/S	entrée ou sortie;
ε	erreur quadratique moyenne relative;
F	matrice décrivant le modèle d'état final du spectromètre après transformation;
FIFO	première entrée, première sortie;
$f(\bullet)$	fonction de contrainte de positivité;
FPGA	(field programmable gate array);
FR	fichier de registre;
Φ	matrice de transition dans une représentation dans l'espace d'état;
G	matrice d'état simplifiée;
g	vecteur de la réponse impultionnelle du système;
HDL	langage de description de matériel;
h	réponse impultionnelle modifiée;

I	innovation;
ITGE	intégration à très grande échelle;
J	critère d'optimisation;
k	gain de Kalman;
KALSPL	algorithme de kalman et spline;
$\hat{I}$	position des pics dans un spectrogramme;
λ	longeur d'onde;
L	opérateur de Laplace;
LD	ordre du polynome du dénominateur;
LMS	moindres carrés moyens (least mean square);
LN	ordre du polynome du numérateur;
M	ordre de la matrice d'état du système;
m	nombre de pics dans un spectre;
M/A	multiplieur/accumulateur;
max	valeur maximal;
MAC	multilpieur/accumulateur;
MC	micro- contrôleur
min	valeur minimale;
M_g	taille de la réponse impulsionnelle du système;
M_V	nombre de points nécessaires pour l'approximation de la réponse impulsionnelle du système;
MUX	multiplexeur;

M_v	nombre de points nécessaires pour l'approximation de g;
μ	paramètre de relaxation dans l'algorithme LMS;
N	nombre des échantillons;
n	indice des échantillons;
NBX	nombre de bits pour la représentation des variables;
N(s)	opérateur Laplacien d'une équation;
η	bruit qui entache la mesure;
O	ordre de l'équation;
P	position de processeur élémentaire;
PE	processeur élémentaire;
PPC	puces par couche de silicium;
q	pas de progression d'un signal;
R	rendement;
r	variante de bruit;
RAM	random access memory;
RF	fichier de registre (register file);
RISC	machine à ensemble d'instruction réduit;
RTL	register transfer level;
SIMD	une seule instruction, plusieurs données;
TEXPO98	exposition en microélectronique organisée par la CMC 1998;
Tn	matrice de transition dans l'algorithme LMS;
t	temps;

t_p	temps de calcul en parallèle;
t_s	temps de calcul séquentiel;
UAL	unité arithmétique logique;
UF	unité fonctionnelle;
UM	unité de mémore;
VHDL	langage rapide de description du matériel;
VLIW	mot d'instruction très long;
VLSI	intégration à très grande échelle (very large scale integration);
x	signal d'entrée du système;
$\hat{x}$	résultat final de reconstitution;
y	signal de sortie;
$\tilde{y}$	signal de sortie bruitée;
[]	référence;
σ^2	variance;
z	vecteur d'état;

Chapitre 1

Introduction

1.1 PROBLÉMATIQUE

La mesure est certainement la démarche scientifique la plus fondamentale. La précision de la mesure constitue un préalable primordial pour le développement des sciences et à la mise en œuvre de certaines techniques. La mesure peut être définie comme l'extraction de l'information désirée à partir d'un signal de mesure, et la présentation de cette information sous forme utile appelée résultat de mesure.

Le résultat de conversion $\tilde{y}(t)$ d'un signal $\dot{x}(t)$ qui n'est pas accessible par mesure directe est sujet au bruit de diverses origines. Ceci pourrait être expliqué en utilisant le schéma général de mesure d'un système linéaire tel que décrit par la figure1.1. Dans cette figure nous avons à mesurer un signal $\dot{x}(t)$ à l'aide d'un système de mesure qui a pour fonction de convertir les grandeurs physique en signaux électriques. La bande passante limitée du système de mesure ainsi que le bruit $\eta(t)$ associé à différentes sources introduisent des imperfections sur le résultat de mesure $\tilde{y}(t)$. Tout signal de mesure peut

être caractérisé par sa nature physique et les particularités de son modèle mathématique nécessaire pour son analyse. Le modèle mathématique qui décrit la relation entre les signaux de cette figure, pour un système de mesure linéaire et invariant dans le temps, pourrait être donné en utilisant l'équation de convolution:

$$\tilde{y}(t) = \int_{-\infty}^{\infty} g(t-\tau)\dot{x}(\tau)d\tau + \eta(t) \qquad (1.1)$$

où $\dot{x}(t)$ est le signal à mesurer, $g(t)$ est la réponse impulsionnelle du système de conversion, $\eta(t)$ est le bruit considéré de mesure additif associé généralement aux imperfections du système et à l'effet de l'environnement et $\tilde{y}(t)$ est le résultat de conversion bruité de la mesure.

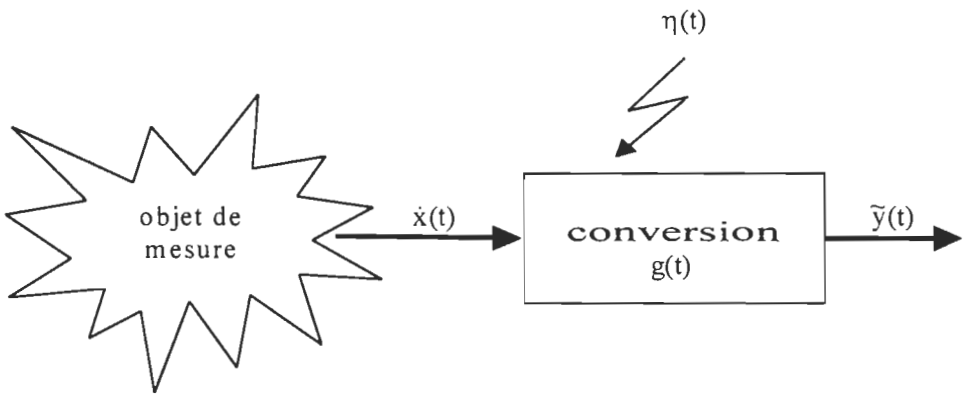

Figure 1.1 Schéma descriptif du système de conversion.

La modélisation des systèmes de mesure a pour but principal d'établir le lien entre la grandeur réelle et celle mesurée pour que la représentation des résultats fournis par l'instrument traduise la réalité.

Toute mesure peut être considérée comme une séquence d'opérations élémentaires, qui consistent à transformer, au moins l'une des caractéristiques mesurées, en signal.

Conséquemment, tout système de mesure peut être considéré comme un ensemble de blocs fonctionnels organisés de manière à accomplir le traitement nécessaire de la mesurande en réalisant des fonctions tout à fait primitives.

La transformation de l'information dans le processus de mesure, tel que décrit par l'approche systémique [24] est exposée à la figure 1.2 . Cette figure représente le schéma général d'un système de mesure.

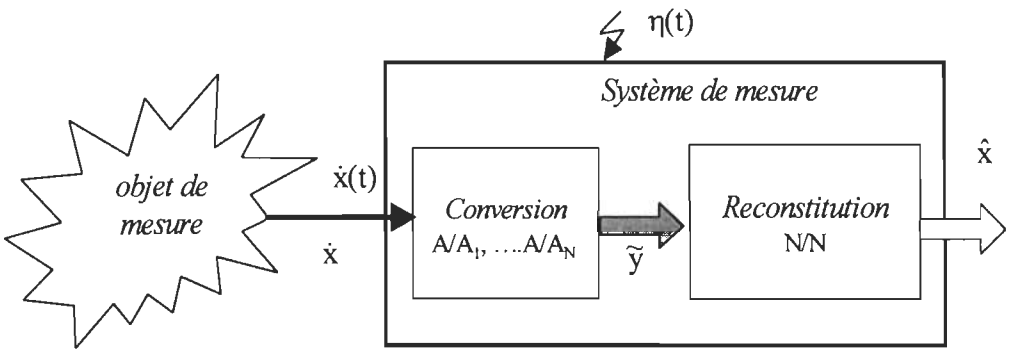

Figure 1.2 Schéma général d'un système de mesure.

Dans cette figure $\dot{x}$ ou $\dot{x}(t)$ est le signal représentant le signal d'entrée tel qu'il pourrait être mesuré par un système avec une très haute exactitude, $\tilde{y}$ est la séquence du résultat brut de mesure issue d'un système qui est sujet au bruit de conversion *A/A* et *A/N* et $\hat{x}$ est le résultat final de mesure soit l'estimé de x. Ce processus de mesure peut être décomposé logiquement en deux parties distinctes: la conversion et la reconstitution.

Conversion: Durant cette phase, nous transformons l'information de la grandeur physique mesurée en signal facilement interprétable par le bloc de reconstitution. Dans le cas de mesure électrique, cette transformation se fait en deux étapes ; une conversion *A/A* pour transformer la grandeur physique sous forme de signal électrique, et ensuite, une conversion *A/N* pour discrétiser le signal en vue d'un traitement numérique.

Reconstitution: Cette opération a pour objectif d'effectuer l'interprétation du résultat de conversion pour aboutir au résultat final de la mesure. Durant la phase de reconstitution, différents algorithmes peuvent être utilisés pour faire l'interprétation du résultat de conversion et l'extraction de l'information sous sa forme utile.

Dans le cas où l'effet du bruit serait négligeable, le système peut être considéré comme déterministe et les méthodes conventionnelles de reconstitution peuvent être appliquées pour éliminer l'effet du système sur la mesure finale. Tandis que la correction des imperfections des systèmes stochastiques et l'élimination de leurs effets ne sont pas triviales, plusieurs méthodes de reconstitution de signaux ont été développées pour cette fin. Ces méthodes peuvent être classifiées en six groupes selon [1], nous précisons ici uniquement ceux touchant de près notre travail :

1. Méthodes directes qui sont basées sur la numérisation du modèle de la relation liant l'entrée $x(t)$ et la sortie $\tilde{y}(t)$.
2. Méthodes itératives qui utilisent la sortie du système $\tilde{y}(t)$ et sa réponse impulsionnelle pour faire la reconstitution de $x(t)$ par approximation successive [2-3].

3. Méthodes variationnelles, basées sur la minimisation ou la maximisation d'un critère définissant la qualité de reconstitution.

Ces méthodes utilisent le modèle mathématique du système et les informations a priori sur le bruit pour faire la reconstitution de la mesurande.

Dans les analyses spectrométriques, la mesure du spectre d'abosorption d'une substance (spectre d'absorption en fonction du longueur d'onde) est affectée par des erreurs systématiques dues aux imperfections et limitations de l'instrument de mesure : le spectromètre. Ces erreurs provoquent la distorsion des caractéristiques de la grandeur `s mesurer (par exemple recouvrement des pics voisins, présence des artéfacts, etc.) et compliquent son interprétation exacte.

L'information dans un spectre est contenue dans la position et l'amplitude des pics qu'il contient. En effet, la longueur d'onde du spectre et son amplitude permettent d'identifier certaines composantes chimiques ou organiques constituant la matière mesurée. La position des pics dans un spectre et leur amplitude est une mesure importante en spectrométrie.

Plusieurs algorithmes de reconstitution de signaux basés sur des méthodes stochastiques peuvent être mis en œuvre pour faire la reconstitution du spectre et l'estimation de la position des pics et la correction de leur amplitude [4-6]. Les performances de ces algorithmes varient en fonction de l'application et des conditions d'utilisation. Les développements, algorithmique et architectural, ouvrent des horizons de

plus en plus larges vers l'implantation en technologie VLSI d'une classe d'algorithmes spécialisés en reconstitution de mesurande. Les exigences concernant la miniaturisation des systèmes, leur connexion, leur consommation en puissance et leur facilité de développement et de maintenance sont sans aucun doute les véritables raisons de leur intégration. Les exploits récents dans le domaine de la micro-électronique jumelés avec les nouveaux logiciels de conception assistée par ordinateur rendent possible l'amélioration de la qualité de la mesure en facilitant l'implantation d'algorithmes sophistiqués utilisés pour faire la reconstitution de mesurande.

Dépendamment des exigences concernant la précision et la rapidité de la mesure, nous pouvons faire le choix entre l'utilisation de processeurs commerciaux (DSP, MC, FPGA, etc.) ou le développement d'un processeur spécialisé. Quand la précision est le seul critère, l'utilisation d'un DSP est généralement satisfaisante. L'utilisation d'un processeur spécialisé est plus justifiée quand l'application exige une bonne précision et rapidité de calcul ou si l'algorithme utilisé exige une grande capacité de calcul.

La proposition d'une architecture dédiée est demandée pour exécuter les opérations de calcul et profiter du parallélisme inhérent dans l'algorithme. Une architecture systolique sera utilisée pour résoudre le calcul matriciel [7]. L'implantation de l'algorithme dans cette architecture assurera un compromis entre, le temps de calcul, la surface d'intégration, la puissance consommée et la précision des résultats.

1.2 ÉTAT DE LA RECHERCHE

H. T. Kung et C. E. Leiserson de l'université Carnegie-Malone ont introduit les réseaux systoliques pour la première fois en 1979 [8]. Il s'agissait d'une proposition très théorique qui a été appliquée dans divers domaines de la recherche scientifique, surtout dans le domaine de traitement numérique du signal. H. T. Kung en 1982 donne une variété d'architectures systoliques applicables aux algorithmes standard de traitement de signal tels que la FFT, la convolution, la déconvolution ou encore l'arithmétique matricielle. La simplicité, la régularité et le parallélisme massif de ces architectures ont encouragé l'application de ses diverses configurations à plusieurs types de problèmes réels, entre autres pour l'implantation du filtre de Kalman et la méthode de gradiant LMS.

Pour accomplir l'implantation du filtre de Kalman dans une architecture typiquement systolique, plusieurs travaux ont porté sur la résolution du calcul matriciel et les techniques qui s'y rattachent pour faire face à la demande excessive de capacité de calcul. Ainsi, M. Moonen, P. Van Dooren et J. Vandewalle ont proposé en 1990 [21] une architecture systolique dédiée au calcul matriciel telle que la triangularisation de matrice et la multiplication matrice-vecteur. Ils ont aussi démontré la possibilité d'implanter efficacement des algorithmes de traitement de signal comme le filtre de Kalman, la résolution des équations de Riccati et Lyapunov [21]. S.Y. Kung et J.N. Hwang ont présenté une architecture similaire pour traiter le problème du filtre de Kalman [23]. Dans [23] les auteurs présentent une architecture dédiée basée sur un réseau linéaire pour la mise en œuvre du filtre de Kalman pour le contrôle d'un système dynamique. Ils prouvent que

pour un tel système avec N états et M composantes d'observations, le réseau utilise N(N+1) processeurs élémentaires et 4N+6M itérations.

M.R. Azimi, T. Lu et E. M. Nebot ont appliqué le même modèle pour l'implantation de l'estimateur récursif de Kalman et la génération de l'estimation. Dans la littérature on peut trouver plusieurs articles qui traitent de l'implantation du filtre de Kalman non-stationnaire [9].

L'application de l'architecture systolique pour la résolution des équations du filtre de Kalman non stationnaire et variant est plus répandue. Une classe d'algorithmes basés sur le filtre de Kalman a été développée par [1] et [6]. Ces algorithmes ont été développés en interaction aux développements d'architectures en technologie VLSI. Des architectures parallèles dédiées à l'implantation de ce filtre en technologie VLSI sont proposées dans [1]. Ces architectures exploitent au maximum le parallélisme inhérent dans les algorithmes. M. Newel et J. Rasure ont utilisé le réseau systolique pour l'implantation d'un réseau de neurones. Cette architecture pourrait être adaptée à l'implantation d'autres algorithmes tels que la FFT, la DCT, la convolution et l'arithmétique matriciel. La principale limitation de cette architecture réside dans son manque d'autonomie [7].

L'architecture VLIW ('Very Long Instruction Word', mot d'instruction très long) est moins répandue pour l'implantation du filtre de Kalman ou pour la reconstitution du mesurande. Elle trouve son application dans le domaine du traitement d'image et le multimédia. Ainsi, D. Bursky décrit dans [10] le processeur MPACT. Ce circuit intégré est

dédié au multimédia et dépasse les exigences et conditions d'exécution d'un système multimédia. La puce englobe environ 1,4 millions de transistors et est basée sur un modèle VLIW programmable. Le processeur exécute des instructions du type (SIMD). Tandis que dans [11] on propose un processeur basé sur le modèle VLIW appelé HIPAR-DSP, ce processeur est principalement dédié à l'implantation des algorithmes de traitement d'image. En se basant sur les exigences algorithmiques résultantes, les auteurs ont déduit une architecture VLIW pour un processeur spécifique à jeu d'instructions réduit avec circulation de données parallèles. Le processeur se compose de 4 ou 16 chemins de données parallèles avec des mémoires locales pour stocker les données couplées avec une mémoire partagée à accès du type matriciel. Les architectures de commande, de mémoire et d'arithmétique du processeur sont parfaitement équilibrées et adaptées aux flots de données des algorithmes ayant pour résultat une capacité de traitement élevée pour un large éventail des algorithmes de traitement d'image.

Bien que les idées fondamentales et les techniques des machines VLIW et de SIMD (une seule instruction plusieurs données, Single Instruction Multiple Data) soient différentes, elles partagent certaines caractéristiques communes du point de vue matériel. Le plus évident est: le parallélisme des unités fonctionnelles qui exécutent simultanément, sous la commande d'une seule instruction, plusieurs opérations. Dans [12] les auteurs proposent une nouvelle architecture, appelée Homogène-VLIW (HVLIW), qui peut maintenir les différents mérites et même réduire quelques inconvénients de VLIW et de SIMD. Un des problèmes principaux de la conception est: comment établir un système de mémoire qui convient aux machines de SIMD et de VLIW. Ils proposent une hiérarchie de

mémoire basée sur une mémoire tampon pour résoudre le problème. Le résultat d'analyse prouve que la mémoire basée sur la hiérarchie de mémoire tampon équipée d'une connexion de bus a non seulement servi de mémoire temporaire entre l'unité centrale de traitement et la mémoire, mais réalise également l'exécution d'un débit de plus de 90% d'une mémoire traditionnelle.

Le VIPER (VLIW Integer Processor), proposé dans [13] contient quatre unités fonctionnelles pipelinées et peut réaliser 100 MIPS comme sommet de performance à 25 MHz. Le processeur est capable d'exécuter des opérations de branchement à plusieurs directions, deux opérations de load/store, et jusqu'à quatre opérations arithmétiques et logiques à chaque cycle d'horloge avec le plein accès de chaque unité fonctionnelle UF au fichier de registre RF. Le processeur est intégré avec une unité de contrôleur avec mémoire d'instructions large et une mémoire de données, exigeant 450000 transistors et une surface de 12,9 par 9,1 millimètres en technologie 1,2 μm.

La G-HINGE [14] est une machine VLIW qui exploite le parallélisme au niveau d'instruction dans des applications intensives de grandeur scalaire et de mémoire. L'architecture de mémoire offre la possibilité d'exécuter multiples opérations non homogènes en parallèle. La base de cette architecture est la configuration des modules de mémoire programmable indépendamment. Les instructions de mémoire parallèle et les opérations de flot de données sont mises ensemble dans des mots d'instructions très large (VLIW).

Pour exploiter le parallélisme au niveau d'instruction, les compilateurs pour VLIW et les processeurs superscalaires utilisent souvent l'échéancier statique du programme. Cependant, le code disponible à nouveau peut être sévèrement restreint dû aux dépendances ambigues entre les instructions de mémoire. Dans [15] on présente un mécanisme matériel simple, désigné sous le nom du tampon de conflit mémoire, qui facilite la planification du code statique en présence des dépendances écriture/lecture de la mémoire. L'exécution du programme correcte est assurée par le tampon de conflit de mémoire et le code de réparation fourni par le compilateur. Avec cet ajout, une accélération significative qui dépasse le modèle agressif d'établissement de code peut être réalisé pour des programmes non numériques et numériques.

D'autres travaux ont porté sur le développement d'outil software pour la conception des architectures VLIW plus appropriées et mieux adaptées à certaines applications ou algorithmes. Ainsi dans [16] on trace les grandes lignes des étapes générales de conception d'un outil de synthèse pour réaliser les co-processeurs dédiés à certaines applications scientifiques données ayant des calculs itératifs intensifs particulièrement avec des répétitions. On décrit la réalisation d'un co-processeur basé sur un modèle VLIW, on a même produit un code parallèle d'accompagnement.

VLIW, multi-contexte ou (windowed-registre) sont des architectures qui peuvent exiger cent registres ou plus. Il peut être difficile de concevoir un fichier de registre avec tant de registres qui répond à des exigences de vitesse assez sévères. R. Yung et N.C.

Wilhelm [17] ont proposé de résoudre ce problème. Ils tirent profit des valeurs des registres qui sont sautées dans le pipeline du processeur en complétant les valeurs sautées avec des valeurs assurées par une petite banque de registre.

Zhizhong Tang et al. ont proposé une nouvelle architecture basée sur le modèle VLIW appelée GPMB (Global Pipelining of Multi-Branch). L'architecture de GPMB [18] peut manipuler des programmes branchement-intensifs efficacement. Avec le concept de la fonction de la prochaine adresse, GPMB calcule correctement la prochaine adresse. Ce principe est implanté dans ce système d'une façon matérielle et logicielle.

[19] présente une étude de projet d'un processeur à mot d'instruction très long VLIW dédié au traitement de signal vidéo VSP en se concentrant sur les différents points qui affectent l'architecture du processeur. Les architectures VLIW fournissent un parallélisme élevé et une excellente programmabilité utilisant des langages de très haut niveau, mais exigent une attention particulière à la conception VLSI. L'architecture proposée est flexible, avec une grande largeur de bande et un RF complètement connecté. Ce design vise entre 32-64 opérations par instruction par cycle avec des fréquences de base excédant 500 MHz. Les applications avancées de transmission des données dans le domaine du multimédia exigent un plus grand parallélisme dans l'architecture d'usage général telle que le DSP. Quelques architectures RISC peuvent fournir assez de capacité de traitement comparable à l'architecture d'usage universel mais ne sont pas adéquates pour les nouvelles applications en télécommunication. Tandis que l'architecture VLIW s'est avérée être une très bonne approche pour résoudre ce problème, le principal inconvénient est la limitation des ressources matérielles comparativement à la complexité des unités

fonctionnelles multiple et au nombre d'instructions qui augmente d'une manière considérable quand un programme n'offre pas assez de parallélisme au niveau des instructions. Dans [20] une architecture RISC est proposée. Elle est basée à la fois sur le principe des architectures RISC et le principe du mot normal d'instruction (RNIW).

1.3 OBJECTIFS DE RECHERCHE

Ce travail vise principalement à contribuer au développement d'un système de mesure intégré. Le développement architectural permet certainement une amélioration de la qualité des systèmes de mesure. Cette contribution se concrétise par la proposition d'architectures parallèles pour la mise en œuvre d'une classe d'algorithmes de reconstitution de signal basés sur le filtre de Kalman et la méthode des moindres carrés.

L'atteinte de cet objectif passe par la réalisation des sous-objectifs suivants:

- étude de l'algorithme de reconstitution de signal basé sur le filtre de Kalman et les fonctions Spline [6] pour l'estimation de la position et de l'algorithme LMS pour la correction d'amplitude de données spectrométriques.
- adaptation des algorithmes de correction en vue d'une implantation en technologie VLSI.
- proposition d'une architecture pour l'algorithme de Kalman-Spline (KALSPL) et Least-Mean-Square (LMS).
- Proposition d'une architecture capable d'accepter une classe d'algorithmes basés sur le filtre de KALMAN.

- Modélisation et synthèse des architectures proposées en technologie CMOS 1.5 μm.
- Synthèse des résultats.

1.4 MÉTHODOLOGIE

La recherche dans le domaine de la technologie VLSI a beaucoup avancé surtout durant la dernière décennie. Ainsi plusieurs architectures ont été proposées pour l'implantation des algorithmes de traitement de signal. Une recherche bibliographique va permettre une mise à jour de l'état de la recherche scientifique dans ce domaine et facilitera l'orientation du projet pour mieux rencontrer les objectifs préétablis.

La connaissance approfondie des algorithmes à implanter est nécessaire pour accomplir l'efficacité dans le design de l'architecture à proposer. Ceci implique une étude assez poussée des algorithmes choisis pour la reconstitution de mesurande. Pour mieux comprendre le flot de données dans les architectures, nous proposons une étude avec simulation sur Matlab.

Sachant qu'une bonne proposition résulte d'un équilibre entre l'algorithme et l'architecture [1], l'étude algorithmique doit tenir compte de l'architecture visée. Ainsi une étude des architectures visées s'impose pour mieux établir l'équilibre entre les deux études et faire une bonne proposition qui tient compte du flot de données, de la capacité de calcul, de la bande d'entrée-sortie et de la bande mémoire.

L'étude des algorithmes et des architectures permet de les adapter pour une implantation en technologie VLSI. L'adaptation peut aller de la simple transformation jusqu'à la modification de la structure de l'algorithme ou même de l'architecture. Le calcul séquentiel serait inadéquat dans le cas d'algorithmes parallèles à cause de la limitation dans les performances et les ressources des processeurs à usage général. Les architectures parallèles dédiées permettent de tirer profit de ce parallélisme inhérent et augmenter la performance du processeur. Vue le type de calcul dans les algorithmes de reconstitution de mesurande, la proposition d'architecture systolique serait très bien adaptée pour ce type d'algorithmes. Pour rendre le processeur plus flexible à la mise en œuvre d'une classe d'algorithmes de traitement de signal, la proposition d'une architecture VLIW hautement parallèle serait inévitable.

Le développement des outils de conception assistée par ordinateur et des langages de description du matériel rend possible la réalisation des architectures proposées. Ainsi, nous pouvons modéliser l'architecture systolique en VHDL et simuler le design pour s'assurer de sa fonctionnalité logique.

La synthèse des modèles VHDL dans une technologie CMOS permet d'avoir une idée sur le dessin de masque du processeur et de vérifier sa fonctionnalité physique grâce à des simulations d'après synthèse.

Chapitre 2

Étude et analyse des algorithmes

2.1 INTRODUCTION

Par définition, un signal est le support physique de l'information. Mathématiquement, les signaux sont représentés par une fonction d'une ou plusieurs variables qui régit leurs interactions. Généralement, les signaux doivent être traités soit pour en extraire de l'information, soit pour les rendre porteurs d'information. Ces traitements sont effectués à l'aide de systèmes de traitement des signaux. Le développement des processeurs à application spécifique a permit la mise en œuvre d'algorithmes très complexes rendant possible le traitement plus efficace et donnant un sens aux diverses théories du traitement de signal.

Toute grandeur physique peut être représentée par un signal électrique grâce à l'utilisation d'un système de mesure électrique. Le processus de mesure décrit à la figure 1.2 donne une idée sur les étapes de transformation de l'information contenue dans le signal mesuré. Elle met l'accent sur les perturbations que peut subir un signal durant ce processus. Ces perturbations proviennent généralement de la limitation de la bande passante du système et de l'introduction des blocs de conversion dans le processus [24-25].

Le traitement numérique de signal est nécessaire pour l'amélioration de la qualité des systèmes de mesure électronique. Plusieurs algorithmes basés sur les méthodes stochastiques ont été développés et efficacement utilisés pour faire la reconstitution de mesurande et l'élargissement de la bande passante des systèmes électriques et électroniques [26-29].

La méthode des moindres carrés est l'une des méthodes les plus populaires et les plus utilisées dans le domaine de traitement de signal et ce grâce aux résultats très satisfaisants qu'elle offre. Le filtre de Kalman est une variante de cette méthode [29]. Ce filtre est basé sur certaines suppositions concernant la nature du bruit qui perturbe la mesure et ses propriétés statistiques. Ce filtre est largement utilisé pour la reconstitution de mesurande. Dans la littérature on peut trouver plusieurs travaux traitant de l'application de ce filtre dans les problèmes standards de traitement de signal et plus spécifiquement la reconstitution de mesurande [4-6].

Au chapitre 1 nous avons décrit le résultat de mesure $\tilde{y}(t)$ d'un signal $x(t)$ par la figure 1.1. Nous avons aussi donné le modèle mathématique qui décrit la relation entre les signaux de cette figure avec Eq. (1.1). L'intégrale de convolution dans cette équation peut être exprimée sous la forme mathématique suivante:

$$y(t)=g(t)*x(t)+\eta(t) \tag{2.1}$$

Pour éviter la complexité de la convolution et faciliter la résolution de Eq. (1.1) ou Eq. (2.1), nous transformons le produit de convolution dans le domaine temporel en simple multiplication dans le domaine fréquentiel.

$$Y(f) = G(f)X(f) + \eta(f) \tag{2.2}$$

Dans le cas des systèmes déterministes, la résolution de Eq. (2.2) est triviale, elle se simplifie à une inversion de la fonction de transfert du système puisque η est nul. Dans le processus de mesure décrit à la figure (1.2) le bruit est inévitable et il faut tenir compte de son influence. Comme G(f) agit souvent comme un filtre passe bas, son inverse agira comme un filtre passe haut, d'où l'amplification des composantes de haute fréquence du signal Y(f) et du bruit $\eta(f)$.

$$X(f) = G^{-1}(f)\big(G(f)X(f) + \eta(f)\big) \tag{2.3}$$

La résolution de Eq. (2.2) dans ce cas est fort complexe et nécessite l'utilisation d'algorithmes puissants pour extraire le signal x(t). Comme le problème vient du bruit, il faut donc contrôler l'effet de son amplification dans les régions à haute fréquence, on peut donc minimiser l'erreur résiduelle en utilisant des algorithmes basés sur la méthode des moindres carrées.

2.2 ANALYSE DES ALGORITHMES

2.2.1 Algorithme de KALSPL

Nous avons décrit la relation entre les signaux dans un système de mesure au chapitre 1. Nous reprenons ici le théorème de convolution dans le cas d'un modèle de données spectrométriques :

$$y(\lambda) = \int_{-\infty}^{\infty} g(\lambda - \lambda')x(\lambda')d\lambda' \tag{2.4}$$

$$\tilde{y}_n = y(\lambda_n) + \eta_n \quad \text{pour} \quad n = 1,2,...,N \tag{2.5}$$

où η_n est l'erreur aléatoire introduite par le bruit, λ est la langueur d'onde, λ_n est sa valeur discrète, $y(\lambda)$ est le résultat de convolution du signal d'entrée parfait $x(\lambda')$ et la réponse impultionnelle du système (spectromètre) $g(\lambda)$. $\tilde{y}_n$ est le résultat de mesure délivré par le système à la sortie et qui est sujet au bruit. La correction des données spéctromètriques consiste à la résolution numérique de Eq.(2.4) basée sur les informations à priori de $g(\lambda)$ et $\{\tilde{y}_n\}$.

En supposant que :

$$x(\lambda + \lambda_{min}) \Rightarrow x(\lambda) \text{ pour } \lambda \in [0, \lambda_{max} - \lambda_{min}]$$

$$g(\lambda + \lambda'_{min}) \Rightarrow g(\lambda) \text{ pour } \lambda \in [0, \lambda'_{max} - \lambda'_{min}]$$

$$y(\lambda + \lambda_{min} + \lambda'_{min}) \Rightarrow y(\lambda) \text{ pour } \lambda \in [0, \lambda_{max} - \lambda_{min} + \lambda'_{max} - \lambda'_{min}]$$

$x^{(k)}(\lambda)$ est la k^{iem} dérivée de $x(\lambda)$, $k = 1,2,.....$

$y^{(k)}(\lambda)$ est la k^{iem} dérivée de $y(\lambda)$, $k = 1,2,.....$

$x_n = x(\lambda_n)$, $y_n = y(\lambda_n)$ pour $n = 1,2,....N.$

$x_n^{(1)} = x^{(1)}(\lambda_n)$, $x_n^{(2)} = x^{(2)}(\lambda_n)$, pour $n = 1,2,....N.$

On peut normaliser Eq. (2.4) de façon à ce que $x(\lambda)$, $g(\lambda)$ et $y(\lambda)$ soit nulles pour $\lambda<0$. Cette équation intégrale pourrait être représentée dans le domaine de Laplace avec une équation algébrique de la forme:

$$Y(s) = G(s)X(s) \tag{2.6}$$

où $Y(s) = L\{y(t)\}$, $G(s) = L\{g(t)\}$ et $X(s) = L\{x(t)\}$ avec L est l'opérateur de Lapalce.

Si on suppose que :

$$G(s) = \frac{N_g(s)}{D_g(s)} \tag{2.7}$$

et que N(s) et D(s) sont polynomiales d'ordre LN et LD respectivement, avec LD>LN et $d_{LD}=1$, de sorte que :

$$N(s) = \sum_{k=0}^{LN} n_k s^k \quad \text{et} \quad D(s) = \sum_{k=0}^{LN} d_k s^k \quad \text{avec LD>LN et } d_{LD}=1. \tag{2.8}$$

Eq. (2.6) pourrait être remplacée par une équation différentielle ordinaire linéaire,

$$\sum_{k=0}^{LD} d_k y^{(k)}(\lambda) = \sum_{k=0}^{LN} n_k x^{(k)}(\lambda) \tag{2.9}$$

Ce modèle du spectre mesuré $y(\lambda)$ peut avoir forme d'une variable d'état en utilisant une variable auxiliaire :

$$v_0(\lambda) = L^{-1}\left\{\frac{X(s)}{D(s)}\right\} \tag{2.10}$$

Ce qui permet de représenter $x(\lambda)$ et $y(\lambda)$ comme :

$$x(\lambda) = \sum_{k=0}^{LD} d_k v_0^{(k)}(\lambda) \tag{2.11}$$

$$y(\lambda) = \sum_{k=0}^{LN} n_k v_0^{(k)}(\lambda) \tag{2.12}$$

En utilisant le vecteur :

$\mathbf{v} = \left[v_0^{(0)} \quad v_0^{(1)} \quad \ldots \quad v_0^{(LD-1)} \right]^T$ comme vecteur d'état.

et on obtient la représentation en forme de variable d'état :

$$\mathbf{v}^{(1)}(\lambda) = \mathbf{A}\mathbf{v}(\lambda) + \mathbf{b}x(\lambda) \tag{2.13}$$

$$y(\lambda) = \mathbf{c}^T\mathbf{v}(\lambda) \tag{2.14}$$

où

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \ldots & 1 \\ -d_0 & -d_1 & -d_2 & -d_3 & \ldots & -d_{LD-1} \end{bmatrix}, \dim(\mathbf{A})=LD\times LD;$$

$\mathbf{b} = \left[0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1\right]^T$, $\dim(\mathbf{b})=LD$;

$\mathbf{c} = \left[n_0 \quad n_1 \quad n_2 \quad \ldots \quad n_{LN} \quad 0 \quad \ldots \quad 0\right]^T$, $\dim(\mathbf{c})=LD$.

Comme le traitement des données de mesure se fait généralement par ordinateur, nous aurons donc besoin de discrétiser le modèle d'état Eq. (2.13) et (2.14). Il existe plusieurs approches pour la discrétisation. Nous distinguons en particulier la discrétisation basée sur la schématisation d'Euler [6] : $\mathbf{v}_n^1 = \frac{\mathbf{v}_{n+1} - \mathbf{v}_n}{\Delta\lambda}$, $n = 0,1,2,\ldots$. ce qui donne le modèle d'état suivant:

$$\mathbf{v}_{n+1} = \Phi \mathbf{v}_n + \phi x_n$$

$$\tilde{y}_n = \mathbf{c}^T \mathbf{v}_n + \eta_n$$

avec $\Phi = I_{Ld} + \Delta\lambda \mathbf{A}$, $\mathbf{r} = \Delta\lambda \mathbf{b}$, $\mathbf{I}_{Ld}$ est la matrice identité d'ordre Ld.

L'apport essentiel de cette représentation est la réduction de l'ordre du modèle. La principale limitation dans le calcul de la solution avec l'algorithme basé sur le filtre Kalman provient des dimensions des vecteurs d'état qui sont de l'ordre du nombre d'échantillons de la réponse impulsionnelle $\mathbf{g}(\lambda)$.

L'approche à la correction des données spectrométriques proposée dans [69] est basée sur la modélisation ou l'approximation exact du spectre $x(\lambda)$ avec des fonctions spline cubic [70].

$$\hat{x}(\lambda) = \sum_{k=0}^{3} p_{n,k} (\lambda - \lambda_n)^k \quad \text{pour } \lambda \in [\lambda_n, \lambda_{n+1}] \tag{2.15}$$

continue avec ses deux dérivées première et seconde $\hat{x}^{(1)}(\lambda)$ et $\hat{x}^{(2)}(\lambda)$. Les conditions de continuité de la fonction $\hat{x}(\lambda)$ et $\hat{x}^{(1)}(\lambda)$ permettent d'exprimer les paramètres $p_{n,k}$ avec des valeurs discrètes du spectre exact $x(\lambda)$ et de sa dérivée $x^{(1)}(\lambda)$.

$$p_{n,0} = x_n \tag{2.16}$$

$$p_{n,1} = x_n^{(1)} \tag{2.17}$$

$$p_{n,2} = \frac{1}{\Delta\lambda}\left(\frac{3}{\Delta\lambda}(x_{n+1} - x_n) - x_{n+1}^{(1)} - 2x_n^{(1)}\right) \tag{2.18}$$

$$p_{n,3} = \frac{1}{\Delta\lambda^2}\left(-\frac{2}{\Delta\lambda}(x_{n+1} - x_n) + x_{n+1}^{(1)} + x_n^{(1)}\right) \tag{2.19}$$

Tandis que la condition de la continuité de la deuxième dérivée $\hat{x}^{(2)}(\lambda)$ pourrait être écrite sous la forme :

$$3x_{n+1} - \Delta\lambda x^{(1)}_{n+1} = 3x_{n-1} + \Delta\lambda x^{(1)}_{n-1} + 4x^{(1)}_{n} \tag{2.20}$$

pour n=1,2,3,.....,N-1.

La substitution de Eq. (2.16)-(2.19) dans Eq. (2.15) donne pour $\lambda \in [\lambda_n, \lambda_{n+1}]$:

$$\hat{x}(\lambda) = w_{00}(\lambda - \lambda_n)x_n + w_{01}(\lambda - \lambda_n)x^{(1)}_n + w_{10}(\lambda - \lambda_n)x_{n+1} + w_{11}(\lambda - \lambda_n)x^{(1)}_{n+1} \tag{2.21}$$

où $w_{00}(\lambda) = \frac{1}{\Delta\lambda^3}(2\lambda^3 - 3\lambda^2\Delta\lambda + \Delta\lambda^3)$

$$w_{01}(\lambda) = \frac{1}{\Delta\lambda^2}(\lambda^3 - 2\lambda^2\Delta\lambda + \Delta\lambda^2)$$

$$w_{10}(\lambda) = \frac{1}{\Delta\lambda^3}(-2\lambda^3 - 3\lambda^2\Delta\lambda)$$

$$w_{11}(\lambda) = \frac{1}{\Delta\lambda^2}(\lambda^3 - \lambda^2\Delta\lambda)$$

De la même façon, selon Eq. (2.13), on peut approximer la variable d'état $\mathbf{v}(\lambda)$ par :

$$\hat{\mathbf{v}}(\lambda) = w_{00}(\lambda - \lambda_n)\mathbf{v}_n + w_{01}(\lambda - \lambda_n)\mathbf{v}^{(1)}_n + w_{10}(\lambda - \lambda_n)\mathbf{v}_{n+1} + w_{11}(\lambda - \lambda_n)\mathbf{v}^{(1)}_{n+1} \tag{2.22}$$

L'intégration de l'équation (2.13), où $x(\lambda)$ et $\mathbf{v}(\lambda)$ sont remplacés par $\hat{x}(\lambda)$ et $\hat{\mathbf{v}}(\lambda)$:

$$\hat{\mathbf{v}}^{(1)}(\lambda) = \mathbf{A}\hat{\mathbf{v}}(\lambda) + \mathbf{b}\hat{x}(\lambda) \tag{2.23}$$

sur l'intervalle $[\lambda_n, \lambda_{n+1}]$ mène à la relation suivante :

$$\mathbf{A}_1\mathbf{v}_{n+1} - \mathbf{b}_{10}x_{n+1} - \mathbf{b}_{11}x^{(1)}_{n+1} = \mathbf{A}_0\mathbf{v}_n - \mathbf{b}_{00}x_n - \mathbf{b}_{01}x^{(1)}_n \tag{2.24}$$

avec

$$\mathbf{A}_1 = \mathbf{I} - \frac{\Delta\lambda}{2}\mathbf{A} + \frac{\Delta\lambda^2}{12}\mathbf{A}^2, \qquad \mathbf{A}_0 = \mathbf{I} + \frac{\Delta\lambda}{2}\mathbf{A} + \frac{\Delta\lambda^2}{12}\mathbf{A}^2, \qquad \mathbf{b}_{00} = \left(\frac{\Delta\lambda}{2}\mathbf{I} + \frac{\Delta\lambda^2}{12}\mathbf{A}\right)\mathbf{b},$$

$$\mathbf{b}_{10} = \left(\frac{\Delta\lambda}{2}\mathbf{I} - \frac{\Delta\lambda^2}{12}\mathbf{A}\right)\mathbf{b}, \qquad \mathbf{b}_{01} = \frac{\Delta\lambda^2}{12}\mathbf{b} \quad \text{et} \qquad \mathbf{b}_{11} = -\frac{\Delta\lambda^2}{12}\mathbf{b}.$$

où **I** est la matrice identité. On supposant que le spectre exacte x(λ) peut être adéquatement modélisé par :

$$x_{n+1} = x_n + \Delta\lambda u_n \tag{2.25}$$

où u_n sont des réalisations de variables aléatoire $\underline{u}_n$ approximant le modèle d'état du spectre mesuré est transformé sous la forme :

$$\mathbf{z}_{n+1} = \Phi\mathbf{z}_n + \phi u_n \tag{2.26}$$

$$\tilde{y}_n = \mathbf{h}^T\mathbf{z}_n + \eta_n \tag{2.27}$$

où $\mathbf{z}_n = \left[x_{n-1} \quad x_{n-1}^{(1)} \quad x_n \quad x_n^{(1)} \quad \mathbf{v}_n^T\right]^T$, dim($\mathbf{z}_n$)=LD+4;

$$\Phi = \mathbf{M}^{-1}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 3 & \Delta\lambda & 0 & 4\Delta\lambda & 0 & \dots & 0 \\ 0 & 0 & \mathbf{b}_{00} & \mathbf{b}_{01} & & \mathbf{A}_0 & \end{bmatrix}, \text{ dim}(\Phi)=(LD+4)\times(LD+4);$$

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & -3 & -\Delta\lambda & 0 & \dots & 0 \\ 0 & 0 & -\mathbf{b}_{10} & -\mathbf{b}_{11} & & \mathbf{A}_1 & \end{bmatrix}, \text{ dim}(\mathbf{M})=(LD+4)\times(LD+4);$$

$\phi = \mathbf{M}^{-1}\left[\Delta\lambda \quad 0 \quad 0 \quad \dots \quad 0\right]^T$, dim($\phi$)=LD+4;

$\mathbf{h} = \left[0 \quad 0 \quad 0 \quad 0 \quad \mathbf{c}^T\right]^T$, dim($\mathbf{h}$)=LD+4.

L'algorithme de KALSPL est ainsi basé sur les suppositions suivantes concernant les séquences $\{u_n\}$ et $\{\eta_n\}$:

- u_n sont des réalisations identiques de variables aléatoires $\underline{u}_n$ à moyenne nulle et une estimation a priori $\breve{\sigma}_u^2$ de leurs variances σ_u^2 est disponible.
- η_n sont des réalisations identiques de variables aléatoires $\underline{\eta}_n$ à moyenne nulle et une estimation a priori $\breve{\sigma}_\eta^2$ de leurs variances σ_η^2 est disponible.
- Les variables $\underline{u}_n$, $\underline{\eta}_n$ sont noncorollées, i.e., $E[\underline{u}_n \quad \underline{\eta}_n] = 0$, $E[\underline{u}_n \quad \underline{u}_m] = \sigma_u^2 \delta_{nm}$ et $E[\underline{\eta}_n \quad \underline{\eta}_m] = \sigma_n^2 \delta_{nm}$.

L'utilisation de variables aléatoires à moyenne nulle pour la modélisation de la première dérivée du spectre exact $x(\lambda)$ est très bien justifiée dans les applications spectrométriques vue que le spectre idéal est supposé avoir la forme d'un train d'impulsion dont la moyenne des dérivées est proche de zéro.

Sous les suppositions qu'on vient d'énumérer, le filtre de Kalman [29] pourrait être directement appliqué pour l'estimation du vecteur d'état $\mathbf{z}_n$ en se basant sur les données spectrométriques et les prés-estimés $\breve{\sigma}_u^2$ et $\breve{\sigma}_\eta^2$:

$$\sum\nolimits_{n+1}^{+} = \Phi \sum\nolimits_{\nu} \Phi^T + \breve{\sigma}_u^2 \phi \phi^T \tag{2.28}$$

$$\sum\nolimits_{n+1} = \sum\nolimits_{n+1}^{+} - \frac{\sum\nolimits_{n+1}^{+} \mathbf{h}\mathbf{h}^T \sum\nolimits_{n+1}^{+}}{(\mathbf{h}^T \sum\nolimits_{n+1}^{+} \mathbf{h} + \breve{\sigma}_\eta^2)} \tag{2.29}$$

$$\mathbf{k}_{n+1} = \frac{1}{\breve{\sigma}_\eta^2} \sum\nolimits_{n+1} \mathbf{h} \tag{2.30}$$

$$\hat{\mathbf{z}}_n = \Phi\hat{\mathbf{z}}_{n-1} + \mathbf{k}_\infty(\tilde{y}_{n-1} - \mathbf{h}^T\Phi\hat{\mathbf{z}}_{n-1}) \tag{2.31}$$

où $\sum_{n+1}^{+}$ est la matrice de covariance de l'erreur de prédiction, $\sum_{n+1}$ est la matrice de covariance de l'erreur d'estimation, **h** est le vecteur d'observation, Φ est la matrice d'état avec dim(Φ)=LD×LD et $\mathbf{k}_{n+1}$ est le vecteur du gain de Kalman stationnaire qui peut être calculé en avance, dim($\mathbf{k}_\infty$)=LD. Le résultat d'estimation du vecteur $\hat{\mathbf{z}}_n$ permet de retrouver les estimés de $\hat{x}_n$ et $\hat{x}_n^{(1)}$, et on utilise Eq. (2.21) pour reconstruire le spectre $\hat{x}(\lambda)$.

On peut améliorer les résultats de reconstitution si on incorpore une contrainte de positivité sur les valeurs estimées :

$$\hat{x}_n := \begin{Bmatrix} \hat{x}_n \text{ si } \hat{x}_n \geq 0 \\ 0 \text{ si } \hat{x}_n < 0 \end{Bmatrix} \tag{2.32}$$

L'algorithme de Kalman tel que décrit dans [4, 29] exige une capacité de calcul relative à l'ordre du modèle d'état qui décrit le système. L'ordre du modèle mathématique d'un système de mesure dépend de la largeur de sa réponse impultionnelle $g(\lambda)$. Afin d'éliminer la dépendance entre l'ordre du modèle mathématique du système et la largeur de $g(\lambda)$, il faut réduire l'ordre du modèle mathématique du système et la capacité arithmétique exigée pour la résolution numérique de l'algorithme de Kalman. L'algorithme de Kalman-spline a été dérivé en profitant de l'approximation de $g(\lambda)$ par la convolution de deux fonctions symétriques $g^-(\lambda)\,\&\,g^+(\lambda)$ et $x(\lambda)$ avec une fonction spline cubique [6]. Cette approximation étend la portée de Eq. (2.31) à des systèmes avec des réponses

impultionnelles $g(\lambda)$ de différentes largeurs sans changement significatif sur M=Mv+4 où Mv est le nombre d'éléments nécessaires pour l'approximation de $g(\lambda)$. Par conséquent, l'estimation du vecteur d'état est exécutée en deux étapes: la première utilisant $g^{-}(\lambda)$ et le résultat de cette estimation, $\hat{x}_n^{int}$, sera réutilisé en ordre inversé durant la deuxième étape utilisant $g^{+}(\lambda)$ pour reconstruire le spectre final $\hat{x}_n$.

2.2.2 Algorithme basé sur la méthode des moindres carrés moyens LMS

Dans la méthode du gradient déterministe on a:

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \mu(\mathbf{p} - \mathbf{R}\mathbf{a}_n) = \mathbf{a}_n + \mu E[e_n \mathbf{a}_n] \tag{2.33}$$

$$e_n = y_n - \mathbf{g}_n^T \mathbf{a}_n \tag{2.34}$$

avec y_n le signal de sortie du système, $\mathbf{g}_n$ sont les coefficients du filtre et $\mathbf{a}_n$ est le vecteur d'estimation. $\mathbf{R}$ est la matrice d'autocorrélation définie par : $\mathbf{R}=E\left[\mathbf{a}_n \quad \mathbf{a}_n^T\right]$ et $\mathbf{p}$ est le vecteur d'intercorrélation du vecteur $\mathbf{a}_n$ avec soit même.

Pour passer de l'algorithme du gradient déterministe Eq. (2.33) à l'algorithme traitant directement les données observées Eq. (2.34), il faut donc se débarrasser de l'opérateur de l'espérance mathématique E qui n'est pas calculable à partir des données. Pour cela le critère à minimiser sera:

$$J(\mathbf{g}) = |e_n|^2 = (\tilde{y}_n - \mathbf{g}_n^T \mathbf{a}_n)(\tilde{y}_n - \mathbf{g}_n^T \mathbf{a}_n)^T \tag{2.35}$$

$$\mathbf{g}_{n+1} = \mathbf{g}_n - \frac{1}{2}\mu\frac{d(J(\mathbf{g}))}{d\mathbf{g}} \tag{2.36}$$

$$|e_n|^2 = \left(y_n - \mathbf{g}_n^T\hat{\mathbf{a}}_n\right)\left(y_n - \mathbf{g}_n^T\hat{\mathbf{a}}_n\right)^T \tag{2.37}$$

$$\frac{d}{d\mathbf{a}}\left(|e_n|^2\right) = -2\tilde{y}_n\mathbf{g}_n + 2\mathbf{g}_n\mathbf{g}_n^T\mathbf{a}_n \tag{2.38}$$

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \mu\mathbf{g}_n(\hat{y}_n - \mathbf{g}_n^T\mathbf{a}_n) \tag{2.39}$$

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \mu\mathbf{g}_n e_n \tag{2.40}$$

$$e_n = \tilde{y}_n - \mathbf{g}_n^T\mathbf{a}_n \tag{2.41}$$

L'algorithme des moindres carrés moyens (LMS) est utilisé pour la correction de l'estimation de l'amplitude des pics. Le vecteur **a** est estimé selon l'équation récurrente Eq. (2.39). où $\mathbf{g}_n = \left[\mathbf{g}(n-\hat{l}_1)\,\mathbf{g}(n-\hat{l}_2)\cdots\right]^T$, dim(**g**)=$N_g$, $\hat{\mathbf{l}}_k$ est le vecteur des positions des pics, et μ est le coefficient de relaxation. Le choix de cet algorithme est motivé par sa similarité au filtre de Kalman utilisé pour l'estimation du vecteur d'état. La seule différence entre Eq. (2.31) et Eq. (2.39) est que cette dernière est non stationnaire, la pondération de l'erreur **g**(n) change à chaque itération.

Pour améliorer les résultats de mesure, une contrainte de positivité est appliquée sur les résultats de correction des amplitude. Ainsi, nous appliquons la contrainte de positivité sur $\mathbf{a}_n$, il s'agit d'une contrainte de positivité dure qui tout simplement élimine les valeurs négatives du vecteur. Ce qui pourrait être exprimé comme :

$$\hat{a}_n := \begin{Bmatrix} \hat{a}_n \text{ si } \hat{a}_n \geq 0 \\ 0 \text{ si } \hat{x}_n < 0 \end{Bmatrix} \tag{2.42}$$

Pour résumer l'étude algorithmique nous rappelons l'ensemble des opérations à exécuter dans l'ordre :

1. Chargement des données du modèle mathématique du système;
2. Exécution de l'algorithme KALSPL utilisant comme entrée le résultat bruité de la mesure $\tilde{y}_n$, les données du modèle du système et le nombre d'échantillon du signal d'entrée (N). $\{\hat{x}_n^{int}\}$ = KALSPL($\{\tilde{y}_n\}$, modèle du système, N), avec n=1,2,...,N;
3. Exécution de l'algorithme KALSPL utilisant comme entrée le résultat de la deuxième étape $\{\hat{x}_n^{int}$, les données du modèle du système et le nombre d'échantillon du signal d'entrée (N). $\{\hat{x}_n\}$ = POS(KALSPL($\{\hat{x}_n^{int}\}$, modèle du système, N)) avec n=N,N-1,...,1;
4. L'opérateur POS a pour but d'introduire, durant la deuxième étape de filtrage, la contrainte de positivité sur l'estimation en tronquant les valeurs négatives du premier élément dans le vecteur qui est le spectre $\hat{x}_n$;
5. Au fur et à mesure que l'on calcul les échantillons de $\hat{x}_n$, on applique la régle de la première dérivée pour la recherche du maximum d'une fonction sur chaque échantillon. Ceci permet d'estimer la position des pics durant la deuxième étape de filtrage en détectant simplement les positions où il y a un changement de signe du deuxième élément dans le vecteur d'état $\hat{\mathbf{z}}_n$;
6. La dernière étape dans le processus est la correction de l'amplitude des pics estimés durant le phase précédente (étape 5) en appliquant l'algorithme LMS.

7. Sur les éléments du vecteur d'amplitudes on introduit la contrainte de positivité pour éliminer les éléments négatifs de ce dernier.

2.3 ADAPTATION DES ALGORITHMES À L'IMPLANTATION EN TECHNOLOGIE VLSI

Cette étude permet de faire le point sur les détails dans les deux algorithmes pour mieux comprendre leur fonctionnement. Nous les avons simulé sur Matlab pour pouvoir étudier l'effet de quantification et tirer le flux de données pour en déduire l'architecture. Bien que ces deux algorithmes donnent des résultats excellents sur Matlab, leurs implantation sur ASIC n'est pas une tâche facile car il faut emmagasiner les données pour des utilisations ultérieures, ce qui demande l'association d'une mémoire externe au processeur. La dynamique des données pose un problème au niveau de la normalisation des données pour effectuer les calculs à virgule fixe, ce qui nous oblige à augmenter le nombre de bits alloués à la représentation des données stationnaires. Une simplification très importante du premier algorithme, en vue de la réduction de la surface et du temps de calcul est rendue possible, mais on ne peut pas faire la même chose avec l'autre l'algorithme LMS, ce qui rend le contrôle du processeur pas mal complexe.

2.3.1 Algorithme de KALSPL

L'équation générale du filtre de KALSPL peut s'écrire de différentes manières. Du point de vue mathématique, toutes ces écritures sont valables du moment qu'elles respectent les règles fondamentales du traitement du signal. Mais du point de vue implantation, il est nécessaire d'arranger les termes qui peuvent être mis ensemble pour pouvoir démontrer une forme très simple de l'algorithme en question. La simplicité de la forme nous permet d'arriver facilement à une architecture réduite et à un flot de données assez simple à gérer.

Ainsi, nous pouvons décomposer Eq. (2.31) comme suit :

$$\hat{\mathbf{z}}_{n+1/n} = \Phi\hat{\mathbf{z}}_n \tag{2.43}$$

$$\hat{y}_n = \mathbf{h}^T\hat{\mathbf{z}}_{n+1/n} \tag{2.44}$$

$$I_n = \tilde{y}_n - \hat{y}_n \tag{2.45}$$

$$\hat{\mathbf{z}}_{n+1} = \Phi\hat{\mathbf{z}}_n + \mathbf{k}_\infty I_n \tag{2.46}$$

L'organigramme de calcul de Eq. (2.31) est similaire à l'ordre des équations (2.43-46). Nous pouvons donc établir le flot de données de cet algorithme de la manière décrite à la figure (2.1) [1].

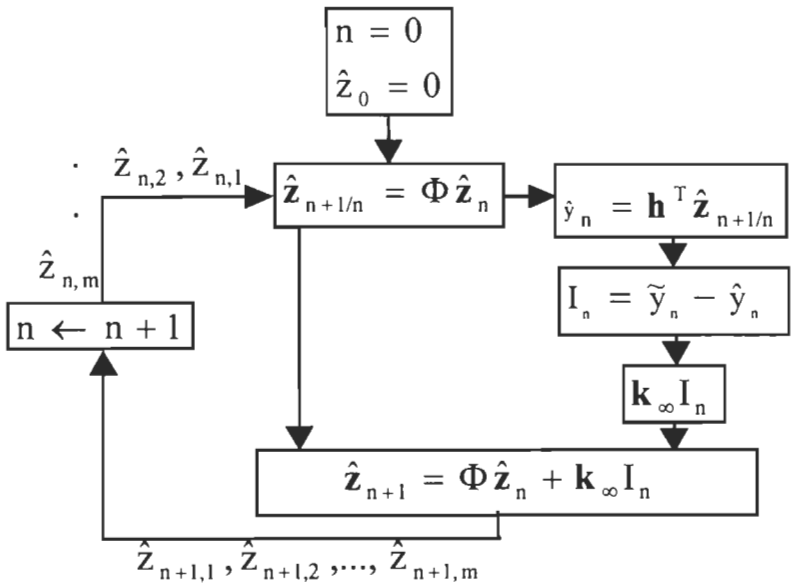

Figure 2.1 Flot de données non simplifié pour l'algorithme de KALSPL.

Grâce à la nature récursive des équations (2.31) et (2.39), les architectures systoliques sont très bien adaptées pour leur implantation. La figure (2.2) montre l'architecture systolique proposée pour le flot de données de la figure (2.1). En examinant ce flot de données, nous pouvons constater la dépendance dans le calcul entre les équations (2.43-46). Cette dépendance ralentit l'exécution de notre algorithme car malgré la rapidité du réseau systolique, les multiplications vecteur-vecteur et vecteur scalaire causent une véritable perte de temps.

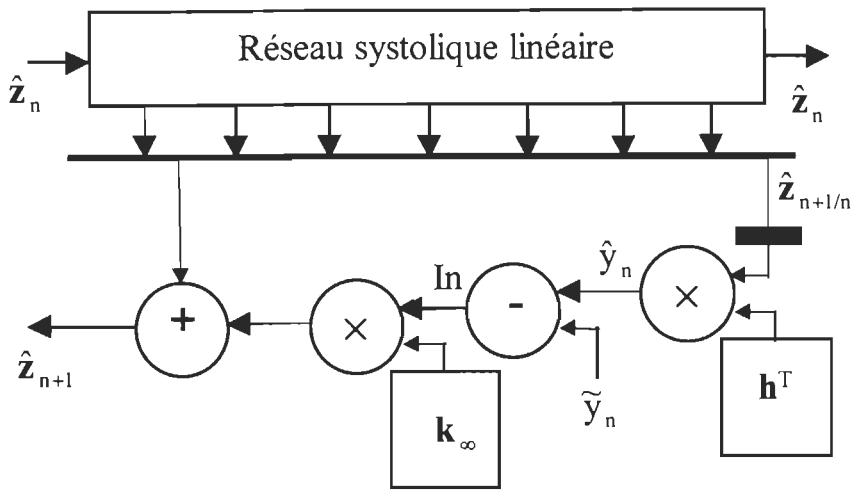

Figure 2.2 Architecture systolique proposée pour le flot de données de la Fig. (2.1).

Une bonne compréhension de Eq. (2.31) permet de la réécrire sous une autre forme, qui du point de vue implantation serait beaucoup plus profitable, et ce en éliminant la dépendance entre Eq. (2.43) et (2.44), ceci pourrait être fait en définissant :

$$\mathbf{b} = \mathbf{h}^T \Phi \tag{2.47}$$

l'Eq. (2.31) devient :

$$\hat{\mathbf{z}}_n = \Phi \hat{\mathbf{z}}_{n-1} + \mathbf{k}_\infty (\tilde{y}_{n-1} - \mathbf{b}^T \hat{\mathbf{z}}_{n-1}) \tag{2.48}$$

Sachant que $\mathbf{h}^T$ et Φ sont constantes, nous pouvons donc effectuer le calcul de $\mathbf{b}$ avant de rentrer les données au processeur. De cette façon nous éliminons la dépendance entre les deux termes de Eq.(2.31) en communiquant directement le résultat de calcul de $\hat{\mathbf{z}}_{n+1}$ aux deux branches du flot de données de la figure (2.1). Le nouveau flot de données est montré à la figure (2.3). Figure (2.4) montre l'architecture correspondante de ce flot de données.

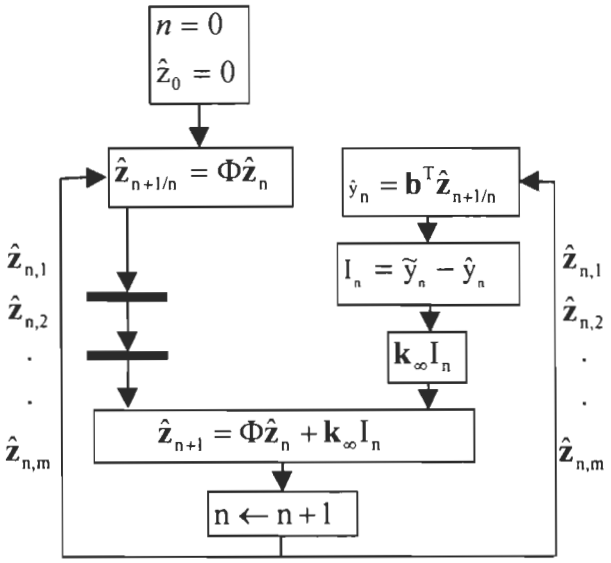

Figure 2.3 Flot de données de l'algorithme KALSPL correspondant à Eq. (2.48).

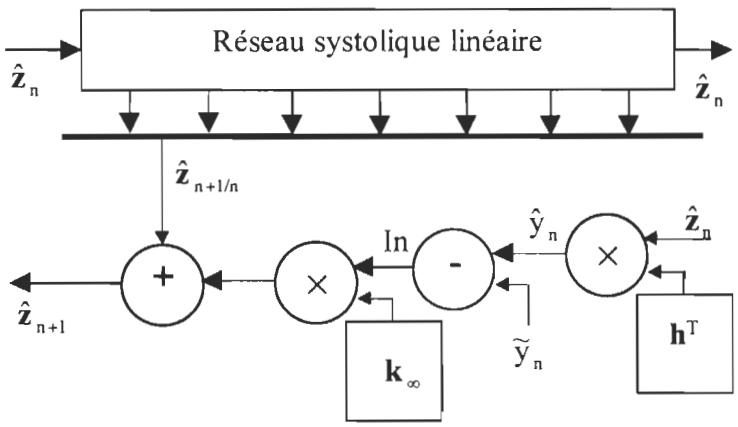

Figure 2.4 Architecture correspondante du flot de données de l'Eq. (2.48).

Pour réduire la surface d'intégration de l'algorithme, nous pouvons réarranger les termes de Eq. (2.31) sous la forme :

$$\hat{\mathbf{z}}_{n+1} = (\Phi - \mathbf{k}_\infty \mathbf{h}^T \Phi)\hat{\mathbf{z}}_n + \mathbf{k}_\infty \tilde{y}_n \tag{2.49}$$

$$\mathbf{G} = (\Phi - \mathbf{k}_\infty \mathbf{h}^T \Phi) \tag{2.50}$$

$$\hat{\mathbf{z}}_{n+1} = \mathbf{G}\hat{\mathbf{z}}_n + \mathbf{k}_\infty \tilde{\mathbf{y}}_n \tag{2.51}$$

Cette nouvelle écriture nous permet d'avoir un gain considérable au niveau surface d'intégration et temps de calcul. Le flot de données de cette nouvelle représentation est donné à la figure (2.5). La figure (2.6) montre le schéma bloc d'une architecture systolique qui pourrait être modélisée pour la mise en œuvre de Eq. (2.51).

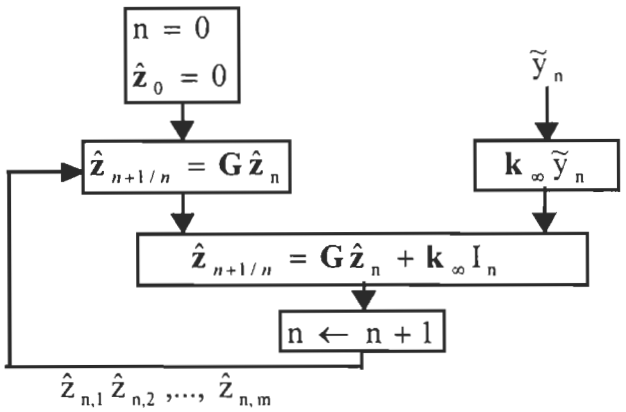

Figure 2.5 Flot de données simplifié pour KALSPL selon Eq. (2.51).

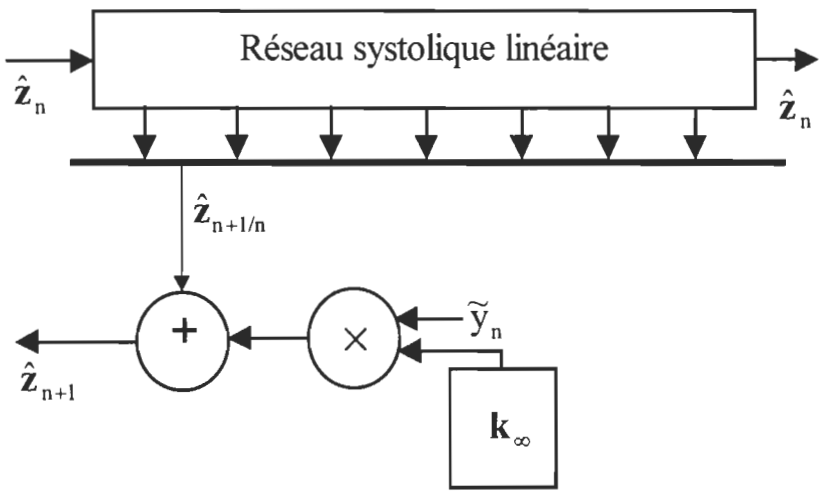

Figure 2.6 Architecture proposée pour la mise en œuvre de Eq. (2.51).

Nous pouvons chercher plus d'économie de surface en augmentant la matrice **G** d'une colonne qui sera le gain de Kalman et une ligne de zéros. Cette écriture permet de

mettre l'accent sur le réseau systolique et d'éliminer plusieurs blocs dans l'architecture de la figure (2.6). En effet nous pouvons encore une fois, et à partir de Eq. (2.51) démontrer la forme matricielle finale de notre algorithme. Nous pouvons récrire Eq. (2.51) dans une forme plus compacte:

$$\begin{bmatrix} \hat{\mathbf{z}}_n \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{k}_\infty \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{z}}_{n-1} \\ \tilde{y}_{n-1} \end{bmatrix} = \mathbf{F} \begin{bmatrix} \hat{\mathbf{z}}_{n-1} \\ \tilde{y}_{n-1} \end{bmatrix} \tag{2.52}$$

La matrice **F** est calculée en avance puisque elle ne dépend pas de $\tilde{y}_n$, avec dim(**F**)=M×(M+1). Avec ce calcul préliminaire, la complexité de calcul est réduite de $2N(M^2+2M+8)$ à $2N(M^2+M)$ [64-66]. Les opérations dans Eq. (2.52) peuvent être facilement répartis sur un réseau systolique linéaire. Le flot de données établit par cette écriture est montrée à la figure (2.7).

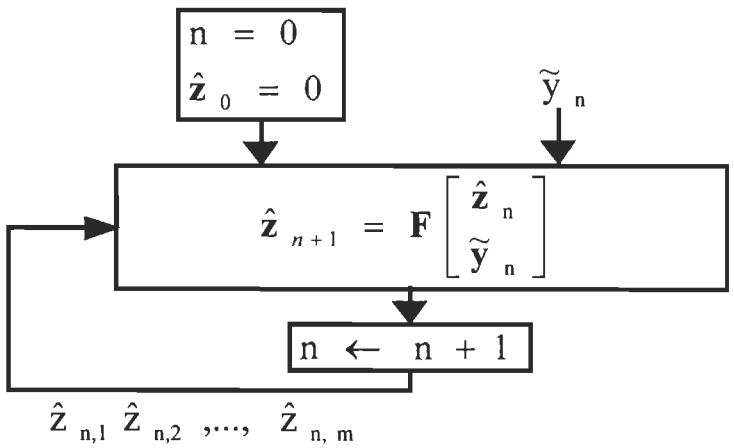

Figure 2.7 Flot de données décrit par Eq. (2.52).

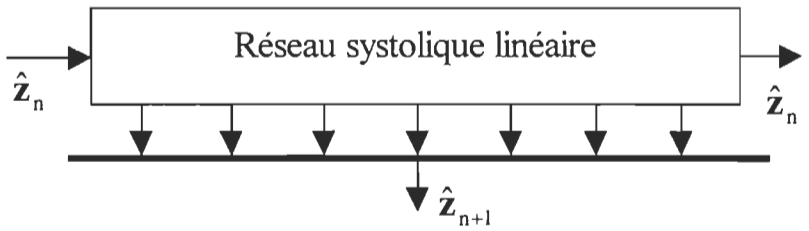

Figure 2.8 Architecture systolique proposée pour la mise en œuvre de Eq. (2.52).

Pour ce flot de données nous proposons l'architecture de la figure (2.8). Cette architecture nous permet d'atteindre des performances très considérable au niveau du temps de calcul et surface d'intégration. Les flot de données ainsi que les architectures discutées auparavant ne tiennent pas compte de la contrainte de positivité et la détection du changement de signe de la première dérivée. Le flot de donnée complet pour l'algorithme Kalman-Spline avec contrainte de positivité et estimation de position des pics est donné à la figure (2.9). Dans cette figure la fonction f(.) note la contrainte de positivité tandis que la fonction d(.) représente la détection du changement de signe pour l'estimé du première dérivée de x_n.

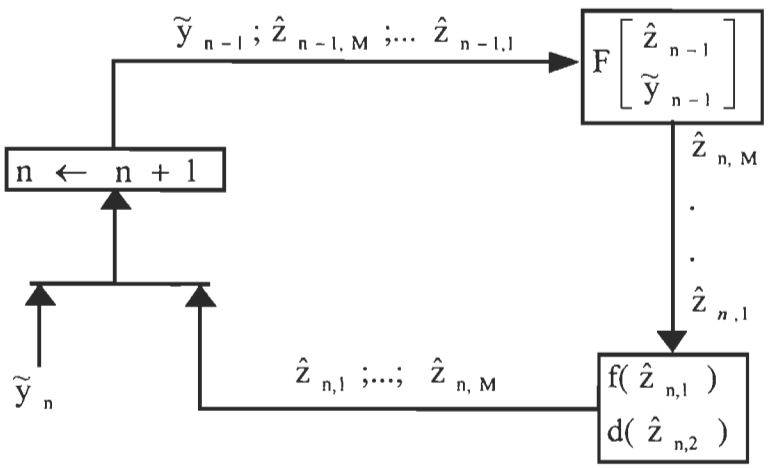

Figure 2.9 Flot de données complet de l'algorithme KALSPL.

2.3.2 Algorithme des moindres carrés moyens LMS

L'algorithme LMS est décrit par Eq. (2.39) que nous reprenons ici :

$$\mathbf{a}_{n+1} = \mathbf{a}_n + \mu \mathbf{g}_n (\hat{y}_n - \mathbf{g}_n^T \mathbf{a}_n) \tag{2.53}$$

En suivant la même procédure qu'à la section 2.3.1, nous pouvons essayer de manipuler les termes de cette équation pour pouvoir aboutir à une formulation plus simple tel que fait pour Eq. (2.31).

$$\mathbf{a}_{n+1} = (\mathbf{I} - \mu \mathbf{g}_n \mathbf{g}_n^T)\mathbf{a}_n + \mu \mathbf{g}_n \tilde{y}_n \tag{2.54}$$

$$\mathbf{a}_{n+1} = \mathbf{P}_n \mathbf{a}_n + \mathbf{r}_n \tilde{y}_n \tag{2.55}$$

$$\begin{bmatrix} \mathbf{a}_n \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_n & \mathbf{r}_n \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{a}_{n-1} \\ \tilde{y}_{n-1} \end{bmatrix} = \mathbf{T}_n \begin{bmatrix} \mathbf{a}_{n-1} \\ \tilde{y}_{n-1} \end{bmatrix} \tag{2.56}$$

L'utilisation d'un flot de données quasi identique à KALSPL nous permettra de garder le même type d'architecture. Malheureusement, la nature dynamique du vecteur $\mathbf{g}_n$ ne permet pas l'application de cette transformation sur l'algorithme LMS. L'inspection de la forme de Eq. (2.55) met en évidence le changement de la matrice $\mathbf{P}_n$ et le vecteur $\mathbf{r}_n$ à chaque itération n. L'implantation de cette équation serait trop coûteuse au niveau surface d'intégration et nombre de ports d'E/S de l'architecture. La dimension de $\mathbf{g}(\lambda)$ est relativement grande, de l'ordre 300 à 1000 points. Pour la mise en œuvre de Eq. (2.55) en utilisant une architecture systolique telle que celle de figure (2.8) il faudrait avoir l'accès simultané à toutes les données de la matrice $\mathbf{T}_n$. Ces données seront de l'ordre de 300×300 voir 1000×1000 ce qui est énorme et exige une mémoire parallèle ou plusieurs mémoires montées en parallèle avec plusieurs unité de génération d'adresse UGA. Et même avec une

mémoire parallèle le nombre de port d'entrée sortie du circuit sera la principale limitation car ce nombre dépendra du nombre de pics dans un spectre, ce nombre est variable et pourrait être limité à 20. L'utilisation d'un circuit avec une telle dimension n'est pas imaginable pour l'application visée par ce projet soit la spectrométrie. Toutefois, il est possible d'introduire une transformation semblable à celle proposée dans [30]. Ainsi, nous introduisons la transformation suivante:

$$\hat{y}_n = g_{n-1}^T \hat{a}_{n-1} \quad , \quad \hat{a}_0 = \left[\tilde{y}_{p1} \quad \tilde{y}_{p2} \quad \cdots \quad \tilde{y}_{pL}\right] \tag{2.57}$$

$$e_n = \tilde{y}_n - \hat{y}_n \tag{2.58}$$

$$I_n = \mu e_n \tag{2.59}$$

$$\hat{a}_n = \hat{a}_{n-1} + g_n I_n \tag{2.60}$$

Eqs. (2.58) et (2.59) sont faciles à exécuter car il s'agit simplement d'un calcul scalaire, mais Eq. (2.57) et (2.60) sont un peu plus compliquées car elles impliquent la multiplication de vecteur par un autre vecteur. Eqs. (48) à Eq. (51) peuvent être exécutées tel que proposé dans [30]. Le diagramme de flot de données de cet algorithme comme décrit par ces équations est donné à la figure (2.10). Pour ce flot de donnée l'architecture proposée dans [30] serait très bien adaptée pour son implantation.

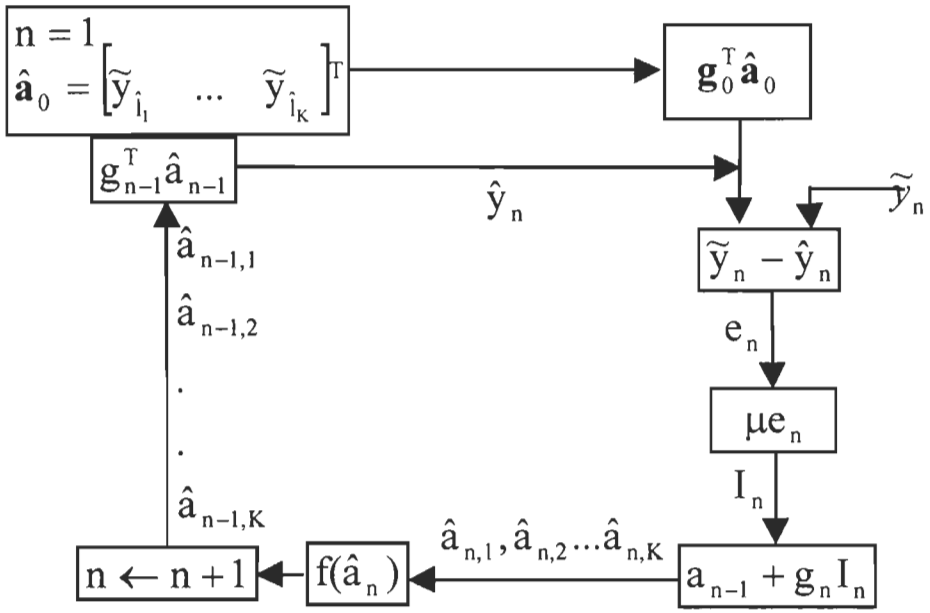

Figure 2.10 Flot de données final de l'algorithme LMS.

2.4 ÉTUDE DE QUANTIFICATION

Les algorithmes de reconstitution des signaux implantés dans des circuits numériques ne peuvent manipuler que des signaux numériques. Les amplitudes de ces signaux ne peuvent prendre qu'un nombre fini de valeurs. Il faut donc choisir un ensemble fini de nombre réels. Cet ensemble est déterminé par le nombre de bits utilisés pour représenter ces amplitudes. Ainsi, la précision de leurs représentations dépend du nombre de bits utilisés pour représenter les données et le nombre de bits utilisés pour effectuer les calculs. La plage de variation de ces amplitudes est $[A_{min}, A_{max}]$, où A_{min} et A_{max} représentent respectivement la plus petite et la plus grande amplitude représentable. Ces caractéristiques définissent la dynamique des signaux [31-32].

2.4.1 Représentation numérique des nombres réels

La représentation numérique et le type d'arithmétique utilisé a une influence significative sur le comportement et l'exécution de l'architecture pour tous les algorithmes. Dans une réalisation de circuits à applications spécifiques, pour assurer le support architectural des algorithmes de traitement numérique, on doit choisir le type de représentation convenable à l'application pour assurer la précision requise. En pratique, nous pouvons faire le choix entre deux types, soit ; la représentation en virgule flottante, ou la représentation en virgule fixe. Dans la représentation en virgule flottante, on consacre certain nombre de bits, généralement assez élevé pour pouvoir représenter fidèlement les chiffres sur une grande plage. De toute évidence la représentation en virgule flottante permet une plus grande exactitude dans le traitement des données numérisées comparativement à la représentation en virgule fixe. En contre partie, la représentation en virgule flottante coûte beaucoup plus cher en temps de calcul et en surface d'intégration que la représentation en virgule fixe. De plus, il n'est pas toujours justifié d'utiliser la représentation en virgule flottante pour exécuter les algorithmes [33]. Quoique la représentation en virgule flottante est plus précise que la représentation en virgule fixe, elle demeure valable uniquement pour la simulation et la validation du niveau de précision de la représentation en virgule fixe, puisqu'on l'utilise comme référence.

La précision est l'un des principaux soucis lors de l'implantation des algorithmes de traitement numérique, mais elle est loin d'être le seul, car la surface d'intégration, le temps de calcul et la puissance consommée sont des critères aussi importants, selon l'application

ciblée. Ces quatres critères trouvent une certaine satisfaction lors de la représentation en virgule fixe à condition que le nombre de bits soit raisonnable.

Le point de départ pour la représentation en virgule fixe est une suite de valeurs représentables et équidistantes sur l'axe réel. En principe, le nombre de points et la distance entre points voisins est arbitraire, mais le plus souvent on choisit l'ensemble symétrique par rapport à l'origine. Évidemment le nombre de valeurs représentables détermine le nombre de bits qui sont nécessaires pour leur représentation. Plus précisément, si la dynamique des signaux numériques est l'intervalle [-1, +1], alors l'ensemble des valeurs représentables par NBX bits est, à quelques détails près, $\{-1,...,-2\times 2^{-(NBX-1)},-2^{-(NBX-1)},0,2^{-(NBX-1)},2\times 2^{-NBX(-1)},...,1\}$. Les détails dépendent de la représentation précise que l'on adopte. La façon la plus naturelle consiste à réserver un bit pour le signe et d'écrire avec les NBX bits restant le module comme fraction binaire (Fig. (2.11)). C'est la représentation par signe et module qui s'exprime par l'équation suivante :

$$\text{valeur} = (-1)^{c_0} \sum_{i=1}^{NBX-1} C_i 2^i \qquad (2.61)$$

On remarque qu'en utilisant le signe et le module, on ne parvient pas à représenter +1 ni à représenter -1 dans l'ensemble de Eq. (2.61).

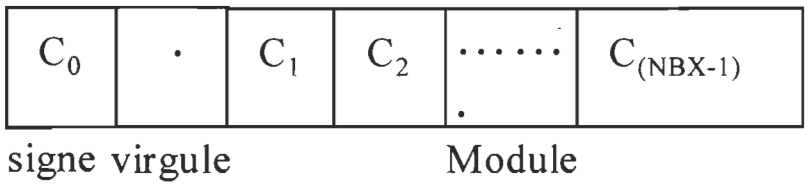

Figure 2.11 Représentation en virgule fixe en signe et module.

Une autre méthode, la représentation par complément à 2, est très répandue, parce qu'elle présente des avantages pour les opérations arithmétiques. Elle consiste à remplacer le nombre négatif $-\Omega$ par $2-\Omega$ et à développer ce dernier comme fraction binaire (Fig. (2.12)).

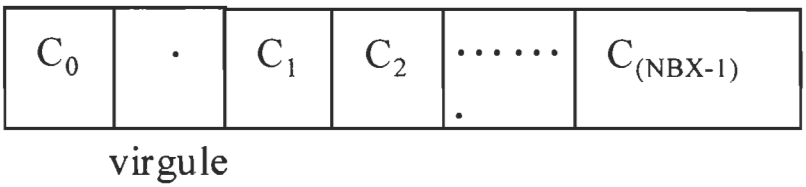

Figure 2.12 Représentation en virgule fixe en complément à deux.

$$\text{valeur} = C_0 \sum_{i=1}^{NBX-1} C_i 2^i \tag{2.62}$$

La valeur zéro possède la double représentation 00 ...0 et 10 ...0. La plage de variation des signaux, ou la dynamique, n'est donc pas $[-1,+1]$, mais $\left[(-1+2^{-(NBX-1)}),(+1-2^{-(NBX-1)})\right]$.

Dans le cas général d'un ensemble de valeurs représentables :

$$\{X_{min}, X_{min}+q, X_{min}+2q, \ldots, X_{max}-2q, X_{max}-q, X_{max}\} \tag{2.63}$$

on établit la correspondance bijective avec l'ensemble qui respecte l'ordre et on utilise, pour chaque valeur de Eq. (2.62), la représentation binaire de l'élément correspondant de Eq. (2.61). Par conséquent, la distance q entre deux valeurs successives, définissant le pas de quantification, est liée au nombre de bits de la représentation par signe et module :

$$q = \frac{X_{\max} - X_{\min}}{2^{(NBX-1)}}$$

On remarque qu'il y a toujours un nombre impair de valeurs représentables. Dans la représentation par complément à 2, la valeur -1 devient 10 ...0, mais +1 reste sans représentation. Par contre, la valeur possède la représentation unique 00 ...0, ce qui est un avantage certain. La dynamique exacte est donc $[-1, 1 - 2^{-(NBX-1)}]$. La relation entre le pas de quantification et le nombre de bits, pour l'ensemble général des valeurs représentables Eq. (2.63), est :

$$q = \frac{X_{\max} - X_{\min}}{2^{NBX} - 1}$$

Ainsi, il y a un nombre pair de valeurs représentables. Dans le cas le plus fréquent, où des 2n valeurs représentables n-1 sont négatives, n-2 positives et la dernière zéro, on convient que la dynamique est $\left[X_{mim}, \left|X_{\min}\right|\right]$, bien qu'en appliquant rigoureusement la définition, elle sera de $\left[X_{mim}, \left|X_{\min}\right| - q\right]$ [1] [31-32].

2.4.2 Analyse des résultats de simulation

Un des points les plus importants à discuter avant d'aborder la conception du circuit intégré est l'étude de quantification. Cette dernière permet de conclure sur le type de représentation, le nombre de bits nécessaire pour représenter les données et la taille des composantes arithmétiques telle que le multiplieur accumulateur.

Une étude de quantification a été effectuée pour déterminer la largeur des mots nécessaire pour représenter les données et les blocs de calcul et ainsi assurer une bonne précision utilisant un calcul à virgule fixe et une représentation en complément à deux. Nous avons réalisé cette étude avec le logiciel MATLAB, qui nous permet, dans un premier lieu, de simuler la fonctionnalité de l'algorithme en effectuant les calculs à virgule flottante. Par la suite nous avons modifié les programmes utilisés dans la première étape pour quantifier les signaux. Ainsi nous avons rajouté un programme de quantification de matrice et un autre pour la quantification des vecteurs. Ensuite nous avons réalisé un autre programme qui fait la multiplication matrice-matrice élément par élément et la quantification du résultat après chaque opération arithmétique élémentaire. Comme dernière étape nous avons réalisé un autre programme très simple pour le calcul de l'erreur relative de quantification. Cette dernière est définie comme étant la différence entre le signal estimé avec un certain nombre de bits NBX $\hat{x}_{NBX}(n)$ et le signal original sans bruit $\dot{x}(n)$.

Tous les programme utilisés durant cette étude sont présentés en annexe, le programme principal fait appel à d'autres programmes développés par Mohamed Ben-

Slima, entre autres le programme de formulation du modèle et de l'estimation des paramètres. La qualité de la reconstitution est évaluée selon l'erreur quadratique moyenne relative :

$$\varepsilon(\dot{x},\hat{x}_{\infty}) = \frac{\left\|\{\hat{x}_{\infty,n}\}-\{\dot{x}_n\}\right\|_2}{\left\|\{\dot{x}_{\infty,n}\}\right\|_2} \tag{2.64}$$

$$\varepsilon(\dot{x},\hat{x}_{q}) = \frac{\left\|\{\hat{x}_{q,n}\}-\{\dot{x}_n\}\right\|_2}{\left\|\{\dot{x}_{\infty,n}\}\right\|_2} \tag{2.65}$$

$$\varepsilon(\dot{x}_{\infty},\hat{x}_{q}) = \frac{\left\|\{\hat{x}_{q,n}\}-\{\dot{x}_{\infty,n}\}\right\|_2}{\left\|\{\dot{x}_{\infty,n}\}\right\|_2} \tag{2.66}$$

Les figures (2.13) et (2.14) donnent les résultats de quantification exprimés en terme d'erreur quadratique relative moyenne Eq. (2.65), ε, pour l'algorithme de KALSPL et LMS respectivement. Ces résultats sont obtenus pour différents nombre de bits utilisés pour les blocs de calcul (CBN) des variables $\mathbf{z}_n$ et $\mathbf{a}_n$ et différents nombre de bits utilisés pour la représentation des données (DBN) soit le modèle mathématique du système **F**, sa réponse impulsionnelle $\mathbf{g}_n$ et la mesure $\tilde{y}_n$. Ces résultats montrent que 24 bits pour les blocs arithmétiques et la représentation des données sont nécessaire pour garantir une précision acceptable pour l'estimation des paramètres du spectrogramme. Un exemple de signaux quantifiés est donné aux figures (2.15-17).

Dans cet exemple on utilise des données synthétiques avec :

- un signal d'entrée $\tilde{y}_n$ de N= 300 points;

- m=5 pics,
- un modèle mathématique d'ordre sept, $M=M_v+4=7$.

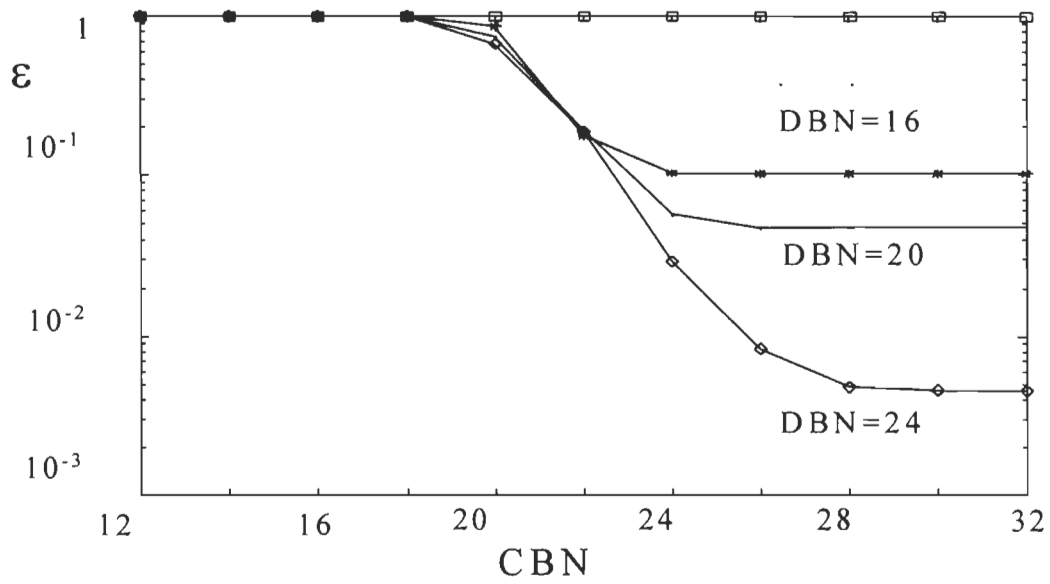

Figure 2.13 Résultats de quantification de l'algorithme de KALSPL.

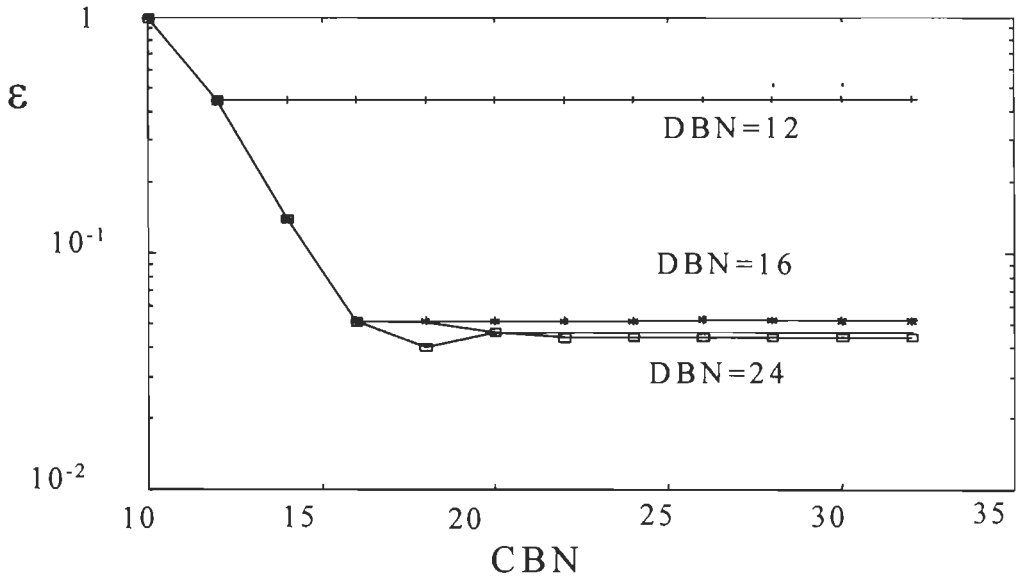

Figure 2.14 Résultats de quantification de l'algorithme LMS.

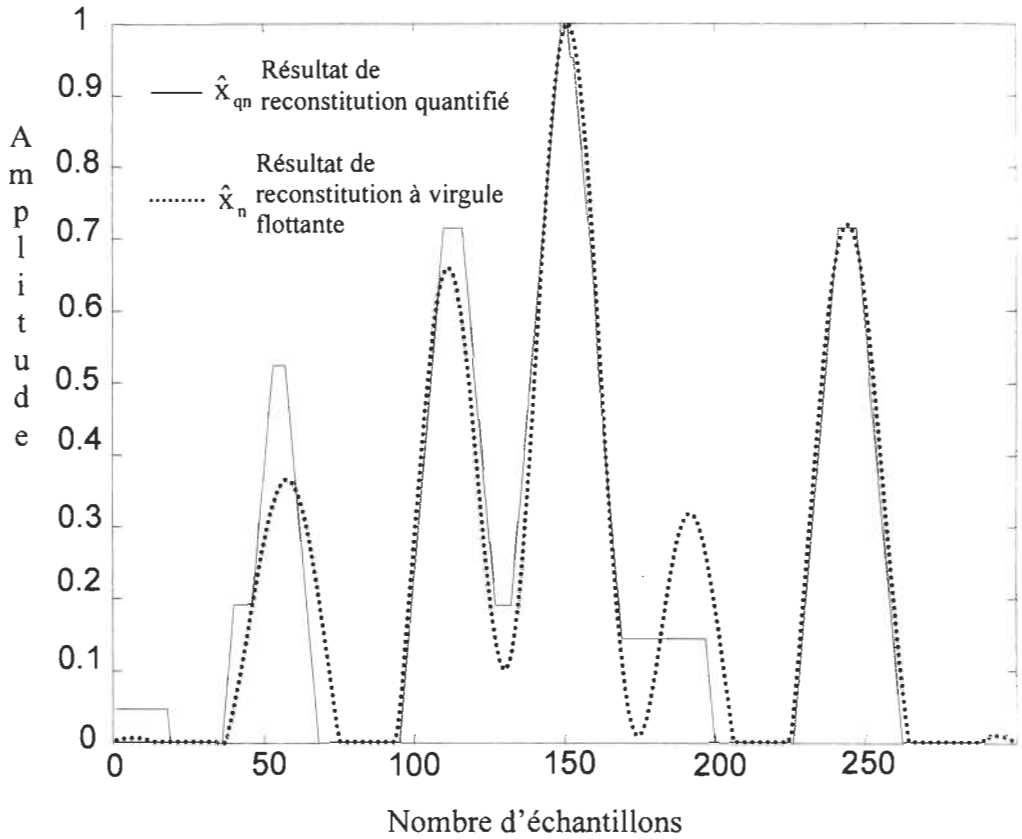

Figure 2.15 Exemple de signaux synthétiques quantifiés DBN=16 et CBN=16.

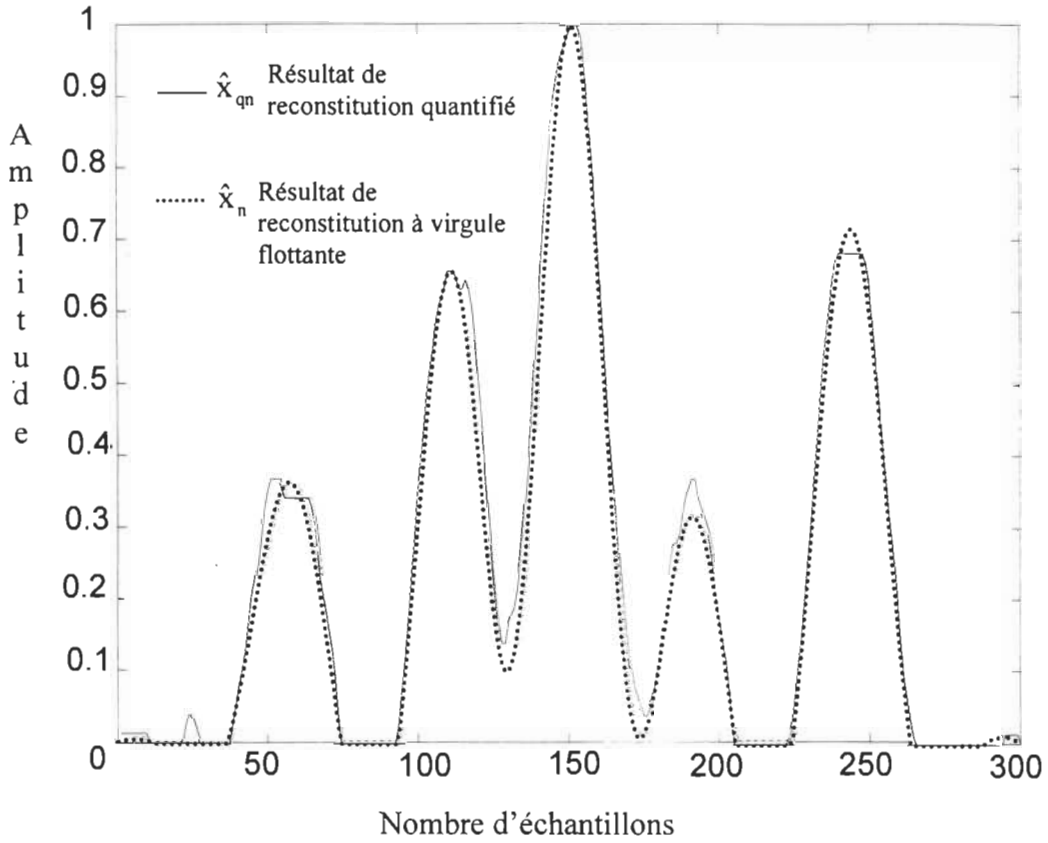

Figure 2.16 Exemple de signaux synthétiques quantifiés DBN=16 et CBN=24.

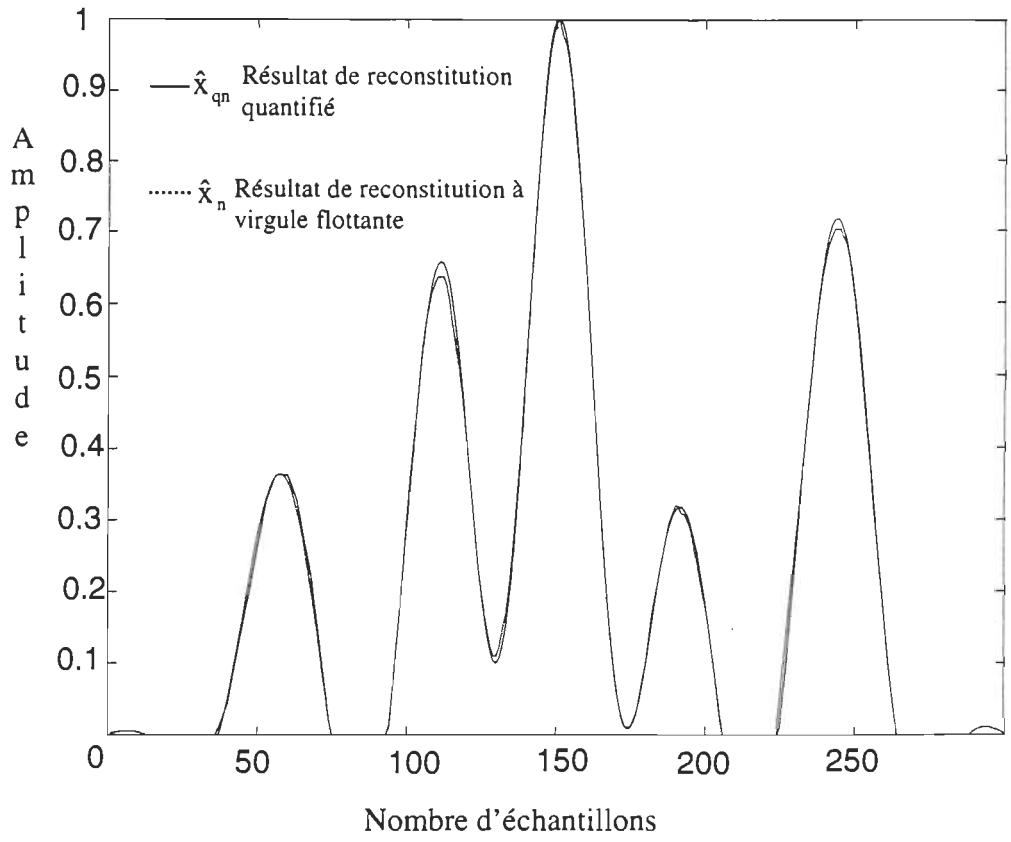

Figure 2.17 Exemple de signaux synthétiques quantifiés DBN=24 et CBN=24.

2.5 RÉSULTATS ET DISCUSSION

Cette étude a permis de mettre le point sur les détails dans les deux algorithmes, pour mieux comprendre leur fonctionnement. Une bonne compréhension des algorithmes à implanter et nécessaire pour la proposition d'architectures simples et efficaces profitant de la particularité de chaque algorithme. Nous avons simulé les algorithmes KALSPL et LMS sur Matlab pour pouvoir étudier l'effet de quantification et tirer le flux de données pour en

déduire l'architecture. Bien que ces deux algorithmes donnent des résultats excellents sur Matlab, leurs implantation sur ASIC n'est pas une tâche facile car il faut emmagasiner les données pour des utilisations ultérieures, ce qui demande l'association d'une mémoire externe au processeur. La dynamique des données pose un problème au niveau de la normalisation des données pour effectuer les calculs à virgule fixe, ce qui nous oblige à augmenter le nombre de bits alloués à la représentation des données stationnaires.

Une transformation du premier algorithme a permis de simplifier le flot de données de ce dernier. Cette transformation permet de réaliser des réductions substantielles au niveau de la surface d'intégration et du temps de calcul. Tandis qu'au niveau de la précision en utilisant le calcul à virgule fixe, les résultats de simulation ont montré que la forme transformée exige plus de bits pour représenter les données et les variables pour assurer une précision acceptable. En effet la transformation introduite dans l'équation (2.52) cause un élargissement de la plage de variation des constantes, donc la dynamique des valeurs de la matrice **F** et du vecteur **k.**

Dans la version de Kalman-spline la matrice Φ est dense, et nous pourrons proposer l'utilisation d'un réseau systolique linéaire pour effectuer le produit matrice-vecteur. Pour profiter du parallélisme inhérent dans les deux algorithmes et réaliser une bonne conception d'architecture systolique, des transformations ont étés introduites sur l'écriture des équations décrivant l'algorithme LMS. Toutes les opérations nécessaires dans les deux algorithmes peuvent être efficacement réparties sur l'ensemble des Processeurs

Élémentaires PE du réseau systolique linéaire avec des contraintes E/S (ENTRÉE-SORTIE) très réalistes.

Les critères qui seront utilisés durant la phase d'évaluation des performances seront en premier lieu la précision et la surface d'intégration et par la suite le temps de calcul. Pour cette raison nous sommes appelés à faire un compromis entre la surface d'intégration et la précision, ce compromis s'illustre dans la définition d'une erreur acceptable qui permettra une bonne précision des résultats délivrés par le processeur avec le minimum de bits possible.

Chapitre 3

Proposition d'une Architecture Systolique

3.1 INTRODUCTION

Le développement technologique et l'évolution rapide de l'intégration à très grande échelle (VLSI) ont donné naissance aux processeurs numériques de traitement de signal. Ces processeurs, de plus en plus spécialisés, ont pris le nom d'ASIC 'Application Specific Integrated Circuit'. Ce sont tout simplement des circuits intégrés conçus pour un objectif spécifique et qui regroupent, sur une même puce, l'ensemble des éléments fonctionnels nécessaires à cette tâche.

Les exigences concernant la miniaturisation des systèmes, leur connectivité, la puissance consommée et leur fréquence de fonctionnement sont les principales raisons de

leur intégration en technologie VLSI. Cette intégration permet de réduire les coûts et le temps de conception en plus de faciliter le développement et la maintenance, remplaçant ainsi les cartes à circuit imprimées contenant de nombreux circuits commerciaux par un seul ASIC [34]. À la différence des circuits à usage général, qui présentent une flexibilité d'utilisation suffisante aux traitement numérique de signaux grâce à leur jeu d'instructions, les ASICs sont optimisés pour des algorithmes spécialisés et dédiés à répondre à des besoins de plus en plus croissants des applications hautement spécialisées entre autres : la spectrométrie, les télécommunications, le contrôle, etc.

La progression technologique que connaît la technologie VLSI jumelée aux nouveaux développements des outils de conception assistée par ordinateur 'CAD tools' et des langages de description du matériel 'HDL' permettent un design, à la fois efficace et intelligent avec une facilité exceptionnelle des systèmes de très grande complexité.

Dépendamment des exigences concernant la précision, la vitesse, la surface d'intégration et l'environnement matériel souhaité, la mise en oeuvre des algorithmes décris dans le chapitre 2 peut se faire suivant une variété de processeurs de signaux (Micro-Contrôleur, DSPs, FPGA, et ASICs). Toutes ces solutions ont leurs avantages et leurs faiblesses [35]:

- Les processeurs à usage général (MCs, DSPs, etc.) disponibles sur le marché sont moins chers et plus flexibles, mais ils sont limités du point de vue précision et vitesse.

- Les processeurs dédiés sont souvent mieux adaptés à la résolution d'une classe de problèmes, mais ils coûtent plus chers et ils sont moins flexibles si un changement dans l'algorithme s'impose.

Pour tirer avantage du parallélisme inhérent dans les deux algorithmes du chapitre 2, nous optons pour le choix de l'ASIC. Ce choix est motivé par la vitesse et la précision requises pour les nombreuses applications visées par les deux algorithmes notamment dans les domaines médical, environnemental et industriel.

Généralement le cycle de développement d'un circuit dédié à une classe d'applications se décompose en quatre étapes majeures, soit définition des tâches, conception, modélisation et synthèse.

3.2 DÉFINITION DES TÂCHES

Cette étape permet de déterminer les performances exigées par les applications cibles. C'est durant cette phase que l'on doit prendre la décision concernant le type de processeur à utiliser. Cette décision tient compte du type de données à traiter, le temps de traitement, le niveau de précision des résultats, l'environnement matériel disponible et les coûts. L'évaluation requise pour cette étape est fondamentale car elle permet de faire les ajustements nécessaires pour rencontrer les objectifs préétablis.

Les algorithmes de reconstitution présentés au chapitre 2 nécessitent un traitement numérique de données relativement fort complexe. La complexité provient de l'implication de multiplication matrice-vecteur, ainsi que d'autres opérations élémentaires. Ces algorithmes prennent une certaine importance au niveau pratique lorsqu'ils rencontrent les deux critères : performance au niveau théorique (simulation) et possibilité d'implantation en technologie ITGE (VLSI).

Les développements que connaît la micro-électronique permettent de concrétiser certains types d'architectures au niveau intégration. Le passage d'une technologie de $0.8\mu m$ à une de l'ordre de $0.2\mu m$ permet d'envisager l'implantation réaliste d'architecture fortement parallèle, comme les architectures systoliques, VLIW, RISC, SISC dont la contribution est vitale pour certaines classes d'algorithmes où la performance est définie par des exigences très strictes au niveau précision, surface d'intégration, temps de calcul, puissance consommée et coût.

Il est clair que les contraintes, ci haut énumérées sont difficiles à satisfaire voire même impossible, car les solutions qui peuvent être proposées pour rencontrer les unes sont en contradiction avec d'autres. Ainsi, pour augmenter la précision de calcul et réduire le temps de traitement des données d'un algorithme dans un processeur de signaux, on peut envisager :

- l'augmentation du nombre de bits pour mieux représenter les données, et du même fait réduire l'erreur de quantification.

- exploiter le parallélisme pour augmenter la vitesse en ajoutant d'autres unités de traitement arithmétique.

Cependant, le coût en surface d'intégration et en puissance consommée peuvent devenir critique.

Le processeur à concevoir doit utiliser le modèle mathématique du système, les résultats de mesure délivrés par le convertisseur analogique-numérique (A/N) et la réponse impulsionnelle du système pour réaliser les opérations principales des deux algorithmes soit, le filtrage du signal mesuré, la reconstitution du mesurande, l'estimation de la position des pics et la correction d'amplitude.

Le processeur aura à estimer les paramètres du spectrogramme à étudier et délivrer le résultat final de la signature spectrale, (positions et amplitudes des pics) [35]. La similitude entre les deux algorithmes KALSPL et LMS nous permet de les implanter dans le même processeur et ainsi utiliser la même architecture pour les deux. Cette manoeuvre nous permet de réduire les coûts de façon substantielle.

La méthodologie de conception doit tenir compte de la puissance consommée qui est directement liée au nombre de bits et aux ressources utilisés. Cette étape est détaillée à la section 2.4 du chapitre 2. L'analyse des deux algorithmes nous permet d'évaluer les besoins du point de vue implantation en technologie VLSI. En effet, nous pouvons

facilement prévoir la nécessité d'une mémoire de taille de $3N + R$ avec N le nombre d'échantillons et R le nombre de résultats à délivrer.

L'évolution de la technologie des circuits intégrés conduit à imposer de nouveaux développements architecturaux pour augmenter l'efficacité des processeurs et ainsi établir des critères d'évaluation. L'architecture à proposer doit permettre de satisfaire les critères d'évaluation d'une implantation [1], à savoir :

- la performance qui se traduit par le temps de calcul nécessaire pour réaliser une reconstitution pour tous les points d'échantillonnage,
- le niveau de précision acceptable,
- la surface d'intégration,
- la facilité de réalisation de l'architecture dédiée,
- les caractéristiques électriques,
- le temps de réalisation du circuit.

Le temps de réalisation est fonction de la complexité des outils et des méthodes disponibles pour la conception par ordinateur. Il est donc difficile de l'évaluer à l'avance. Cependant, posséder une connaissance approfondie de l'algorithme à implanter est un point essentiel pour arriver à une architecture satisfaisant les critères d'évaluation [1].

3.3 CONCEPTION D'UNE ARCHITECTURE DÉDIÉE

Au chapitre 2, nous avons établi que la partie dominante dans les calculs des deux algorithmes est la multiplication matrice-vecteur décrite par Eq. (2. 52) et (2. 53). Les architectures systoliques sont très bien adaptées pour ce genre de calcul et se prêtent bien à une implantation en VLSI.

Le concept de réseau systolique a été proposé pour la première fois par H.T.Kung et C.E. Leiserson en 1979, réalisant une extension du pipeline synchrone [36]. Ces réseaux consistent en un certain nombre de cellules élémentaires interconnectées ensembles, chacune a la tâche d'exécuter un calcul très simple. Le travail de ces cellules est synchronisé par l'horloge du système, d'où le nom systolique faisant allusion à la circulation sanguine dans le corps humain. Les données se présentent devant les cellules élémentaires qui les utilisent pour réaliser le traitement requis.

Ces cellules sont dites élémentaires car le traitement qui leur est confié ne dépasse pas les simples opérations de lecture des données issues des cellules situées en amont, évaluation d'une série de résultats et communication de ceux-ci vers les cellules situées en aval. Les informations circulent de manière synchrone orchestrées par l'horloge.

La simplicité, la régularité des communications et le contrôle décentralisé des cellules élémentaires permettent leurs interconnexions sous diverses configurations

(linéaire, arbre, etc.). Dans la littérature on peut trouver plusieurs propositions d'architectures et d'études de leurs connexions [36-40].

Les réseaux systoliques sont des processeurs intégrés spécialisés, dont les caractéristiques dominantes sont le parallélisme massif, le mode opératoire synchrone, le contrôle décentralisé, la communication localisée, la régularité et la modularité.

3.3.1 Parallélisme massif

Il y a essentiellement deux façons de bâtir un système rapide. La plus conventionnelle est de disposer de composantes plus rapides, mais cette approche est fortement limitée par la technologie. L'autre façon de faire est de construire un système parallèle basé sur le principe de décomposition du problème en un nombre de sous problèmes traités en parallèle.

Le concept de parallélisme rompt l'approche classique qui consiste à gagner de la vitesse en effectuant plus rapidement chaque opération. En calcul parallèle, le gain de vitesse provient de la réalisation simultanée de plusieurs opérations. Plusieurs modèles d'architecture parallèles ont été proposés [36].

Le degré de parallélisme d'un processeur dédié est fonction de l'algorithme à implanter. On peut accomplir un parallélisme massif et efficace uniquement si l'algorithme permet d'introduire un pipeline rentable et une réalisation simultanée et indépendante des

opérations. Lorsqu'on cherche à tirer partie d'une architecture parallèle pour exécuter un algorithme donné, le problème principal peut se résumer de la façon suivante: comment répartir l'algorithme sur les processeurs, de telle sorte qu'ils soient tous actifs en permanence pendant la durée de l'exécution de l'algorithme, et ce, en effectuant du travail utile [36].

L'optimisation et la gestion efficace des ressources est une condition nécessaire pour un parallélisme rentable. L'optimisation doit tenir compte de l'ensemble des tâches à réaliser et non seulement d'une partie car il est nécessaire de justifier le parallélisme en terme d'amélioration des performances, par exemple la réduction considérable du temps de calcul.

Pour le flot de données de la figure (2. 7) nous pouvons proposer l'architecture parallèle suivante de la figure (3.1).

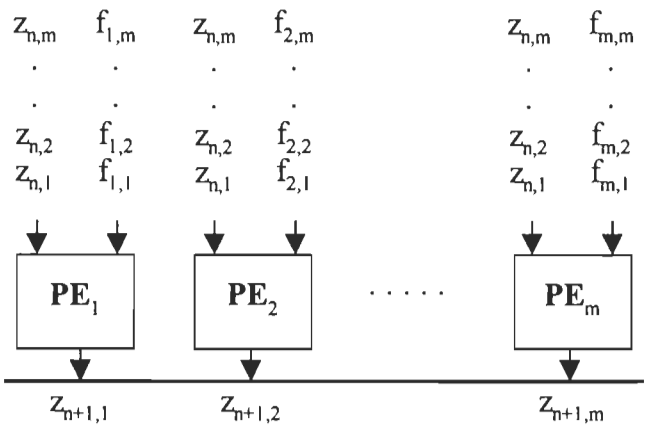

Figure 3.1 Architecture parallèle pour les algorithmes KALSPL et LMS

Le nombre de PE est fonction de l'ordre du modèle mathématique du système et du nombre de pics dans le spectrogramme à étudier. L'ordre du modèle mathématique du spectromètre est de M_v+4 avec $3 \leq Mg \leq 6$, M_v est le nombre de fonctions nécessaire pour l'approximation de la réponse impulsionnelle du système. Chaque processeur élémentaire aura à calculer un élément du vecteur $\mathbf{z}_n$ correspondant à sa position et émettre le résultat sur le bus de donnée au bon moment. L'architecture des PE sera décrite à la section 3.3.7

Le nombre de pics des spectrogrammes visés par cette application varie entre $0<m<20$. Il faut donc se baser sur l'ordre du modèle mathématique du système pour fixer le nombre de processeur élémentaire à mettre en parallèle, ce qui limite le nombre de pics à traiter par le processeur à M_v+4.

Pour remédier à cela, nous posons l'hypothèse que dans le cas où le nombre d'amplitude de pic à traiter est supérieur à M avec $M=M_v+4$, on pourrait calculer les *m* premières amplitudes de façon normale, et aussitôt que l'on détecte le pic suivant on laisse tomber le premier et on amorce le calcul des (M+1) restant et ainsi de suite; de cette façon le vecteur des amplitudes pourrait être calculé de manière séquentielle quelque soit sa longueur. La séquence de calcul pourrait être décrite par : $\{a(1),...,a(M)\},\{a(2),...,a(M+1)\},\{a(3),...,a(M+2)\}$, etc. Cette hypothèse a été validée dans le logiciel MATLAB et les résultats sont très satisfaisants.

Le traitement effectué par cette architecture est hautement parallèle. Les éléments du vecteur $\mathbf{z}_n$ sont tous calculés en même temps de façon indépendante. Les paramètres du système ainsi que les résultats du calcul précédent sont directement envoyés en parallèle aux PEs. Cette architecture permet de réduire le temps de calcul en procédant à une décomposition du problème global, qui est la multiplication matrice-vecteur, en plusieurs sous problèmes (multiplication vecteur ligne par vecteur colonne). Quand plusieurs éléments fonctionnent simultanément, la coordination et la communication deviennent des aspects de première importance, surtout dans la technologie VLSI où le coût du routage domine largement sur la puissance, le temps et la surface requis pour faire l'implantation des algorithmes.

3.3.2 Mode opératoire synchrone et pipeliné

L'architecture de la figure (3.1) permet le calcul parallèle des éléments du vecteur $\mathbf{z}_n$. Ces éléments ne seront pas tous utilisés en même temps mais plutôt en ordre, un après l'autre, ce qui permet d'imposer un fonctionnement pipeliné au réseau de processeurs. De plus, on peut éliminer les connexions entre les PEs et le bus de donnée et ne garder que celle reliant ce dernier au premier processeur élémentaire. De cette façon nous définissons l'entrée du réseau pipeliné et sa sortie. Le fonctionnement pipeliné est l'une des caractéristiques les plus importantes dans l'approche systolique. Il offre la possibilité d'utiliser la même donnée par tous les processeurs élémentaires, et ainsi équilibrer le rapport entre le nombre d'entrée-sortie et la charge de calcul.

La figure (3.2) montre l'architecture avec les modifications nécessaires pour rencontrer le mode opératoire désiré. Le fonctionnement de tous les processeurs élémentaires est orchestré par l'horloge du système. Les données seront utilisées par le premier processeur durant le cycle d'horloge (n), par le suivant durant le cycle suivant (n+1) et ainsi de suite.

Le fonctionnement pipeliné introduit une latence d'ordre M+1. Cette dernière est due au fait que le processeur doit effectuer toutes les multiplications et accumulations des résultats intermédiaires avant de sortir le premier résultat final. Mais une fois que les M premier cycles d'horloge seront écoulés, nous aurons un résultat à la sortie à chaque cycle d'horloge d'où un débit égal à la fréquence d'horloge. Cette architecture donne les mêmes performances que l'architecture de la figure (3.1) puisque le pipeline sera plein en permanence.

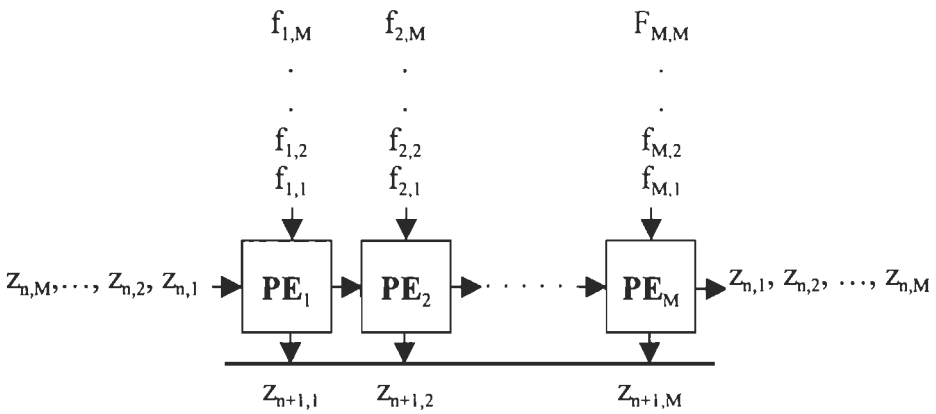

Figure 3.2 Architecture piplinée de la Fig. 3.1

Les données en direction des PEs doivent être retardées par un délai correspondant à la position du processeur moins un ($D_d = P-1$), avec D_d est le retard et P est la position du processeur élémentaire. Ce retard est nécessaire pour que les opérations soient effectuées dans le bon ordre. Le retard peut être créé de différentes manières, soit par l'insertion de registre ou par un signal de commande pour activer les éléments de mémoire qui contient les données.

3.3.3 Contrôle décentralisé

Le contrôle dans les architectures conventionnelles tel que l'architecture Von Neumann ou Harvard est centralisé dans l'unité de contrôle. Ce sont des architectures programmables avec un jeu d'instruction bien défini qui leur permet un haut niveau de flexibilité. Dans le cas d'architectures systoliques, le contrôle est décentralisé car ce genre d'architecture n'est pas programmable et les instructions sont définies durant la conception et contenues dans les processeurs élémentaires. Dans la conception de notre processeur, certains signaux de contrôle sont générés par la machine à état, qui a pour but de guider le processeur dans son chemin pour respecter les flots de donnée établis au chapitre 2.

L'emploi de la même architecture pour les deux algorithmes nous oblige à utiliser un contrôle synchronisé sur l'horloge et d'avoir recours à l'usage de plusieurs signaux de contrôle. À l'exception de quelques uns, tous les signaux de contrôle sont communiqués au premier processeur élémentaire qui lui les communique aux suivants via des registres de

temporisation ou de synchronisation car tous les processeurs fonctionnent de la même manière et exécutent les mêmes tâches d'une façon pipelinée. Dans ce cas on parle de contrôle systolisé.

L'effort de contrôle dans les architectures systoliques se fait au moment de la conception, car durant cette phase on doit définir un réseau systolique composé de processeurs élémentaires qui devront, en fonction de l'emplacement des données et des connexions du réseau, réaliser le bon traitement sur la bonne donnée et sortir le résultat au bon moment. Il s'agit là d'une tâche extrêmement complexe, car il n'est pas évident de définir une même architecture systolique pour deux algorithmes avec deux flots de données différents. La similitude entre les deux équations est plus au niveau écriture qu'au niveau du flot de données.

3.3.4 Communications localisées

En examinant de près le problème de décomposition d'une tâche globale en plusieurs sous tâches, on s'aperçoit rapidement que la condition principale du succès n'est pas de disposer de processeurs mis en parallèle et qui calculent vite, mais plutôt que ces processeurs puissent communiquer entre eux de façon efficace; c'est à dire qu'ils puissent échanger de l'information aussi rapidement qu'ils accèdent à leurs propres données [38].

Tout réseau de communication pour une répartition spatiale ou une autre, doit respecter le concept de localité: pour une répartition spatiale donnée des processeurs, il

n'est possible de relier efficacement un processeur qu'avec son voisinage. En conséquence, un algorithme ne pourra être réparti de telle sorte que si chaque processeur n'ait à communiquer qu'avec ses voisins, au sens du réseau de communication de l'architecture. En fonction du mode de connexion choisi, de la technologie visée et de la fréquence de fonctionnement désirée, plusieurs configurations systoliques peuvent être utilisées. Dans cette section nous en proposons quelques unes et nous discuterons les points pour et contre afin de fixer le choix final de l'architecture à modéliser. Selon la façon que les données sont communiquées aux processeurs élémentaires, on peut définir deux types d'architectures systoliques, soit l'architecture semi-systolique, soit l'architecture systolique pure [41].

La figure (3.3) montre une proposition d'architecture semi-systolique où les données d'entrée du réseau systolique sont communiquées à tous les PEs en même temps. On dit que ces données sont diffusées à travers le réseau de processeur.

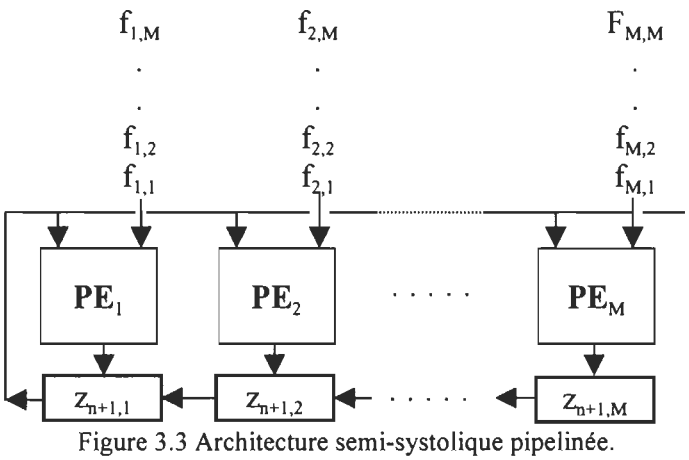

Figure 3.3 Architecture semi-systolique pipelinée.

Cette architecture est massivement parallèle avec un fonctionnement pipeliné. Le pipeline cette fois est au niveau de la sortie des résultats de traitement plutôt que sur la communication des données aux PEs. Cette architecture réduit de $2\times M$ le temps nécessaire pour les calculs $tp=\frac{ts}{2\times M}$, tp est le temps de calcul parallèle, ts est le temps de calcul séquentielle et m est le nombre de processeurs. La mise en oeuvre des deux algorithmes dans cette architecture peut s'avérer très laborieuse du moment qu'elle permette d'accélérer la vitesse de calcul avec un minimum de PE.

L'implantation de cette architecture sur silicium respectant la contrainte de modularité et tenant compte de l'expansion du système peut poser un problème au niveau de la communication des données et la collecte des résultats. En effet, la communication des données ou la collecte des résultats de tous les processeurs élémentaires du réseau systolique, durant chaque cycle, nécessite l'utilisation d'un bus de donnée commun à tous les processeurs. Au fur et à mesure que le nombre de processeur élémentaires augmente, la longueur du bus de donnée deviendra importante; l'expansion de ces communications non localisées pour rencontrer l'augmentation du système sans ralentir la fréquence d'horloge est une tâche très difficile.

Pour résoudre ce problème nous proposons deux architectures systoliques avec des communications entièrement localisées. La figure (3.4) montre la première configuration systolique pour l'implantation des deux algorithmes du chapitre 2. Cette architecture présente une solution très efficace au problème de localité des communications car toutes

les données et les résultats des PEs sont communiqués entre les processeurs voisins de façon pipelinée.

Cependant, cette configuration est limitée au niveau de la vitesse de calcul. En effet, on constate au moment de la collecte des résultats des processeurs élémentaires qu'il faut attendre que le dernier processeur ait fini de calculer pour pouvoir chercher les résultats de toutes les cellules dans l'ordre. Le pipeline dans cette configuration perd son efficacité et son utilité car il faut le vider après le calcul de chaque échantillon pour commencer un autre. De cette façon les processeurs élémentaires effectueront du travail utile durant m cycles et seront au repos durant m autres cycles.

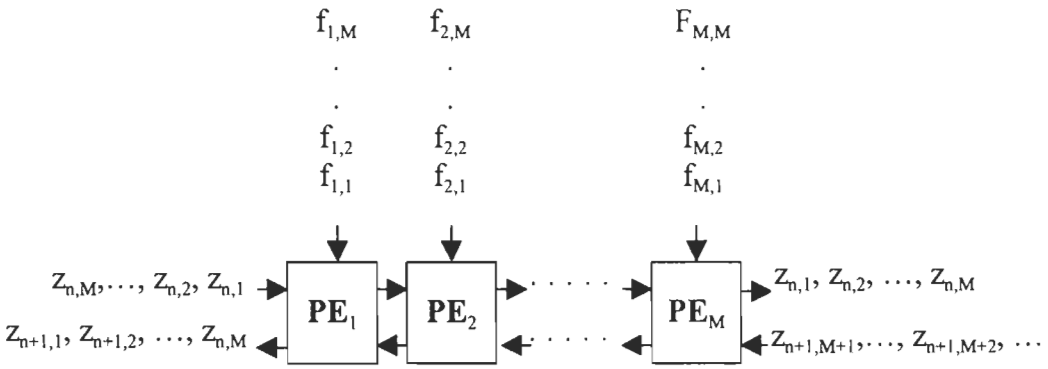

Figure 3.4 Architecture systolique entièrement pipelinée.

La deuxième solution est la configuration de figure (3.5). Cette architecture permet d'éviter le ralentissement de la vitesse de calcul tout en respectant la contrainte de localité des communications. Le calcul des composantes du vecteur $\mathbf{z}_{n+1}$ est réparti cette fois sur tous les PEs. Dans chaque processeur on calcule une partie et on accumule le résultat sur l'entrée. L'élément $z_{n+1,1}$ du vecteur Z_{n+1} est calculé au fur et à mesure qu'il se déplace vers

la sortie du réseau suivi du $Z_{n+1,2}$ et ainsi de suite. Cette configuration nous oblige à rajouter un PE qui va calculer le produit $k_M \times \tilde{y}_n$.

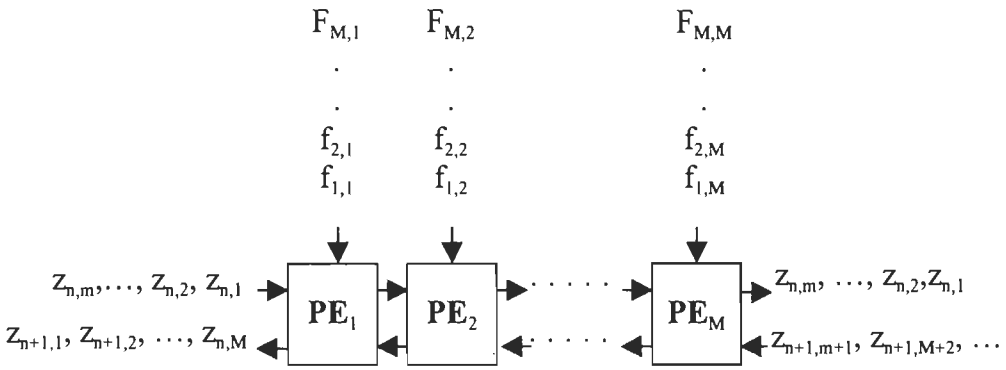

Figure 3.5 Architecture systolique rapide avec communications localisées.

Une seconde conséquence du mode de connexion local concerne le rapport entre entrées-sorties et calcul. Puisqu'il n'est pas possible de relier chaque processeur à tous les autres, il est de la même façon difficile d'imaginer que chaque processeur soit relié directement à l'environnement extérieur de la machine, qu'on appelle l'hôte du système censé de fournir les données de l'algorithme et centraliser ses résultats. D'une façon ou d'une autre, cela signifie qu'il faut que la complexité de l'algorithme repose sur les calculs et non sur la quantité d'entrées-sorties qu'il doit effectuer [38].

Le réseau systolique a pour but de balancer la capacité d'entrées-sorties ('I/O bound') en fonction de la capacité de calcul ('compute bound') dans le cas où le volume de calcul est plus important que les entrées-sorties. Pour réduire le nombre d'entrées-sorties des données, on peut facilement envisager de mémoriser les données des deux algorithmes dans des piles circulaires à l'intérieur des PEs. Cette solution s'avère très efficace pour

l'algorithme KALSPL car le nombre de données est M_v+5, alors que pour l'algorithme LMS le nombre est de N. Donc, on peut difficilement imaginer de les stocker dans des piles circulaires. Pour l'implantation des deux l'algorithmes ces données seront stockées dans une mémoire.

La complexité des circuits intégrés disponibles à l'heure actuelle rend possible la réalisation à un faible coût de systèmes massivement parallèles, pour peu que le mode de communication des processeurs respecte la contrainte de localité que nous venons d'énoncer. De même, la classe d'application est bien délimitée: le volume de calculs à effectuer doit primer largement sur le transfert de données à réaliser. Par suite, plutôt que de calculateurs généraux, il s'agit de processeurs spécialisés que l'on adjoint à un processeur hôte de type conventionnel.

3.3.5 Régularité

Une contrainte technologique essentielle qu'un concepteur d'architectures intégrées en circuits VLSI ne peut pas oublier est le coût d'une puce. Ce coût a toujours été un élément décisif dans le développement des circuits dédiés; il doit être suffisamment bas, par rapport au circuit à vocation générale, pour justifier leur faible quantité de production. Les coûts des ASIC peuvent être classés en deux ensembles : coûts récurrents et coûts non récurrents.

Les coûts non récurrents interviennent au niveau de l'étude des algorithmes à implanter, la définition des tâches, la conception, la modélisation et la synthèse. Les coûts récurrents sont plutôt liés à la production et à la fabrication. Ils diminuent rapidement due à l'évolution de la technologie des circuits intégrés et des techniques de production. Mais ces diminutions sont aussi valide pour les circuits à usage général d'où la nécessité de réduire les coûts de développement pour les ASICs pour qu'ils restent compétitifs et attirants du point de vue économique.

Ces coûts sont, pour une large mesure, déterminés par la superficie de la puce. En effet, le procédé de fabrication impose le coût de la tranche, le rendement de fabrication et la densité de défectuosité par unité de surface. Hennessy a proposé une façon simple de prédire le coût [33]: $CP = \frac{CT}{PPC \times R}$ avec CP= Coût de la Puce, CT= Coût de la Tranche, PPC= Puces par Couche et R= Rendement.

Nous pouvons immédiatement dire que le nombre de puces par tranche est inversement proportionnel à la surface de la puce. Quant au rendement, il dépend du procédé de fabrication et de la technologie utilisée. Dans la plupart des procédés de fabrication, il est inversement proportionnel au carré ou plus de la surface de la puce [33]. Nous arrivons à la conclusion que les coûts sont proportionnels au cube (ou à une puissance supérieure) de la surface de la puce: coût de la puce = f (surface de la puce^3)

De plus la taille de la puce dépend à la fois de la technologie, des portes nécessaires à la réalisation des fonctions et du nombre de broches d'entrées-sorties sur la périphérie d'une puce. Le concepteur a avantage à réduire au maximum la taille de la puce pour réduire les coûts. Heureusement, les coûts du design des circuits à applications spécifiques peuvent être réduits considérablement avec le choix d'architectures appropriées en adoptant une approche de conception systémique. En effet, si la structure du système peut être décomposée en quelques blocs fonctionnels simples et réguliers qui ne nécessitent que des interfaces modestes pour être reliés entre eux, on peut donc réaliser des économies substantielles. Ceci est spécialement vrai pour la conception en VLSI où les circuits peuvent comporter plusieurs milliers de composantes. Les réseaux systoliques offrent une grande régularité facilitant considérablement la tâche du concepteur. Il suffit de faire le design du premier processeur élémentaire et le copier autant de fois que requis, pour former le réseau au complet. Dans toutes les architectures proposées dans ce chapitre, on constate une parfaite régularité car tous les processeurs élémentaires sont identiques. La figure 3.6 donne une idée sur l'architecture interne d'un processeur élémentaire qu'on pourrait utiliser dans l'architecture de la figure (3.5) par exemple. L'utilisation des outils de conception assistée par ordinateur et des langages de description du matériel à haut niveau permet d'accomplir la tâche de conception avec une grande facilité et de réaliser d'importantes économies du temps et des coûts.

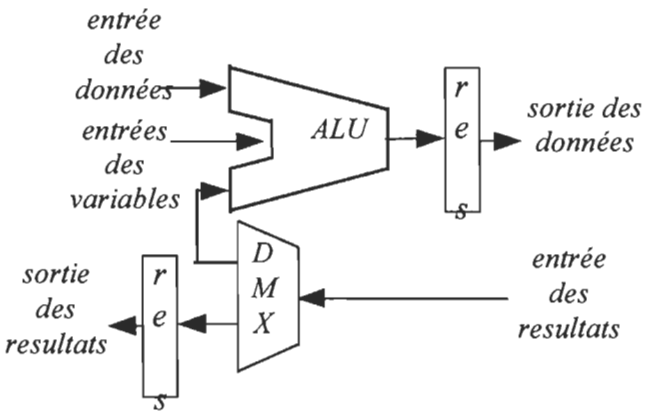

Figure 3.6 Architecture interne du processeur élémentaire utilisé dans la configuration proposée à la Fig. (3.5)

Dans cette architecture l'unité arithmétique logique (UAL) réalise un calcul très simple du type $Z_{n+1,i}^{sortie} = F_{i,j} \times Z_{n,i} + Z_{n+1,i}^{entrée}$. La simplicité des blocs formant l'architecture systolique de la figure (3.5) nous facilite sa conception et sa régularité et nous permet d'avoir une idée approximative sur la surface d'intégration.

3.3.6- Modularité

L'un des plus grands avantages des architectures systoliques est leur modularité qui leur donne une grande flexibilité vis à vis des problèmes de taille plus importante. La possibilité de mettre en cascade plusieurs processeurs pour accomplir des calculs dans des problèmes d'ordre plus important est préservée dans ce design. La figure. (3.7) montre un exemple de mise en série de deux processeurs, configuration de la figure (3.5), pour le calcul d'un problème d'ordre supérieur à 7 $M_v + 4 \geq 8$.

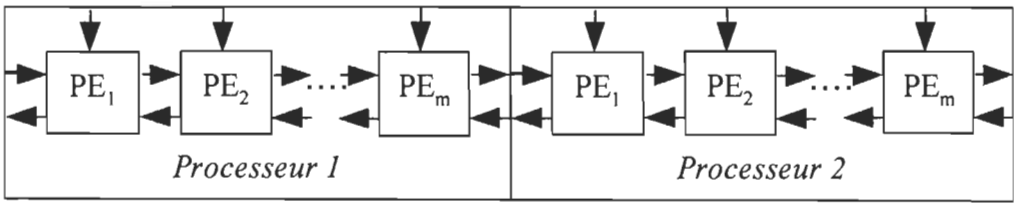

Figure 3.7 Mise en série de plusieurs processeurs.

3.3.7 Proposition d'architecture systolique pour KALSPL

En examinant le flot de données de la figure (2.9), nous pouvons constater qu'il s'agit tout simplement d'une multiplication matrice-vecteur. L'architecture systolique proposée à la figure 3.4 est très bien adaptée à cet algorithme. Puisque la technologie visée pour faire la fabrication est de l'ordre de $1.5\mu m$, nous pouvons utiliser l'architecture de la figure (3.3) ou (3.2) sans que cela nuise d'une manière considérable au fonctionnement de l'architecture car les délais de communication resteront toujours négligeables par rapport à ceux des UALs.

Nous utiliserons l'architecture de la figure (3.2) avec diffusion des résultats car cette architecture implique un contrôle systolique décentralisé, tandis que l'architecture de la figure (3.3) nécessite un contrôle parallèle centralisé ce qui rompt la condition de modularité.

Pour l'application de la contrainte de positivité un autre PE est nécessaire. Ce dernier prend le nom de bloc de décision car il a aussi la tâche de vérifier le signe du

deuxième élément dans le vecteur d'état pour détecter la première dérivée du signal $\hat{x}$ et d'estimer la position des pics du spectrogramme.

La figure (3.8) montre le schéma bloc de l'architecture proposée pour l'implantation de l'algorithme KALSPL. La figure (3.9) donne une idée sur le schéma interne du processeur élémentaire tandis que la figure (3.10) représente le schéma interne du bloc de décision utilisé dans la conception de cette architecture.

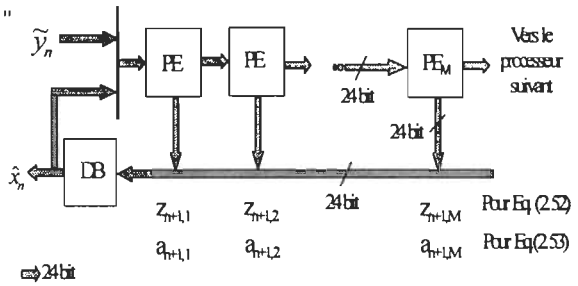

Figure 3.8 Architecture systolique pour l'implantation de l'algorithme KALSPL

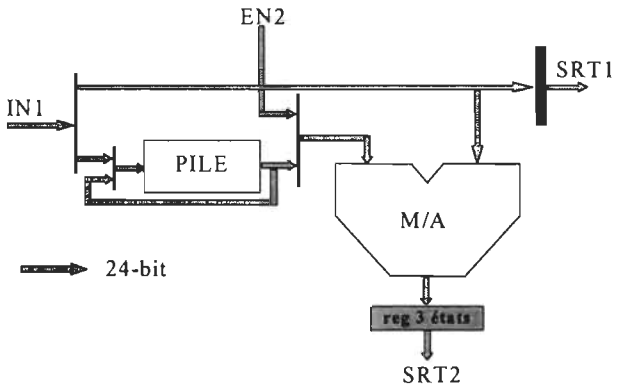

Figure 3.9 Architecture interne du processeur élémentaire.

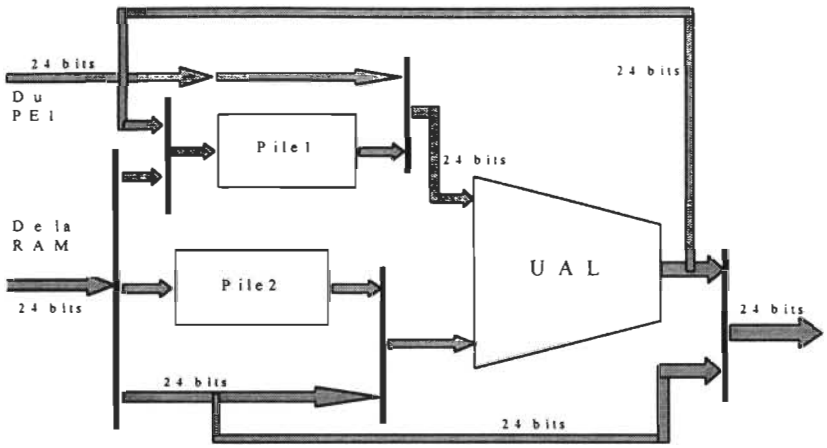

Figure 3.10 Schéma interne du Bloc de Décision.

Dans l'architecture interne du processeur élémentaire de la figure (3.9) nous proposons une pile circulaire du type FIFO pour le stockage des données de l'algorithme de KALSPL. Cette solution reste valable voir même très pratique dans le cas où l'ordre du modèle mathématique du système est fixe et connu d'avance 'lors de la conception'. Mais du moment où ce fameux ordre est variable un problème de modularité se pose.

Pour résoudre ce problème nous devons remplacer la pile circulaire par une mémoire d'une taille assez importante pour pouvoir couvrir une classe plus large de problème. De cette façon nous assurerons la modularité de notre processeur.

3.3.8 Proposition d'architecture systolique pour l'algorithme LMS

D'après le flot de données de la figure (2.10) nous pouvons proposer comme architecture semi-systolique parallèle pour l'implantation de l'algorithme LMS, la

configuration présentée à la figure (3.11). Cette architecture a été proposée dans [1]. La figure (3.12) présente l'architecture interne du processeur élémentaire de cette architecture.

Cette architecture est extrêmement performante en ce qui concerne le temps de calcul (1 cycle par échantillon), mais elle nécessite une mémoire parallèle. De part sa différence par rapport à l'architecture proposée pour KALSPL figure (3.8), cette architecture entraîne des coûts en surface très élevés. Il sera donc pratique pour nous d'adapter l'architecture de la figure (3.8) au flot de donnée de l'algorithme LMS plutôt que de développer une autre architecture. La figure (3.13) donne le schéma de l'architecture de la figure (3.8) adaptée pour l'implantation de l'algorithme LMS.

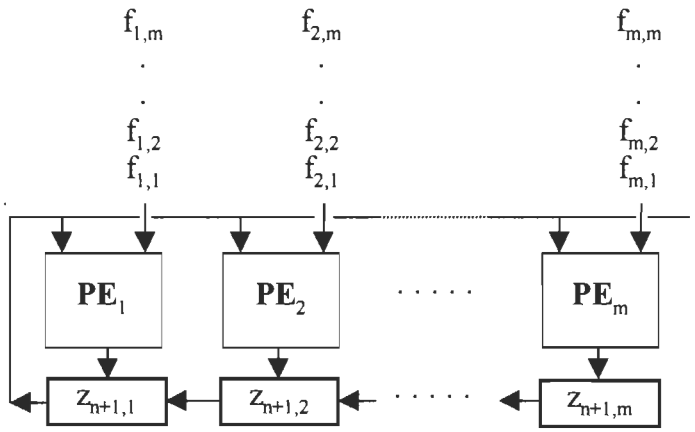

Figure 3.11 Architecture semi-systolique pour l'implantation de l'algorithme LMS.

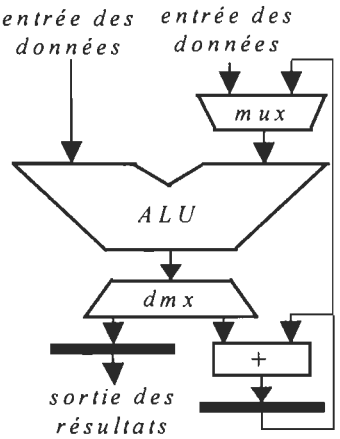

Figure 3.12 Architecture interne du processeur élémentaire de l'architecture de la figure 3.11.

La conception d'une architecture systolique requiert essentiellement la décomposition du système en PEs simples connectés entre eux par des retards appropriés. Les opérations impliquées dans Eq. (2.52) sont efficacement implantées dans le réseau systolique linéaire de la figure (3.8). Dans cette architecture les principes fondamentaux des architectures systoliques tels que la communication locale entre PEs voisins, le pipeline de la structure et le parallélisme massif dans l'exécution des opérations de calculs sont préservés.

Les opérations arithmétiques sont décomposées de façon que chaque PE calcule une partie de la solution globale de Eq. (2.52). Puisque M≥7 sept PEs sont rangés linéairement pour former le réseau systolique. L'architecture proposée est extrêmement modulaire et facilement adaptable. Ainsi, multiples processeurs peuvent être mis en cascade si M≥8 pour exécuter les tâches de calcul. La régularité du réseau systolique dans cette structure en fait une solution très attrayante à l'implantation en technologie VLSI d'algorithmes très

complexes. L'usage efficace de ce système peut réduire considérablement l'effort de calcul de M^2+M à $2M+1$.

La conception architecturale optimale des PEs n'est pas triviale si l'application a un ensemble d'opérations complexes. La conception du PE qui constitue la partie la plus importante dans le réseau est décrite à la figure (3.9). Le coeur de cette figure est le multiplieur-accumulateur (M/A) à bit parallèle. Puisque le processeur fonctionne à virgule fixe la mise à l'échelle des données se fait durant le transfert des résultats de calcul entre le l'accumulateur et le registre trois-états par un simple décalage. L'élément de mémoire FIFO est un vecteur de 32 mots de 24 bits chaque. Il est utilisé pour stocker le modèle mathématique du système que l'on trouve dans Eq. (2.52) et Eq. (2.53). Ces données sont figées dans ces piles et sont adressées au rythme de l'horloge ce qui permet de réduire la bande d'E/S requise pour les deux algorithmes. Le décision-bloc (DB) a été conçu selon [35] pour:

- introduire la contrainte de positivité sur l'estimation en tronquant les valeurs négatives du premier élément du vecteur d'état;
- estimer la position des pics pendant la deuxième étape de filtrage en détectant simplement les positions où il y a un changement de signe du deuxième élément dans le vecteur d'état;
- Corriger l'amplitude des pics estimés durant les phases précédentes en appliquant l'algorithme LMS.

Quand de multiples processeurs sont utilisés, le DB dans le premier processeur garde ce même comportement, pendant que les autres agissent comme une simple connexion entre le bus interne et le port d'E/S

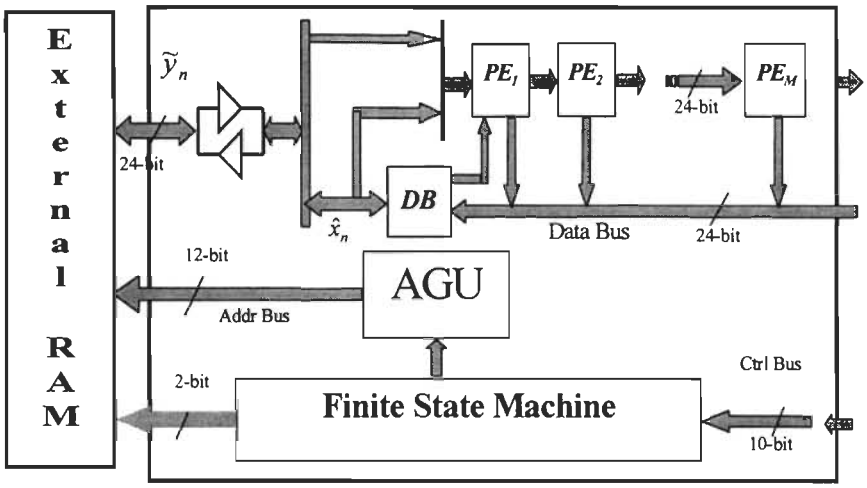

Figure 3.13 Architecture proposée pour l'implantation de KALSPL et LMS

3.3.9 Fonctionnement de l'architecture proposée

Les tableaux 3.1 et 3.2 présentent les séquences de fonctionnement de l'architecture de la figure 3.13 pour l'algorithme de KALSPL .

Tableau 3.1 Séquences de fonctionnement de KALSPL pour l'architecture de la Fig. (3.9).

CLK	PE_1	PE_2	...	PE_M
1	$Ac_1 = 0$	$Ac_2 = Z$	...	$Ac_m = Z$
2	$Aç = Aç + F_{1,1} \times Z_{0,}$	$Ac_2 = 0$	...	$Ac_m = Z$
3	$Aç = Aç + F_{1,2} \times Z_{0,}$	$Aç = Aç + F_{2,1} \times Z_{0,}$	...	$Ac_m = Z$
4	$Ac_1 = Ac_1 + F_{1,3} \times Z_{0,3}$	$Aç = Aç + F_{2,2} \times Z_{0,}$	...	$Ac_m = Z$
5	...	$Aç = Aç + F_{2,3} \times Z_{0,}$	...	$Ac_m = 0$
6	...	...	...	$Ac_m = Ac_m + F_{m,1} \times Z_{0,1}$

7	...	...	...	$Aç_m = Aç_m + F_{m,2} \times Z_0$
8	...	...	...	$Aç_m = Aç_m + F_{m,3} \times Z_0$
9	...	...	...	...
10	$Aç = Aç + F_{1,m} \times Z_{0,m}$	...	...	...
11	$Ac_1 = Ac_1 + k_1 \times y_0$	$Aç_2 = Aç_2 + F_{2,m} \times Z_0$	...	...
12	$Z_{1,1} = Ac_1$	$Aç_2 = Aç_2 + k_2 \times y_0$	...	...
13	$Ac_1 = 0$	$Z_{1,2} = Ac_2$	...	...
14	$Aç = Aç + F_{1,1} \times Z_{1,1}$	$Ac_2 = 0$	...	$Aç_m = Aç_m + F_{mm} \times Z_0$
15	$Aç = Aç + F_{1,2} \times Z_{1,}$	$Aç_2 = Aç_2 + F_{2,1} \times Z_{1,}$	...	$Aç_m = Aç_m + k_m \times y_0$
16	$Aç = Aç + F_{1,3} \times Z_{1,}$	$Aç_2 = Aç_2 + F_{2,2} \times Z_{1,}$	...	$Z_{1,m} = Ac_m$

Tableau 3.2 Séquences de fonctionnement de LMS pour l'architecture de la Fig. (3.9).

CYCLES	BLOC-DECISION	PE_1
1	Load $\hat{a}_{1,1} = \tilde{y}_{p1}$	$Ac_1 = 0$
2	Load $\hat{a}_{1,2} = \tilde{y}_{p2}$	$Ac_1 = 0$
...	...	$Ac_1 = 0$
L	Load $\hat{a}_{1,L} = \tilde{y}_{pL}$	$Ac_1 = 0$
(L+1)	$e_1 = \tilde{y}_1 - 0$	$Ac_1 = 0$
(L+2)	$I_1 = e_1$	$Ac_1 = 0$
(L+3)	$\hat{a}_{2,1} = \hat{a}_{1,1} + g_{1,1} I_1$	$Ac_1 = 0$
$(L+4)^{ème}$	$\hat{a}_{2,2} = \hat{a}_{1,2} + g_{1,2} I_1$	$Aç = Aç + \hat{a}_{2,1} g$
...	...	$Aç = Aç + \hat{a}_{2,2} g_2$
(2 L+2)	$\hat{a}_{2,L} = \hat{a}_{1,L} + g_{1,L} I_1$	...
(2L+3)	NOP	$Aç = Aç + \hat{a}_{2,L} g$
2 (L+2)	NOP	$T\_S\_R = Ac_1$
(2L+5)	$e_2 = \tilde{y}_2 - \hat{y}_2$	$Ac_1 = 0$
2 (L+3)	$I_2 = e_2$	$Ac_1 = 0$

3.4 MODÉLISATION DE L'ARCHITECTURE PROPOSÉE

La modélisation d'un système à l'aide des outils de conception assistée par ordinateur pourrait être décomposée en trois étapes : définition des exigences du design, description du design en VHDL et simulation du code au niveau logique.

3.4.1 Définition des exigences du design

Avant de commencer à écrire le code des blocs qui forment l'architecture, il faut choisir une méthodologie de conception. Plusieurs méthodes de conception existent entre autres la méthode du 'top-down design'. Cette méthode est basée sur la décomposition du système en plusieurs blocs fonctionnels simples. Une fois la décomposition faite, il faut définir l'interface entre eux, décrire l'architecture interne de chacun pour pouvoir établir les exigences de chaque bloc et sa fonctionnalité.

Avoir une idée claire sur les exigences de chaque bloc et sa fonction dans le design permet d'établir la bonne stratégie de modélisation et d'accomplir un travail efficace. La décomposition de l'architecture proposée suivant la méthode du 'top-down design' pourrait se faire en premier lieu en concordance avec la figure. (3.13) et ensuite selon les figure 3.9 et 3.10. L'interface entre les blocs ainsi que leur fonctionnalité sont bien claires sur les figures.

3.4.2 Description du design en VHDL

Après avoir décidé de la méthodologie de modélisation, nous pouvons décrire les modules et les blocs constituant notre architecture à l'aide d'un langage de description du matériel. Nous avons choisi le VHDL comme langage de modélisation pour ses avantages multiples 'puissant, flexible, indépendant de la technologie et du circuit, portable, rapidité de conception et standard'.

La séquence d'opérations à être exécutée par l'architecture est donné dans la Table. 3.3. Chaque groupe est exécuté de façon séquentielle et chaque opération par groupe est exécutée d'une manière parallèle. Une machine à état est utilisée pour synchroniser toutes les opérations du flot de données dans l'architecture. L'évaluation des performances est estimée par le nombre de cycles nécessaire pour exécuter chaque opération (Tab. 3.3). Une mémoire RAM est utilisée pour stocker N données estimées $\hat{x}_n^{\text{int}}$ et $\hat{x}_n$, les N données d'entrée $\tilde{y}_n$, les M×(M+1) éléments de la matrice F, les Ng éléments du vecteur g, et les 2K éléments qui sont les résultats $\hat{\boldsymbol{l}}$ et $\hat{\boldsymbol{a}}$ pour un total de $(2N+M^2+M+N_g^2+2K)$ mots de 24 bits. La méthodologie du top-down design a été adoptée pour faciliter la conception, ainsi l'architecture a été décomposée en plusieurs blocs simples qui sont utilisés fréquemment avec une simple interface; par conséquent, une réduction substantielle en terme de temps de développement de l'architecture est atteinte. Le contrôle dans cette architecture est simple. Un contrôle systolique est réalisé puisque tout les PEs font la même chose avec un simple décalage. Le modèle comportemental de cette architecture a été fait en VHDL en utilisant

les outils de CAO de Mentor-Graphics au niveau de transfert entre registres (RTL) pour la modélisation et la simulation et il a été synthétisé avec les outils de Synopsys dans la technologie Mitel 1.5 µm CMOS. Le résultat de synthèse est 8743 porte inverseur pour un PE, 10980 porte inverseur pour le bloc de décision (DB) et 72K porte inverseur pour toute l'architecture sans la machine à état. Nous avons estimé la surface de cette architecture à 150 mm^2 dans cette technologie. Le nombre de broches de ce processeur est montré à la Fig. (3.13) et il totalise 96 broches sans les broches de l'alimentation et la testabilité. Le tableau 3.4 donne les résultats de synthèse du code VHDL obtenus avec Synopsys.

Table 3.3 Séquences des opérations pour les applications spectrométriques et le nombre de cycles pour les exécuter.

Groupes	Opérations	# de cycles
G0	**0** initialisation, chargement des paramètres du modèle	M^2+2M+N_g+1
G1	**1** $\{\hat{x}_n^{int}\} = \text{SPL\_KAL}(\{\tilde{y}_n\}, \mathbf{F}, N)$, n=1,2,…,N	(2M+3)N
G2	**2** $\{\hat{x}_n\} = \text{POS}(\text{SPL\_KAL}(\{\hat{x}_n^{int}\}, \mathbf{F}, N))$, n=N,N-1,…,1	(2M+3)N
	3 $\hat{I} = \text{PEAK\_POS}(\{\hat{x}_n\})$	0
G3	**4** $\hat{\boldsymbol{a}} = \text{POS}(\text{PEAK\_MAG}(\{\hat{x}_n^{\wedge}\}, \boldsymbol{g}, \hat{\boldsymbol{I}}, \mu))$	$\leq(L+4)N_g$

La validation de l'architecture proposée a été faite avec des données synthétiques. Les résultats finaux obtenus par ce processeur, i.e. la signature spectrale des données, ont été utilisés pour valider les performances de l'architecture proposée sur le plan de la vitesse et la précision avec M = 7, N=Ng=300, et nombre de pics m=2. Comparativement au DSP56001 de Motorola, l'architecture proposée est 100 fois plus rapide. Dans cet exemple, les temps de calcul pour toute la séquence de données spectrométrique sont 131 ms et 1 ms pour le DSP56001 et l'architecture proposée respectivement. Les deux systèmes ont la même précision mais la surface exigée pour l'architecture systolique est plus élevée.

L'étape de modélisation est la plus longue dans le chemin de développement d'un circuit dédié. C'est durant cette étape que l'on prend les décisions les plus importantes, on aborde certains détails et on adopte une vraie stratégie de modélisation.

La description comportementale permet de faciliter la tâche du concepteur en permettant une description à haut niveau. La description structurelle est nécessaire pour relier les modules et blocs entre eux et former la structure finale.

Tableau 3.4 Résultats de synthèse des composantes fondamentales

Composant	Portes	Surface estimée en mm^2	Délai ns
PE	8743.01	13.76	78.25
DB	10980.34	17.29	76.11
MAC	7522.54	11.84	77.34

Les programmes VHDL ainsi que les fichiers de stimulus sont donnés à l'annexe A. Ces programmes décrivent les structures présentées aux figures 3.9, 3.10, 3.13. Pour des besoins de simulation nous avons rajouté un bloc qu'on a appelé unité de mémoire.

3.4.3 Simulation du code VHDL

La simulation du code VHDL au niveau RTL 'Register Transfer Level' a pour but la vérification de la fonctionnalité logique de notre design. Cette étape est d'une importance primordiale dans le cycle de développement du circuit car elle permet de déceler toutes les erreurs tôt dans la modélisation et d'apporter les corrections nécessaires que ce soit au niveau de la conception ou de la modélisation. Le tableau 3.3 donne le nombre de cycles nécessaire pour faire la reconstitution pour chaque étape, les opérations sont divisées en

groupes. Toutes les opérations du même groupe sont exécutées en parallèle. Les groupes sont exécutés séquentiellement. Les résultats de reconstitution avec un spectrogramme de deux pics sont montrés à la figure (3.13). Le tableau 3.5 résume les performances de l'architecture comparativement au DSP56001 (Motorola).

Tableau 3.5 Comparaison entre les performances de l'architecture proposée et le DSP56001(Motorola).

CRITÈRES D'ÉVALUATION	**DSP56001**	SYSKAL_SPL
Précision	Bonne	très Bonne
Nombre de cycles	1188127	15000
Nombre de portes	60000	70000

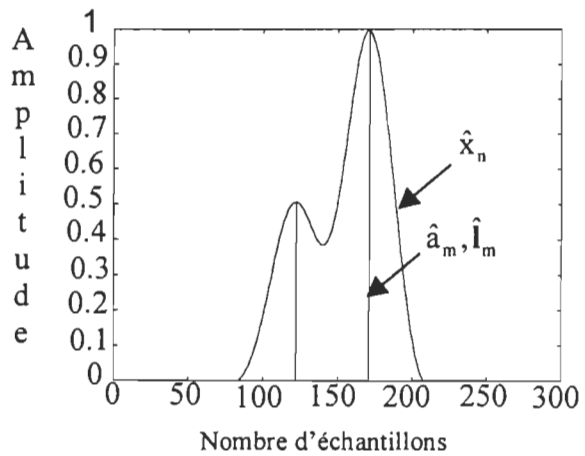

Figure 3.14 Résultats de simulation avec des données synthétiques.

3.5 RÉSULTATS ET DISCUSSION

L'utilisation de circuits à applications spécifiques pour résoudre les équations du filtre de Kalman-Spline et LMS est bien justifiée. L'implication d'opérations arithmétiques très complexes nécessite une très bonne capacité de calcul. Le nombre de ces opérations est

fonction de la taille du modèle mathématique du système, de la résolution de l'échantillonneur et de la réponse impulsionnelle du système $g(n)$.

Le stockage des résultats intermédiaires entre les deux passages constitue la plus grande faiblesse de l'algorithme du point de vue d'implantation sur silicium car il faut disposer d'une mémoire externe avec une taille assez grande.

La réduction de l'équation du système à une forme matricielle assure une grande économie en terme de surface d'intégration, puissance consommée et temps de calcul. Les performances du réseau systolique ont été démontrées pour faire la multiplication matrice vecteur.

La répartition du calcul sur les cellules élémentaires permet de réduire le temps de calcul d'un problème d'ordre $O(N^2)$ pour un calcul séquentiel à un temps d'exécution d'ordre $O(N)$ en utilisant un réseau de N cellules.

La modularité des réseaux systoliques permet de les adapter à plusieurs types et tailles des problèmes. En plus, il est possible de systoliser plusieurs problèmes[38], car le modèle systolique se prête bien au calcul numérique (multiplication matrice-vecteur, convolution, déconvolution etc...).

La modélisation de l'architecture en VHDL a été possible grâce à la performance des outils de conception 'Mentor Graphics, Synopsys et Cadence'. Nous avons décomposé la modélisation du processeur en plusieurs étapes, ce qui a facilité cette tâche de conception. Nous avons tiré profit de la régularité et de la modularité de l'architecture en ce qui concerne la phase du développement car nous avons mis tous les efforts sur le design d'un processeur élémentaire et nous l'avons reproduit plusieurs fois.

Chapitre 4

Proposition d'une architecture VLIW

4.1 GÉNÉRALITÉS

L'architecture des ordinateurs est l'un des domaines les plus mouvementés dans l'histoire de l'ingénierie. Suivre tous les avancements récents est l'un des défis qui est posé aux chercheurs dans le domaine des systèmes informatiques. Le but de la recherche dans ce domaine est d'améliorer le rapport qualité-prix des systèmes ordonnés [33]. Le prix et la performance des systèmes informatiques sont déterminés par la technologie VLSI, les architectures des processeurs et des mémoires des systèmes, la technologie du compilateur et du système d'exploitation et bien entendu les applications ciblées.

L'approche des machines VLIW a suscité beaucoup d'intérêt récemment, étant donné qu'elles sont des architectures mieux appropriées pour exploiter le parallélisme extrait par les compilateurs ou la répartition algorithmique pour la gestion efficace des ressources. L'architecture proposée au chapitre 3 est non programmable ce qui constitue sa principale limitation. Pour assurer la flexibilité du processeur à la mise en œuvre d'autres algorithmes de reconstitution de mesurande, nous proposerons dans ce chapitre une

architecture programmable et hautement parallèle basée sur l'approche VLIW. En plus de sa flexibilité, cette architecture offre les même avantages que ceux de l'architecture systolique. En effet, les machines VLIW permettent de tirer profit du parallélisme au niveau des instructions, d'où leur mot d'instruction très long . La longueur de l'instruction vient du fait que plusieurs sous-instructions sont mises ensemble pour former une seule instruction globale capable d'exécuter plusieurs tâches en un cycle d'horloge et ce en commandant plusieurs unités fonctionnelles totalement indépendantes.

Le concept VLIW a été développé à partir des premières recherches sur les architectures et les compilateurs de micro-code. Le premier travail sur l'approche VLIW est enregistré dans [42]. Le concept des architectures VLIW a été analysé et depuis lors exploré par beaucoup de projets de recherche [46]. Les efforts de recherche et de développement dans la conception des microprocesseurs sont habituellement appliqués à optimiser le nombre des transistors dans un circuit, accélérer ou ralentir la fréquence de l'horloge et réduire la consommation pour maximiser la rentabilité et améliorer les performances suivant les besoins. Les changements architecturaux principaux sont rares. Le mot très long d'instruction (VLIW) est considéré comme un changement radical dans la philosophie de conception des microprocesseurs, cette technologie a maintenant atteint le marché pour des applications de calcul dans le multimédia. La raison majeure du développement de ces nouvelles machines est le standard de traitement visuel (exemple : MPEG). La conférence de HOTCHIPS96 à Stanford a montré une acceptation massive du modèle VLIW dans le domaine des co-processeurs. Certains modèles de machines commerciales basées sur le principe VLIW sont apparus sur le marché. Les premiers

exemples de cette approche sont : L'Intel I860 [43] [44] qui peut exécuter deux opérations par cycle, le système 6000 [45] de la compagnie IBM qui peut exécuter quatre opérations parallèles et le Multiflow TRACE [46] qui a été conçu pour permettre l'exécution concourante de jusqu'à 28 opérations par cycle. Le dernier processeur DSP de Texas Instrument, TMX320, est basé sur l'approche VLIW [47]. En 1994, la compagnie HP a introduit son premier microprocesseur du nom MAX avec une extension de l'ensemble de ses instructions. Par après, tous les concepteurs principaux de microprocesseur ont répliqué avec des processeurs offrant des extensions d'instruction semblables. Tout dernièrement, en 1997, le géant Intel a introduit cette extension d'instruction sur son (MMX) x86, on prévoit que MMX sera l'un des processeurs les plus utilisés dans le domaine du traitement numérique du signal [48-49].

Plusieurs recherches sont en cours pour améliorer la performance de cette architecture qui est considérée comme une architecture d'avenir. Récemment les chercheurs d'IBM ont proposé une solution au problème de la compatibilité binaire de différentes générations des processeurs VLIW. L'instruction VLIW est implantée comme étant un arbre d'instructions [50]. Bien que toutes ces architectures poursuivent le but d'exécuter des opérations multiples concurremment, aucune d'elles n'a été établie en utilisant l'approche idéale de l'architecture VLIW. Les variations introduites sur le principe de base sont nécessaires parce que la mise en oeuvre du modèle idéal VLIW est contrainte à plusieurs considérations technologiques, la plus critique est la difficulté de bâtir un Fichier de Registres RF qui a un nombre excessivement grand de ports pour mémoriser et alimenter chaque unité fonctionnelle (UF) avec ses données respectives et ce d'une manière parallèle.

La machine VLIW idéale a un certain nombre d'unités fonctionnelles, qui lui permettent d'exécuter concurremment plusieurs opérations. Ces unités fonctionnelles sont alimentées par une mémoire avec une grande largeur de bande de données qui permet la lecture de deux opérandes et recevoir une écriture par unité fonctionnelle dans chaque cycle d'horloge. La miniaturisation et l'avancement technologique réalisés au cours des dernières années dans le domaine de VLSI nous permettent d'implanter ce type d'architecture sur une même puce avec des contraintes très réalistes au niveau des coûts. On s'attend à ce que la tendance de puces avec une grande capacité continue, rendant la réalisation d'un modèle VLIW complet sur une même puce une pratique courante dans un proche avenir.

Puisque le modèle VLIW idéal suppose que n'importe quel UF peut avoir accès à n'importe quel registre dans un cycle à travers un réseau entièrement connecté reliant les UFs et les registres, nous nous référons à cette architecture en tant que VLIW entièrement connectée. C'est un modèle simple, pourtant attrayant et extrêmement utile pour le parallélisme des machines. La seule préoccupation pour le programmeur est l'écriture et la répartition du code, plusieurs travaux portent sur le développement des compilateurs capables d'exploiter le parallélisme offert par une telle architecture [51][55]. Cependant, ce modèle idéal est impraticable pour une réalisation sur silicium (excepté pour un nombre limité de UFs) dû au grand nombre de ports exigés par le FR centralisé [46].

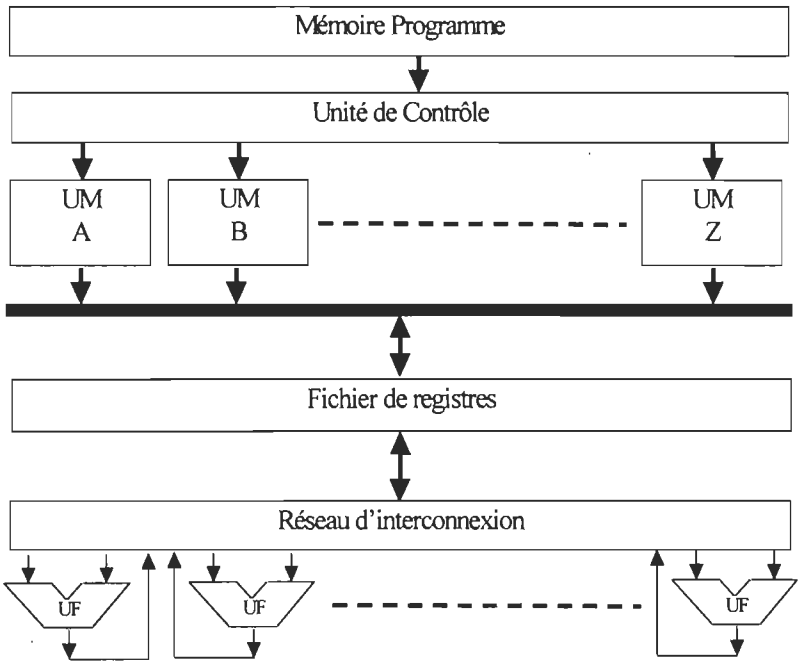

Figure 4.1 Schéma général d'une architecture VLIW idéale.

Par conséquent, une architecture VLIW plus réaliste doit avoir son code divisé dans des FRs multiples avec un nombre de ports limités. Ce type d'architecture VLIW est connu sous le nom de LC-VLIW (Limited Connectivity VLIW), architecture VLIW à connexions limitées. L'utilisation de cette architecture offre plusieurs avantages, entre autres :

- La faisabilité technique et technologique ;
- La modularité : chaque bloc de l'architecture peut être conçu et adapté à la taille désirée de la machine finale ;
- Possibilité d'utiliser des composantes standards, disponibles immédiatement, peu coûteux ou utiliser une approche de conception de custom/semi-custom.
- La facilité de sa testabilité et son entretien, puisque les blocs défectueux peuvent facilement être substitués.

Une solution pratique pour la réalisation de l'architecture VLIW sur silicium est la limitation de la pleine connectivité entre les registres et les UFs afin de réduire le nombre de ports nécessaires pour chaque FR. Cette approche a été suivie par l'équipe de conception du Multiflow TRACE qui a apporté une solution à ce problème en limitant le nombre de ports. Cette dernière a conçu un modèle VLIW avec des RFs multiples [45]. Le concept de base est de sacrifier une partie de la performance d'exécution d'une VLIW idéale pour une structure plus simple de registre de données et un cycle d'horloge plus court. Ce but est réalisé en utilisant un processeur VLIW divisé où chaque bloc a un nombre égal de UFs et tous les UFs par bloc sont complètement reliés à un FR local. La communication entre les différents blocs est réalisée par un certain nombre de bus dédiés qui relient tous les RFs. Un tel modèle est appelé LC-VLIW (limited connectivity VLIW). Quand le contenu d'un registre est requis pour un bloc autre que celui dans lequel il est stocké, un mouvement spécifique doit être exécuté pour copier les données et les transférer là où elles sont requises. Par conséquent le compilateur doit se charger d'échéancier les mouvements de données pour une gestion efficace des ressources.

Il est clair qu'en raison de la connectivité limitée et les mouvements registre-registre, le modèle d'exécution ne ressemble plus à une architecture avec mémoire homogène partagée. En fait, les accès aux registres peuvent avoir différentes latences, ayant pour résultat une contrainte supplémentaire qui nous empêche de réaliser le parallélisme optimal obtenu en utilisant un modèle VLIW idéal. Cependant, il y a des avantages clairs avec le LC-VLIW, entre autres la possibilité d'implanter physiquement la machine sur silicium.

La figure 4.2 montre une architecture LC-VLIW se composant de quatre blocs de deux unités fonctionnelles chacune. Dans chaque bloc, les deux unités fonctionnelles sont complètement interconnectées au RF local, rapportant 4 ports de sortie et 2 ports d'entrée pour les accès de l'unité fonctionnelle. En outre, chaque RF a 4 ports supplémentaires d'entrée pour des transferts de données inter-registre. L'interconnexion entre les RFs de chaque bloc est accomplit à travers un certain nombre de bus de transfert de donnée qui relient la sortie de chaque port du RF au port d'entrée spécifique au transfert entre bloc.

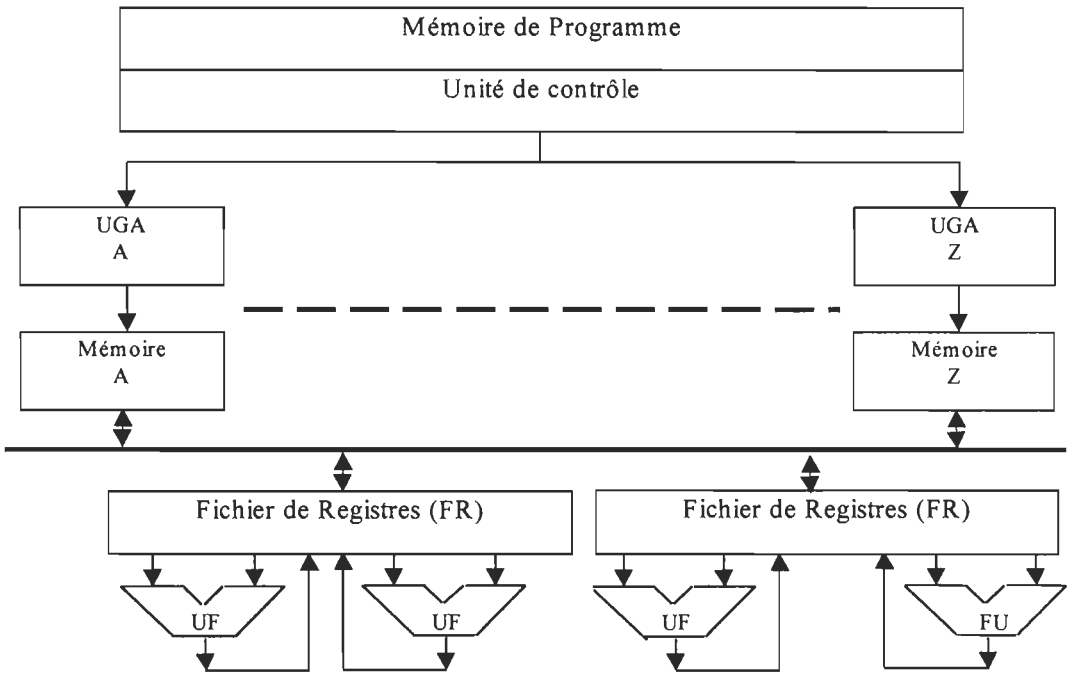

Figure 4.2 Architecture VLIW avec connectivité limitée, LC-VLIW.

Le nombre de bus de transmission exigés dans une telle architecture est égal à la largeur de bande. Notez que quand la deuxième constante (le nombre total des UF) est moins que la largeur de bande, les bus ne sont pas reliés à tous les ports d'entrée ; ceci exige l'adoption d'un arrangement approprié pour assurer la pleine connectivité. Cette

architecture exige une unité de commande très simple et un simple indicateur conditionnel, puisque chaque indicateur conditionnel peut être écrit par au plus une unité fonctionnelle et être lu seulement par une seule unité de commande. Par conséquent il n'y a aucun besoin de dupliquer les registres de commande et de condition. L'avantage principal du modèle décrit est le nombre limité de ports exigés par chaque RF et la distance limitée qui sépare intuitivement chaque unité fonctionnelle de l'autre RF. La conséquence directe est un temps amélioré de retard dans le chemin de donnée.

Pour une architecture VLIW, une autre considération importante est qu'un registre ne peut pas être écrit concurremment. C'est parce que l'architecture VLIW idéale garantit la pleine connectivité, ayant pour résultat la possibilité de multiples écritures au même emplacement de registre. Seulement les lectures concourantes sont permises à partir du même registre et dans le pire des cas tous les ports de sortie d'un fichier de registre peuvent accéder au même registre (c.-à-d. cellules de mémoire).

Dans [56] on prouve que pour un grand nombre de ports, la période d'accès au FR peut être modélisée par une fonction logarithmique du nombre des ports de sortie. Ceci rend le nombre de ports par RF un paramètre fondamental pour la conception d'une architecture canalisée où chaque retard constitutif doit être réduit au minimum afin de réaliser le meilleur débit global. On démontre également que la complexité de surface est une fonction qui croît avec le carré du nombre total des ports.

4.2 CARACTÉRISATION DE L'ARCHITECTURE PROPOSÉE

L'objectif de cette section est de proposer une architecture programmable, basée sur le modèle VLIW, dédiée à l'implantation des algorithmes de reconstitution de mesurande basés sur le filtre de Kalman. Une telle architecture a été proposée dans [1] pour une classe d'algorithmes basée sur le filtre de Kalman. Nous tendrons dans ce qui suit de proposer une architecture VLIW mieux adaptée et mieux détaillée. Nous partirons du modèle idéal de l'approche VLIW, et nous introduiserons des changements au niveau structurel de l'architecture pour l'adapter aux besoins des algorithmes visés, soit Kalman et LMS.

La conception d'un modèle idéal de l'architecture VLIW est loin d'être triviale. Toutefois, il est possible de proposer une architecture qui tire profit de ce modèle sans qu'elle respecte tous les principes de base [1, 56, 67, 68]. Nous proposons l'architecture de la figure Cette architecture est composée des éléments suivant :

1. Une unité de contrôle avec une mémoire programme interne ;
2. Plusieurs unités de mémoire accessibles en parallèle munies d'UGAs ;
3. Un fichier de registre FR ;
4. Des bus de données et d'adresses ;
5. Un bloc de traitement composé de quatre unités fonctionnelles.

Le modèle idéal de l'architecture VLIW peut être considéré comme une machine horizontalement programmée capable de décoder simultanément et exécuter différentes micro-instructions sur différentes unités fonctionnelles. Chaque unité fonctionnelle est

complètement indépendante et logiquement équivalente aux autres de sorte qu'il n'y ait aucune dépendance structurale entre les unités, rapportant un modèle uniforme d'architecture hautement parallèle [57-60]. Dans l'architecture proposée à la figure 4.3, nous utilisons quatres unités fonctionnelles, trois d'entre elles sont des simples M/A, et ce grâce à la nature matricielle des calculs impliqués dans les algorithmes visés, et la quatrième est une unité arithmétique logique qui sera conçue suivant les mêmes critères et aura sensiblement les mêmes tâches que le bloc de décision de l'architecture systolique proposée au chapitre 3.

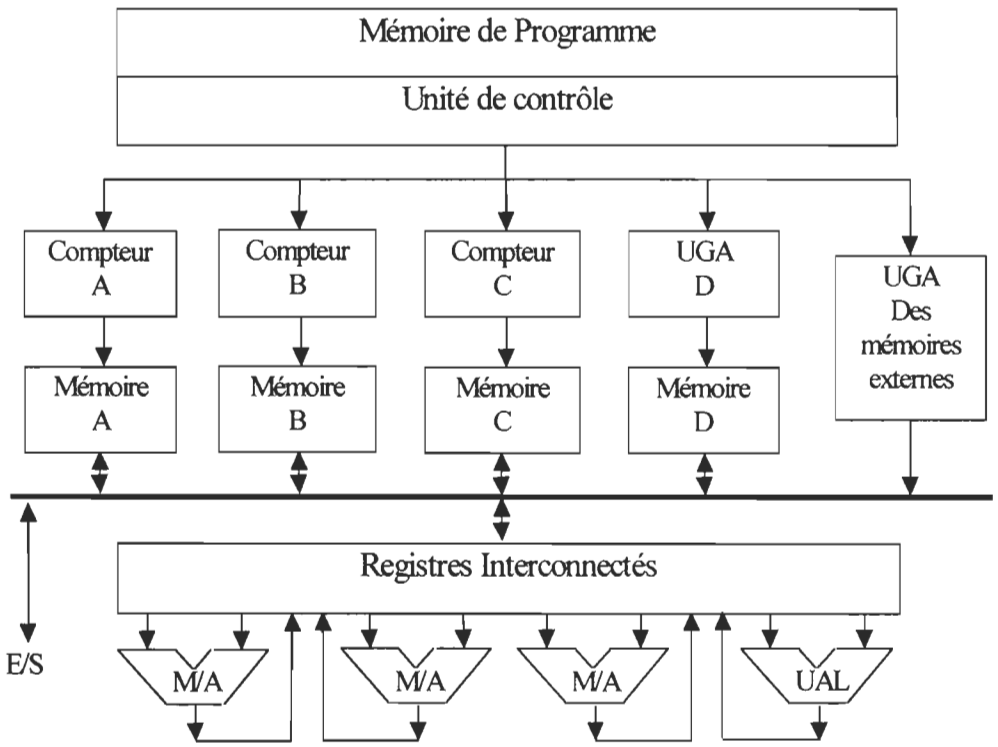

Figure 4.3 Architecture VLIW proposée pour l'implantation d'une classe d'algorithme basée sur le filtre de Kalman.

La figure 4.4 montre l'entité des deux composantes principales qui forment le bloc de traitement, soit le M/A et l'UAL, cette figure donne une idée sur la largeur des bus et lignes

nécessaires pour la commande. Tous les éléments de traitement dans ce bloc fonctionnent avec une représentation à point fixe en complément à 2 avec une largeur de mot de 24 bits. Toutes ces unités fonctionneront concurremment pour profiter du parallélisme inhérent dans les algorithmes visés. Bien que les flots de données dans les deux architectures soient complètement différents, ces unités ressemblent étrangement au réseau systolique linéaire de l'architecture du chapitre 3.

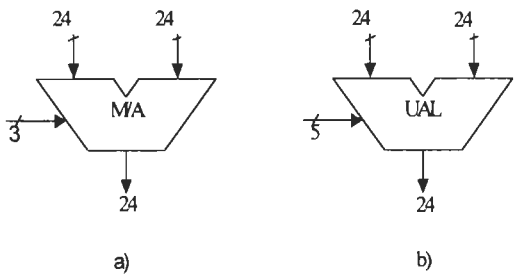

Figure 4.4 Entité du a) multiplieur-accumulateur b) unité arithmétique et logique.

Pour que les UFs puissent fonctionner en parallèle il leur faut un accès simultané à toutes les données nécessaires pour exécuter l'instruction parallèle en cours. Pour cela, il faut disposer de mémoires complètement indépendantes accessibles en parallèles. Sur la figure 4.3, on utilise quatre mémoires indépendantes, chacune d'entre elles est munie de sa propre unité de génération d'adresse. Les trois mémoires A, B et C seront utilisées pour stocker le modèle mathématique du système, un peu à l'image des FIFO utilisées dans l'architecture systolique. La gestion des données dans les mémoires doit tenir compte de leur position et de leurs connexions avec les unités de traitement. La figure 4.5 montre clairement la structure des mémoires utilisées et les liens entre ces dernières et le reste du processeur. On remarquera que les mémoires A, B et C n'ont aucun lien direct. Ceci vient

du fait qu'il n'y a aucune raison du point de vu algorithmique pour déplacer les données du modèle d'une mémoire à l'autre et du même fait que ces mémoires communiquent entre elles. L'écriture de ces mémoires se fait uniquement lors du chargement du modèle mathématique du système, donc il est tout à fait normal qu'elles disposent d'un bus de données les reliant à l'extérieur. Durant la phase de traitement ces mémoires sont accessibles en lecture seulement. Il faut aussi noter que la phase de chargement n'est pas très longue et n'arrive que lorsqu'on change de système ou on déconnecte le processeur. Il serait donc raisonnable de les charger séquentiellement et de n'utiliser qu'un seul bus de chargement pour toutes ces mémoires qu'on appellera Data_Bus1. Chacune des mémoires alimente l'unité fonctionnelle correspondante. Ainsi, la mémoire A communique ces données à l'unité fonctionnelle A et la mémoire B à l'unité fonctionnelle B et ainsi de suite. La mémoire D est en place pour recevoir les résultats de calcul intermédiaires; elle communique donc avec les quatres unités fonctionnelles. Cette mémoire est accessible en lecture et en écriture pour toutes les unités fonctionnelles. Cette mémoire communique aussi avec les trois mémoires externes qui contiennent **g**, **y** et les résultats de reconstitution **x**. Cette communication ne se fait pas en parallèle et on n'aura besoin que d'un seul bus reliant ces mémoires et la mémoire D. Puisque le bus Data_Bus1 n'est utilisé que durant la phase de chargement des données pour les mémoires A, B et C, il pourrait être utilisé pour assurer le lien entre les mémoires externes et la mémoire D.

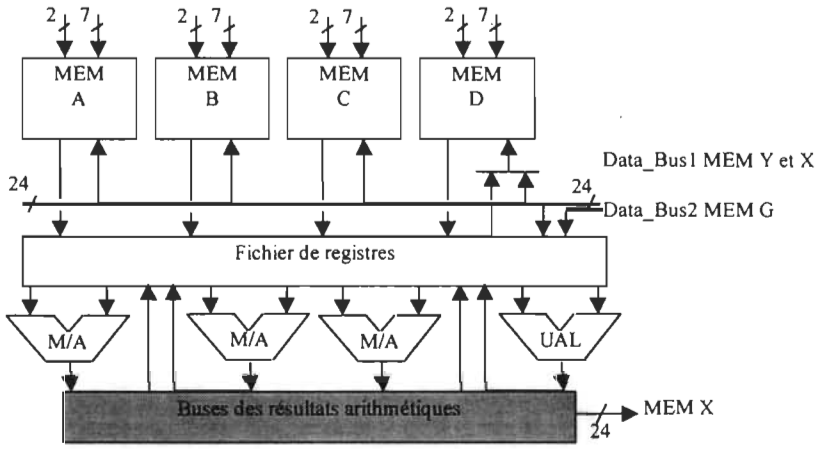

Figure 4.5 Schéma de la structure des mémoires et leurs connexions.

Les résultats de calcul sont directement écrits en mémoire D pour l'usage interne. Une partie de ces résultats est stockée dans la mémoire externe **x** tel que spécifié dans l'algorithme. Toutes les mémoires internes sont du type à deux port et elles disposent d'une ligne de sélection du boîtier et une autre pour distinguer la lecture et l'écriture.

Pour adresser les mémoires, on utilise des unités de génération d'adresse. Les mémoires A, B et C sont adressés d'une manière très simple. Les données dans ces mémoires sont lues dans l'ordre. Cet adressage est appelé auto-incrément. Un simple compteur est largement suffisant pour remplir cette tâche. Tandis que pour la mémoire D l'adressage est un peu plus élaboré. Cette mémoire nécessite un adressage indirect (compteur plus déplacement ou base). L'usage d'une unité de génération d'adresse avec une unité de calcul d'adresse et un pointeur est nécessaire. La figure 4.6 montre le schéma interne de l'unité de génération d'adresse de la mémoire D.

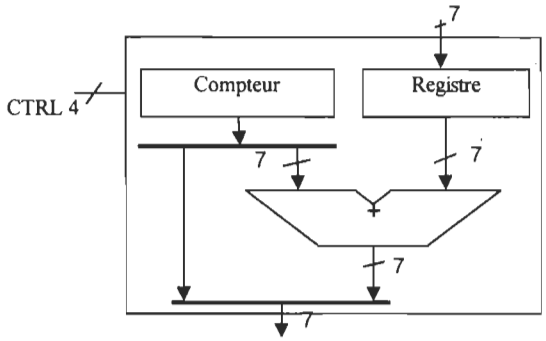

Figure 4.6 Schéma interne de UGA de la mémoire D.

Le fichier des registres est l'un des éléments les plus important dans la conception d'une architecture VLIW. Un design réussi d'une telle architecture est sans doute celui qui établi l'équilibre entre les performances exigées et les ressources disponibles. Le modèle d'une architecture VLIW idéale est fortement limité par le nombre de ports disponibles dans le fichier des registres. Ce nombre détermine la possibilité de réalisation de l'architecture et même ses performances. La disponibilité d'un port de sortie du FR pour un transfert de données pendant l'exécution d'une instruction est garantie puisqu'une opération de transfert de donnée maintient l'unité fonctionnelle du bloc source inactif pour ce cycle. Ceci ne peut pas être fait avec le bloc de destination puisque la fente d'exécution pour son unité fonctionnelle pourrait contenir une opération normale. Par conséquent, un nombre approprié de ports d'entrée est ajouté à chaque FR pour réaliser les opérations de mouvement des données entre les blocs. C'est la tâche du compilateur d'insérer automatiquement une exécution de mouvement en indiquant le registre de source et de destination et la banque. La banque de source est automatiquement détectée selon la fente d'instruction dans laquelle l'exécution de mouvement est insérée.

Le nombre de ports réservés qui doivent être ajoutés à chaque banque de fichier de registre est lié par deux constantes : le nombre de transmissions interbancaires permis par instruction et le nombre total d'unités fonctionnelles dans les autres blocs. La première constante est intuitivement claire, puisque nous ne pouvons pas avoir plus d'opération de transmission que la largeur de bande ayant lieu par instruction, et dans le pire des cas toutes les instructions visent à diriger des données vers le même RF. La deuxième constante résulte du fait qu'il n'y a aucun besoin de copier un registre par un bus externe de nouveau à sa même banque mémoire.

Bien que la technologie des mémoires multi-port ait été abordée pendant plusieurs années [4-20], il n'y a aucune technologie consolidée pour la conception des RFs avec un grand (plus de 16) nombre des ports [62-63]. Même si un tel dispositif de mémoire pourrait être conçu, il peut souffrir d'une dégradation des performances étant donné que chaque cellule de mémoire doit piloter tous les ports de sortie (c.-à-d. tous les chemins de données) dans le pire des cas. Un tel système peut considérablement affecter les avantages d'avoir un grand nombre de ports sur un RF pour réaliser une grande largeur de bande de données aux unités fonctionnelles multiples dans le modèle idéal VLIW. Ce problème exact a été considéré par les concepteurs du Multiflow TRACE [46], qui ont noté avec concision que tout nombre raisonnablement grand d'unités fonctionnelles exige un nombre impassiblement grand de ports pour le RF. Le seul compromis raisonnable pour l'implantation d'un tel système est de diviser le RF.

L'architecture proposée est beaucoup plus simple que le modèle idéal, du point de vu conception du FR. En fait nous n'avons pas le choix que d'opter pour une architecture LC-VLIW. Grâce à certaines simplifications algorithmiques, nous pouvons constater que les registres reliant les mémoires A, B et C au M/A correspondant n'ont pas besoin d'être connectés aux autres registres, mémoires ou UF. On note aussi que les trois M/A doivent avoir accès à la même donnée pour l'exécution des deux algorithmes, un bus reliant les trois avec un registre sera donc réservé, et on l'appellera R_Bus. La figure 4.7 montre la connexion entre le FR et le reste de l'architecture.

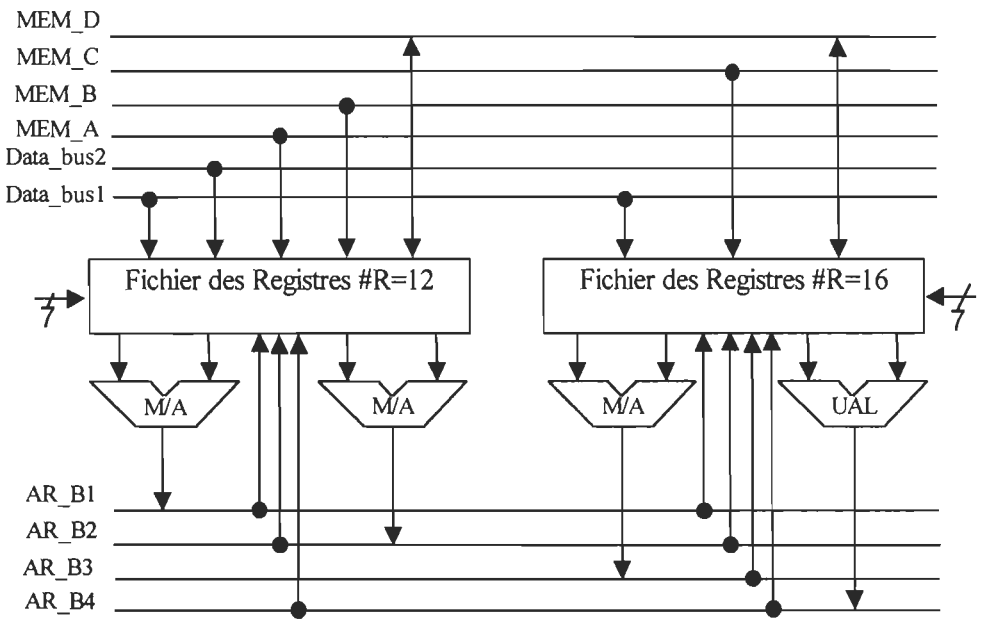

Figure 4.7 Structure et organisation des fichiers de registres.

4.3 UNITÉ DE CONTRÔLE

Si le chemin des données est simple à réaliser, l'autre partie qui permet de respecter ce chemin en fonction des exigences algorithmiques est l'unité de contrôle. La réalisation d'une unité de contrôle optimale est une tâche souvent complexe. Le microséquenseur dans cette architecture aura sensiblement le même rôle que la machine à état jouait dans l'architecture systolique du chapitre 3. Il sera chargé de guider le processeur, selon le micro-programme, dans son flot de données afin de respecter l'exécution des opérations de l'algorithme. Il y a deux types de contrôle : le contrôle cablé et le contrôle microprogrammé. Le contrôle est une façon de simplifier la conception du micro-séquenseur. Il offre aussi l'avantage de réduire les coûts matériels. La machine à état réalisée pour l'architecture du chapitre 3 en est un exemple. Ce type de contrôle ne serait pas intéressant dans la mesure où on cherche à assurer une certaine flexibilité du processeur et serait donc une possibilité à écarter [33].

Le contrôle microprogrammé est plus approprié pour rencontrer nos objectifs. Il donne une flexibilité logicielle assez importante. La microprogrammation permet de modifier le jeu d'instruction en changeant simplement le contenu de la mémoire de contrôle sans toucher à l'implantation matériel. Une unité de contrôle typique se compose de :

- Une mémoire vive qui contient le programme ;

- Un compteur programme qui permet de poursuivre le fonctionnement du processeur en permettant à la mémoire de charger un autre mot d'instruction dans le registre d'instructions;
- Un registre d'instruction directement connecté à la mémoire programme;
- Une logique de contrôle avec multiplexeur de condition ainsi que d'un registre de retour de sous-routine;
- Un multiplexeur d'adresse.

Grâce aux différentes lignes de contrôle, le micro-séquenceur est capable d'exécuter plusieurs tâches en même temps. La mémoire programme permet de charger le code qui décrit les tâches à exécuter par le processeur. Il s'agit d'une mémoire avec des mots assez larges pour orienter l'ensemble du processeur y compris le micro-séquenceur lui-même. La description du champ d'instruction du processeur est donnée à la figure 4.8. Alors que le tableau 4.1 montre la liste détaillée et le nombre de broches utilisées pour le contrôle. La figure 4.9 montre un exemple de la structure détaillée de l'unité de contrôle telle qu'utilisée dans les travaux [1], [67] et [68]. Dans [68] et [1] une proposition d'enjeu d'instructions et définition détaillée du registre d'instruction sont définies.

MEM	SYNC	BCC	CTE	Din1	Din2	OPE1	Dout1	Din3	Din4	OPE2	Dout2

Figure 4.8 Champ d'instruction du microséquenceur.

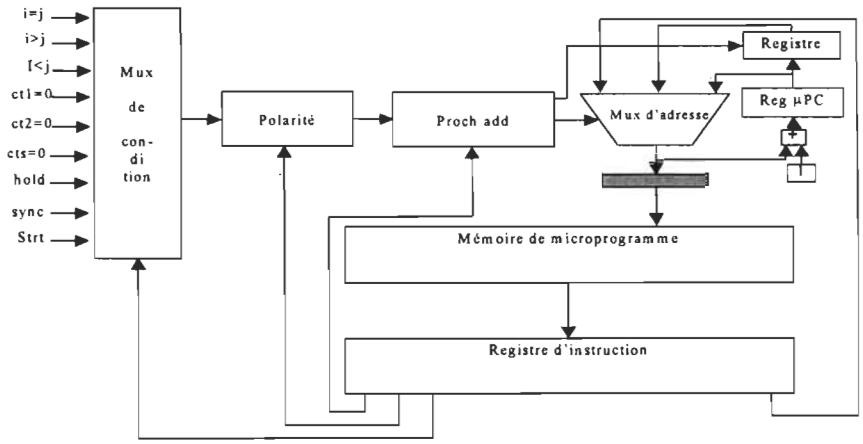

Figure 4.9 Structure détaillée de l'unité de contrôle.

Le tableau 4.1 donne les signaux de commande pour cette architecture. L'objectif de ce chapitre étant de proposer et non modéliser une architecture pour la mise en œuvre de l'algorithme de Kalman et LMS. Nous avons donné les grandes lignes du design.

Tableau 4.1 Lignes de contrôle de l'architecture proposée.

NOM BRE	NOM DE BROCHE	FONCTION
7	adresse	Champ qui contient une adresse immédiate
4	S_cond_b	Contrôle la condition de branchement
1	polarité	Contrôle la polarité du signe multiplexé
1	S_cts	Contrôle le chargement du compteur d'échantillonnage
1	S_ct1	Contrôle le chargement du compteur de boucle 1
1	S_ct2	Contrôle le chargement du compteur de boucle 2
1	En_ctA	Contrôle le compteur de la mémoire A
1	En_ctB	Contrôle le compteur de la mémoire B
1	En_ctC	Contrôle le compteur de la mémoire C
2	S_proch_	Contrôle le multiplexeur d'adresse du microprogramme
1	En_ctD	Contrôle le compteur de la mémoire D
1	En_ctUG	Contrôle le compteur de l'unité de génération d'adresse des mémoires
6	Cntl_UG	Contrôle des registres et pointeur de l'unité de génération d'adresse
2	S_ad_D	Contrôle la sélection de l'adresse de la mémoire D

4	S_PI1_F	Contrôle l'adresse pour le port I1 du fichier de registre 1
1	L1	Contrôle l'écriture dans port I1 dans le RF 1
1	W1	Contrôle la lecture du port 1 dans le RF 1
4	S_P2_FR	Contrôle l'adresse pour le port 2 du RF 1
1	L2	Contrôle l'écriture dans port 2 dans le RF 1
1	W1	Contrôle la lecture du port 2 dans le RF 1
4	S_P3_FR	Contrôle l'adresse pour le port 3 du fichier de registre 1
1	L3	Contrôle l'écriture dans port 3 dans le fichier de registre
1	W3	Contrôle la lecture du port 3 dans le fichier de registre
.4	S_P4_FR	Contrôle l'adresse pour le port 4 du fichier de registre 1
1	L4	Contrôle l'écriture dans port 4 dans le fichier de registre
1	W4	Contrôle la lecture du port 4 dans le fichier de registre
4	S_P5_FR	Contrôle l'adresse pour le port 5 du fichier de registre 1
1	L5	Contrôle l'écriture dans le port 5 dans le fichier de registre
1	W5	Contrôle la lecture du port 5 dans le fichier de registre
4	S_P6_FR	Contrôle l'adresse pour le port 6 du fichier de registre 1
1	L6	Contrôle l'écriture dans le port 6 dans le fichier de registre
1	W6	Contrôle la lecture du port 6 dans le fichier de registre
3	Instr_M/	Contrôle l'opération exécutée par le M/A 1
3	Instr_M/	Contrôle l'opération exécutée par le M/A 2
3	Instr_M/	Contrôle l'opération exécutée par le M/A 3
5	Instr_UA	Contrôle l'opération exécutée par le UAL
8	Ctrl_M_E	Contrôle des mémoires externes

4.4 RÉSULTATS ET DISCUSSION

La réalisation de l'architecture VLIW est fonction de plusieurs paramètres, la taille du fichier de registres, la largeur de la mémoire du programme et la complexité du contrôle. Pour rendre cette approche pratique, il faut opter pour des solutions assez concrètes pour résoudre ces problèmes. Dans le cadre de ce chapitre l'objectif était de faire une simple

d'algorithmes basés sur le filtre de Kalman. Nous avons proposé une architecture suffisamment flexible avec une description complète de la l'architecture et de ses blocs principaux.

L'architecture proposée est assez parallèle pour satisfaire les exigences au niveau rapidité mais pas assez pour atteindre les performances de l'architecture systolique. Les chemins de données dans cette architecture sont plus complexes que ceux de l'architecture systolique, ce qui rend la conception du contrôle très difficile.

Les avantages principaux de l'approche VLIW sont : 1) le matériel est beaucoup plus simple (sauf le micro-séquenceur), des composantes conventionnelles, ce qui engendre un temps plus court de conception du processeur au niveau de la modélisation VHDL. 2) la consommation en puissance peut devenir significative à cause de la duplicité des ressources mais demeure intérieure à l'architecture systolique. Ainsi moins de matériel est nécessaire au délai d'exécution. Les inconvénients principaux de l'approche initiale de l'architecture VLIW sont : 1) le manque de portabilité, le transfert du code d'un processeur VLIW vers un autre de la prochaine génération exige une recompilation. Le compilateur a un contrôle total des ressources du processeur. 2) la planification statique des opérations ne tient pas compte du parallélisme au niveau des instructions dû aux données d'entrée spécifiques sur lesquelles le processeur travaille. 3) l'exécution est au-dessous des processeurs conventionnels aux architectures superscalaires.

Le modèle LC-VLIW exige une modification importante du code idéal de VLIW. Le code d'un modèle VLIW idéal doit être divisé dans un certain nombre de sous-programmes à exécuter sur les blocs de l'architecture LC-VLIW. Chacun de ces sous-programmes doit être (ressource-contrainte) programmée selon l'architecture du bloc lui-même. En outre, la largeur de bande limitée doit être considérée comme une contrainte dans le processus d'écriture du programme.

La modélisation d'un décodeur d'instruction pourrait être une solution pratique pour l'implantation de cette architecture. En effet le décodeur permettrait de réduire le nombre de bits nécessaires pour la commande des différentes composantes de l'architecture sans modifier sa structure.

Chapitre 5

Synthèse de résultats et conclusion

L'objectif de ce travail est de proposer des architectures dédiées à la mise en oeuvre d'algorithmes de reconstitution de mesurande. Une connaissance approfondie de l'algorithme à implanter est nécessaire pour réussir une bonne conception qui tire profit des particularités de chaque algorithme et application. L'étude algorithmique réalisée au chapitre 2 a permis de mieux comprendre les algorithmes et d'établir un flot de données visant une implantation dans une architecture systolique. En effet, dans la version de KALSPL la matrice Φ est dense. Nous proposons l'utilisation d'un réseau systolique linéaire pour effectuer le produit matrice-vecteur. Pour profiter du parallélisme inhérent dans les deux algorithmes et réaliser une bonne conception d'architecture systolique, certaines transformations au niveau de la structure des équations ont été nécessaires pour réduire les coûts et augmenter les performances du processeur. Avec ces transformations, la complexité de calcul est réduite de $2N(M^2+2M+8)$ à $2N(M^2+M)$ où M est l'ordre du modèle d'état du système et N est le nombre d'échantillons [65]. La précision des résultats dans les applications spectrométriques est très importante; l'étude de quantification permet de faire une analyse pour assurer une précision acceptable. La représentation numérique et

Chapitre 5

Conclusion

L'objectif de ce travail est de proposer des architectures dédiées à la mise en oeuvre d'algorithmes de reconstitution de mesurande. Une connaissance approfondie de l'algorithme à implanter est nécessaire pour réussir une bonne conception qui tire profit des particularités de chaque algorithme et application. L'étude algorithmique réalisée au chapitre 2 a permis de mieux comprendre les algorithmes et d'établir un flot de données visant une implantation dans une architecture systolique. En effet, dans la version de KALSPL la matrice Φ est dense. Nous proposons l'utilisation d'un réseau systolique linéaire pour effectuer le produit matrice-vecteur. Pour profiter du parallélisme inhérent dans les deux algorithmes et réaliser une bonne conception d'architecture systolique, certaines transformations au niveau de la structure des équations ont été nécessaires pour réduire les coûts et augmenter les performances du processeur. Avec ces transformations, la complexité de calcul est réduite de $2N(M^2+2M+8)$ à $2N(M^2+M)$ où M est l'ordre du modèle d'état du système et N est le nombre d'échantillons [65]. La précision des résultats dans les applications spectrométriques est très importante; l'étude de quantification permet de faire une analyse pour assurer une précision acceptable. La représentation numérique et

le type d'arithmétique utilisé a une influence significative sur le comportement et l'exécution de l'architecture. Une étude de quantification a été performée pour déterminer la largeur des mots pour représenter les données et les blocs de calcul et ainsi assurer une bonne précision utilisant un calcul à virgule fixe et une représentation en complément à deux. Les résultats de simulation prouvent que le nombre de bits nécessaires pour assurer un certain niveau de précision pour l'équation transformée Eq.(2.43) est supérieur à celui nécessaire pour l'implantation discrète de Eq. (2.24).

Les opérations arithmétiques des deux algorithmes sont décomposées d'une telle façon que chaque PE calcule une partie de la solution globale. Puisque $M \geq 7$ sept PEs sont rangés linéairement pour former le réseau systolique. L'architecture proposée est extrêmement modulaire et facilement adaptable. Ainsi, multiples processeurs peuvent être mis en cascade si $M \geq 8$ pour exécuter les tâches de calcul. La régularité du réseau systolique dans cette structure en fait une solution très attrayante à l'implantation en technologie VLSI d'algorithmes très complexes. L'usage efficace de ce système peut réduire considérablement l'effort de calcul de M^2+M à $2M+1$ où M est l'ordre du modèle mathématique du système étudié.

Une architecture systolique dédiée à la correction de données spectrométriques pour le micro-spectromètre de basse résolution a été proposée; son application de masse dans l'interception écologique peut être prévue dans un avenir proche. La similarité entre les deux algorithmes (Kalman-spline et LMS) a été exploitée pour obtenir une architecture

systolique linéaire basée sur le produit matrice-vecteur dans une exécution bit-parallèle pour minimiser le temps de calcul. Une étude comparative avec le DSP56001 a été performée et nous avons montré que notre architecture nécessite 100 fois moins de cycles d'horloge [63]. Cette architecture supporte la mise en œuvre de plusieurs algorithmes de traitement numérique de signal et de l'arithmétique matricielle. L'architecture proposée sera utilisée dans le développement d'un appareil spectrométrique intégré. Dans cette architecture les principes fondamentaux des architectures systoliques tel que la communication locale entre PEs voisins, le pipeline de la structure et le parallélisme massif dans l'exécution des opérations de calculs sont préservés.

La modélisation de l'architecture en VHDL a été possible grâce à la performance des outils de conception 'Mentor Graphics, Synopsys'. Nous avons décomposé la modélisation du processeur en plusieurs étapes, ce qui a facilité cette tâche de conception. Nous avons tiré profit de la régularité et de la modularité de l'architecture en ce qui concerne la phase du développement car nous avons mis tous les efforts sur le design d'un processeur élémentaire et nous l'avons reproduit plusieurs fois.

Les résultats de cette étude d'implantation visant une architecture systolique ont fait l'objet de plusieurs publications [64-66]. Ainsi, la première partie a été soumise au 65ème congrès de l'ACFAS et elle a porté sur l'étude de l'implantation visant une architecture systolique. La deuxième a été soumise au congrès canadien en génie électrique et informatique CCGEI98, dans laquelle nous avons présenté l'architecture détaillée ainsi que la validation des performances. La dernière publication a été présentée à TEXPO98 dans le

cadre du Symposium en micro-électronique organisé par la CMC et cette dernière a fait l'objet d'une présentation de l'architecture globale et des résultats de synthèse ainsi qu'une vue globale du projet d'intégration d'un processeur dédié à la correction de données spectrométriques.

Le stockage des résultats intermédiaires entre les deux passages constitue la plus grande faiblesse de l'algorithme du point de vue implantation sur silicium car il faut disposer d'une mémoire externe avec une taille assez grande, ce qui enlève les chances de faire un traitement en temps réel. Aussi la nature dynamique du vecteur $\mathbf{g}_n$ nécessite une UGA assez complexe pour effectuer l'adressage mémoire et exige une bande d'entrée-sortie assez importante. Il faut travailler au niveau algorithmique pour trouver une façon de calculer le vecteur $\mathbf{g}_n$ à l'intérieur du processeur au lieu de passer par l'adressage mémoire.

L'architecture systolique proposée n'est pas programmable et souffre d'un manque de flexibilité. Pour rendre le processeur flexible à la mise en œuvre d'autres algorithmes nous proposons l'architecture LC-VLIW. C'est une architecture aussi parallèle que l'architecture systolique et qui offre la possibilité de programmation. En plus de sa flexibilité, cette architecture offre les même avantages que ceux de l'architecture systolique. En effet, les machines VLIW permettent de tirer profit du parallélisme au niveau des instructions.

L'architecture VLIW pose certains problèmes au niveau de conception. L'implantation du modèle idéal VLIW est contrainte à plusieurs considérations

technologiques, la plus critique est la difficulté de bâtir un fichier de registres (RF) qui a un nombre excessivement grand de ports pour mémoriser les données et alimenter chaque unité fonctionnelle (UF) avec ses données respectives et ce d'une manière parallèle. Une solution pratique pour la réalisation de l'architecture VLIW sur silicium est la limitation de la pleine connectivité entre les registres et les UFs afin de réduire le nombre de ports nécessaires pour chaque FR. La conception d'une unité de contrôle d'un modèle VLIW est aussi compliquée que le design de son unité de mémoire pour contenir le programme. Malgré tous ces inconvénients, l'architecture VLIW reste une architecture d'avenir imposant ainsi un changement dans la philosophie du développement matériel dans le domaine d'informatique et le multimédia.

Bibliographie

[1] D. Massicotte, "Une approche à l'implantation en technologie VLSI d'une classe d'algorithmes de reconstitution de signaux", Ph.D. Thèse, École Polytechnique de Montréal, 1995.

[2] P. A. Jansson, "Deconvolution with Applications in Spectroscopy", 2nd Edition, Academic Press, 1998.

[3] A. Miekina, R.Z. Morawski, and A. Barwicz, "The Use of Deconvolution and Iterative Optimization for Spectrogram Interpretation", *IEEE Trans. Instr. & Meas.*, Vol. 46, No. 4, pp. 1049-1053, 1997.

[4] D. Massicotte, R. Z. Morawski, A. Barwicz, "Kalman-filter-based Algorithm of Spectrometric Data Correction", Part I: An Iterative Algorithm of Deconvolution", *IEEE Trans. Instr. & Meas.*, Vol. 46, No 3, pp. 678-684, 1997.

[5] P.Brouard, L'application du filtre de Kalman à l'approximation de mesurandes par une fonction spline – étude d'un algorithme et son implantation sur DSP, Mémoire de maîtrise en éléctronique industrielle, UQTR, 1995.

[6] M. Ben Slima, "Algorithmes de reconstitution de mesurandes dynamiques avec régularisation multiple", Thèse de doctorat, INRS-Télécommunications, June 1996.

[7] M. Newell and J. Rasure, "A VLSI System for Real-Time Linear Operations and Transforms", IEEE Trans. on Signal Processing, Vol. 39, No. 8, pp. 1914-1917, August 1991.

[8] H. T. Kung, "Why Systolic Architectures", *IEEE Trans. Computer*, pp. 37-46, January 1982.

[9] M. R. Azimi, T. Lu et E. M. Nebot, "Parallel and sequential block Kalman filtering and their implementation using systolic arrays", IEEE Trans. on Signal Processing, Vol. 39, Iss. 1, pp. 137-147, 1991.

[10] D. Bursky ; "Reprogrammable IC takes on graphics, video, and audio", Electronic Design, Vol. 43, Iss :21, pp. 133-4, 136-7, Oct 1995.

[11] P. Pirsch; J. Kneip; K. Ronner; "Parallelization resources of image processing algorithms and their maping on a programmable parallel videosignal processor", 1995 IEEE Symposium on Circuits and Systems, Vol. 1, pp. 562-5, 1995.

[12] L. Wang; C.-T Kan; J.W. Jou; "A dual mode homogeneous VLIW computer and its memory hierarchy", International Computer Symposium Conference Proceedings, pp. 457-64, Vol. 1, 1994.

[13] J. Gray; A. Naylor; A. Abnous et N. Bagherzadeh, "VIPER : A25-MHz peak VLIW microprocessor", Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 4.1.1-4.1.5, 1993.

[14] R. Milikowski et W.G. Vree; "Non Homogenous parallel memory operations in a VLIW machine"; Parallel Processing : CONPAR 94 – VAPP VI. Third Joint International Conference on Vector and Parallel Processing Proceedings pp. 485-96, 1994.

[15] D.M. Gallagher;W.Y. Chen; S.A. Mahlek; J.C. Gyllenhaal; W. –M.W. Hwu; "Dynamic memory disambiguation using the memory conflict buffer"; Sigplan Notices, Vol. 29 Iss. 11 pp.183-93.

[16] E. Ercanli et C. Parachristou; "A register file and scheduling model for application specific processor synthesis", 33rd Design Automation Conference, pp. 35-40, 1996.

[17] R. Yung et N.C. Wilhelm, "Caching processor general registers", International Conference on computer Design : VLSI in Computers and Processors, pp. 307-12.

[18] Z. Tang; C. Zhang; L. Sifei; T. Yu, "A new architecture for branch-intensive loops", Proceedings. Advances in Parallel and Distributed Computing, pp. 241-6.

[19] S. Dutta; A. Wolf; W. Wolf et K.J. O'Connor, "Design issues for very-long-instruction-word VLSI video signal processors", VLSI Signal Processing Conference, IX, pp. 95-104.

[20] H. Qing et H.C. Huan, "RNIW : a novel general-purpose DSP architecture", 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, pp. 3302-5, Vol. 6.

[21] M. Moonen ; P. Van Dooren et J. Vandewalle, "Combined Jacobi-type algorithms in signal processing", SVD and Signal Processing, II. Algorithms, Analysis and Application, Amsterdam, Netherlands, 25-27 June 1990.

[22] W.I. Mitchell, "Systolic array for Kalman filtering with algorithm-based fault tolerance", SVD and Signal Processing, II. Algorithms, Analysis and Application, Amsterdam, Netherlands, 25-27 June 1990.

[23] S.Y. Kung et J.N. Hwang, "Systolic array designs for Kalman filtering", IEEE Trans. on signal Processing, Vol. 39, pp. 171-82, Jan. 91.

[24] A. Barwicz, "System Approach to Electrical Measurements", Conference Record IEEE IMTC/93, Irwine, California, May 18-20, pp. 397-402, 1993.

[25] D. Alba et G.R. Meira, "Inverse optimal filtering method for the instrumental spreading correction in size excution chromatography", Journal of liquid chromatography, Vol. 7, N^0 14, pp. 2833-2862, 1984.

[26] M. Ben Slima, R. Z. Morawski et A. Barwicz, "Spline-based Variational Methode with Constraints for Spectrometric Data Correction", IEEE Trans. Instrumentation and Measurement, Vol. 41, N^o 6, pp. 786-790, Dec 1992.

[27] D. Massicotte, R.Z. Morawski, et A. Barwicz, "Efficiency of Constraining the Set of Feasible Solutions in Kalman-Filter-Based Algorithms of Spectrometric Data Correction", Conference Record IEEE IMTC/93, Irwine, California, pp. 496-499, May 18-20 1993.

[29] M. Najim, Modélisation et identification en traitement du signal, Masson, 1988.

[30] D. Massicotte, M.A. Santerre, Y. Savaria and A. Barwicz, "Structure de calculs parallèles pour le filtrage de Kalman dans la reconstitution de signaux", CCECE'92, Toronto, pp. TM4.16.1-TM4.16.4, 1992.

[31] M. Kunt et al., "Techniques modernes de traitement numérique des signaux", Presse polytechniques et universitaires romandes, vol. 1, 1991.

[32] M. Béllanger, "Traitement numérique du signal-théorie et pratique", Masson, 1990.

[33] J.L Hennessey ET D. A. Patterson, "Architectures des ordinateurs :une approche quantitative", McGRAW-HILL 1992.

[35] M. Ben Slima, L. Szczecinski, D. Massicotte, A. Barwicz, and R.Z. Morawski, "Algorithmic Specification of an Specialized Processor For Spectrometric Applications", *IEEE Instrum&Meas. Technology Conference (IMTC/97)*, Ottawa, Canada, pp. 90-95, 19-21 May 1997.

[36] H. T. Kung, "Why Systolic Architectures", IEEE Trans. Comp., pp. 37-46, January 1982.

[37] L. Snyder, "Introduction to the Configurable Highly Parallel Computer", IEEE Trans. Comp., pp. 47-56, January 1982.

[38] P. Quinton et Y. Robert, "Algorithmes et architectures systoliques", Masson, 1989.

[39] A. Barwicz et al., "An Application-Specific Processor Dedicated to Kalman-Filter-Based Correction of Spectrometric data", IEEE Trans. Instr. & Meas. vol. 44, N^0. 3, June 1995.

[40] M. Newell et J. Rasure, "A VLSI System for Real-Time Linear Operations and Transforms", IEEE Trans. Signal Processing, vol. 39. N^0. 8, August 1991.

[41] S.Y. Kung, "VLSI Array Processor, Information and System Science Service", Englewood Cliffs, Prentice Hall, 1988.

[42] J. A. Fischer, "Very long instruction word architectures and the ELI-52", International Symposium of Computer Architecture, June, 1983.

[43] Mark Atkhins, "Performance and the i860 microporcessor". *IEEE Micro*, 11(5), October 1991.

[44] Intel. *i860-64 Bit Microprocessor, Assembler and Linker Reference Manual,* 1989.

[45] C. Stephens, B. Cogswell, J. Heinlein, G. Palmer and J. P, Shen. Instruction Level Profiling and evaluation of the IBM Rs/6000. In The 18th Annual International Symposium on Computer Architecture, ACM SIGARCH, page 180, May 1991.

[46] Robert P. Colwell, Robert P. Nix, John J. O'Donnell, David B. Papworth, and Paul K. Rodman. "A VLIW Architecture for a Trace Scheduling Compiler". *IEEE Trans. On Computers*, 37(8) :967, August 1988.

[47] G. . Slavenburg, Philips Semiconductors, "The Trimedia Tm-1 PCI VLIW Mediaporcessor", Hotchips VIII, Stanford, August 1996.

[48] U. Weiser, Intel Israel, "Intel MMX Technology", Hotchips VIII, Stanford, August 1996.

[49] Linley Gwennap, Intel's MMX Speeds Multimedia, Microprocessor Report, Vol. 10, Nr. 3, March 5, 1996.

[50] J.H. Moreno, M. Moudgill, K. Ebcioglu, E. Altman, B. Hall, R. Miranda, S. K. Chen, A. "Polyak, Architecture, compiler, and simulation of a tree-based VLIW processor", International Symposium of Computer Architecture, June, 1997.

[51] P.Fahler, P; Nagel, C.; Rammig, F. J., Kastas U., "Design of VLIW architecture constructed from standard RISC chips : a case study of hardware/software codesign", Microporocessing and microprogramming, vol. 38 Iss :1-5 p.61-68, Sept 1993.

[52] W. Wilcke, "MIMD computers for scientific application", International Journal of high speed computing, Vol :5, Iss : 3, p.403-411, Sept 1993.

[53] M.A., Schuette; J.P.Shen, "Instruction level experimental evaluation of the Multiflow TRACE 14/300 VLIW computer", Journal of supercomputimg Vol :7 Iss :1-2, p.249-271, May 1993.

[54] S. Moon, S.D Carson, "Generalized multiway branch unit for VLIW microprocessors", IEEE Trans. On Parallel and distributed systems, Vol. 6 Iss :8, p.850-62.

[55] S. Novack; A. Nicolam, "A hierarchical apporcah to instruction-level parallelisation", International Journal of parallel Programming, Vol :23 Iss :1, p.35-62, Feb.95.

[56] Andrea Capitano, Nikil Dutt and Alexander Nicolau, "Multi ported register file complexity analysis", Technical Report 92-58, UC Irvine, ICS Dept., 1992.

[57] M. Kuga, K. Murakami, and S. Tomita, "DSNS : Yet Another Superscalar Processor Architecture", *Computer Architecture News (SIGARCH)*, 19(4) :14, June 1991.

[58] K. Ebcioglu, "Some design Ideas for a VLIW Architecture for Sequential Natured Software", In *Parallel Processing, Proc. IFIP WG 10.3 Working Conference on Parallel Processing,* 1988.

[59] Roni Potasman, "Perculation Based Compiling for Evaluation of Parallelism and Hardware Design Trade-Offs", PhD thesis, University of California, Irvine, Dept. of Information and Computer Science. 1992.

[60] Alexander Nicolau, "Perculation Scheduling : a Parallel Compilation Technique", Technical report, TR 85-678, Cornell University, 1984.

[61] M.L Anido, D.J. Allerton and E.J Zaluska, "A three-port/three-access register file for concurrent processing and I/O communication in a RISC like graphics engine", In The 16th Annual International Symposium on COMPUTER ARCHITECTURE, page 354, 1989.

[62] W. Maly et al. Memory chip for 24-port register file. Technical Report, Carnegie Mellon University, 1990.

[63] Dan Gajski. Dual port register file. Personal Communication, 1989.

[64] K. Sakkay, D. Massicotte, A. Barwicz, "A Specialised Architecture for reconstruction algorithms resolution based on Kalman Filter", ACFAS Conference, June '97, Quebec.

[65] K. Sakkay, D. Massicotte, A. Barwicz, "A Systolic Architecture for a Signal Reconstruction based on Kalman-Spline and LMS Filters", Canadian Congress in Electrical and Computer Engineering, Waterloo-'98, Ontario.

[66] K. Sakkay, D. Massicotte, A. Barwicz, "A Parallel Processor Design Dedicated to Spectrometric Applications", TEXPO98, Canadian Microelectronic Corporation, Ottawa 1998, Ontario.

[67] M.A. Santerre, D. Massicotte, A. Barwicz et Y. Savaria, "Architecture d'un processeur spécialisé pour la reconstitution de signaux basée sur le Filtre de Kalman", CCGEI'92, Toronto, Ontario, pp.. MM7.3.1-MM7.3.4, , 13-16 Sept., 1992.

[68] M.A Santerre, "Architecture d'un processeur spécialisé pour la reconstitution des signaux basée sur le filtrage de Kalman", Mémoire de maîtrise en électro-industrielle, 1996.

[69] M. Ben Slima, R. Z. Morawski et A. Barwicz, "Kalman-Filter-Based Algorithms of Spectrometric Data Correction-Part II: use of Splines for Approximation of Spectra", IEEE Trans. Instrumentation and Measurement, Vol. 46, N° 3, pp. 685-689, June 1997.

[70] J.H. Ahlberg, E.N. Nilson and J.L. Walsh, "The Theory of Splines ans Their Applications", New York: Acadimic, 1967.

ANNEXE A

```
function [y]=Quantif(x,B,Am);

% x   : vector or scalar
% B   : Bit number
% Am  : amplitude range

q=Am*2^(-B+1);
q1=round(x/q);
y=q1*q;
z=(abs(y)>Am/2);
a=find(z);
      if isempty(a)==0
   K=length(a);
      for n=1:K
   i=a(n);
   y(i)=Am/2*sign(y(i));
      end
      end

end
```

```
function [Y]=Quantification(x,NB);

if x==0
   for i=1:NB;
   y(i)='0';
   end;
end;

if x>=0
   sign='0';

else
   sign='1';

end;

for i=1:(NB-1)
   x=abs(x)*2;
   if x>=1
      y(i)='1';
      x=x-1;
   elseif x<1
      y(i)='0';
      x=x;
   else
      x=0;
      y(i)=0;
   end;

end;

B=strcat(sign,y(1),y(2),y(3),y(4),y(5),y(6),y(7),y(8),y(9),y(10),y(11),y(
12),y(13),y(14),y(15),y(16),y(17),y(18),y(19),y(20),y(21),y(22),y(23),y(2
4),y(25),y(26),y(27),y(28),y(29),y(30),y(31));
Y=[B];
end
```

```
function y=Quantifm(x,B,Am);

% x  : Matrix
% B  : Bit number
% Am : Amplitude range

[N,M]=size(x);
      for i=1:M;
   y(:,i)=Quantif(x(:,i),B,Am);
      end

end
```

```
% FILTRE DE KALMAN AVEC q(t) GAUSSOIDALE %

function [xrq,Kn,x1q,ZZ]=discr(y,LD,PHI,b,H,POS,Beta,Bc,Bd)

% APPLICATION DU FILTRE DE KALMAN #1 %
  Initialisation des matrices et des vecteurs

N=length(y);
Z1q=zeros(LD+3,1);
Zq=zeros(LD+3,1);
Sn=1e4*eye(LD+3);
xr=zeros(1,N);


        for k=1:50

   sig = PHI*Sn*PHI' + Beta*b*b';
   S=sig*H;
   Sn=sig-(S*H'*sig)/(1+H'*S);
   Kn  = Sn*H;
        end

   %G=(PHI-Kn*H'*PHI)*.25;
   %Gq=Quantifm(G,Bd,2);
%  Kn=Kn*0.01;
%  Knq=Quantif(Kn,Bd,2);
   Knq=Kn;
   yq=Quantif(y/75,Bd,2);

        for k=1:N

   for   i=1:LD+3
   Tq=0;

      for   j=1:LD+3

   A=FHI(i,j)*Zq(j);
   B=H'*A;
   Aq=Quantif(A,Bc,2);
   Bq=Quantif(B,Bc,2);
   T=Tq+Aq;
   Tq=Quantif(T,Bc,2);
      end

   I=Knq(i)*yq(k);
   Iq=Quantif(I,Bc,2);
   F=Tq*4+Iq;
   Z1q(i)=Quantif(F,Bc,2);
```

```
   end

   ZZ(:,k)=Z1q(:);
   Zq=Z1q;
   Z1q=0;

   x1q(N+1-k)=Quantif(Zq(1),Bc,2);

         end

break
                APPLICATION DU FILTRE DE KALMAN #2
% Initialisation des matrices et des vecteurs

Zq=zeros(LD+3,1);
Z1q=zeros(LD+3,1);

      for k=1:N

   for   i=1:LD+3
   Tq=0;

      for   j=1:LD+3
   A=Gq(i,j)*Zq(j);
   Aq=Quantif(A,Bc,2);
   T=Tq+Aq;
   Tq=Quantif(T,Bc,2);
      end

   I=Knq(i)*x1q(k);
   Iq=Quantif(I,Bc,2);
   F=Tq*10+Iq;
   Z1q(i)=Quantif(F,Bc,2);
   end
   Zq=Z1q;
   Z1q=0;

if POS
      if Zq(1)<0
   Zq(1)=0;
      end
end

   xrq(N+1-k)=Quantif(Zq(1)*75,Bc,2);

         end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FILTRE DE KALMAN AVEC g(t) GAUSSOIDALE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
load data_kamal
POS=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%      INITIALISATION DES PARAMETRES       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialisation des matrices et des vecteurs

N=length(y);
M=length(PHI);
Z=zeros(M+1,1);
xr=zeros(1,N);
G=(PHI-Kn*H'*PHI);
F=[G Kn; zeros(1,M+1)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% APPLICATION DU FILTRE DE KALMAN #1 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


         for k=1:N

   Z(M+1)= y(k);
   Z     = F*Z;
   x1(N+1-k)=Z(1);
         end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION DU FILTRE DE KALMAN #2 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialisation des matrices et des vecteurs

Z=zeros(M+1,1);

         for k=1:N

   Z(1:M)=Z(1:M);
   Z(M+1)= x1(k);
   Z     = F*Z;

if POS
      if Z(1)<0
   Z(1)=0;
      end
```

```
end

    xr(N+1-k)=Z(1);
       end
```

```
function [y,xrq]=CCGEI981(NF,DBN,CBN,ABN);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % FILTRE DE KALMAN AVEC v(t) GAUSSOIDALE %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load data_kamal
POS=1;

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%     INITIALISATION DES PARAMETRES     %%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialisation des matrices et des vecteurs

RS=10*NF;
N=length(y);
M=length(PHI);
Z=zeros(M+1,1);
xrq=zeros(N,1);
G=(PHI-Kn*H'*PHI);
NG=10;
NFG=16;
NGK=NG*NFG;
F=[G/NFG Kn/NGK; zeros(1,M+1)];

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%     QUANTIFICATION DES PARAMETRES     %%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

F=Quantifm(F,DBN,2);
y=Quantif(y/NF,DBN,2);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%% APPLICATION DU FILTRE DE KALMAN #1  %%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        for k=1:N


   Z(M+1)= y(k);
   Z     = 16*Z;
   Z(1:M)= Quantifm(Z(1:M),CBN,2);
   Z     = Produimv(F,Z,ABN) ;
   Z     = Quantifm(Z,CBN,2);
   x1(N+1-k)=Z(1);

        end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%    APPLICATION DU FILTRE DE KALMAN #2  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  Initialisation des matrices et des vecteurs

Z=zeros(M+1,1);
x1(:)=x1*NG;
         for k=1:N

   Z(M+1)= x1(k);
   Z      = 16*Z;
   Z(1:M)= Quantifm(Z(1:M),CBN,2);
   Z      = Produimv(F,Z,ABN) ;
   Z      = Quantifm(Z,CBN,2);


if POS
      if Z(1)<0
   Z(1)=0;
      end
end

   xrq(N+1-k,1)=Z(1);
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%    RESULT PRINTING      %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xrq(:,1)=xrq(:,1)*RS;
plot(xrq,'b');
```

```
load data_kamal;
i=1;
j=1;
POS=1;
erq=zeros(8,8);
xr=EuroS;
mu=0.2;

   for DBN=12:2:32;
   j=1;

      for CBN=12:2:32;
ABN=DBN+CBN;
xrq=CCGEI981(1000,DBN,CBN,ABN);
erq(i,j)=norm(xrq-xr)/norm(xr);
[AmpliPeak,B]=LMSPQ(y,g,mu,POS,t,Pos,DBN,CBN,ABN);
erqa(i,j)=norm(AmpliPeak-Amp)/norm(Amp);

j=j+1;
      end

   i=i+1;
   end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     %%% APPLICATION DE LA METHODE LMS POUR L'ESTIMATION DES AMPLITUDES

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [AmpliPeak,B]=LMSPQ(y,g,mu,POS,t,Pos,DBN,CBN,ABN)

%  y     : results of measurements
%  t     : time-axis
%  g     : calibrated transoperator
%  mu    : relaxation coefficient
%  Pos   : Peak Positions

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%     QUANTIFICATION DES PARAMETRES        %%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

NF=1000;
y=Quantif(y/NF,DBN,2);
g=Quantif(g,DBN,2);
g=fftshift(g);
dt=t(2)-t(1);
N=length(y);
M=length(Pos);
gn=zeros(M,1);

L1=ceil((Pos-t(1))/dt);

      for i=1:M
   j=L1(i);
   B(i)=y(j+1);
      end
A=B(:);

   for i=1:N

      for k=1:M

         if  (i+N/2-L1(k))<=N & (i+N/2-L1(k))>0
        gn(k)=g(i+N/2-L1(k));
         else
        gn(k)=0;
         end
      end

   ga=Produimv(gn',A,ABN) ;
   z=y(i)-ga;
   z=Quantif(z,CBN,2);
   yest=gn*z;
   I=Quantif(mu*yest,CBN,2);
   A=A+I;
   A=Quantif(A,CBN,2);
   end
```

```
if POS
A=A.*(A>0);
end

AmpliPeak=NF*A;
AmpliPeak=AmpliPeak';
end
```

```
% -DETERMINATION DU MODELE D'ETAT D'UN SYSTEME DE MESURE-
%      LE CANAL DE MESURE EST MODELISE PAR UNE EQUATION
%      DIFFERENTIELLE LINEAIRE AVEC PARAMETRES CONSTANTS
%         (EDLPC)
%
%              (utilise avec l'algorithme sol_kal1.m)
%
%
%      x(n+1) = PHI*x(n) + b*E(n)
%                 %
%      y(n)   = H'*x(n)  + M(n)
%                 %
% Incorporation de la contrainte de la deuxieme derivee du spline


function [PHI,b,H]=mod_etat(A,b1,c,dt,l)

W00=dt/2;W01=dt^2/12;W10=dt/2;W11=-W01;



P1=eye(l) - W10*A - W11*A^2;      Matrice P1
P0=eye(l) + W00*A + W01*A^2;    % Matrice P0
B00=W00*b1 + W01*A*b1;        Vecteur B00
B01=W01*b1;           Vecteur B01
B10=W10*b1 + W11*A*b1;        % Vecyeur B10
B11=W11*b1;           Vecteur B11

M1=[0 0 1 0 zeros(1,l);1 0 0 0 zeros(1,l);
    0 1 0 0 zeros(1,l);0 0 3 -dt zeros(1,l);
    zeros(l,1) zeros(l,1) -B10 -B11 P1];

M0=[0 0 1 0 zeros(1,l);0 0 1 0 zeros(1,l);
    0 0 0 1 zeros(1,l);3  dt 0 4*dt zeros(1,l);
    zeros(l,1) zeros(l,1) B00 B01 P0];


b=inv(M1)*[dt;0;0;0;zeros(l,1)];          Vecteur o

H=[0;0;0;0;c];                    Vecteur H

PHI=inv(M1)*M0;
end
```

```
function [MV]=Produimv(M,V,Bit2)

   for i=1:length(M(:,1));

Aq=0;

      for j=1:length(V);
   A=Aq+M(i,j)*V(j);
   Aq=Quantif(A,Bit2,2);
      end

%MV(i,1)=Aq;
MV(i,1)=A;
   end
```

```
function [MV]=Produivv(M,V,Bit2)

   for i=1:length(M(:,1));

Aq=0;

      for j=1:length(V);
   A=M(i)*V(i);
   Aq=Quantif(A,Bit2,2);
      end

MV(i,1)=Aq;
%MV(i,1)=A;
   end
```

```
--Univesite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Labratoire de Micro-Electronique et de Micro-Systemes
--Hiver 97
--Kamal SAKKAY
--Porgramme VHDL du demultiplexeur 1 a 2 utilise dans SYS_KAL_SPL
------------------------------------------------------------
------------------------------------------------------------
---------------------------LIBRARY--------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
------------------------------------------------------------
------------------ Demultiplexer Entity --------------------
ENTITY demux12 IS

      PORT        (     DIN         :  IN       BIT_24;
                        DOUT0       :  OUT      BIT_24;
                        DOUT1       :  OUT      BIT_24;
                        Sel_dm      :  IN       std_logic);

END demux12;
------------------------------------------------------------
------------------------------------------------------------
--------------- Demultiplexer Architecture -----------------

ARCHITECTURE behav_demux12 OF demux12 IS

BEGIN

      demultiplexeur_Process  : PROCESS(sel_dm, DIN)

      BEGIN

            CASE Sel_dm IS

            WHEN  '0'   =>

               DOUT0    <=    DIN;

            WHEN  '1'   =>

               DOUT1    <=    DIN;


                        WHEN OTHERS =>

                              DOUT0       <=          (OTHERS=>'Z');
                              DOUT1       <=          (OTHERS=>'Z');


                  END CASE;
```

```
      END PROCESS demultiplexeur_process;

END behav_demux12;
-------------------------------------------------------------
-------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de Micro-Electronique et Micro-systemes
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du Processeur_Elementaire_1 utilise pour
--la conception de SYS_KAL_SPL_LMS
-------------------------------------------------------------------------
----------------------------- LIBRARY USED ------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
USE WORK.ALL;
-------------------------------------------------------------------------
------------ Processeur_Elementaire ENTITY DESCREPTION IS ------------

ENTITY D_B IS

PORT        ( DIN1      :     IN          BIT_24;
              DIN2      :     IN          BIT_24;
              DOUT      :     OUT         BIT_24;
              ETA       :     IN          BIT_2;
              Instr     :     IN          BIT_2;
              Pc        :     IN          std_logic;
              CLK       :     IN          std_logic;
              EN_DB     :     IN          std_logic;
              ACTV      :     OUT         std_logic);

END D_B;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
--------- Processeur_Elementaire ARCHITECTURE DESCREPTION IS ----------

ARCHITECTURE struct_DB OF D_B IS

COMPONENT mux2a1

      PORT        (     DIN0        :     IN          BIT_24;
                        DIN1        :     IN          BIT_24;

                        DOUT        :     OUT   BIT_24;
                        Sel_mx      :     IN          std_logic);


END COMPONENT;

COMPONENT   demulxe

            PORT  (     DIN         :  IN       BIT_24;
                        DOUT0       :     OUT   BIT_24;
                        DOUT1       :     OUT   BIT_24;
                        Sel_dm      :     IN    std_logic);
```

```
END COMPONENT;

COMPONENT   stack

            PORT  (     DIN         : IN        BIT_24;
                        DOUT        : OUT       BIT_24;
                        CLK         : IN        std_logic;
                        EN_St       : IN        std_logic;
                        Sel         : IN        std_logic);

END COMPONENT;


SIGNAL            Dmx_Sta           :     BIT_24;
SIGNAL            Sta_Mac           :     BIT_24;
SIGNAL            Mac_3SR           :     BIT_24;
SIGNAL            Dmx_Mac           :     BIT_24;

BEGIN

      U1    :     demulxe

            PORT  MAP   (INPUT1, Dmx_Mac, Dmx_Sta, Sel_dm);

      U2    :     mac

            PORT  MAP   (Dmx_Mac, Sta_Mac, MCO,
                                    Mac_3SR, CLK, CLR_I, EN_mac_I);

      U3    :     stack

            PORT  MAP   (Dmx_Sta, Sta_Mac, CLK,
                                    EN_St_I, Sel_dm);

      U4    :     Tr_State_register

            PORT  MAP   (Mac_3SR, OUTPUT, EN_3str_I,
                                    EN_3str_O, CLK);

      U5:   Contr_register

            PORT  MAP   (EN_St_I, EN_St_O, CLK);

      U6    :     Contr_register

            PORT  MAP   (EN_mac_I, EN_mac_O, CLK);

      U7 :  Contr_register

            PORT  MAP   (CLR_I , CLR_O, CLK);

END struct_DB;
-------------------------------------------------------------
-------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de Micro-Electronique et Micro-Systemes
--Hiver 97
--Kamal SAKKAY
--Programme du registre1 qui servira pour la transmission de commande
--entre les blocs
-------------------------------------------------------------------------
------------------------- Library used IS -------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
-------------------------------------------------------------------------
---------------- REGISTRE1 ENTITY DESCRIPTION IS ------------------------
ENTITY contr_Register IS

      PORT  (DIN  :     IN    std_logic;
             DOUT :     OUT   std_logic;
             CLK  :     IN    std_logic);

END contr_Register;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------- REGISTRE1 ARCHITECTURE DESCRIPTION IS --------------------

ARCHITECTURE behav_register OF contr_register IS

BEGIN

      PROCESS

      BEGIN

            WAIT UNTIL CLK='1' and (CLK'EVENT);

                  DOUT<=DIN;

      END PROCESS;

END behav_register;
-------------------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de Micro-Electronique et Micro-Systemes
--Hiver 97
--Kamal SAKKAY
--Porgramme VHDL du bolc de decision utilise dans
--la conception de Sys_Kal_Spl_Lms
-------------------------------------------------------
------------------- LIBRARY USED ARE ------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
-------------------------------------------------------
---------- Deci_Bloc ENTITY DESCRIPTION IS ------------
ENTITY LOGIC_BLOC IS

        PORT   ( DIN1  :        IN      BIT_24 ;
                 DOUT  :        OUT     BIT_24 ;
                 S     :        IN      BIT_VECTOR(1 DOWNTO 0) ;
                 Shift :        IN      BIT ;
                 CLK   :        IN      STD_LOGIC ;
                 ACTV  :        OUT     STD_LOGIC );

END LOGIC_BLOC;
-------------------------------------------------------
-------------------------------------------------------
-------- Deci_Bloc ARCHITECTURE DESCRIPTION IS ---------

ARCHITECTURE behav_LOGIC_BLOC OF LOGIC_BLOC IS
BEGIN

        PROCESS
        VARIABLE    Inter   :       BIT_24;
        VARIABLE    D       :       STD_LOGIC ;

        BEGIN

                WAIT UNTIL CLK'EVENT and CLK='1';

                        Inter   :=      DIN1 ;

                        CASE    S       IS

                                WHEN    "00"    =>

                                IF      Inter(23)='1'       THEN
                                Inter   := "000000000000000000000000" ;
                                END IF;

                                WHEN    "01"    =>

                                IF      Inter(23)='1'       THEN
                                D := '1' ;
```

```
                    ELSE
                    D := '0' ;
                    END IF;

                    WHEN  OTHERS          =>

                END CASE;

                CASE  Shift    IS

                    WHEN  '0' =>

                    Inter     :=(Inter(23) & Inter(18 DOWNTO 0) & "0000");

                    WHEN  '1' =>

                    Inter     := Inter ;

                END CASE;

                DOUT  <=      Inter ;
                ACTV  <=      D ;

        END PROCESS;

END behav_LOGIC_BLOC;
-------------------------------------------------------
-------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole D'Ingenierie
--Laboratoire de micro-elctronique
--Hiver97
--Kamal SAKKAY
--Programme VHDL du Registre utilise dans la
--conception de SYS_KAL_SPL_LMS
--------------------------------------------------------------------
----------------------LIBRARY USED----------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
--------------------------------------------------------------------
------------ Registre Entity Description----------------------

ENTITY Tr_State_register IS
  PORT (        DIN             :       IN      BIT_24;
                DOUT            :       OUT     BIT_24;
                EN_IN           :       IN              std_logic;
                CLK             :       IN      std_logic);
END Tr_State_register;
--------------------------------------------------------------------
--------------------------------------------------------------------
----------- Registre Architecture Description -----------------
ARCHITECTURE behav_Tr_State_register OF Tr_State_register IS
BEGIN

  PROCESS(CLK)

  BEGIN

        WAIT UNTIL (CLK='1') and (CLK'EVENT);

                IF EN_IN='1' THEN

                        DOUT<=DIN;
                ELSE
                        DOUT<=(OTHERS=>'Z');

                END IF;

  END PROCESS;

END behav_Tr_State_register;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

```
--Univesite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoir de micro-elctronique et micro-systemes
--Kamal SAKKAY
--Programme VHDL de la pile utilisee dans la conception du
--processeur Sys_Kal_Spl_Lms
-------------------------------------------------------------------
---------------------------LIBRARY USED--------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
----------------------ENTITY DESCREPTION IS---------------------

ENTITY stack IS

      PORT        (     DIN         : IN  BIT_24;
                        DOUT        : OUT BIT_24;
                        CLK         : IN  std_logic;
                        EN_St       : IN  std_logic;
                        Sel         : IN  std_logic);

END stack;
-------------------------------------------------------------------
-------------------------------------------------------------------
-----Architecture de la pile qui contient le vecteur g et a-------

ARCHITECTURE behav_stack OF stack IS

BEGIN

      PROCESS

      VARIABLE    St_Tmp      :     St_Vec;
      VARIABLE    Inter       :     BIT_24;

      BEGIN

            WAIT UNTIL (CLK='1') and (CLK'EVENT);

                  IF (EN_St='1') and (Sel='0') THEN


                        DOUT<=St_Tmp(0);
                        Inter:=St_Tmp(0);


                              FOR i IN 0 TO (St_Dep-1)LOOP
                              St_Tmp(i):=St_Tmp(i+1);
                              END LOOP;

                        St_Tmp(St_Dep):=Inter;
```

```
                    ELSIF Sel='1' THEN

                                FOR i IN 0 TO (St_Dep-1)LOOP
                                St_Tmp(i):=St_Tmp(i+1);
                                END LOOP;

                          St_Tmp(St_Dep):=DIN;

                    END IF;

      END PROCESS;

END behav_stack;
--------------------------------------------------------------------
--------------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole D'Ingenierie
--Laboratoire de micro-elctronique
--Hiver97
--Kamal SAKKAY
--Programme VHDL du Registre utilise dans la
--conception de SYS_KAL_SPL_LMS
-------------------------------------------------------------
-----------------------LIBRARY USED--------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
-------------------------------------------------------------
------------ Registre Entity Description------------------

ENTITY DATA_Register IS

   PORT     (     DIN         :     IN    BIT_24;
                  DOUT        :     OUT   BIT_24;
                  CLK         :     IN    std_logic);

END DATA_Register;
-------------------------------------------------------------
-------------------------------------------------------------
----------- Registre Architecture Description -------------

ARCHITECTURE behav_register OF DATA_Register IS

BEGIN

   PROCESS(CLK)

   BEGIN

            WAIT UNTIL (CLK='1') and (CLK'EVENT);

                              DOUT<=DIN;

   END PROCESS;

END behav_register;
-------------------------------------------------------------
-------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de Micro-Electronique et Micro-systemes
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du Processeur_Elementaire_1 utilise pour
--la conception de SYS_KAL_SPL_LMS
--------------------------------------------------------------------------
----------------------------- LIBRARY USED ---------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
USE WORK.ALL;
--------------------------------------------------------------------------
------------ Processeur_Elementaire ENTITY DESCREPTION IS ------------

ENTITY Processing_Element_1 IS

      PORT        ( INPUT1          :     IN    BIT_24;
                    INPUT2          :     IN    BIT_24;
                    OUTPUT          :     OUT   BIT_24;
                    MCO             :     OUT   BIT_24;
                    CLK             :     IN    std_logic;
                    CLR_I           :     IN    std_logic;
                    CLR_O           :     OUT   std_logic;
                    EN_3str_I       :     IN    std_logic;
                    EN_3str_O       :     OUT   std_logic;
                    EN_mac_I        :     IN    std_logic;
                    EN_mac_O        :     OUT   std_logic;
                    EN_St_I         :     IN    std_logic;
                    EN_St_O         :     OUT   std_logic;
                    Sel_mx          :     IN    std_logic;
                    Sel_dm          :     IN    std_logic);

END Processing_Element_1;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------- Processeur_Elementaire ARCHITECTURE DESCREPTION IS ----------

ARCHITECTURE struct_PE OF Processing_Element_1 IS

COMPONENT   mac

      PORT  (     MC          :     IN    BIT_24;
                  MP          :     IN    BIT_24;
                  DOUT  :  OUT      BIT_24;
                  CLK   :     IN    std_logic;
                  CLR   :     IN    std_logic;
                  EN          :     IN    std_logic);

END COMPONENT;

COMPONENT   demux12
```

```
      PORT        (     DIN         :  IN       BIT_24;
                        DOUT0       :     OUT   BIT_24;
                        DOUT1       :     OUT   BIT_24;
                        Sel_dm      :     IN    std_logic);

END COMPONENT;

COMPONENT   stack

            PORT  (     DIN         : IN        BIT_24;
                        DOUT        : OUT       BIT_24;
                        CLK         : IN        std_logic;
                        EN_St       : IN        std_logic;
                        Sel         : IN        std_logic);

END COMPONENT;

COMPONENT   DATA_Register

   PORT     (     DIN         :     IN    BIT_24;
                  DOUT        :     OUT   BIT_24;
                  CLK         :     IN    std_logic);

END COMPONENT;

COMPONENT   Tr_State_register

            PORT  (     DIN         : IN        BIT_24;
                        DOUT        : OUT       BIT_24;
                        EN_IN       : IN        std_logic;
                        CLK         : IN        std_logic);

END COMPONENT;

COMPONENT   Contr_register

            PORT  (     DIN         : IN        std_logic;
                        DOUT        : OUT       std_logic;
                        CLK         : IN        std_logic);

END COMPONENT;

COMPONENT   mux2a1

      PORT        (     DIN0        :     IN          BIT_24;
                        DIN1        :     IN          BIT_24;

                        DOUT        :     OUT   BIT_24;
                        Sel_mx      :     IN          std_logic);


END COMPONENT;
```

```
SIGNAL          Mux_Mac          :     BIT_24;
SIGNAL          Dmx_Sta          :     BIT_24;
SIGNAL          Sta_Mux          :     BIT_24;
SIGNAL          Mac_3SR          :     BIT_24;
SIGNAL          Dmx_Mac          :     BIT_24;

BEGIN

      U1    :     demux12

            PORT  MAP   (INPUT1, Dmx_Mac, Dmx_Sta, Sel_dm);

      U2    :     mux2a1

            PORT  MAP   (Sta_Mux, INPUT2, Mux_Mac, Sel_mx);

      U3    :     mac

            PORT  MAP   (Dmx_Mac, Mux_Mac, Mac_3SR, CLK,
                                    CLR_I, EN_mac_I);

      U4    :     stack

            PORT  MAP   (Dmx_Sta, Sta_Mux, CLK,
                                    EN_St_I, Sel_dm);

      U5    :     DATA_Register

            PORT  MAP   (Dmx_Mac, MCO, CLK);

      U6    :     Tr_State_register

            PORT  MAP   (Mac_3SR, OUTPUT, EN_3str_I, CLK);

      U7:   Contr_register

            PORT  MAP   (EN_St_I, EN_St_O, CLK);

      U8    :     Contr_register

            PORT  MAP   (EN_mac_I, EN_mac_O, CLK);

      U9:   Contr_register

            PORT  MAP   (CLR_I , CLR_O, CLK);

      U10:  Contr_register

            PORT  MAP   (EN_3str_I, EN_3str_O, CLK);

END struct_PE;
------------------------------------------------------------
------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de Micro-Electronique et Micro-systemes
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du Processeur_Elementaire_1 utilise pour
--la conception de SYS_KAL_SPL_LMS
-------------------------------------------------------------------------
---------------------------- LIBRARY USED ---------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
USE WORK.ALL;
-------------------------------------------------------------------------
------------ Processeur_Elementaire ENTITY DESCREPTION IS ------------

ENTITY Processing_Element IS

      PORT        ( INPUT1          :     IN    BIT_24;
                    OUTPUT          :     OUT   BIT_24;
                    MCO             :     OUT   BIT_24;
                    CLK             :     IN    std_logic;
                    CLR_I           :     IN    std_logic;
                    CLR_O           :     OUT   std_logic;
                    EN_3str_I       :     IN    std_logic;
                    EN_3str_O       :     OUT   std_logic;
                    EN_mac_I        :     IN    std_logic;
                    EN_mac_O        :     OUT   std_logic;
                    EN_St_I         :     IN    std_logic;
                    EN_St_O         :     OUT   std_logic;
                    Sel_dm          :     IN    std_logic);

END Processing_Element;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
--------- Processeur_Elementaire ARCHITECTURE DESCREPTION IS ----------

ARCHITECTURE struct_PE OF Processing_Element IS

COMPONENT   mac

      PORT  (     MC          :     IN    BIT_24;
                  MP          :     IN    BIT_24;
                  DOUT  :  OUT      BIT_24;
                        CLK   :     IN    std_logic;
                        CLR   :     IN    std_logic;
                        EN          :     IN    std_logic);

END COMPONENT;

COMPONENT   demux12

      PORT        (     DIN         :  IN       BIT_24;
```

```
                              DOUT0       :     OUT   BIT_24;
                              DOUT1       :     OUT   BIT_24;
                              Sel_dm      :     IN    std_logic);

END COMPONENT;

COMPONENT   stack

            PORT  (     DIN         : IN        BIT_24;
                              DOUT        : OUT       BIT_24;
                              CLK         : IN        std_logic;
                              EN_St       : IN        std_logic;
                              Sel         : IN        std_logic);

END COMPONENT;

COMPONENT   DATA_Register

   PORT     (     DIN         :     IN    BIT_24;
                  DOUT        :     OUT   BIT_24;
                  CLK         :     IN    std_logic);

END COMPONENT;

COMPONENT   Tr_State_register

            PORT  (     DIN         : IN        BIT_24;
                              DOUT        : OUT       BIT_24;
                              EN_IN       : IN        std_logic;
                              CLK         : IN        std_logic);

END COMPONENT;

COMPONENT   Contr_register

            PORT  (     DIN         : IN        std_logic;
                              DOUT        : OUT       std_logic;
                              CLK         : IN        std_logic);

END COMPONENT;

SIGNAL            Dmx_Sta           :     BIT_24;
SIGNAL            Sta_Mac           :     BIT_24;
SIGNAL            Mac_3SR           :     BIT_24;
SIGNAL            Dmx_Mac           :     BIT_24;

BEGIN

      U1    :     demux12

            PORT  MAP   (INPUT1, Dmx_Mac, Dmx_Sta, Sel_dm);

      U2    :     mac
```

```
            PORT  MAP   (Dmx_Mac, Sta_Mac, Mac_3SR, CLK,
                                    CLR_I, EN_mac_I);

      U3    :     stack

            PORT  MAP   (Dmx_Sta, Sta_Mac, CLK,
                                    EN_St_I, Sel_dm);

      U4    :     DATA_Register

            PORT  MAP   (Dmx_Mac, MCO, CLK);

      U5    :     Tr_State_register

            PORT  MAP   (Mac_3SR, OUTPUT, EN_3str_I, CLK);

      U6    :     Contr_register

            PORT  MAP   (EN_St_I, EN_St_O, CLK);

      U7    :     Contr_register

            PORT  MAP   (EN_mac_I, EN_mac_O, CLK);

      U8 :  Contr_register

            PORT  MAP   (CLR_I , CLR_O, CLK);

      U9 :  Contr_register

            PORT  MAP   (EN_3str_I,EN_3str_O, CLK);

END struct_PE;
-------------------------------------------------------------
-------------------------------------------------------------
```

```
-- University OF Quebec at Trois-Rivieres
-- Egineering school
-- VLSI LAB
-- Winter 97
-- Kamal SAKKAY
-- VHDL descreption of the testbench for the Processing_Element
-- used in PE_S_K_S_L.
-- This architecture is used to stimulate and record test vectors
-- on the RTL VHDL model.  The input vectors are located in
-- u/hping/sakkay/DESIGN_SKSL/HDLs/EUROSENSOR/mactvIn.dat
-- and both the stimuli and the outputs are recorded in mactvOut.dat.
--------------------------------------------------------------------
------------------------ LIBRARIES USED IS -------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
--------------------------------------------------------------------

PACKAGE Test_TYPES_Euro IS

CONSTANT PERIOD              :  TIME      := 50 ns ; --20 MHz Clock
CONSTANT HALF_PERIOD    :  time    := PERIOD / 2 ;

END Test_TYPES_Euro;
--------------------------------------------------------------------
---------------------------- LIBRARIES USED ------------------------
LIBRARY STD ;
LIBRARY IEEE ;
USE STD.textio.ALL ;
USE IEEE.std_logic_1164.ALL ;
USE IEEE.std_logic_textio.ALL ;
USE WORK.PE_TYPES_Euro.ALL ;
USE WORK.Test_TYPES_Euro.ALL ;
USE WORK.ALL ;
--------------------------------------------------------------------
--------------------------------------------------------------------
-------------------- TestBench ENTITY DESCREPTION ------------------

ENTITY PE1_TB IS

END PE1_TB;

--------------------------------------------------------------------
--------------------------------------------------------------------
----------------- TestBench ARCHITECTURE DESCREPTION ---------------

ARCHITECTURE behav_PE1_TB OF PE1_TB IS

COMPONENT  Processing_Element_1

      PORT        ( INPUT1          :     IN    BIT_24;
                          INPUT2          :     IN    BIT_24;
                          OUTPUT          :     OUT   BIT_24;
                          MCO             :     OUT   BIT_24;
                          CLK             :     IN          std_logic;
```

```
                        CLR_I           :     IN          std_logic;
                        CLR_O           :     OUT   std_logic;
                        EN_3str_I :     IN          std_logic;
                        EN_3str_O :     OUT   std_logic;
                        EN_mac_I  :     IN          std_logic;
                        EN_mac_O  :     OUT   std_logic;
                        EN_St_I   :  IN       std_logic;
                        EN_St_O   :  OUT      std_logic;
                        Sel_mx          :     IN          std_logic;
                        Sel_dm          :     IN    std_logic);

END COMPONENT;

   SIGNAL INPUT1        :     BIT_24 ;
      SIGNAL INPUT2           :     BIT_24 ;
   SIGNAL OUTPUT        :     BIT_24 ;
   SIGNAL MCO                 :     BIT_24 ;
   SIGNAL CLK                 :     STD_LOGIC ;
   SIGNAL CLR_I         :     STD_LOGIC ;
      SIGNAL CLR_0            :     STD_LOGIC ;
   SIGNAL EN_3str_I     :     STD_LOGIC ;
   SIGNAL EN_3str_O     :  STD_LOGIC ;
      SIGNAL EN_mac_I   :  STD_LOGIC ;
      SIGNAL EN_mac_O   :     STD_LOGIC ;
      SIGNAL EN_St_I          :  STD_LOGIC ;
      SIGNAL EN_St_O          :  STD_LOGIC ;
      SIGNAL Sel_mx           :  STD_LOGIC ;
      SIGNAL Sel_dm           :  STD_LOGIC ;

BEGIN

   U1 : Processing_Element_1

      Port Map ( INPUT1, INPUT2, OUTPUT, MCO, CLK, CLR_I,
                              CLR_0,EN_3str_I, EN_3str_O, EN_mac_I,
                              EN_mac_O, EN_St_I, EN_St_O, Sel_mx,
Sel_dm) ;


      STIMULUS : PROCESS

      FILE              PETVin        : TEXT IS IN
"../../stimilus/PEtvIn.dat" ;
   FILE                 PETVout       : TEXT IS OUT
"../../stimilus/PEtvOut.dat" ;
   VARIABLE       INline        : LINE ;
      VARIABLE    OUTline     : LINE ;
   VARIABLE       INPUT1_IN   : BIT_24 ;

   BEGIN

      -- Initialisation

            EN_mac_I          <= '0';
```

```
            EN_St_I           <= '0';
            EN_3str_I   <= '0';
            CLR_I             <= '1';
            Sel_dm            <= '0';
            Sel_mx            <= '0';

                        -- Get Data to the FIFO Element


      FOR i IN 7 DOWNTO 0 LOOP

                                                              -- Get a DATA

                                                        READLINE( PETVin ,
INline ) ;
                                                        READ( INline ,
INPUT1_IN ) ;

                                                        -- Synchronize

                                                        CLK     <= '0' ;
                                                              Sel_dm <= '1'
;

                                                        -- Assign input
values

                                                        INPUT1 <= INPUT1_IN
;

                                                        -- Clock Generation
& Enable

                                                        WAIT FOR
HALF_PERIOD;
                                                        CLK           <= '1'
;
                                                        WAIT FOR HALF_PERIOD
;


      END LOOP;

            --Initailisation before the computing cycle
            CLK           <= '0';
            Sel_dm     <= '0';
            EN_St_I       <= '1';
            CLR_I         <= '1';

            WAIT FOR HALF_PERIOD ;
      CLK           <= '1' ;
      WAIT FOR HALF_PERIOD ;
```

```
                                                                    --
Computing cycle

      FOR i IN 7 DOWNTO 0 LOOP

                                                          -- Get a DATA
                                                    READLINE( PETVin ,
INNline ) ;
                                                    READ    ( INline ,
INPUT1_IN ) ;

                                                    -- Synchronize
                                                    CLK         <= '0' ;
                                                          EN_mac_I
      <= '1' ;
                                                          EN_3str_I   <=
'0' ;
                                                          EN_St_I
      <= '1' ;
                                                          CLR_I
      <= '0' ;

                                                    -- Assign input
values

                                                    INPUT1 <= INPUT1_IN
;

                                                    -- Clock Generation
                                                    WAIT FOR HALF_PERIOD
;
                                                    CLK <= '1' ;
                                                    WAIT FOR HALF_PERIOD
;

      END LOOP;


            -- Check for end of file

            IF ENDFILE(PETVin) THEN

          --Synchronize

            CLK         <= '0' ;
                  EN_3str_I   <= '1' ;
                  EN_mac_I    <= '0' ;
```

```
                EN_St_I          <= '0' ;

                --Write output values

                WRITE(OUTline, OUTPUT) ;
           WRITELINE( PETVout, OUTline );

                WAIT FOR HALF_PERIOD ;
        CLK                    <= '1' ;
                WAIT FOR HALF_PERIOD ;

                WAIT;

                END IF;

   END PROCESS;

END behav_PE1_TB;


CONFIGURATION testbench2 OF PE1_TB IS

  FOR behav_PE1_TB
  END FOR ;

END testbench2 ;
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de micro-electronique
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du package des definitions utilise dans PE_SKSL
-------------------------------------------------------------------------
-------------------------------------------------------------------------
------------------------ LIBRARIES USED IS ------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
-------------------------------------------------------------------------

PACKAGE SKSL_TYPES IS


      SUBTYPE     BIT_48      IS STD_LOGIC_VECTOR     (47 DOWNTO 0);
      SUBTYPE     BIT_24      IS STD_LOGIC_VECTOR     (23 DOWNTO 0);
      SUBTYPE     BIT_ADRS IS STD_LOGIC_VECTOR  (12 DOWNTO 0);
      SUBTYPE     BIT_2       IS STD_LOGIC_VECTOR     (1  DOWNTO 0);
      SUBTYPE     BIT_6       IS STD_LOGIC_VECTOR     (5  DOWNTO 0);
      CONSTANT    mu                              :     BIT_6   :="000001";
      CONSTANT    I                               :     BIT_48
:="000000000000000000000000000000000000000000000000";
      CONSTANT IZ                           :     BIT_24
:="000000000000000000000000";
      CONSTANT    N                               :
      BIT_ADRS:="0010000000000";
      CONSTANT    ECH_Y                     :     BIT_ADRS:="0000001100100";
      CONSTANT    ECH_Xi                    :     BIT_ADRS:="0010001100100";
      CONSTANT    ECH_Xr                    :     BIT_ADRS:="0100001100100";
      CONSTANT    ECH_g                     :     BIT_ADRS:="0110001100100";
      CONSTANT St_Dep               :     INTEGER := 7;
      CONSTANT    MO                              :     INTEGER := 7;


      TYPE        St_Vec      IS    ARRAY (St_Dep DOWNTO 0) OF BIT_24;
      TYPE        Z_Vec       IS    ARRAY ((MO-2) DOWNTO 0) OF BIT_24;
      SUBTYPE     E_Gene      IS    STD_LOGIC_VECTOR  (5 DOWNTO 0);

END SKSL_TYPES;

-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de micro-electronique
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du package des definitions utilise dans PE_SKSL
--------------------------------------------------------------------------
--------------------------------------------------------------------------
------------------------ LIBRARIES USED IS -------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
--------------------------------------------------------------------------

PACKAGE SKSL_TYPES IS


      SUBTYPE     BIT_48      IS STD_LOGIC_VECTOR     (47 DOWNTO 0);
      SUBTYPE     BIT_24      IS STD_LOGIC_VECTOR     (23 DOWNTO 0);
      SUBTYPE     BIT_ADRS    IS STD_LOGIC_VECTOR     (12 DOWNTO 0);
      SUBTYPE     BIT_2       IS STD_LOGIC_VECTOR     (1  DOWNTO 0);
      SUBTYPE     BIT_6       IS STD_LOGIC_VECTOR     (5  DOWNTO 0);
      CONSTANT    mu                                  :     BIT_6   :="000001";
      CONSTANT    I                                   :     BIT_48
:="000000000000000000000000000000000000000000000000";
      CONSTANT IZ                             :     BIT_24
:="000000000000000000000000";
      CONSTANT    N                                   :
      BIT_ADRS:="0010000000000";
      CONSTANT    ECH_Y                       :     BIT_ADRS:="0000001100100";
      CONSTANT    ECH_Xi                      :     BIT_ADRS:="0010001100100";
      CONSTANT    ECH_Xr                      :     BIT_ADRS:="0100001100100";
      CONSTANT    ECH_g                       :     BIT_ADRS:="0110001100100";
      CONSTANT St_Dep               :     INTEGER := 7;
      CONSTANT    MO                                  :     INTEGER := 7;


      TYPE        St_Vec      IS    ARRAY (St_Dep DOWNTO 0) OF BIT_24;
      TYPE        Z_Vec       IS    ARRAY ((MO-2) DOWNTO 0) OF BIT_24;
      SUBTYPE     E_Gene      IS    STD_LOGIC_VECTOR  (5 DOWNTO 0);

END SKSL_TYPES;

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

```
-- University OF Quebec at Trois-Rivieres
-- Egineering school
-- VLSI LAB
-- Winter 97
-- Kamal SAKKAY
-- VHDL description of the testbench for the Processing_Element
-- used in PE_S_K_S_L.
-- This architecture is used to stimulate and record test vectors
-- on the RTL VHDL model.  The input vectors are located in
-- u/hping/sakkay/DESIGN_SKSL/HDLs/EUROSENSOR/mactvIn.dat
-- and both the stimuli and the outputs are recorded in mactvOut.dat.
--------------------------------------------------------------------
------------------------ LIBRARIES USED IS -------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
--------------------------------------------------------------------

PACKAGE Test_TYPES_Euro IS

CONSTANT PERIOD         :  TIME       := 50 ns ; --20 MHz Clock
CONSTANT HALF_PERIOD    :  time       := PERIOD / 2 ;

END Test_TYPES_Euro;
--------------------------------------------------------------------
--------------------------- LIBRARIES USED -------------------------
LIBRARY STD ;
LIBRARY IEEE ;
USE STD.textio.ALL ;
USE IEEE.std_logic_1164.ALL ;
USE IEEE.std_logic_textio.ALL ;
USE WORK.SKSL_TYPES.ALL ;
USE WORK.Test_TYPES_Euro.ALL ;
USE WORK.ALL ;
--------------------------------------------------------------------
--------------------------------------------------------------------
-------------------- TestBench ENTITY DESCREPTION ------------------

ENTITY PE_TB IS
END PE_TB;

--------------------------------------------------------------------
--------------------------------------------------------------------
----------------- TestBench ARCHITECTURE DESCREPTION ---------------

ARCHITECTURE behav_PE_TB OF PE_TB IS

COMPONENT  Processing_Element

      PORT        ( INPUT1          :     IN    BIT_24;
                          OUTPUT          :     OUT   BIT_24;
                          MCO             :     OUT   BIT_24;
                          CLK             :     IN          std_logic;
                          CLR_I           :     IN          std_logic;
-                         CLR_O           :     OUT   std_logic;
```

```
                        EN_3str_I :     IN          std_logic;
                        EN_3str_O :     OUT   std_logic;
                        EN_mac_I  :     IN          std_logic;
                        EN_mac_O  :     OUT   std_logic;
                        EN_St_I   :  IN       std_logic;
                        EN_St_O   :  OUT      std_logic;
                        Sel_dm          :     IN    std_logic);

END COMPONENT;

   SIGNAL INPUT1        :     BIT_24 ;
   SIGNAL OUTPUT        :     BIT_24 ;
   SIGNAL MCO                 :     BIT_24 ;
   SIGNAL CLK                 :     STD_LOGIC ;
   SIGNAL CLR_I         :     STD_LOGIC ;
      SIGNAL CLR_0            :     STD_LOGIC ;
   SIGNAL EN_3str_I     :     STD_LOGIC ;
   SIGNAL EN_3str_O     :  STD_LOGIC ;
      SIGNAL EN_mac_I   :  STD_LOGIC ;
      SIGNAL EN_mac_O   :     STD_LOGIC ;
      SIGNAL EN_St_I          :  STD_LOGIC ;
      SIGNAL EN_St_O          :  STD_LOGIC ;
      SIGNAL Sel_dm           :  STD_LOGIC ;

BEGIN

   U1 : Processing_Element

      Port Map ( INPUT1, OUTPUT, MCO, CLK, CLR_I, CLR_0,
                              EN_3str_I, EN_3str_O, EN_mac_I,
                              EN_mac_O, EN_St_I, EN_St_O, Sel_dm) ;


      STIMULUS : PROCESS

      FILE              PETVin        : TEXT IS IN
"../../stimilus/PEtvIn.dat" ;
   FILE                 PETVout       : TEXT IS OUT
"../../stimilus/PEtvOut.dat" ;
   VARIABLE       INline        : LINE ;
      VARIABLE    OUTline    : LINE ;
   VARIABLE       INPUT1_IN  : BIT_24 ;

   BEGIN

      -- Initialisation

            EN_mac_I          <= '0';
            EN_St_I           <= '0';
            EN_3str_I   <= '0';
            CLR_I             <= '1';
            Sel_dm            <= '0';

                        -- Get Data to the FIFO Element
```

```
      FOR i IN 7 DOWNTO 0 LOOP

                                                        -- Get a DATA
INNline ) ;                                             READLINE( PETVin ,
INPUT1_IN ) ;                                           READ( INline ,

                                                        -- Synchronize
                                                        CLK    <= '0' ;
;                                                             Sel_dm <= '1'

values                                                  -- Assign input

;                                                       INPUT1 <= INPUT1_IN

& Enable                                                -- Clock Generation

HALF_PERIOD;                                            WAIT FOR
;                                                       CLK          <= '1'
;                                                       WAIT FOR HALF_PERIOD

      END LOOP;


            CLK          <= '0';
            Sel_dm    <= '0';
            EN_St_I      <= '1';
            CLR_I        <= '0';

            WAIT FOR HALF_PERIOD ;
      CLK          <= '1' ;
      WAIT FOR HALF_PERIOD ;


                                                                                --
Computing cycle

      FOR i IN 7 DOWNTO 0 LOOP
```

```
INline ) ;

INPUT1_IN ) ;



      <= '1' ;

'0' ;

      <= '1' ;

      <= '0' ;

values

;



;

;

      END LOOP;


            -- Check for end of file

            IF ENDFILE(PETVin) THEN

           --Synchronize

            CLK         <= '0' ;
                  EN_3str_I    <= '1' ;
                  EN_mac_I     <= '0' ;
                  EN_St_I            <= '0' ;

                  --Write output values

                  WRITE(OUTline, OUTPUT) ;
            WRITELINE( PETVout, OUTline );
```

```
      -- Get a DATA

READLINE( PETVin ,

READ    ( INline ,

-- Synchronize

CLK         <= '0' ;
      EN_mac_I

      EN_3str_I   <=

      EN_St_I

      CLR_I

-- Assign input

INPUT1 <= INPUT1_IN

-- Clock Generation

WAIT FOR HALF_PERIOD

CLK <= '1' ;
WAIT FOR HALF_PERIOD
```

```
                WAIT FOR HALF_PERIOD ;
        CLK                     <= '1' ;
                WAIT FOR HALF_PERIOD ;

                WAIT;

                END IF;

   END PROCESS;

END behav_PE_TB;


CONFIGURATION testbench2 OF PE_TB IS

  FOR behav_PE_TB
  END FOR ;

END testbench2 ;
```

```
--Universite du Quebec a Trois_Rivieres
--Ecole d'ingenierie
--Laboratoire de micro-electronique et micro-systemes
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du multiplexeur 3 a 1 avec selction
-------------------------------------------------------------------
----------------------------Library used-------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES.ALL;
-------------------------------------------------------------------
---------------------MULTIPLEXER 3_1 ENTITY----------------------
ENTITY mux3a1 IS

      PORT        (           DIN0        :     IN          BIT_24;
                              DIN1        :     IN          BIT_24;
                              DIN2        :     IN          BIT_24;
                              DOUT        :     OUT         BIT_24;
                              Sel_mx      :     IN          BIT_2);


END mux3a1;
-------------------------------------------------------------------
------------------- MULTIPLEXER 3_1 ARCHITECTURE -------------------

ARCHITECTURE behav_mux3a1 OF mux3a1 IS

BEGIN

      Multiplexeur_process:PROCESS(Sel_mx, DIN0, DIN1, DIN2)

      BEGIN

            CASE  Sel_mx      IS

                  WHEN  "00"        =>

                        DOUT  <=    DIN0;

                  WHEN  "01"        =>

                        DOUT  <=    DIN1;

                  WHEN  "10"        =>

                        DOUT  <=    DIN2;


                  WHEN  OTHERS      =>

                        DOUT  <=    (OTHERS=>'Z');
```

```
            END CASE;

      END PROCESS Multiplexeur_process;

END behav_mux3a1;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
--Universite du Quebec a Trois_Rivieres
--Ecole d'ingenierie
--Laboratoire de micro-electronique et micro-systemes
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du multiplexeur 2 a 1 avec selction
-------------------------------------------------------------------
----------------------------Library used-------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
-------------------------------------------------------------------
---------------------MULTIPLEXER 2_1 ENTITY-----------------------
ENTITY mux2a1 IS

      PORT        (     DIN0        :     IN          BIT_24;
                        DIN1        :     IN          BIT_24;
                        DOUT        :     OUT         BIT_24;
                        Sel_mx      :     IN          std_logic);


END mux2a1;
-------------------------------------------------------------------
------------------- MULTIPLEXER 2_1ARCHITECTURE -------------------

ARCHITECTURE behav_mux2a1 OF mux2a1 IS

BEGIN

      Multiplexeur_process:PROCESS(Sel_mx, DIN0, DIN1)

      BEGIN

            CASE  Sel_mx      IS

                  WHEN  '0'         =>

                        DOUT  <=    DIN0;

                  WHEN  '1'         =>

                        DOUT  <=    DIN1;

                  WHEN  OTHERS      =>

                        DOUT  <=    (OTHERS=>'Z');

            END CASE;

      END PROCESS Multiplexeur_process;

END behav_mux2a1;
-------------------------------------------------------------------
```

```
-- University OF Quebec at Trois-Rivieres
-- Egineering school
-- VLSI LAB
-- Winter 97
-- Kamal SAKKAY
-- VHDL description of the testbench for the multiplier accumulator
-- used in PE_S_K_S_L.
-- This architecture is used to stimulate and record test vectors
-- on the RTL VHDL model.  The input vectors are located in
-- u/hping/sakkay/DESIGN_SKSL/HDLs/EUROSENSOR/mactvIn.dat
-- and both the stimuli and the outputs are recorded in mactvOut.dat.
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------

PACKAGE Test_TYPES_Euro IS

CONSTANT PERIOD         :  TIME       := 50 ns ; --20 MHz Clock
CONSTANT HALF_PERIOD    :  TIME       := PERIOD / 2 ;

END Test_TYPES_Euro;
--------------------------------------------------------------------
---------------------------- LIBRARIES USED -------------------------
LIBRARY STD ;
LIBRARY IEEE ;
USE STD.textio.ALL ;
USE IEEE.std_logic_1164.ALL ;
USE IEEE.std_logic_textio.ALL ;
USE WORK.SKSL_TYPES.ALL ;
USE WORK.Test_TYPES_Euro.ALL ;
USE WORK.ALL ;
--------------------------------------------------------------------
--------------------------------------------------------------------
-------------------- TestBench ENTITY DESCREPTION --------------------

ENTITY mac_TB IS
END mac_TB;

--------------------------------------------------------------------
--------------------------------------------------------------------
------------------ TestBench ARCHITECTURE DESCREPTION ----------------

ARCHITECTURE behav_mac_TB OF mac_TB is

   SIGNAL MC            :     BIT_24 ;
   SIGNAL MP            :     BIT_24 ;
   SIGNAL DOUT          :     BIT_24 ;
   SIGNAL CLK           :     STD_LOGIC ;
   SIGNAL CLR           :     STD_LOGIC ;
   SIGNAL EN            :     STD_LOGIC ;


COMPONENT mac
```

```
   PORT      (    MC          :     IN    BIT_24;
                  MP          :     IN    BIT_24;
                  DOUT  :  OUT      BIT_24;
                        CLK   :     IN    std_logic;
                        CLR   :     IN    std_logic;
                        EN          :     IN    std_logic);

END COMPONENT;

BEGIN

   U1 : mac

      Port Map ( MC , MP , DOUT, CLK, CLR, EN) ;


      STIMULUS : process

      FILE              macTVin : TEXT IS IN  "../../stimilus/mactvIn.dat"
;
   FILE                 macTVout: TEXT IS OUT
"../../stimilus/mactvOut.dat" ;
   VARIABLE       INline  : LINE ;
      VARIABLE    OUTline : LINE ;
   VARIABLE       MC_IN   : BIT_24 ;
   VARIABLE       MP_IN   : BIT_24 ;

   BEGIN

            WHILE NOT(ENDFILE(macTVin)) LOOP

            --Get a DATA

      READLINE( macTVin , INline ) ;
      read( INline , MC_IN ) ;
      read( INline , MP_IN ) ;

      --Assign input values

      MC <= MC_IN ;
      MP <= MP_IN ;

            --Synchronize & Clear

      EN     <= '0';
            CLR <= '1';
      CLK <= '0';
            WAIT FOR HALF_PERIOD ;
      CLK <= '1';
            WAIT FOR HALF_PERIOD ;

      --Clock Generation & Enable

            EN     <= '1' ;
```

```
            CLR <= '0' ;
        WAIT FOR HALF_PERIOD ;
        CLK <= '1' ;
        WAIT FOR HALF_PERIOD ;

        --Write output values

        WRITE(OUTline , MC_IN) ;
        WRITE(OUTline , MP_IN) ;
              WRITE(OUTline , DOUT) ;
        WRITELINE( macTVout , OUTline );

              --Check for end of file

              IF ENDFILE(macTVin) THEN

             --Synchronize

                    CLK <= '0';

              WAIT FOR HALF_PERIOD ;
           CLK <= '1';
                    EN     <= '0';
           WAIT FOR HALF_PERIOD ;

           WRITE(OUTline , MC_IN) ;
           WRITE(OUTline , MP_IN) ;
                    WRITE(OUTline , DOUT) ;
           WRITELINE( macTVout , OUTline );

                    WAIT ;

                    END IF;

        END LOOP;

     END PROCESS ;

END behav_mac_TB;


CONFIGURATION testbench1 OF mac_TB IS

  FOR behav_mac_TB
  END FOR ;

END testbench1 ;
```

```
--Universite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Laboratoire de micro-electronique
--Hiver 97
--Kamal SAKKAY
--Programme VHDL du multiplieur accumulateur utilise dans PE_S_K_S_L
--------------------------------------------------------------------------
--------------------------------------------------------------------------
------------------------ LIBRARIES USED IS -------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.SKSL_TYPES.ALL;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
---------------------- Mac Entity Description ----------------------
ENTITY mac IS

   PORT      (    MC          :     IN    BIT_24;
                  MP          :     IN    BIT_24;
                  DOUT        :     OUT   BIT_24;
                  CLK         :     IN    std_logic;
                  CLR         :     IN    std_logic;
                  EN          :     IN    std_logic);

END mac;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
-------------------- Mac Architecture Description --------------------

ARCHITECTURE behav_mac OF mac IS

BEGIN

      PROCESS

            VARIABLE    MC:   BIT_24;
            VARIABLE    MP:   BIT_24;
            VARIABLE    ACC:  BIT_48;

      BEGIN


            WAIT UNTIL (CLK='1') and (CLK'EVENT);

                  IF (EN='1') THEN


                  ACC:= SIGNED(MP) * SIGNED(MC) + SIGNED(ACC);

                  ELSIF (CLR='1') THEN


                        ACC:=I;
```

```
                END IF;

            DOUT  <=    ACC(47 DOWNTO 24);

      END PROCESS;

END behav_mac;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
--Univesite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Labratoire de Micro-Electronique et de Micro-Systemes
--Hiver 97
--Kamal SAKKAY
--Porgramme VHDL du demultiplexeur 1a 3 utilise dans SYS_KAL_SPL
------------------------------------------------------------
------------------------------------------------------------
--------------------------LIBRARY---------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
------------------------------------------------------------
------------------ Demultiplexer Entity --------------------
ENTITY demulxe13 IS

      PORT        (     DIN         :  IN       BIT_24;
                        DOUT0       :  OUT      BIT_24;
                        DOUT1       :  OUT      BIT_24;
                        DOUT2       :  OUT      BIT_24;
                        Sel_dm      :  IN       BIT_2);

END demulxe13;
------------------------------------------------------------
------------------------------------------------------------
--------------- Demultiplexer Architecture -----------------

ARCHITECTURE behav_demulxe OF demulxe13 IS

BEGIN

      demultiplexeur_Process  : PROCESS(sel_dm, DIN)

      BEGIN

            CASE Sel_dm IS

            WHEN  "00"  =>

               DOUT0    <=    DIN;

            WHEN  "01"  =>

               DOUT1    <=    DIN;

            WHEN  "10"  =>

               DOUT2    <=    DIN;

                        WHEN OTHERS =>

                              DOUT0       <=          (OTHERS=>'Z');
                              DOUT1       <=          (OTHERS=>'Z');
```

```
                                DOUT2       <=          (OTHERS=>'Z');

                    END CASE;

      END PROCESS demultiplexeur_process;

END behav_demulxel3;
-------------------------------------------------------------
-------------------------------------------------------------
```

```
--Univesite du Quebec a Trois-Rivieres
--Ecole d'Ingenierie
--Labratoire de Micro-Electronique et de Micro-Systemes
--Hiver 97
--Kamal SAKKAY
--Porgramme VHDL du demultiplexeur 1 a 2 utilise dans SYS_KAL_SPL
------------------------------------------------------------
------------------------------------------------------------
--------------------------LIBRARY---------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.PE_TYPES_Euro.ALL;
------------------------------------------------------------
------------------ Demultiplexer Entity --------------------
ENTITY demux12 IS

      PORT        (     DIN         :  IN       BIT_24;
                        DOUT0       :  OUT      BIT_24;
                        DOUT1       :  OUT      BIT_24;
                        Sel_dm      :  IN       std_logic);

END demux12;
------------------------------------------------------------
------------------------------------------------------------
--------------- Demultiplexer Architecture -----------------

ARCHITECTURE behav_demux12 OF demux12 IS

BEGIN

      demultiplexeur_Process  : PROCESS(sel_dm, DIN)

      BEGIN

            CASE Sel_dm IS

            WHEN  '0'   =>

               DOUT0    <=    DIN;

            WHEN  '1'   =>

               DOUT1    <=    DIN;


                        WHEN OTHERS =>

                              DOUT0       <=          (OTHERS=>'Z');
                              DOUT1       <=          (OTHERS=>'Z');


                  END CASE;
```

```
      END PROCESS demultiplexeur_process;

END behav_demux12;
-------------------------------------------------------------
-------------------------------------------------------------
```