UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES
COMME EXIGENCE PARTIELLE DE LA
MAÎTRISE EN ÉLECTRONIQUE INDUSTRIELLE

ÉTUDE DE L'IMPLANTATION EN TECHNOLOGIE INTÉGRÉE À TRÈS GRANDE ÉCHELLE (VLSI) D'UN ALGORITHME DE RECONSTITUTION DE SIGNAUX BASÉ SUR LE FILTRAGE DE KALMAN

par : Philippe A. PANGO

Juillet 1995

Université du Québec à Trois-Rivières

Service de la bibliothèque

Avertissement

L'auteur de ce mémoire ou de cette thèse a autorisé l'Université du Québec à Trois-Rivières à diffuser, à des fins non lucratives, une copie de son mémoire ou de sa thèse.

Cette diffusion n'entraîne pas une renonciation de la part de l'auteur à ses droits de propriété intellectuelle, incluant le droit d'auteur, sur ce mémoire ou cette thèse. Notamment, la reproduction ou la publication de la totalité ou d'une partie importante de ce mémoire ou de cette thèse requiert son autorisation.

RÉSUMÉ

Dans ce mémoire, nous proposons une étude de la mise en oeuvre de l'algorithme du filtre de Kalman en technologie VLSI. Nous avons abordé les aspects suivants:

- relativement à l'algorithme, nous avons introduit une équation supplémentaire pour permettre une utilisation maximale de la largeur des mots;

- relativement à la notation employée, nous avons utilisé une notation en complément à deux avec facteur d'alignement égal à 1, pour éviter les débordements et permettre une meilleure récupération du résultat en sortie des multiplieurs.

- enfin, relativement à l'architecture et à sa mise en oeuvre, nous avons

* une étude architecturale incluant une étude d'une réalisation antérieure effectuée dans le cadre du projet, dans le but d'optimiser la structure et de réduire les chemins critiques;

* des simulations des éléments de l'architecture et des simulations des erreurs de reconstitution dues à la notation employée, en tenant compte des effets de troncature dus à la résolution.

* l'incorporation d'une unité opérative dédiée à paralléliser l'ensemble de l'algorithme.

Notre but étant d'étudier tous les aspects liés à la mise en oeuvre du filtre de Kalman dans un processeur, nous avons proposé une architecture microprogrammable qui lui est dédiée et destinée à un algorithme développé pour des signaux stationnaires. Cette structure effectue parallèlement les équations les plus importantes en nombre d'opérations dans l'algorithme. Nous avons procédé à plusieurs améliorations majeures dans le but d'améliorer les deux qualités principales d'un tel processeur: la vitesse d'exécution de l'algorithme et la qualité de reconstitution.

L'architecture que nous proposons comprend des mémoires pour stocker les différents vecteurs entrant en jeu dans la reconstitution de signaux; ces mémoires sont pilotables de l'extérieur à l'initialisation, et elles sont ensuite gérées par un microséquenceur programmable interne, ce qui constitue une solution à programmation horizontale adéquate pour de telles applications.

Les mémoires alimentent une unité arithmétique et logique comportant deux décaleurs (droite et droite/gauche), un soustracteur 16 bits à anticipation de retenue et deux multiplieurs (accumulateurs et additionneurs) 16 bits pipelinés à 100 MHz. les multiplieurs sont cadencés au même rythme que les accès mémoires, ce qui correspond à un débit de 10ns; ce débit permettrait d'effectuer une reconstitution toutes les 3µs, en plus des temps de conversion analogique/numérique et numérique/analogique.

La qualité de reconstitution est nettement améliorée par la largeur des mots (16 bits). La notation adoptée (virgule fixe avec facteur d'alignement de 1) permet une utilisation optimale du nombre de bits, grâce à l'insertion d'un décaleur aidant à la remise à l'échelle des vecteurs de trop petites amplitudes. Le prétraitement effectué sur le transopérateur de l'appareil de mesure H et le gain du filtre de Kalman K permettent d'avoir une reconstitution plus exacte, en leur attribuant une amplitude maximale. D'autre part, la notation employée évite les débordements.

Le double rôle de certaines unités de l'architecture entraîne une réduction du nombre de signaux issus du microséquenceur et donc la taille de la mémoire de microprogramme qui lui est attribuée. Les compteurs des boucles par exemple, en effectuant parallèlement le rôle d'adresseur, permettent une relative économie de surface.

REMERCIEMENTS

Ce travail n'aurait jamais pu se réaliser sans le support moral, matériel et financier de certaines personnes et organismes. J'aimerais en guise de remerciements nommer:

- Madame Yveline Côté, coordinatrice du Programme Canadien de Bourses de la Francophonie (P.C.B.F), dont le suivi et la haute considération qu'elle a des étudiants d'origine Africaine sont à féliciter.

- mon directeur de recherche, M. Andrzej Barwicz de l'Université du Québec à Trois-Rivières (U.Q.T.R) dont la connaissance parfaite des mécanismes et des outils de recherche a été un très grand atout dans l'aboutissement de cette thèse.

- Mon co-directeur de recherche, M. Yvon Savaria de l'École Polytechnique de Montréal (É.P.M) sans qui mon intégration dans le milieu VLSI aurait été impossible.

- Monsieur Daniel Massicotte pour les nombreuses heures de discussion consacrées à ce projet et pour l'étroite collaboration que nous avons sans cesse menée.

- À ma famille qui, si loin de moi, constitue néanmoins le plus grand exemple de réussite que j'ai connu.

- À mon père et à ma mère, qui m'ont inculqué les vertus des études universitaires, de l'ambition et du courage.

- Mon épouse, Lina Ouellet, qui a toujours su apporter l'aide dont j'avais besoin au moment où il le fallait; son apport à ce projet n'est pas quantifiable.

À toutes ces personnes, je dis MERCI.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SYMBOLES

A/A	:	analogique/analogique
A/N	:	analogique/numérique
b_k	:	matrice de pondération du bruit du système
CLA	:	Carry-Look-Ahead (Additionneur à anticipation de retenue).
C_0	:	paramètre de la contrainte de positivité
Cy	:	carry (retenue)
Demux	:	démultiplexeur
dt	:	pas d'échantillonnage
ε	:	erreur de reconstitution
F_k	:	matrice d'état
gain_h	:	facteur de rééchelonnement du transopérateur
gain_k	:	facteur de rééchelonnement du gain du filtre
gain_m	:	facteur de rééchelonnement du signal de mesure
h	:	transopérateur
I	:	innovation
Id	:	Indice de dépassement
I/O	:	Input/Output (Entrées/Sorties)
K	:	gain du filtre de Kalman
Mux	:	multiplexeur
$\eta(t)$	:	bruit
N	:	nombre d'échantillons
N/A	:	numérique/analogique
N.M.M	:	Non-additive Multiply Module (module de produit 4×4 bits)
N/N	:	numérique/numérique

$P_{k/k}$	:	Matrice de covariance de bruit
Q	:	variance du bruit du système
R	:	variance du bruit de mesure
Re	:	covariance de l'innovation
SNR	:	Signal-Noise Ratio (rapport signal-bruit)
UAL	:	Unité Arithmétique et Logique
UGA	:	Unité de génération d'adresse
v_k	:	bruit de mesure
w_k	:	bruit du système
Wn	:	élément de l'arbre de Wallace ayant n bits de même poids en entrée
WT	:	Wallace tree (arbre de Wallace)
x	:	signal d'entrée
$\hat{x}$	:	signal reconstitué
$\hat{y}$	:	estimé de mesure
$\hat{\hat{y}}$	:	estimé de mesure rééchelonné
$\tilde{y}$	:	signal de mesure bruité
$Z_{i/j}$	:	vecteur d'état à l'instant j évalué à l'instant i

1. INTRODUCTION

La métrologie continue de suivre un développement assez rapide. Cet aspect de l'électronique est aujourd'hui présent dans des domaines d'activités très variés. En effet, presque tous les domaines d'activités nécessitent l'acquisition d'informations ou de paramètres de natures diverses. Cela va du simple relevé météorologique à l'évaluation des bancs de poisson en haute mer.

Les systèmes d'acquisition sont aussi variés que les types de données à recueillir. Une des principales caractéristiques de ces systèmes est l'adéquation obligatoire qui doit exister entre le type de données à traiter et le système de mesure qui en a la charge. Les notions de mesure, capteurs, systèmes de mesure seront précisées plus tard dans ce document, mais on peut déjà se permettre d'éclairer un peu plus le lecteur sur la nécessité d'englober le contexte de la mesure sous un même vocable: le système de mesure.

L'instrumentation, comme son nom l'indique, a au départ été la science de l'instrument (de mesure). Intuitivement, ce terme a semblé signifier la conception et l'utilisation d'instruments essentiellement voués à l'acquisition, d'où son orientation vers l'étude et le développement de capteurs; ceci concerne essentiellement l'époque où la physique fondamentale s'est donnée comme mission de quantifier tous les phénomènes étudiés. Cette formulation de ce sous-problème métrologique a révélé ses limites, ce qui a conduit à la mise en place de traitement post-acquisition. Les raisons qui rendent ce traitement nécessaire sont que:

- les données à mesurer sont quelquefois présentées sous des formes ne permettant pas leur connaissance exacte, du moins dans la forme qu'elles ont en sortie du ou des capteurs;

- les acquisitions multicapteurs nécessitent un traitement de synthèse;

- la numérisation des processus oblige un suivi des équations numérisées;

- les capteurs présentent souvent le mesurande sous une forme "illisible" dans sa forme originelle;

- les capteurs eux-mêmes présentent très souvent des mesurandes "inexacts", nécessitant des corrections.

La numérisation des processus a de beaucoup contribué à l'essor des systèmes de mesure ayant une forte concentration sur le traitement de données. En effet, la puissance de calcul des ordinateurs croissant, on peut dorénavant se permettre une acquisition (mesurande) de moins en moins proche de la mesure exacte ou désirée en espérant contrebalancer cela par un traitement numérique plus intense, mais facilité par la précision et surtout la vitesse accrue des calculateurs.

L'outil de mesure s'est progressivement transformé en système de mesure. En effet, l'apparition d'algorithmes de plus en plus complexes, couplés au développement de calculateurs (ordinateurs) à très grande vitesse, ont permis une numérisation progressive de la plupart des systèmes d'acquisition et ont étendu la chaîne d'acquisition de l'information à des étapes bien en amont du simple capteur. Les systèmes d'acquisition sophistiqués sont généralement constitués de capteurs et de leur conditionneur, de séquenceurs chargés de rythmer les acquisitions, de dispositifs chargés de la commande si l'on est dans un milieu aux paramètres changeant (systèmes adaptatifs) et de filtres dont les fonction peuvent être multiples:

- extraction de bruit;

- déconvolution;

- identification du processus.

La reformulation du problème de filtrage à minimum de variance peut être appliquée différemment selon les objectifs recherchés.

Les besoins en mesures de plus en plus précises et rapides de différentes grandeurs physiques, jusqu'à présent souvent non mesurables, résultent notamment du développement de l'ingénierie (la commande en temps réel, la robotique, ...), de la protection de l'environnement, de la médecine et de la pharmacologie.

Un système de mesure [1], dans ce projet de maîtrise, correspond à la structure de la figure 1:

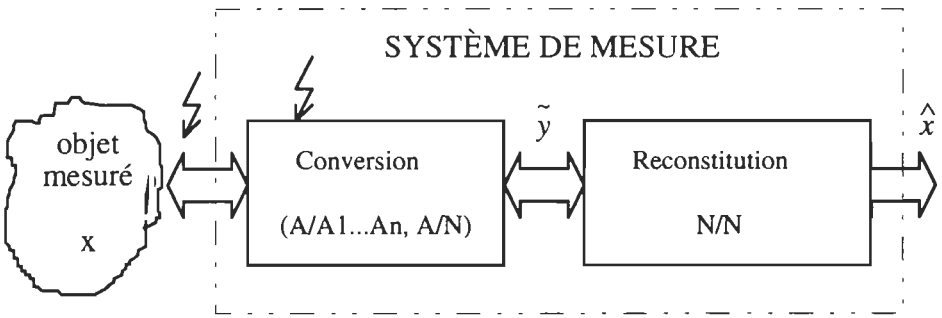


Figure 1 : Système de mesure

Sur cette figure:

- x est une caractéristique empirique d'un objet à mesurer, dont la détermination constitue l'objectif du dispositif de mesure; cette grandeur, qui peut être un vecteur ou un scalaire, est appelée mesurande;

- $\tilde{y}$ est le résultat brut de mesure, un signal électrique (résultat de conversion) qui représente le mesurande et qui est obtenu a la sortie d'un canal de mesure;

- le canal de mesure comprend des capteurs et leurs interfaces, les circuits de conditionnement et de la conversion analogique-numérique;

- $\hat{x}$ est le résultat final de mesure, un estimé du mesurande obtenu à partir du résultat de conversion, en utilisant l'information a-priori concernant les modèles mathématiques du canal de mesure, du bruit de conversion, des grandeurs d'influence, ainsi que du mesurande;

-  représente le bruit de conversion; une perturbation parasite du résultat de conversion qui représente toutes les influences aléatoires agissant sur les signaux dans la chaîne de mesure.

La transformation de l'information de mesure peut être décomposée logiquement en deux parties:

- la conversion, qui transfère l'information de mesure dans un domaine de phénomènes facilement interprétables (e.g signaux électriques, numériques);

- la reconstruction, qui transforme les résultats de la conversion (appelé aussi le résultat brut $\tilde{y}$), en résultat final de mesure $\hat{x}$.

La conversion consiste en une série de transformations du signal reçu de l'objet de mesure en un signal électrique, de préférence numérique (conversion A/A et A/N). Un résultat brut de mesure contient l'information sur la grandeur mesurée, mais il exige une reconstitution pour l'interprétation de cette information et pour sa présentation dans une forme plus utile comme un résultat final de mesure.

La reconstitution consiste en une série de transformations du résultat de conversion (résultat brut de mesure) en résultat final de mesure (conversion N/N). Dans des cas de mesure plus sophistiquées, les méthodes numériques de reconstitution sont utilisées. Par exemple, pour l'amélioration de la qualité des données spectrométriques, la correction dynamique des capteurs, l'interprétation des données séismiques et l'égalisation dans des canaux de télécommunications. L'élément clef dans la reconstitution est le modèle mathématique de la relation entre les signaux x et y. La reconstitution est en général basé sur

le modèle mathématique de la relation entre le résultat brut de mesure et le mesurande. Ce modèle est identifié et estimé pendant le processus d'étalonnage.

Les techniques de traitement numérique ont d'abord été utilisées sous forme d'algorithmes, généralement exprimés en langages évolués [2], puis dans des cartes de traitement numérique de signaux ou DSP (Digital Signal Processor), pour répondre à des exigences de vitesse; par exemple le DSP56000 [3]. Dans ce cheminement, l'étape suivante est la mise au point de circuits intégrés dédiés à des algorithmes précis. Cette étape, une fois réalisée, donne au système de mesure de nombreux avantages:

- rapidité d'exécution (temps réel);
- précision adaptée;
- taille et transportabilité (encombrement moindre).

L'intégration de systèmes de mesures [4] avance vite, grâce aux progrès récents dans les domaines connexes comme la micro-électronique. Une amélioration de la qualité des mesures peut être obtenue en utilisant des algorithmes sophistiqués de traitement des signaux (pour la reconstitution) et des capteurs micromécaniques (pour la conversion), les deux implantés en technologie VLSI.

Bien entendu, les exigences concernant la miniaturisation du système, sa fiabilité, sa consommation d'énergie et la facilité du design et/ou de la maintenance, sont les incitations les plus évidentes à l'intégration. En plus, l'intérêt envers un processeur spécialisé pour la reconstitution de mesurandes augmente si la reconstitution exige l'utilisation de modèles mathématiques nécessitant une capacité de calculs considérable. Dépendamment des exigences concernant la précision, la vitesse de traitement et la fiabilité des mesures, on peut opter pour un processeur de traitement de signaux d'application générale (microcontrôleurs,

DSP, etc.) ou bien, on peut avoir recours aux structures spécialisées, réalisées en technologie VLSI.

Globalement, l'application d'un processeur spécialisé à la place d'un DSP d'application générale est principalement justifiée par les exigences de vitesse; le processeur spécialisé, utilisant le parallélisme inhérent de l'algorithme de reconstitution, assure une vitesse beaucoup plus élevée.

Ce mémoire porte sur la mise en commun de deux champs d'études:

- le traitement des données numériques;
- la microélectronique (Very Large Scale Integration: VLSI/ITGE).

Les objectifs spécifiques de ce projet sont liés à l'application du filtrage de Kalman à la reconstitution des signaux de mesures. Ils se résument ainsi:

- étude d'une première version d'un processeur spécialisé pour un algorithme de reconstitution des signaux basé sur le filtrage de Kalman;
- proposition et conception de l'architecture d'une seconde version d'un processeur;

Cette recherche, particulièrement adaptée au traitement en temps réel, a constitué une étape d'étude nécessaire pour l'élaboration d'une structure intégrée.

Notre démarche sera la suivante. Après avoir défini la problématique dans laquelle s'inscrit ce projet, nous expliciterons la méthodologie adoptée pour résoudre les exigences formulées dans sa définition.

Ce projet a été rendu possible grâce à une collaboration entre plusieurs organismes et institutions:

- Le laboratoire de systèmes de mesure de l'Université du Québec à Trois-Rivières (UQTR) qui poursuit des recherches dans le domaine de la reconstitution des mesures échantillonnées.

- Le laboratoire VLSI de l'École Polytechnique de Montréal (ÉPM) qui appuit ces travaux par les outils dont il dispose pour la conception de circuits intégrées d'applications diverses.

- Le Programme Canadien de Bourses de la Francophonie, qui a financé mes travaux, cours et déplacements.

2. PROBLÉMATIQUE ET MÉTHODOLOGIE

2.1 LA PROBLÉMATIQUE DU TRAITEMENT DU SIGNAL

Comme nous l'avons montré en introduction, la reconstitution de signaux à mesurer constitue un problème fondamental en métrologie. Si le signal est le véhicule de l'information rendant la communication possible [5], sa connaissance exacte et quelquefois son anticipation sont nécessaires au maintien de la qualité de l'information. Par exemple, les filtres adaptatifs utilisés aujourd'hui dans les égaliseurs des lignes téléphoniques, ou dans l'atténuation des interférences électromagnétiques dans une voiture, sont également utilisés en médécine, avec d'autres paramètres, pour éliminer les interférences dans les signaux qui permettent de sonder le fonctionnement du corps humain, ou dans l'industrie, pour diminuer les fausses alarmes dans les systèmes de détection d'intrusions.

Les méthodes de reconstitution sont généralement basées sur certaines suppositions concernant le modèle mathématique de la relation entre les signaux, l'information *a priori* sur le signal à traiter, ainsi que sur le bruit qui entâche le résultat de la conversion $\tilde{y}$. Le problème de reconstitution, posé de façon abstraite, est bien conforme aux nombreuses situations pratiques dans les différents domaines se rapportant aux techniques de mesure et de commande.

L'utilisation des processeurs spécialisés en VLSI pour la reconstitution des mesurandes est bien justifié dans certains cas de mesures spectrophotométriques [30]. Dans ces cas, les mesures, c'est-à-dire le spectre d'absorption de l'échantillon étudié, est sujet à des erreurs systématiques du type instrumental. Pour prendre en considération l'effet total des erreurs de ce type, le modèle du système linéaire unidimensionnel [6] du spectre enregistré $y(\lambda)$ souvent utilisé est:

$$y = \int h(t-\lambda)x(\lambda)d(\lambda) + \eta(t) \qquad (2.1)$$

où $x(\lambda)$ est le spectre sans distorsion, lequel aurait pu être enregistré par un spectromètre avec une résolution parfaite, $g(\lambda)$ est la réponse optique incohérente et non-normalisée. Une correction numérique des données spectrophotométriques consiste en une résolution numérique de l'équation ci-haut, à partir des échantillons mesurés $\tilde{y}(\lambda)$, lesquels sont inévitablement sujets à des erreurs aléatoires, et à partir de la fonction $g(\lambda)$ identifiée *a priori*. Cette opération qui est un cas particulier de reconstitution, est un problème numériquement mal posé. La solution de ce problème passe par une régularisation de l'équation (2.1). De manière générale, la régularisation équivaut à la substitution de l'équation (2.1) par une autre dont l'analyse fournit une solution acceptable et stable (d'un point de vue algorithmique) surtout vis-à-vis des faibles variations de la grandeur mesurée.

Les signaux bruités sont communément qualifiés de processus aléatoires et les dernières quarante années ont donné une abondante théorie sur ce sujet. Le filtrage consiste à extraire d'un signal l'information dont on a vraiment besoin. Le filtrage linéaire par moindres carrés a pris sa véritable importance depuis que sa forme récursive a été introduite par R. E. Kalman [7] et Kalman-Bucy [8]en 1960. Avant d'en arriver là, il serait bien de mentionner les efforts précédents effectués par Wiener. Le résultat final de la solution de Wiener au problème du filtre optimal [9] est un filtre pondéré. En fait, ces travaux montrent comment les valeurs acquises précédemment en entrée du filtre devraient être pondérées, dans le but d'estimer au mieux la valeur présente, ce qui donne une estimation optimale. Malheureusement, la solution de Wiener ne se prête pas bien aux problèmes discrets et aux problèmes multi-entrées et multi-sorties. Ce problème est ici résolu à l'aide du filtre de Kalman. L'efficacité de cette méthode a été démontrée autant pour des signaux évoluant avec douceur que pour ceux ayant des variations brusques. L'algorithme de Kalman nécessite, dans les applications en temps réel, une capacité énorme de calcul de la part du

processeur sur lequel il est implanté. Il y a eu plusieurs propositions de mise en oeuvre de l'algorithme de Kalman. Rao [10] et D. Lawrie [11] ont proposé des architectures intéressantes, mais reflètant malheureusement la lourdeur des équations matricielles du filtrage. D'autres ont préféré une approche systolique [12] [13] [14]. Les études réalisées au laboratoire de systèmes de mesure de l'Université du Québec à Trois-Rivières ont mené à des développements selon deux axes: algorithmiques et architecturaux. Les architectures des processeurs disponibles commercialement sont conçues pour être versatiles et elles ne sont pas optimisées pour un algorithme en particulier. En effet, le calcul est réparti sur plusieurs cycles, en utilisant généralement un seul multiplieur/accumulateur (M/A), ce qui ne permet souvent pas d'atteindre les performances espérées. Dans le but d'accélérer les calculs pour une mise en oeuvre matérielle adaptée, la recherche de tout parallélisme dans l'algorithme va de pair avec le développement d'une structure qui lui serait dédiée. Une première contribution à l'intégration d'un algorithme de reconstitution basé sur le filtre de Kalman a permis d'éviter la longueur des opérations matricielles et de proposer une nouvelle génération d'architectures véritablement dédiées [27]. **Notre contribution a pris la forme d'une participation à l'élaboration d'une structure plus performante. Ce mémoire entre dans le cadre de la mise au point d'une structure intégrée dédiée à la déconvolution de signaux numériques, incluant les plus récents développements algorithmiques. Notre objectif spécifique est donc de proposer une architecture ayant de meilleures performances du point de vue de la vitesse et de la qualité de reconstitution.**

2.2 MÉTHODOLOGIE

La méthodologie que nous adopterons s'inscrit dans un processus qui a consisté en une première contribution à l'intégration d'un algorithme basé sur le filtrage de Kalman pour la reconstitution de signaux, afin d'améliorer la résolution des instruments de mesure.

En tenant compte de l'évolution préalable du projet, c'est-à-dire des versions préalables et les études algorithmiques qui en ont découlé, les éléments principaux de la méthodologie que nous adopterons seront les suivants:

- une recherche bibliographique sur les architectures VLSI pour le filtrage de Kalman;

- une étude de la première version du processeur spécialisé pour la reconstitution de signaux basé sur le filtrage de Kalman;

- une étude de l'algorithme à implanter;

- la recherche du parallélisme dans l'algorithme;

- le choix d'une seconde architecture pour l'algorithme de reconstitution;

- la conception d'une seconde version du processeur spécialisé;

- la rédaction d'un mémoire.

Après avoir explicité au chapitre 2 la problématique du traitement du signal, le chapitre 4 servira à présenter l'ensemble de l'analyse visant à la résoudre. Il sera orienté selon les points suivants:

- le choix de l'algorithme;

- une discussion sur les choix préalables à faire, en ce qui concerne la numérisation des données;

- l'étude de ce qui a déjà été réalisé dans ce domaine;

- une discussion architecturale pour dégager les traits généraux d'une architecture dédiée au filtrage de Kalman;

- la proposition d'une nouvelle architecture.

Enfin, le chapitre 6, pour sa part, résumera l'ensemble des résultats obtenus, et une conclusion sera présentée au chapitre 7.

3. ALGORITHME DE RECONSTITUTION DE MESURANDE BASÉ SUR LE FILTRE DE KALMAN

3.1 ALGORITHME DU FILTRE DE KALMAN DE BASE

3.1.1 Description du modèle

On désire souvent faire la meilleure estimation possible d'un jeu de paramètres décrivant le vecteur d'état d'un système. Pour y arriver, l'information sur l'état du système est tirée de mesures périodiques. Dans des conditions où les mesures sur le système sont fortement perturbées par la présence de bruit, l'utilisation du filtre de Kalman permet une meilleure reconstitution des signaux. L'algorithme de reconstitution est basé sur un modèle auto-régressif, sur lequel le filtrage de Kalman est appliqué.

Le modèle du système:

Ici, "système" désigne le système physique que l'on mesure. Nous considérons que le système est examiné périodiquement et que l'état à l'instant t_k est linéairement lié à l'état à l'instant t_{k+1} par les matrices d'état F et b, pondérant respectivement l'état précédent et un bruit aléatoire considéré comme gaussien de moyenne nulle (bruit du système) $w(k+1)$. Le modèle du système est donc représenté par l'équation d'état :

$$Z_{k+1} = F_k Z_k + b w_k \tag{3.1}$$

Ce modèle est suffisant pour décrire un système représenté par une équation différentielle ordinaire d'ordre n. Si le bruit du système est nul, le système évolue alors de manière déterministe.

Le modèle de mesure:

Supposons maintenant un certain nombre d'observations y_0, y_1, y_2, ..., y_n. Ces mesures sont linéairement liées au vecteur d'état décrivant le processus par la matrice de mesure h_k. Les mesures sont elles aussi perturbées par un bruit dit de mesure, v, qui sera supposé gaussien et de moyenne nulle. Ceci nous donne le modèle de mesure suivant:

$$y_k = h_k Z_k + v_k \tag{3.2}$$

La matrice de mesure est une fonction arbitraire du temps k, mais elle doit bien sûr être connue ou mesurable à chaque instant. Dans le cas d'un système de mesure, la matrice de mesure se réduit à la réponse impulsionnelle (ou transopérateur) du système. Elle a pour rôle d'extraire du vecteur d'état le paramètre recherché; dans notre cas, il s'agit de la mesure; dans d'autres cas, on aurait pu rechercher une dérivée quelconque de celle-ci: vitesse de variation, accélération ... etc ... etc

Hypothèses simplificatrices:

Nous adopterons quelques hypothèses simplificatrices. En considérant qu'à chaque instant de mesure k, le transopérateur de l'appareil de mesure est le même (on parle de processus invariants), il s'en suit que la matrice de mesure h est constante quelque soit k. De plus, observer que la dynamique du système reste la même d'une mesure à l'autre entraîne que la matrice de transition F_k est aussi constante. C'est le cas par exemple quand le système est régi par une équation différentielle à coefficients constants. Le modèle unidimensionnel finalement adopté est le suivant:

$$\begin{aligned} Z_{k+1} &= FZ_k + bw_k \\ \tilde{y}_{k+1} &= h^T Z_{k+1} + v_k \end{aligned} \tag{3.3}$$

pour k=0, 1, ..., N échantillons de mesure de y(t).

Les vecteurs w_k et v_k sont des bruits de moyenne nulle, de distribution gaussienne et de variances respectives Q(k) et R(k). La matrice de mesure et les matrices F et b qui déterminent l'état ont les formes canoniques suivantes:

$$F = \begin{vmatrix} 0 & 1 & 0 & . & . & . & 0 & 0 & 0 \\ 0 & 0 & 1 & . & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & 0 & 0 & 1 \\ 0 & 0 & 0 & . & . & . & 0 & 0 & 1 \end{vmatrix} \quad \dim(F)=M\times M \tag{3.4}$$

$$b=[0\ 0\ 0\ .\ .\ .\ 0\ \ 1]^T \quad \dim(b)= M\times 1$$

$$h^T=[h_0\ \ h_1\ \ h_2\ .\ .\ .\ \ h_{M-2}\ \ h_{M-1}] \quad \dim(h)=1\times M$$

3.1.2 Équations du filtre

À partir d'observations $y_0, y_1, y_2, ..., y_k, ..., y_n$, on désire effectuer une estimation $\hat{x}_k$ de l'entrée x_k.

Notation importante:

Définissons $\hat{Z}_{(i/j)}$ comme une reconstitution de l'état du vecteur d'état Z à l'instant i à partir des j premiers échantillons de mesure. On a trois cas:

- j>i: il s'agit d'un problème d'interpolation ou de lissage;

- j=i: il s'agit d'un problème de filtrage;

- j<i: il s'agit d'un problème de prédiction ou d'extrapolation.

Plaçons-nous dans le cas où l'on a déjà fait la reconstitution de l'échantillon k à l'aide des k premiers échantillons de mesure (soit $\hat{Z}_{(k/k)}$), et que l'on désire faire la reconstitution suivante, soit $\hat{Z}_{(k+1/k+1)}$.

Étape 1 : prédiction de Z

$\hat{Z}_{(k/k)}$ servira tout d'abord à estimer ce que sera la prochaine reconstitution; cette estimation sera donc notée $\hat{Z}_{(k+1/k)}$, car elle est faite au moment où le filtre n'a pas encore "pris connaissance" de l'échantillon k+1.

Étape 2 : prédiction de la mesure

On compose la prédiction précédemment faite avec le transopérateur pour obtenir une prédiction de la mesure k+1 attendue : $\hat{y}_{k+1}$

Étape 3 : L'innovation

C'est à cette étape du filtrage que l'échantillon $\tilde{y}_{k+1}$ à traiter entre dans les équations. La différence entre la mesure effective (qui vient d'être évaluée si le dispositif fonctionne en temps réel) et la prédiction précédemment faite donne l'innovation, c'est-à-dire, l'erreur de mesure par rapport à la valeur prédite, ou plutôt l'erreur de la prédiction $\hat{y}_{k+1}$ par rapport à la mesure; les deux interprétations sont possibles, mais dans un système fortement bruité, le premier cas est souvent le plus plausible, car la prédiction peut être considérée comme la référence par rapport à la mesure qui est plutôt à corriger.

Étape 4 : Le gain du filtre K(k)

C'est à ce niveau que le filtre prend son importance, en établissant que la reconstitution définitive de l'état suivant, $\hat{Z}_{(k+1/k+1)}$, sera égale à la prédiction précédemment faite $\hat{Z}_{(k+1/k)}$, à laquelle il faudra ajouter l'innovation I_k, pondérée par un facteur appelé gain de Kalman K_k. Le filtrage de Kalman utilise un poids K_k de sorte que l'erreur soit minimale au sens des moindres carrés, entre l'entrée et sa reconstitution soit; L'erreur d'estimation est définie comme:

$$e = \tilde{y}_k - y_k . \tag{3.5}$$

Il s'agit donc de minimiser le critère:

$$J = \text{var}(e) = \text{trace}(P_{k/k}) \tag{3.6}$$

où $P_{k/k}$ est la matrice de covariance d'erreur.

Le gain de Kalman K_k peut être calculé indépendamment de toutes mesures et avant même que celles-ci ne soient traitées par un filtre. En effet, dans des cas de systèmes invariants (transopérateur h constant), le gain peut être calculé avant même de commencer le filtrage, à condition que la covariance du processus générateur du bruit du système (w) soit constante et connue. Cette variable constitue une partie de l'information *a priori* nécessaire sur l'état du système. Le vecteur gain de Kalman est calculé par un algorithme itératif et sa valeur tend vers un vecteur constant, que nous noterons K_∞. les équations sont à la page suivante. La figure 2 montre un exemple de reconstitution. Il s'agit de relevés spectrométriques. Ces graphes ont été obtenues par simulation MATLAB. Il s'agit de signaux typiques en spectrométrie. Le signal d'entrée (qui sert de référence à la reconstitution) est reconstitué avec une erreur quadratique e=0.0139.

L'algorithme en lui-même peut atteindre des performances supérieures si l'on peut se permettre quelques modifications algorithmiques. Le coût de ces modifications sont:

- le temps de reconstitution (en ce qui a trait à la vitesse de l'algorithme);

- le coût de production (VLSI), car l'ajout d'une opération supplémentaire implique des suppléments dans les organes et le temps de calcul.

Les équations pour le calcul du gain et celles servant à la reconstitution des échantillons sont présentées à la page suivante; I est la matrice identité. La figure 2 montre un exemple de reconstitution utilisant le filtre de Kalman de base.

Prédiction de la matrice de covariance d'erreur

$$P_{(k+1/k)}=F.P_{(k/k)}.F^T+b.Q(k).b^T \qquad (3.7)$$

Calcul de la covariance de l'innovation

$$Re_{(k+1)}=R_{(k+1)}+h^T.P_{(k+1/k)}.h \qquad (3.8)$$

Calcul du gain du filtre

$$K_{(k+1)}=P_{(k+1/k)}.h.[Re_{(k+1)}]^{-1} \qquad (3.9)$$

Mise à jour de la matrice de covariance d'erreur

$$P_{(k+1/k+1)} = \left[I - K_{(k)}.h\right].P_{(k+1/k)} \qquad (3.10)$$

Prédiction de l'état suivant

$$Z_{k+1/k} = F.Z_{k/k} \qquad (3.11)$$

Estimation de la mesure

$$\hat{y}_{k+1} = h^T.Z_{k+1/k} \qquad (3.12)$$

Calcul de l'innovation

$$I_{k+1} = \tilde{y}_{k+1} - \hat{y}_{k+1} \qquad (3.13)$$

Mise à jour du vecteur d'état

$$Z_{k+1/k+1} = Z_{k+1/k} + K_{\infty}.I_{k+1} \qquad (3.14)$$

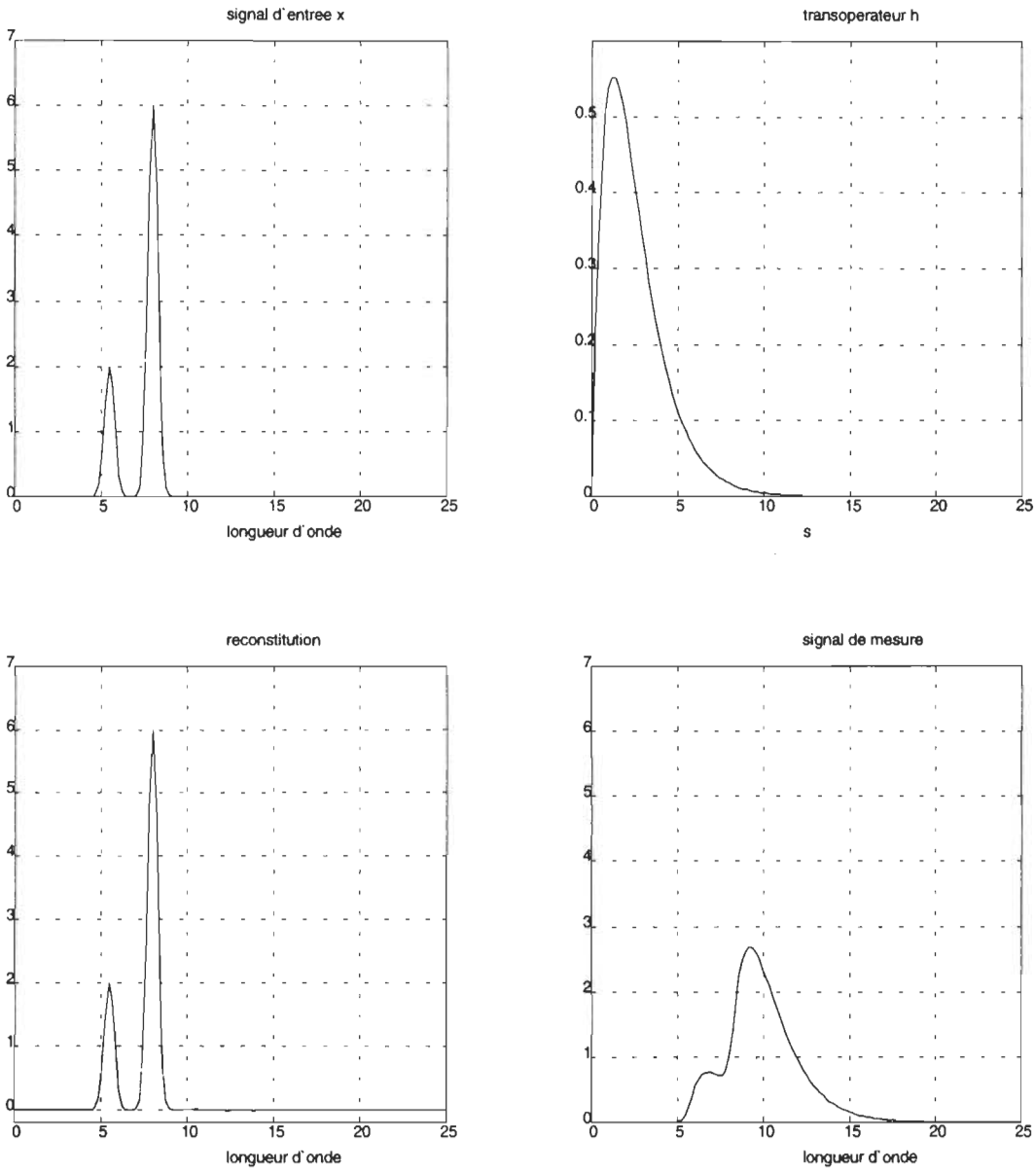


Figure 2 : Exemple de reconstitution par le filtre de Kalman

Avant d'entreprendre toute mise en oeuvre VLSI, il est donc nécessaire de faire un choix algorithmique définitif; celui-ci devrait résulter d'un compromis judicieux entre les performances désirées et les coûts que l'on peut se permettre. Dans les lignes qui suivent, nous expliciterons les différents algorithmes dérivés, pour finalement expliciter le choix de l'algorithme retenu.

3.1.3 Parallélisation des équations

Des quatre équations vectorielles du filtre de Kalman, les équations (3.12) et (3.14) sont les plus longues à exécuter, car elles nécessitent le plus d'opérations mathématiques. Celles-ci peuvent être parallélisées de sorte à augmenter la vitesse de traitement, selon le schéma à la figure 3 [13]. Les deux équations évoluent alors parallèlement sous forme de deux boucles imbriquées: boucle 1: équation (3.14), boucle 2: équation (3.12).

3.2 VARIANTES DES ALGORITHMES

3.2.1 La fenestration

Un avantage du filtrage de Kalman est la possibilité d'agir sur l'algorithme de calcul à chaque point d'échantillonnage et à l'instant d'échantillonnage, connaissant les résultats précédents, en agissant sur l'amplitude des signaux à reconstituer par exemple. La fenestration consiste à corriger la reconstitution. Cette correction se fait en imposant des valeurs maximales et minimales aux échantillons. Elle est basée sur une connaissance préalable de l'ordre de grandeur du signal à reconstituer.

3.2.2 Contrainte de positivité

Elle est elle aussi basée sur une connaissance *a priori* de l'ordre de grandeur du signal à reconstituer, notamment son signe. La contrainte de positivité appliquée à

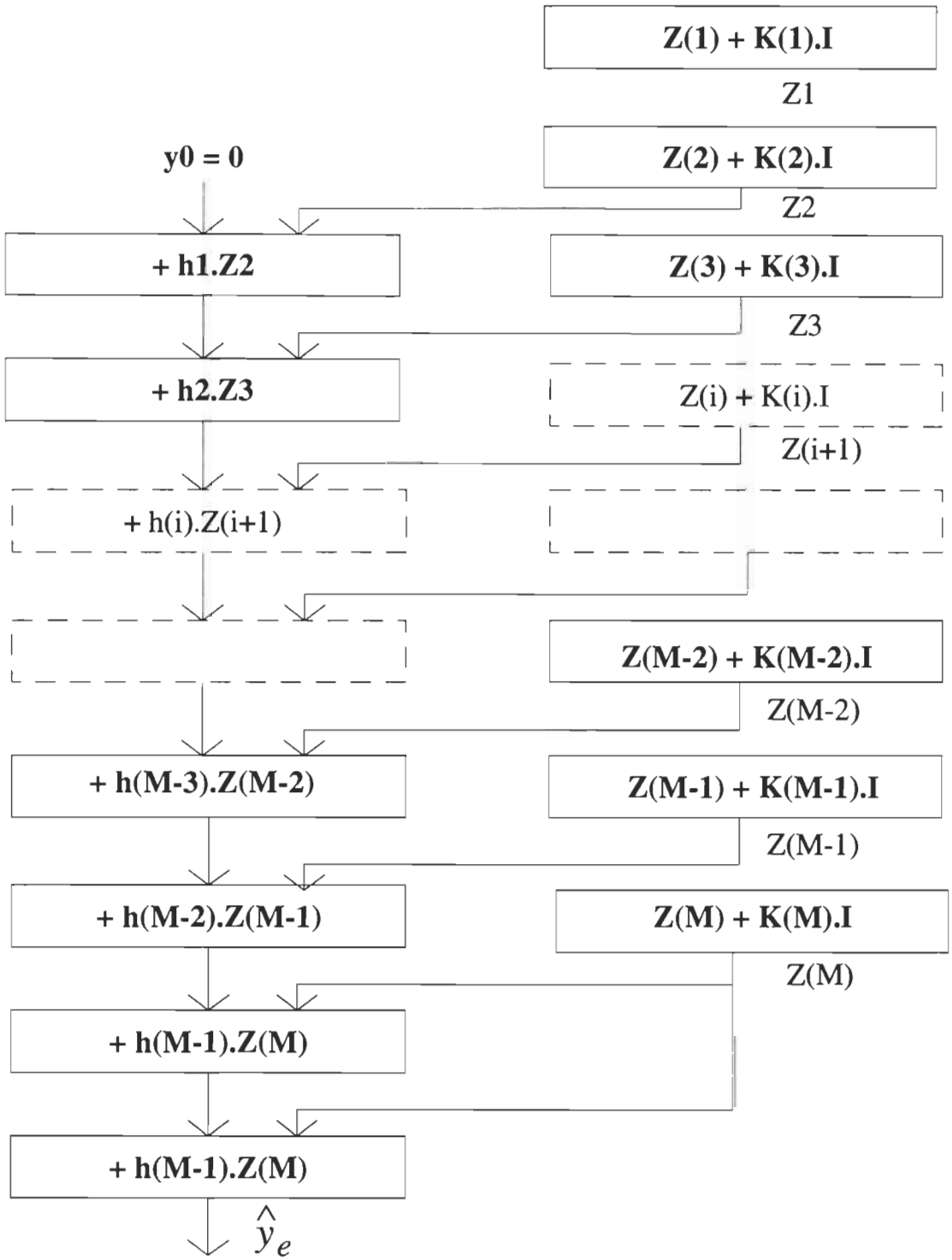


Figure 3 : Parallélisation de l'algorithme

l'algorithme de reconstitution consiste en l'ajout de l'équation supplémentaire suivante [16] [28]:

$$Z_{k+1/k+1,i} = \begin{cases} 0 \rightarrow \text{si } Z_{k+1/k+1,i} \leq 0 \\ Z_{k+1/k+1,i} \rightarrow \text{si } Z_{k+1/k+1,i} > 0 \end{cases} \tag{3.15}$$

où $Z_{k+1/k+1,i}$ est le $i^{ème}$ élément du vecteur $Z_{k+1/k+1}$. Ce cas consiste en l'application d'une contraint de positivité dite "dure". Si l'on examine l'équation (3.15), nous trouvons que la contrainte ajoutée rend cet algorithme non-linéaire. Nous devons en réalité le rendre non-linéaire pour obtenir le filtrage étendu. Cependant, cette solution consistant en une contrainte de positivité "dure", donne des résultats pouvant être améliorés par une contrainte de positivité plus "souple". On remplace alors l'équation (3.15) par:

$$Z_{k+1/k+1,i} = \begin{cases} C_0 \times Z_{k+1/k+1,i} \rightarrow \text{si } Z_{k+1/k+1,i} \leq 0 \\ Z_{k+1/k+1,i} \rightarrow \text{si } Z_{k+1/k+1,i} > 0 \end{cases} \tag{3.16}$$

où C_0 est un facteur inférieur à 1, servant à atténuer les éléments du vecteur d'état qui sont négatifs et dont la valeur d'état doit être atténuée(contrairement à la contrainte dure qui les mets systématiquement à zéro).

3.2.3 Algorithme itératif de Kalman

Cet algorithme est basé sur le modèle suivant: la fonction de transfert est considérée comme une convolution factorielle de signaux $h^k(t)$ [29]:

$$g(t) = h^1(t) * h^2(t) * \ldots * h^K(t) \tag{3.17}$$

où les $h^K(t)$ sont des fonctions connues ressemblant beaucoup à des gaussiennes. Ainsi, le problème de la résolution de l'équation (2.1) est transformé en un problème de résolution d'une série de K équations:

$$y^1(t) = y(t) \qquad (3.18)$$
$$h^k(t) * y^{k+1}(t) = y^k(t) \text{ pour } k=1,2, \ldots ,K$$
$$x^k(t) = y^{k+1}(t)$$

en se servant des échantillons bruités de y(t). L'algorithme reste pratiquement le même, à la différence que l'on répète K itérations mettant en jeu les K fonctions $h^k(t)$. À chaque échantillon à reconstituer, on effectue K fois l'équation:

$$Z^k_{n+1/n+1} = Z^k_{n+1/n} + K_\infty(y^k_{n+1} - (h^k)^T Z^k_{n+1/n}) \qquad (3.19)$$

où k=1,2,...,K.

L'algorithme de Kalman itératif obtient des résultats de reconstitution plus précis, mais il se solde par un temps de reconstitution et une exigence matérielle plus élevés que les algorithmes précédents [27] [29].

Compte tenu de la similitude entre les variantes d'algorithme proposées, nous concentrerons nos efforts sur la réalisation de l'algorithme de base décrit au chapitre 3.1.1, ce qui équivaut à une itération de l'algorithme itératif de Kalman, en y ajoutant la contrainte de positivité.

4. ÉTUDE DE L'IMPACT DE LA NUMÉRISATION DES DONNÉES

Du point de vue de l'unité arithmétique et logique, plusieurs caractéristiques sont à considérer. Les plus essentielles sont:

1. l'algorithme à implanter (équations, opérations mathématiques, contrainte de positivité, fenestration, ... etc ... etc ...)
2. la structure du calculateur (recherche architecturale)
3. notation adoptée pour représenter les données et choix du nombre de bits à allouer à chaque vecteur de reconstitution; H (transopérateur), K (gain), Z (vecteur d'état).

À cette étape, nous nous intéresserons au troisième point, à savoir, l'effet de la numérisation sur la reconstitution. Ce travail est une étape nécessaire pour évaluer partiellement les performances du processeur.

4.1. DISCUSSION SUR LA NOTATION À EMPLOYER

Plusieurs questions pertinentes apparaissent à ce stade. les équations (3.12) et (3.14) sont les principales causes d'éventuels débordements. Il apparaît en effet qu'à la fin de la boucle calculant l'estimation de la mesure $\hat{y}$ (équation (3.12)), on a:

$$\hat{y} = \sum_{i=1}^{M-1} h_i \hat{Z}_{i+1} + h_M \hat{Z}_M \tag{4.1}$$

où h est le transopérateur et $\hat{Z}$ le vecteur d'état.

Plusieurs questions pertinentes apparaissent à ce stade. Compte tenu de la nature itérative du calcul, l'opération de sommation est susceptible de conduire à des débordements. Il faut donc prévoir et adapter la largeur et la nature des chemins de données de façon à les

éviter. Le choix d'une notation doit aussi respecter le fait que les vecteurs h, K, Z et [illegible] de mesure ont des ordres de grandeur très variés. Tous les problèmes qui en découle doivent être solutionnées, étant donné leur influence sur le pré-traitement éventuel des données, ainsi que sur les modifications algorithmiques et/ou architecturales qui pourraient s'en suivre.

En ce qui a trait à la représentation des données, nous avons le choix entre trois notations:

- virgule flottante;
- signe / amplitude;
- virgule fixe avec ou sans complément à 2.

Celles-ci seront étudiées dans les pages qui suivent.

4.1.1 Notation signe / amplitude

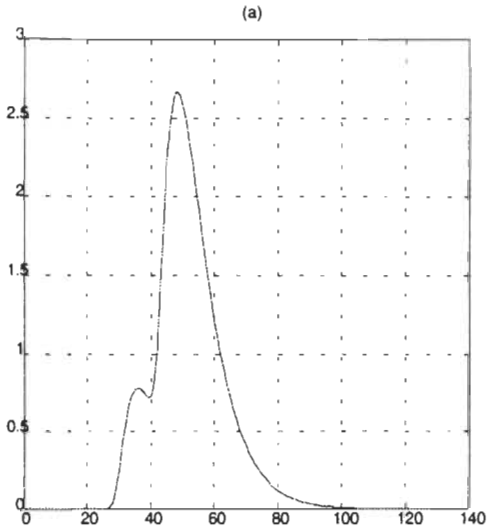
Dans un premier temps, nous nous intéresserons aux notations à virgule fixe car, s'il existe une solution à virgule fixe, il est inutile de discuter plus longtemps sur une UAL à virgule flottante, vu le coût de celle-ci.

Soit N=16, le nombre de bits disponibles. Avec la notation signe/amplitude, les données peuvent être codées de -32767 à +32767. On peut alors penser à rééchelonner le maximum des différents ordre de grandeur sur ces valeurs maximales et minimales, le plus grand des vecteurs imposant aux autres son échelle. Cette procédure fait ressortir deux problèmes:

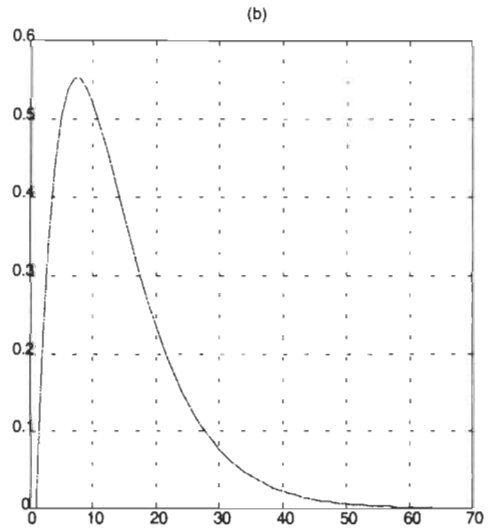
- *L'écrasement*

Dans le cas des signaux présentés à la figure 4, on aurait une échelle imposée par le gain, selon la correspondance suivante:

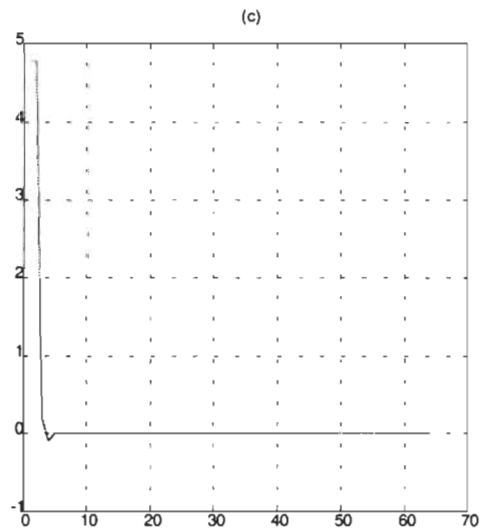
échelle réelle	échelle selon les puissances de 2	notation bina
4.774 ----------	+32767 ----------	0 111111111111111
-4.774 ----------	-32767 ----------	1 111111111111111



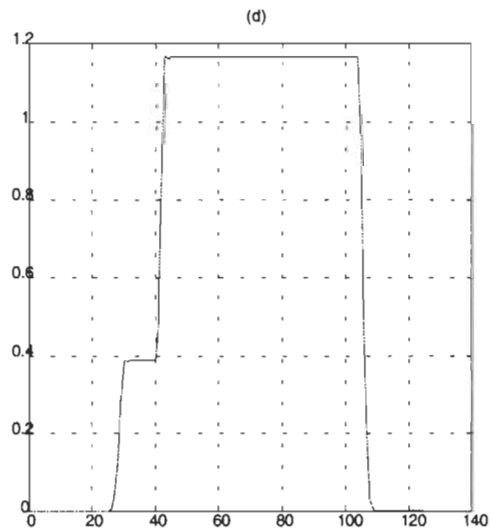
(a) Signal de mesure



(b) Transopérateur



(c) Vecteur gain



(d) Vecteur d'état

Figure 4: Ordre de grandeur des données

Le transopérateur h a pour maximum 0.552, qui correspond à 0 0001110[illegible] le vecteur transopérateur n'utiliserait donc que 12 bits et, la contribution de ses [illegible] composantes risque d'être écrasée. L'ordre de grandeur du vecteur d'état Z est fonction de celui du signal à reconstituer x et du pas d'échantillonnage dt; les simulations ont permis de constater que max(Z) = max(x)*dt, où max(x) est le maximum du signal d'entrée. Or, en général, le pas d'échantillonnage est très petit; si l'on recueille 128 échantillons pendant 2 secondes, cela équivaut à dt = 0.0156. Avec max(x)=6, on tire alors max(Z) = 0.0937, ce qui équivaut à : 0 00001010000011; le vecteur d'état n'utilise alors que 10 bits.

Avant de pousser la discussion jusqu'à déterminer lequel des vecteurs devrait bénéficier du maximum de précision, il serait intéressant de se pencher sur un autre problème, celui du débordement.

- *Le débordement (overflow)*

Pour cette étude, réduisons le nombre de bits à N=8 en reprenant l'exemple précédent. Dans l'échelle à 8 bits, on a la correspondance suivante:

échelle réelle	échelle selon les puissances de 2	notation binaire
4.774	127	0 1111111
.	.	.
2	53.2	0 0110101
.	.	.
1	26.6	0 0011010
.	.	.
0.5	13.9	0 0001101
.	.	.
0	0	0 0000000

Dans l'échelle réelle (correspondant aux valeurs exactes des données), 2*0.5 = 1. Mais sur l'échelle rééchelonnée cette opération est effectuée selon l'opération 53.2*13.9 =

739.48. C'est cette valeur qui va se manifester par un débordement de trois bits au-dela de l'amplitude des opérandes (qui est de 7 bits), tel qu'indiqué dans la multiplication numérique ci-dessous. Le dépassement provient du fait que la notation signe/amplitude représente les données selon les puissances positives de 2; Cela fait apparaître dans le résultat des puissances plus élevées, ce qui peut évidemment produire des résultats en dehors de la plage des opérandes. Pour des opérandes plus grandes, on peut prévoir que la largeur du résultat dépasserait le double du nombre de bits des opérandes, ce qui serait un problème majeur, vu la répétitivité des sommations dans l'algorithme. Remarquons que dans l'exemple ci-dessous, le calcul du signe n'est pas inclus.

```
                0 1 1 0 1 0 1
              * 0 0 0 1 1 0 1
                0 1 1 0 1 0 1
              0 0 0 0 0 0 0
            0 1 1 0 1 0 1
          0 1 1 0 1 0 1
        0 0 0 0 0 0 0
      0 0 0 0 0 0 0
    0 0 0 0 0 0 0
{signe} 0 0 0 1 0 1[0 1 1 0 0 0 1]
```

Dépassement | 7bits

4.1.2 Notation virgule fixe avec facteur d'alignement de 1

Les considérations précédentes nous amènent à chercher une notation qui garantirait que lors d'une multiplication Nbits×Nbits, le résultat puisse s'extraire du mot de 2×Nbits qui en résulte. Pour cela, tout en gardant le signe, on peut rééchelonner les données de sorte à les représenter dans l'intervalle [-2;2]. Elles peuvent alors toutes s'exprimer selon des puissances de deux inférieures ou égales à zéro.

Par exemple:

(bit de signe) 2^{0} 2^{-1} 2^{-2} . . . $2^{-(N-2)}$

C'est une notation à virgule fixe (la virgule se trouvant à la droite du bit le plus fort) de facteur d'alignement égal à 1. Pour l'instant, nous conservons le bit de signe pour respecter le fait qu'il existe des données des deux signes, ce qui n'influe en rien sur l'étude de cette notation. Plus tard, dans la discussion sur la notation complément à deux, nous développerons plus sur ce qu'il advient du signe.

Avec une telle notation, on s'assure de deux choses:

- On peut aller chercher les puissances qui nous intéressent à n'importe quel endroit dans le résultat et, en négligeant les termes de la partie basse du résultat, on ne garde que les puissances présentes dans le multiplieur et le multiplicande.

- Les dépassements (bits au-dela du nombre de bits des opérandes) correspondent au résultat recherché, tandis que les bits de poids plus faible correspondent à un ordre de grandeur inférieur au seuil acceptable par la largeur des opérandes. En effet cette notation entraîne une perte de précision liée à l'abandon des bits de poids faibles. L'effet de cette perte de précision dans tous les résultats issus des multiplieurs peut avoir une incidence relative sur le résultat global de l'algorithme qui, on le sait, comporte de longues boucles susceptibles de créer une accumulation d'erreur. Cependant, cet effet est quantifiable par simulation, en observant l'incidence de troncatures sur le résultat de reconstitution. Les simulations par les programmes MATLAB (Voir annexe C) ont montré que les bits de poids faibles peuvent être négligés sans altérer significativement l'erreur de reconstitution. La figure 10 donne un calcul de l'erreur de reconstitution en fonction du nombre de bits, tenant compte de l'effet de la troncature sur la reconstitution. Il apparaît que l'erreur de troncature n'influe pas lourdement sur le résultat; cette erreur est d'ailleurs d'un ordre de grandeur inférieur à celle sur la numérisation des données.

Ainsi les valeurs maximums et minimums codables sont $+\varepsilon$ et $-\varepsilon$, ε étant de l'ordre de 2. Le problème ne consiste plus alors à éviter un dépassement au-delà de la largeur des opérandes (ce qui est d'ailleurs inévitable à moins de sous-utiliser le nombre de bits), mais plutôt à conditionner les données des vecteurs h, K, Z et du signal de mesure de sorte à

éviter un débordement au-delà de la partie haute du résultat. Dans la notation virgule fixe pour un facteur d'alignement de 1, cela équivaut à chercher une technique permettant de garder le résultat dans l'intervalle [-ε;+ε]. Nous présentons une solution qui consiste en un pré-traitement des données.

4.1.3 Notation complément à deux

La discussion précédente qui a porté sur les débordements et le rééchelonnement éventuel des vecteurs, est en fait valable quelle que soit la notation adoptée. Les résultats obtenus peuvent donc être étendus à la notation complément à deux.

Nous nous devons de choisir une notation qui, tout en permettant d'éviter au besoin les cas de débordements, permet au processeur une meilleure interconnexion avec d'autres dispositifs numériques. Dans le cas des processeurs de traitement des signaux numériques, la notation complément à deux est celle qui est la plus utilisée. Notre processeur étant destiné à "co-habiter" sur une même carte avec des dispositifs divers (mémoires, convertisseurs, logique de contrôle ...), et pour permettre un échange aisé de données entre ceux-ci, sans conversion de notation, **nous avons décidé d'adopter la notation complément à deux**.

Ce choix est aussi justifié par le fait que l'algorithme comporte plusieurs sommations avec des opérandes de signes divers; la notation complément à deux permet de faire des soustractions sans conversion de signe; ceci sera très utile lors de la conception des multiplieurs accumulateurs.

4.2 RÉÉCHELONNEMENT DU TRANSOPÉRATEUR ET DU GAIN

Pour cela, nous exploiterons certaines relations linéaires existant entre les différents paramètres entrant en jeu dans les équations du filtre. Notre objectif est d'utiliser au maximum le nombre de bits disponibles pour tous les vecteurs et cela quel que soit la

notation utilisée. Nous proposons de multiplier le gain et le transopérateur par des facteurs respectifs, les amenant dans l'intervalle [-2;+2]. Pour cela étudions l'effet de telles opérations sur la reconstitution. Le rééchelonnement du transopérateur se fait en le multipliant par un facteur que nous nommons gain_h. Le vecteur gain K, qui est une fonction directe du transopérateur se calcule alors selon les équations (3.7), (3.8), (3.9), et (3.10) modifiées:

$$P_{(k+1/k)}=F.P_{(k/k)}.F^T + b.Q(k).b^T \tag{4.2}$$

$$Re_{(k+1)\_h}=R_{(k+1)} + h^T.P_{(k+1/k)}.h\times(gain\_h)^2 \tag{4.3}$$

$$K_{(k+1)\_h}=P_{(k+1/k)}.h.gain\_h.[Re_{(k+1)\_h}]^{-1} \tag{4.4}$$

$$P_{k+1/k+1} = \left[I - K_{k+1\_h}.h.gain\_h\right]P_{k+1/k} \tag{4.5}$$

où h désigne le transopérateur avant rééchelonnement. L'équation (4.2) est identique à l'équation (3.7). Par contre, dans l'équation (4.3), le terme $h^T.P_{(k+1/k)}.h\times(gain\_h)^2$ étant très grand par rapport à la covariance du bruit de mesure R(k+1), on peut dire que la nouvelle covariance de l'innovation $Re_{(k+1)\_h}$ ("_h" signifie que cette valeur est calculée à partir du transopérateur rééchelonné) est multipliée par le carré de gain_h.

$$Re_{(k+1)\_h} = h^T.P_{(k+1/k)}.h \times (gain\_h)^2 \tag{4.6}$$

$$Re_{(k+1)\_h} \approx Re_{(k+1)}.(gain\_h)^2 \tag{4.7}$$

L'équation de calcul du gain donne alors:

$$\begin{aligned} K_{(k+1)\_h} &= P_{(k+1/k)}.h.gain\_h.[Re_{(k+1)\_h}]^{-1} \\ &= P_{(k+1/k)}.h.gain\_h.[Re_{(k+1).}(gain\_h)^2]^{-1} \\ &= P_{(k+1/k)}.h.[Re_{(k+1).}(gain\_h)]^{-1} \\ &= P_{(k+1/k)}.h.[Re_{(k+1).}]^{-1} / gain\_h \\ &= K_{(k+1)} / gain\_h \\ &= K_{(k+1)\_h} \end{aligned}$$

Ceci nous permet d'énoncer la relation suivante:

Quand le transopérateur est multiplié par un facteur gain_h, le gain résultant est quant à lui divisé par ce même facteur.

Cette assertion exprime la totale linéarité existant entre le transopérateur et le gain d'une part, et, comme nous le démontrerons plus tard, entre ces deux grandeurs et la reconstitution. Le gain réel se reconstruit donc facilement à partir du gain qu'on obtiendrait avec un transopérateur rééchelonné.

Le rééchelonnement de h dans l'intervalle [-2;+2] ne garantit en aucun cas celui de K. Même si l'on sait comment reconstituer le gain réel, celui obtenu après rééchelonnement de h peut être hors de l'intervalle [-2;+2]. Le gain doit lui aussi être multiplié par un facteur arbitraire que nous nommons gain_K. On obtient un gain rééchelonné:

$$K_{(k+1)}\_h\_K = K_{(k+1)} \cdot \frac{gain\_K}{gain\_h} \tag{4.8}$$

où $K_{(k+1)}$ représente le gain réel (qu'on obtiendrait sans rééchelonnement de h). Il est important de noter que $K_{(k+1)}$_h_K correspond au rééchelonnement (selon le facteur gain_K) du gain obtenu, à partir du transopérateur rééchelonné (selon le facteur gain_h), et non du rééchelonnement direct du gain réel. Pour qu'il soit stable, l'algorithme de reconstitution doit utiliser deux vecteurs h et K compatibles, c'est-à-dire, un gain K issu des équations (3.7), (3.8), (3.9) et (3.10); dans ce cas, h*gain_h et $K_{(k+1)}$_h sontcompatibles et permettent de réaliser un système de mesure équivalent, même s'ils ne correspondent pas aux paramètres réels du système qui lui, est caractérisé par h et K. En fait, c'est la multiplication arbitraire du gain $K_{(k+1)}$_h par le facteur gain_K qui entraine une instabilité dans l'algorithme.

Rappelons que l'algorithme de reconstitution obéit aux équations (3.11), (3.12), (3.13) et (3.14) qui sont:

$$Z_{k+1/k} = F.Z_{k/k}$$
$$\hat{y}_{k+1} = h^T.Z_{k+1/k}$$
$$I_{k+1} = \tilde{y}_{k+1} - \hat{y}_{k+1}$$
$$Z_{k+1/k+1} = Z_{k+1/k} + K_\infty.I_{k+1}$$

où K_∞ représente la valeur stationnaire du gain de Kalman.

Dans la suite de cette étude, nommons α et β les facteurs gain_h et gain_K. Le diagramme suivant décrit les relations linéaires entre le vecteur d'état à l'instant k+1, $Z_{k+1/k+1}$, et celui à l'instant k, $Z_{k/k}$, en l'absence de tout rééchelonnement.

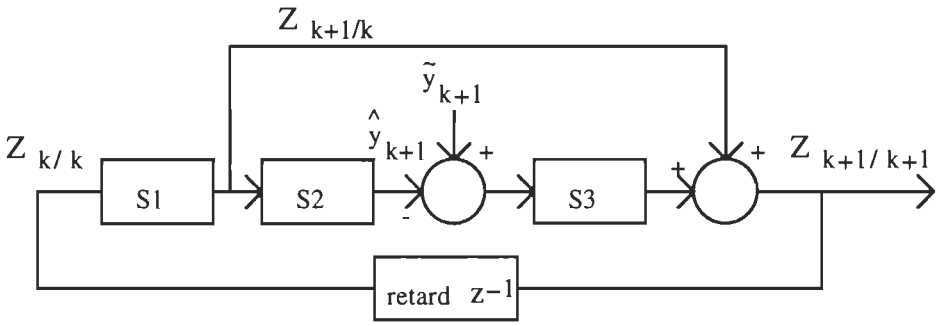


Figure 5 : Diagramme de flot de données de l'algorithme de Kalman

Les fonctions des blocs du diagramme sont:

- S1 ; qui calcule équation (3.11); c'est la multiplication du vecteur d'état $Z_{k/k}$ par la matrice F présentée à la section 4.1. C'est un produit matrice/vecteur, une opération linéaire, donnant comme résultat l'estimation du vecteur d'état, soit $Z_{k+1/k}$.

- S2 ; équation (3.12); C'est le calcul de l'estimé de la mesure $\hat{y}_{k+1}$, donné par le produit de la transposée de h par l'estimation du vecteur d'état. C'est aussi une opération linéaire et si le transopérateur est multiplié par un facteur α, le résultat l'est aussi;

- Le soustracteur: Il calcule l'innovation $I_{k+1} = \tilde{y}_{k+1} - \hat{y}_{k+1}$ (équation (3.13));

- S3: cet opérateur calcule le terme $K_\infty . I_{k+1}$ de l'équation (3.14);

- Le sommateur: Il effectue la somme des deux termes $Z_{k+1/k}$ et $K_\infty . I_{k+1}$ de l'équation (3.14).

La totale linéarité des blocs permet d'écrire:

$$Z_{k+1/k+1} = S1(Z_{k/k}) + S3[\tilde{y}_{k+1} - S2(S1(Z_{k/k}))] \quad (4.9)$$

$$= S1(Z_{k/k}) + S3(\tilde{y}_{k+1}) - S3[S2(S1(Z_{k/k}))] \quad (4.10)$$

Prenons comme hypothèse que le gain (dans le bloc S3) est calculé à partir d'un transopérateur n'ayant subi aucun rééchelonnement. L'expression (4.10) est donc celle d'un système stable. Lorsqu'on multiplie le transopérateur par un facteur α, le diagramme est alors le suivant:

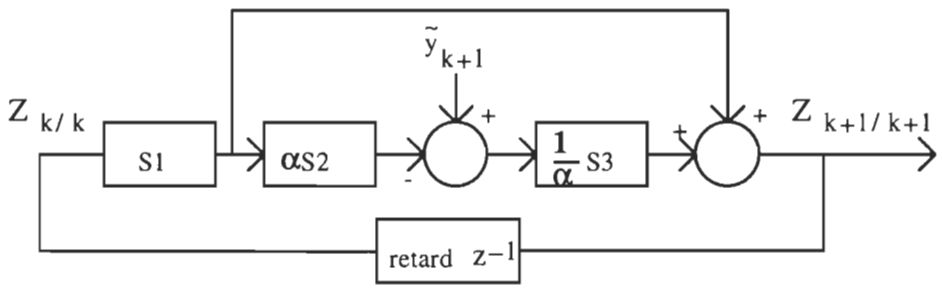


Figure 6 : Système avec transopérateur rééchelonné

ce qui donne:

$$Z_{k+1/k+1} = S1(Z_{k/k}) + \frac{1}{\alpha} S3[\tilde{y}_{k+1} - \alpha . S2(S1(Z_{k/k}))]$$

$$= S1(Z_{k/k}) + \frac{1}{\alpha} S3(\tilde{y}_{k+1}) - S3[S2(S1(Z_{k/k}))]$$

$$= S1(Z_{k/k}) + S3(\frac{1}{\alpha}\tilde{y}_{k+1}) - S3[S2(S1(Z_{k/k}))] \quad (4.11)$$

L'équation (4.11) fait apparaître que, de l'état k à l'état k+1, le fait de multiplier le transopérateur (réel) par un facteur α (ce qui divise implicitement le gain réel par le même facteur) a pour effet d'influencer seulement l'échantillon de mesure; Dans la boucle de rétroaction, α n'a pas d'influence directe sur $Z_{k/k}$, mais plutôt sur l'échantillon de mesure $\tilde{y}_{k+1}$. Le système se comporte comme si le signal d'entrée à reconstituer était préalablement divisé par α. Ceci n'entraine aucune instabilité sur l'algorithme, mais juste un résultat de reconstitution lui aussi divisé par α.

Remarque: c'est d'ailleurs ce qui arrive quand on ajuste le signal de mesure sur la plage du convertisseur A/N.

Une instabilité apparait cependant quand on rééchelonne le gain K par un facteur β; voyons alors comment se présente le diagramme bloc.

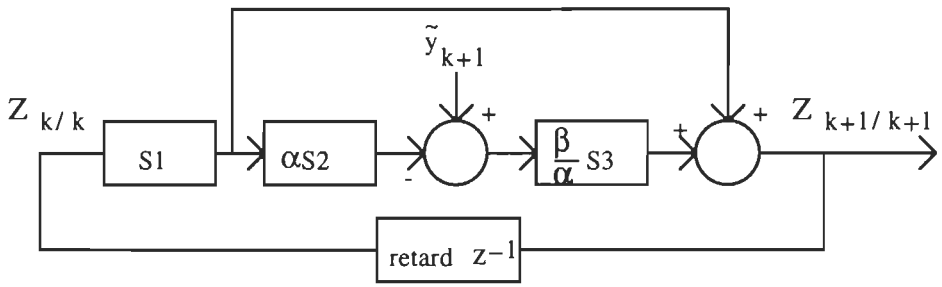


Figure 7 : Système instable

$$Z_{k+1/k+1} = S1(Z_{k/k}) + \frac{\beta}{\alpha} S3[\tilde{y}_{k+1} - \alpha.S2(S1(Z_{k/k}))]$$

$$= S1(Z_{k/k}) + \frac{\beta}{\alpha} S3(\tilde{y}_{k+1}) - \beta.S3[S2(S1(Z_{k/k}))]$$

$$= S1(Z_{k/k}) + S3(\tilde{y}_{k+1}) - S3[S2(S1(Z_{k/k}))] + (\frac{\beta}{\alpha} - 1) S3(\tilde{y}_{k+1}) + (1-\beta) S3[S2(S1(Z_{k/k}))]$$

(4.12)

Les termes en gras différencient ce système (avec rééchelonnement arbitraire du gain) du système stable énoncé en (4.10). Le terme $(\frac{\beta}{\alpha}\mathbf{-1})\mathbf{S3}(\tilde{y})$ a pour effet de multiplier la reconstitution par $(\frac{\beta}{\alpha}\mathbf{-1})$ et n'entraîne aucune instabilité. Par contre, le terme $(\mathbf{1}-\beta)\mathbf{S3}[\mathbf{S2}(\mathbf{S1}(\mathbf{Z_{k/k}}))]$ entraîne une instabilité en réintroduisant dans la rétroaction, le produit du gain (S3), du transopérateur (S2) et de l'état précédent $S1(Z_{k/k})$, quand $\beta \neq 1$; le vecteur d'état Z grandit alors sans cesse, ce qui cause l'instabilité. Pour rendre le système stable, il faut annuler l'effet du facteur $(1-\beta)$. Ceci est réalisable en introduisant une équation qui consiste à diviser l'estimé de la mesure $\hat{y}_{k+1}$ par le facteur β. L'algorithme transformé comporte donc une opération supplémentaire à insérer entre les équations (3.12) et (3.13), soit:

$$\hat{\hat{y}}_{k+1} = \hat{y}_{k+1}/\text{gain\_K} \qquad (4.31)$$

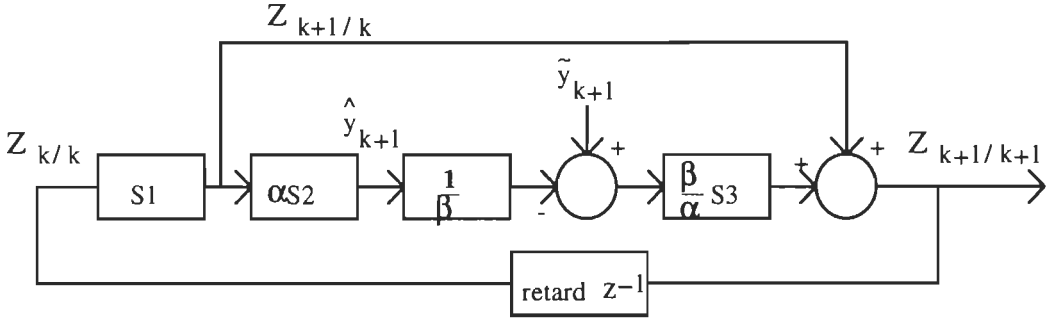


Figure 8 : Algorithme modifié stable

ce qui donne:

$$Z_{k+1/k+1} = S1(Z_{k/k}) + \frac{\beta}{\alpha}S3[\,\tilde{y}_{k+1} - \frac{\alpha}{\beta}S2(\,S1(Z_{k/k})\,)\,] \qquad (4.14)$$

$$= S1(Z_{k/k}) + \frac{\beta}{\alpha}S3(\tilde{y}_{k+1}) + S3[\,S2(\,S1(Z_{k/k})\,)\,] \qquad (4.15)$$

On reconnait l'expression du système stable, sans facteurs de rééchelonnement α et β. Le système se comporte comme si les échantillons de mesure avaient été multipliés par β/α. L'introduction de l'équation supplémentaire permet donc de rééchelonner les vecteurs h et K sans rendre l'algorithme instable. Cette technique est beaucoup plus rentable lorsque les facteurs α et β sont des puissances de 2; on peut alors effectuer l'équation supplémentaire en faisant un simple décalage.

Les vecteurs h et K pouvant alors être rééchelonnés pour utiliser le maximum de bits, le seul problème numérique qui reste à résoudre est celui du débordement. Il s'agit donc de s'assurer que l'équation de sommation (3.12) ne mènera pas à un débordement.

4.3. DÉBORDEMENT ET NOTION D'INDICE DE DÉPASSEMENT

Soit N le nombre de bits. Dans la notation retenue, il peut exister des débordements. L'estimation de la mesure peut être exprimée par la relation:

$$\hat{y} = \sum_{i=1}^{M-1} h_i \hat{Z}_i + h_M \hat{Z}_{M-1} \tag{4.16}$$

Le signal de mesure a été auparavant pré-amplifié (ou atténué) par un facteur **gain_m** qui sert à l'ajuster de façon à couvrir la plage du convertisseur A/N à l'entrée du processeur. Le vecteur d'état résultant se trouve multiplié par ce même facteur. Finalement, Z est de l'ordre de **max(x)*dt*gain_m*gain_K/gain_h** où max(x) est le maximum du signal d'entrée du système de mesure (signal à reconstituer). Pour éviter un dépassement lors de la sommation, il faut normaliser certaines données. Le pire des cas (en terme de débordement), correspond à un transopérateur constant en tout point et égal à la valeur maximale, soit 2. Le dépassement dans la sommation en (4.16) est alors atteint le plus rapidement. Cette considération nous permet de tirer une condition de non-dépassement en

évitant ce cas extrême, c'est-à-dire, le cas où l'on utiliserait un transopérateur constant en tous ses échantillons. Cela atténuerait la contrainte sur la limite de dépassement, même quand le vecteur d'état Z comporte des valeurs maximums. Une telle condition sur h influe fortement sur la sommation d'où découle l'estimation de mesure. Appelons Id, la valeur maximale de $\tilde{y}$. Appelons ce nombre " Indice de dépassement " (Id). Cet indice est lié aux ordres de grandeur des vecteurs h et K, et à celui du signal à reconstituer. La condition de non-dépassement se détermine ainsi:

$$(4.4) \quad \Rightarrow \quad \hat{y} = 2.\left(\sum_{i=1}^{M-1} \hat{Z}_i + \hat{Z}_{M-1}\right)$$

$$\Rightarrow \quad \hat{y} \leq 2.\left(\sum_{i=1}^{M-1} \hat{Z}_{max} + \hat{Z}_{max}\right)$$

$$\Rightarrow \quad \hat{y} \leq 2.M.\hat{Z}_{max}$$

$$\Rightarrow \quad \hat{y} \leq 2.M.dt.max(x).gain\_m.\frac{gain\_K}{gain\_h} \leq 2$$

d'où on tire la condition de non-dépassement:

$$I_d = M.max(x).dt.\frac{gain\_m.gain\_K}{gain\_h} < 1 \qquad (4.17)$$

Cet indice est basé sur une condition très robuste. Son utilisation suppose une information *a-priori* sur le signal à reconstituer, notamment son ordre de grandeur, ce qui constitue une exigence généralement facile à remplir. On peut alors prévoir un dépassement éventuel et ajuster les facteurs gain_h et gain_m au besoin.

4.4 CONSÉQUENCES ALGORITHMIQUES ET ARCHITECTURALES

4.4.1 Algorithme modifié

Il a été montré précédemment que, pour préserver la qualité inhérente au nombre de bits du processeur (16 bits), il est nécessaire de rééchelonner certains vecteurs (transopérateurs et gain) en les multipliant respectivement par les facteurs gain_h et gain_K. Ces opérations qui sont effectuées avant le traitement (hors processeur) peuvent entraîner une instabilité dans l'algorithme de Kalman. Finalement, l'algorithme ne retrouve sa linéarité que si l'estimé de la mesure est à chaque fois divisé par le facteur gain_K. L'algorithme de Kalman modifié, utilisant des vecteurs h et K déja prémultipliés par gain_h et gain_K pour une mise en oeuvre VLSI est spécifié dans l'encadré;

Prédiction de l'état suivant

$$Z_{k+1/k} = F.Z_{k/k}$$

Estimation de la mesure

$$\hat{y}_{k+1} = h^T.Z_{k+1/k}$$

Équation supplémentaire

$$\hat{\hat{y}}_{k+1} = \frac{\hat{y}_{k+1}}{gain\_K} \qquad (4.18)$$

Calcul de l'innovation

$$I_{k+1} = \tilde{y}_{k+1} - \hat{\hat{y}}_{k+1}$$

Mise à jour du vecteur d'état

$$Z_{k+1/k+1} = Z_{k+1/k} + K_{\infty}.I_{k+1}$$

Le graphique à Figure 9 montre 2 cas de reconstitution. Dans ces graphiques, les ordres de grandeurs des vecteurs sont les suivants:

h : [0,0;552]

K : [-0.1;4.77]

Z : [-0.002;1.17]

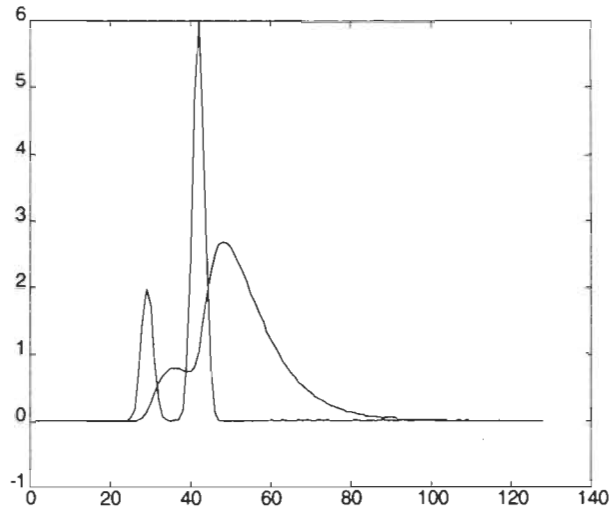
La reconstitution dans une notation à virgule flottante a produit une erreur quadratiqu e = 0.0139. Nous considérons cette performance comme l'idéal à atteindre pour ces données avec cet algorithme.

Nous avons procédé à une reconstitution avec rééchelonnement de tous les vecteurs, selon la méthode proposée plus tôt. Les vecteurs h, K et le signal de mesure sont rééchelonnés par les facteurs gain_h, gain_K et gain_m; L'introduction de l'équation (4.18) permet à l'algorithme d'être stable et d'obtenir une erreur quadratique de reconstitution: ε = 0.0273.

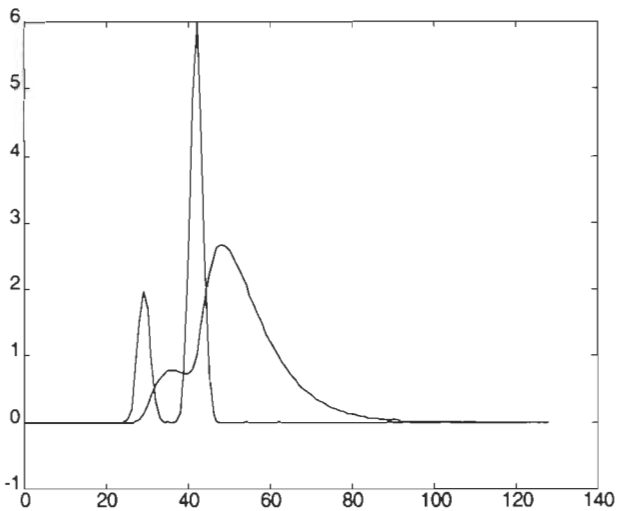
La figure (b) montre la validité de l'algorithme modifié et des opérations préalables sur les vecteurs en vue d'une mise en oeuvre intégrée. Il faut remarquer que la reconstitution (b) a été obtenue sans tenir compte de l'effet de troncature, dû à la numérisation. On peut donc conclure quant à l'effet de l'équation supplémentaire sur la stabilité de l'algorithme. En ce qui concerne l'effet de la troncature des données, nous avons simulé l'effet du nombre de bits des vecteurs sur la qualité de reconstitution. Il apparaît évidemment que la précision de reconstitution croît avec le nombre de bits. La figure 10 montre l'erreur quadratique de reconstitution en fonction du nombre de bits des vecteurs.

4.4.2 Mise en oeuvre de l'équation supplémentaire

Pour ne pas avoir à ajouter à l'UAL un circuit diviseur, on peut substituer l'opération de division (ou multiplication, selon que le gain_K soit inférieur ou supérieur à 1) par un décalage adéquat, pourvu que le facteur gain_K (ou son inverse) soit un facteur de 2. En effet, la notation complément à deux (comme tout autre d'ailleurs) est telle que les opérandes



a) Algorithme classique de Kalman



b) Algorithme de Kalman

Figure 9 : Deux cas de reconstitutions

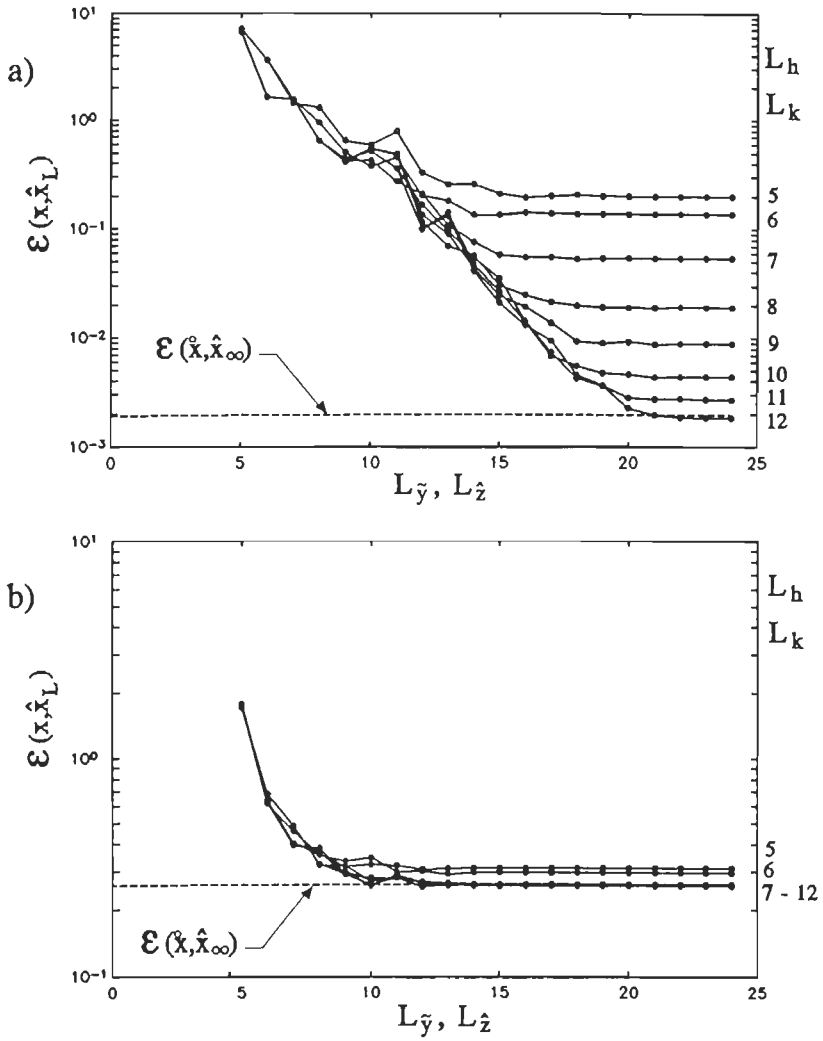


Figure 10 : **L'erreur de reconstitution en fonction de la longueur des mots** du signal de mesure $\tilde{y}$ ($L_{\tilde{y}}$), des vecteurs h et K ($L_{\hat{z}}$)contrainte de positivité C_0=0.25, dt=25/128.

a) Q/R=10^{10}, SNR=∞ dB;

b) Q/R=10^2, SNR=18.6 dB

sont exprimées en fonction de puissances de 2. Un simple décalage vers la gauche équivaut donc à une multiplication par deux, tandis qu'un décalage à droite équivaut à une division par deux. Il suffit donc de choisir le facteur gain_K tel qu'il soit la plus grande puissance de deux (ou l'inverse de celle-ci) tel que le maximum du produit K×gain_K soit inférieur ou égal à 2, maximum codable selon la notation adoptée.

Le coût de l'opération ajoutée est donc un décaleur gauche-droite de 16bits. Dans la section 5.3, nous montrerons que ce coût est d'importance moindre et que cette solution se réalise sans altérer les performances de l'UAL.

4.5 CONCLUSION

Nous avons adopté une notation complément à deux, ce qui est une notation largement utilisée dans les processeurs de signaux numériques. Cette notation, peut être utilisée avec une représentation particulière des données: en rééchelonnant les données sur l'intervalle [-2;2], on s'assure que les bits représenteront des puissances de 2 au plus égales à 1. Il s'en suit que le résultat significatif se situe toujours dans la partie haute du résultat de multiplication. D'autre part, les dépassements peuvent être évités en rééchelonnant les vecteurs h et K par des facteurs choisis de sorte à utiliser, pour chacun des vecteurs, le maximum de bits. Ce rééchelonnement entraîne des instabilités dans l'algorithme, mais celles-ci sont amorties par l'introduction d'une équation supplémentaire qui permet donc à l'algorithme de calculer les états successifs $Z_{k/k}$, avec des vecteurs h et K rééchelonnés.

5. ANALYSE ARCHITECTURALE

5.1 VERSION EXISTANTE DU PROCESSEUR SPECIALISÉ

Il existe plusieurs essais antérieurs de mise en oeuvre du filtre non-stationnaire de Kalman [10], [11], [12], [13], [14]. Ces essais présentent pour la plupart des architectures pouvant être séparées en deux catégories: les architectures systoliques et celles reflétant les équations matricielles de l'algorithme. Dans un cas comme dans l'autre, ces architectures sont limitées soit par les delais des entrées/sorties, soit par le fait que les unités opératives reflètent trop la lourdeur des équations matricielles et n'exploitent pas assez le parallélisme inhérent au calcul. Nous nous intéresserons aux réalisations se rapprochant le plus de notre projet, par le fait qu'elles sont réellement dédiées. Une première version d'un processeur spécialisé pour la reconstitution de mesurande basé sur le filtre de Kalman a été réalisé dans une étape antérieure de ce projet. Cette première version a servi de base de travail pour la mise au point de notre seconde version. L'architecture de cette première version se présentait comme suit [17]:

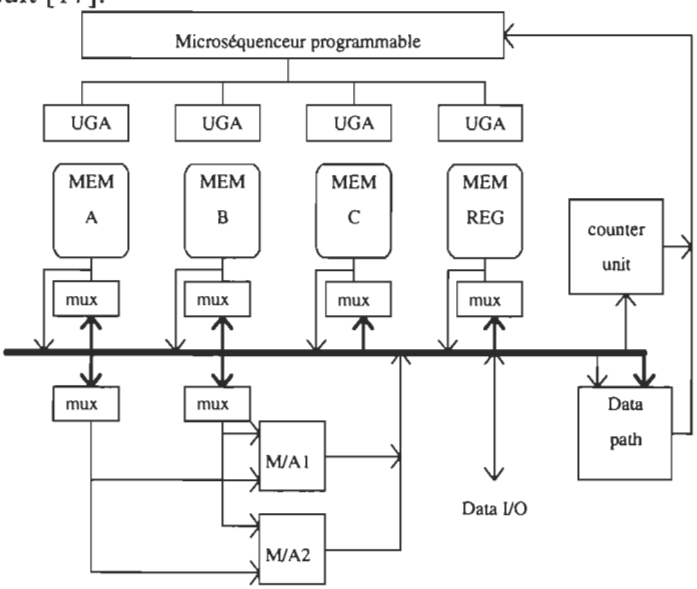


Figure 11 : Première version du processeur

Dans cette architecture, les mémoires servent au stockage des vecteurs et des paramètres de reconstitution. Elles disposent chacune de leur propre bus de données. Les multiplexeurs sont utilisés pour lier les mémoires aux multiplieurs. L'architecture comprend alors six bus de données: quatre pour les mémoires et deux pour les données de chaque multiplieur. Les multiplieurs sont conçus selon l'algorithme de Booth. Pour éviter les délais dans l'adressage, chaque mémoire a son unité de génération d'adresse utilisant deux pointeurs. L'unité opérative contient un comparateur servant pour la contrainte de positivité. Cette architecture permet un transfert parallèle des données vers les mémoires.

L'unité compteur permet de gérer les boucles. Le microséquenceur programmable est similaire au AM2910 de Advanced Micro Devices et il contient une RAM (128*64bits) pour les microcodes. Les vecteurs K et H peuvent être changés à n'importe quel moment durant le traitement grâce à un signal de contrôle, ce qui permet la mise en oeuvre d'algorithmes de reconstitution basés sur la version non-stationnaire du filtre de Kalman.

Une version 8-bits de ce processeur a été fabriquée en technologie CMOS 1.2 microns par la Société Canadienne de Microélectronique (CMC). Elle contient 24000 transistors sans les mémoires et la surface est de 5.1*9.0 mm^2.

Cette structure a été simulée à 45 MHz et elle réduit de moitié le nombre de cycles nécessaires à la reconstitution, comparativement à un DSP56001 [3].

5.2 ANALYSE DES POSSIBILITÉS ET CONTRAINTES ARCHITECTURALES

Cahier de charges:

Ce cahier de charges a été élaboré dans le cadre de ce travail. Nous n'avons utilisé pour seules références que les données disponibles sur la première version. Nos objectifs étaient:

- atteindre une vitesse de traitement égale ou supérieure à celle de la première version, cela malgré une plus grande taille des opérandes (16 bits). Pour cela, nous avons proposé des techniques de pipelinage des multiplieurs;

- améliorer la qualité de reconstitution par l'emploi d'opérandes plus longues (16 bits).

Entre autres, un paramètre à ne pas négliger était la flexibilité en terme de programmation. L'architecture devait accepter le chargement de programmes variés, tout en étant dédiée à l'algorithme de Kalman tel que décrit au chapitre 3.1. Dans ce chapitre, nous discuterons les caractéristiques essentielles de chaque partie du processeur pour en arriver à notre proposition d'architecture.

De manière générale, le processeur se subdivise en ses principales composantes qui sont (voir la Figure 12):

- le microséquenceur programmable;
- le bloc des compteurs et UGA (Unités de Génération d'Adresses);
- le bloc mémoire;
- l'unité opérative ou A.L.U (Unité Arithmétique et Logique).

Le bloc mémoire:

Le but recherché est de paralléliser au maximum les lectures des données devant se rendre vers l'unité opérative. Deux options se sont alors présentées: séparer les différentes grandeurs dans leurs mémoires respectives ou les regrouper dans une super-mémoire de très grande taille. La deuxième option peut avoir l'avantage de délivrer à chaque lecture les opérandes nécessaires à plusieurs opérations.

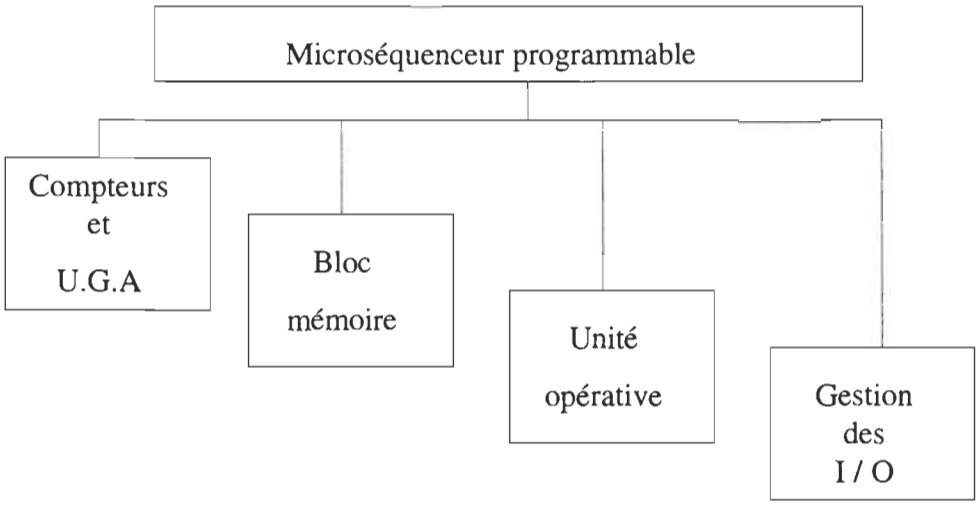


Figure 12 : Première vue d'ensemble du processeur

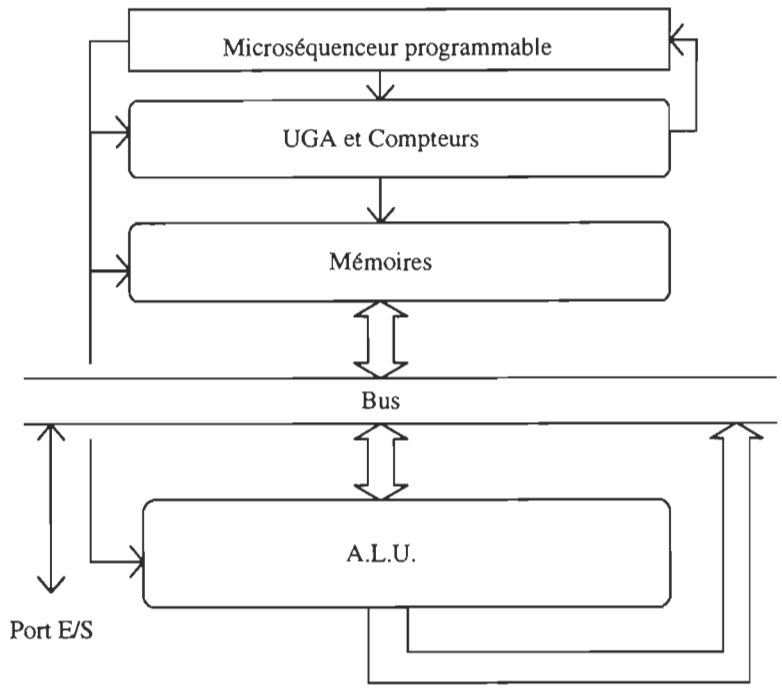


Figure 13 : Seconde vue d'ensemble du processeur

Ceci, en effet, est un concept qui pourrait servir à aligner dans un même processeur une mémoire lente et une UAL très rapide. Un tel besoin se ferait sentir dans le cas où on pousse le pipelinage d'un multiplieur à des délais équivalents à celui d'un additionneur de un bit. Par exemple, si l'on dispose d'un multiplieur/accumulateur à 500 MHz (2ns) et d'une mémoire ayant un délai de lecture double (4ns), il serait intéressant de doubler la largeur de la mémoire de sorte qu'elle fournisse deux groupes d'opérandes au lieu d'un. Ce rapport parfait augmente la vitesse de traitement. Les obstacles à un tel concept sont la non-indépendance des opérandes, c'est-à-dire, le fait que les opérandes sont très souvent en attente l'une par rapport à l'autre, le coût d'une UAL ultra-rapide et la complexité des opérations en mémoires. Pour supporter cette affirmation, rappelons les principales opérations effectuées par l'UAL dans le filtrage de Kalman. Le second multiplieur est alimenté en partie

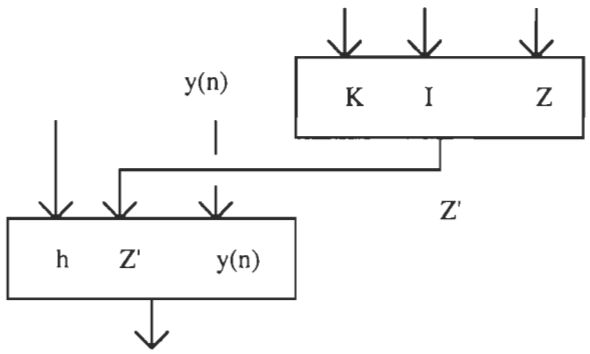


Figure 14 : Équations parallélisées

par des opérandes provenant du premier multiplieur. Il est donc en attente vis-à-vis de celui-ci. Le véritable parallélisme est obtenu quand les deux multiplieurs effectuent chacun leurs opérations sans délais d'attente l'un par rapport à l'autre. Une mémoire plus large ne peut fournir toutes les opérandes nécessaires. Cela nécessiterait entre deux lectures une écriture en mémoire, ce qui nous ramènerait presque aux performances de mémoires séparées. D'autre part, la surface d'un multiplieur/accumulateur croît avec le degré de pipelinage

(nombre d'étage), ce qui peut entraîner des coûts de production élevés. Enfin, on peut supposer une mémoire ayant l'allure de la figure 15; les cases mémoires K, h, I, et Z étant liées par des adresses identiques, la modification d'un vecteur sans toucher aux autres n'est pas chose aisée, à moins que l'on utilise des registres supplémentaires pour effectuer des décalages, ce qui entraîne d'autres coûts (commandes supplémentaires dans la microprogrammation). Néanmoins, cette solution pourrait constituer une alternative intéressante. Une étude plus poussée (qui n'a pu être réalisée) pourrait proposer une troisième version de ce processeur; celle-ci contiendrait en plus du pipelinage de l'UAL, une structure de mémoire plus adaptée.

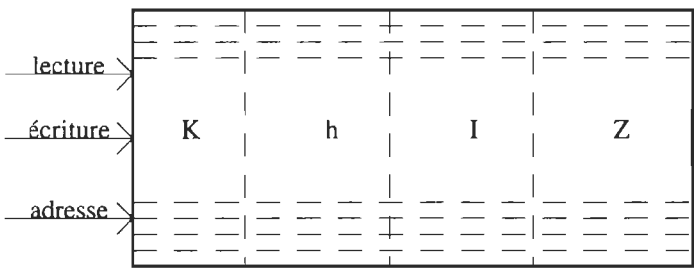


Figure 15 : Structure de mémoire large

Le processeur proposé contiendra donc des mémoires séparées pour les vecteurs K, h et Z et une pour les paramètres nécessaires à la reconstitution (dimensions des vecteurs, initialisations des compteurs, paramètres de reconstitution avec contraintes, etc ...)

La structure de bus:

Afin de réduire le coût de la mise en oeuvre, Il faut choisir la structure de bus minimum pour soutenir l'ensemble des transferts de données dans le processeur, et avec l'extérieur du processeur. Les flots de données vers le processeur concernent le chargement des mémoires et du microprogramme, l'entrée de la mesure (échantillon à reconstituer) issue

du convertisseur analogique/numérique (A/N), ainsi que la sortie de l'échantillon reconstitué. À l'intérieur du processeur, les flots de données concernent les transferts entre les mémoires et l'UAL.

Notons que:

-certains de ces transferts ne sont pas simultanés (phase de chargement et phase de traitement);

- les transferts des mémoires vers l'UAL ne nécessitent pas des bus dédiés;

- il est inutile de prévoir des opérations de transfert de mémoire à mémoire.

On peut alors réduire considérablement la structure de bus proposée dans la première version du processeur [17]. Cette structure sera explicitée à la section 5.3.2.

Les compteurs et les UGA:

Logiquement, nous avons deux boucles imbriquées et quatre mémoires, ce qui justifierait l'utilisation de deux compteurs et de quatre UGA. Cependant, en étudiant l'évolution des boucles et l'adressage requis, on constate qu'il est possible de faire l'économie de deux UGA. Aussi, le processeur étant dédié à un algorithme particulier, on peut dessiner les UGA en fonction du type d'adressage effectué dans les mémoires respectives. Ces éléments seront discutés aux sections 5.3.3 et 5.3.4.

L'unité opérative:

Sa structure nous est dictée par l'algorithme parallélisé présenté à la figure 3. Un degré de parallélisation effectif est obtenu quand les deux opérations multiplication/addition et multiplication/accumulation se font simultanément. À part ces deux opérations, il reste les opérations de division par un multiple de deux (décalage à droite) pour la contrainte de positivité, la remise à l'échelle par le facteur de gain (voir chapitre 4.2.2) qui se fait par un

décalage à gauche ou à droite selon le cas, et le calcul de l'innovation I. Les éléments de l'UAL seront alors les suivants:

- un multiplieur/additionneur : $D \Leftarrow A*B + C$
- un multiplieur/accumulateur : $(C) \Leftarrow A*B + (C)$
- un soustracteur 16 bits : $C = A - B$
- un décaleur à gauche 16 bits : $C = A / 2^n$
- un décaleur gauche/droite : $C = A$ (* ou /) 2^n

Le microséquenceur:

Le microséquencement a pour rôle d'assurer le fonctionnement synchronisé de l'ensemble selon un microprogramme préalablement chargé. Vu le caractère flexible du processeur, on ne peut que choisir une architecture microprogrammable. Cela permet l'utilisation de programmes divers. De plus, sa structure à programmation horizontale rend plus aisée le suivi des micro-opérations générées par les micro-instructions. Notons que le prix à payer pour une grande flexibilité est un coût de production relativement élevé, étant donné la présence d'une surface de mémoire importante: la mémoire de micro-programme.

L'unité de commande doit avoir les caractéristiques suivantes:

- pouvoir exécuter une boucle imbriquée;
- pouvoir effectuer des sauts et des retours de sous-programmes;
- pouvoir s'auto-adresser, c'est-à-dire, générer l'adressage de sa mémoire;
- avoir une mémoire de microprogramme permettant la reconstitution avec des vecteurs de dimension 128 ou moins;
- être "arrêtable" (hold) depuis l'extérieur du processeur.

La structure retenue pour le microséquenceur est présenté à la section 5.3.6.

5.3. L'ARCHITECTURE PROPOSÉE

5.3.1 Description du processeur

Selon l'analyse architecturale et le cahier de charges, nous proposons l'architecture présentée à la figure 16.

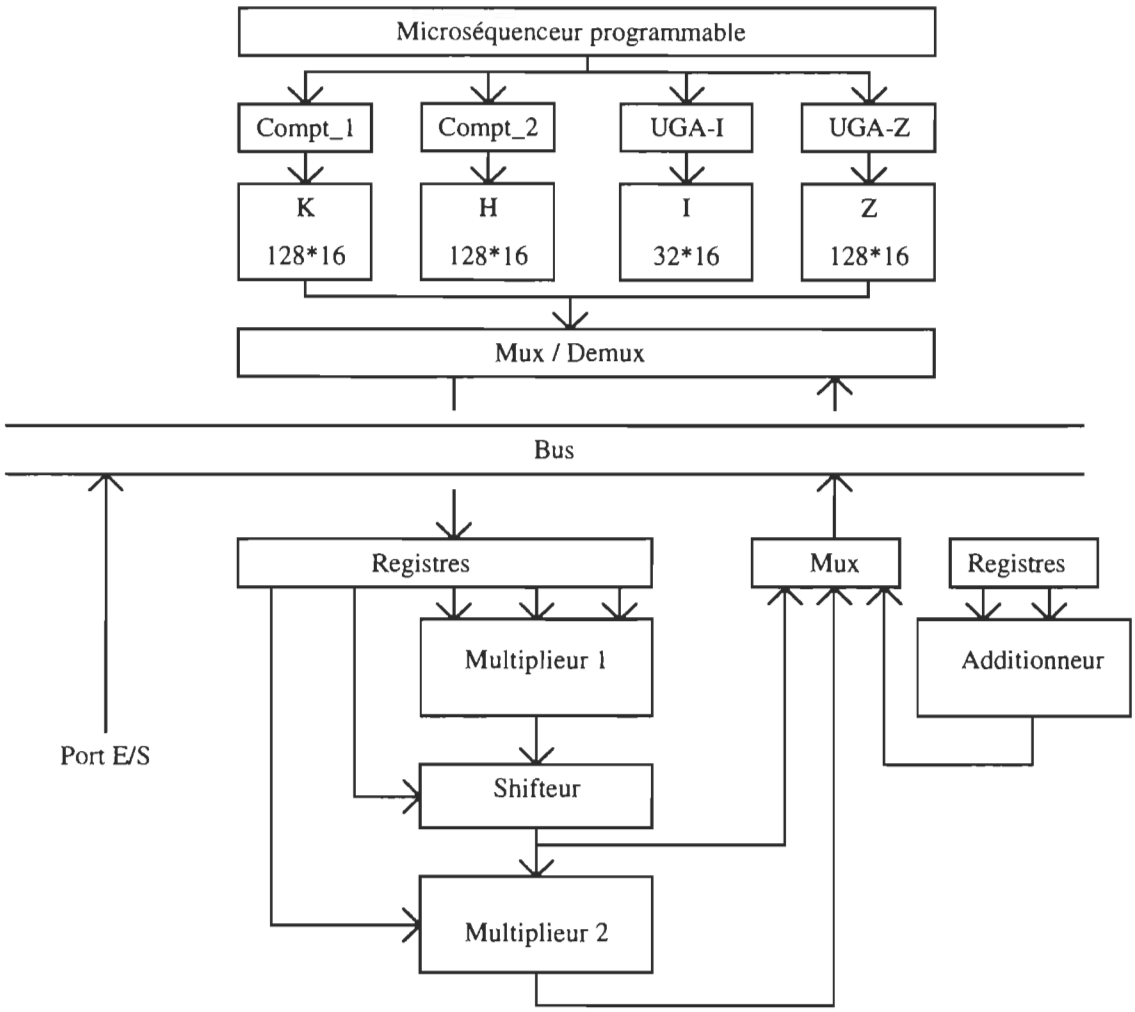


Figure 16 : Structure du processeur

Le processeur proposé comporte 44 broches; ce sont :

Load	: Indique au processeur la phase dans laquelle il se trouve
Vdd (2)	: alimentation
Gnd (2)	: ground
LoadClk	: horloge basse fréquence pour la phase d'initialisation
Clock	: horloge 100mhz pour la phase de traitement
Reset	: remise à zéro
Hold	: suspend l'exécution du microprogramme
SelMem(2:0)	: Sélection de la mémoire à charger
AdrExterne(6:0)	: Bus d'adressage
Data(15:0)	: Bus de données
WriteExt	: commande d'écriture en mémoire à partir de l'extérieur
ConvA/N (3:0)	: signaux de contrôle du convertisseur A/N
ConvN/A (3:0)	: signaux de contrôle du convertisseur N/A

5.3.2 Le bloc mémoire

Les principales données à stocker sont de deux types (hormis le microprogramme):

- les vecteurs de reconstitution (le transopérateur h, le gain K et le vecteur d'état Z);

- les paramètres de reconstitution (la constante de positivité, le facteur gain_K pour le second décaleur, la taille des vecteurs, les valeurs d'initialisation des compteurs etc ...)

Les trois vecteurs sont placés dans trois mémoires de 128 mots de 16 bits: K, H et Z. Les paramètres sont stockées dans une mémoire nommée I comportant 32 mots de 16 bits. Les buffers servent à imposer un état de haute impédance sur le bus d'entrée quand la mémoire délivre une donnée, et vice-versa, pour conserver le caractère bi-directionnel des mémoires.

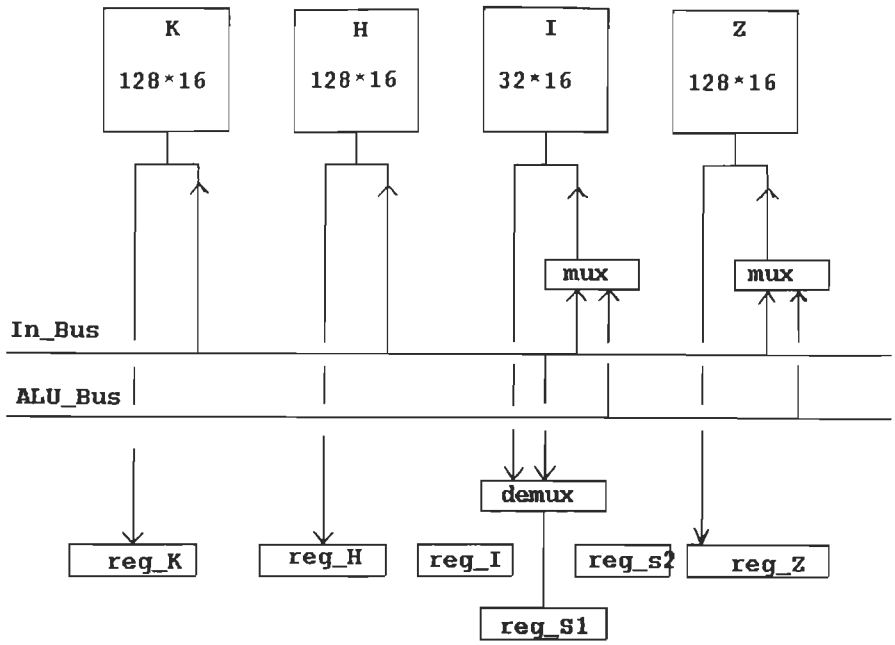


Figure 17 : Bloc mémoire

Cette séparation permet un adressage individuel; en effet, ces mémoires sont adressées par des circuits respectifs (unités de générations d'adresse) et compteurs. Elles ont aux entrées/sorties des tampons commandés par les signaux Read/Write qui permettent d'isoler deux états incompatibles: lecture et écriture.

a) Les Accès mémoires

En ce qui concerne le genre d'accès effectués vers les mémoires, on remarque que lors de l'exécution de l'algorithme, les mémoires H et K sont chargées une fois pour toutes et leurs contenus ne sont plus modifiés. Par contre, I et Z font l'objet de plusieurs écritures. L'innovation I est issue du soustracteur et les éléments du vecteur d'état Z sont issus du multiplieur 1. C'est pourquoi les deux premières mémoires H et K ne sont accessibles que par le bus IN_Bus, qui sert à l'initialisation des vecteurs et les deux autres, I et Z, sont accessibles par les deux bus IN_Bus et ALU_Bus pour les deux phases initialisation et

traitement. Cette alternance est assurée par deux multiplexeurs commandés par un signal issu de l'extérieur du processeur (Load). Ce signal indique d'ailleurs au processeur la phase dans laquelle il se trouve:

- Load = "0" : phase de prétraitement (chargement)
- Load = "1" : phase de traitement.

Cette structure a pour avantage de réduire considérablement la complexité des bus en comparaison avec la première version du processeur. Surtout, elle permet d'éviter les nombreux multiplexeurs pour les transferts de mémoires à mémoires, sans mettre en péril l'exécution de l'algorithme.

b) Les registres

Chaque mémoire a un ou plusieurs registres qui lui sont attribués. Ce sont les portes d'entrées vers l'UAL. Ces registres sont:

- Reg_K : entrée de la mémoire K;
- Reg_H : entrée de la mémoire H;
- Reg_Z : entrée de la mémoire Z;
- Reg_S1 : entrée de la commande du décaleur de contrainte de positivité (issue de la mémoire I);
- Reg_S2 : entrée de la commande du décaleur de remise à échelle (issue de la mémoire I);
- Reg_I : entrée de l'innovation I (issue de la mémoire I).

Ce sont là les six données (opérandes et commandes) dont a constamment besoin l'UAL. Trois d'entre elles sont fournies par la mémoire I. Cette structure de registres a les avantages suivants:

- on évite l'utilisation de bus respectifs dédiés à chacune des mémoires;
- le nombre de signaux de commande requis par le microséquenceur est moindre;

- cela se traduit par une réduction de surface.

Remarquons que le registre de l'innovation Reg_I est accessible directement de l'extérieur (par le bus IN_Bus), cela pour permettre le démarrage des équations. En effet, au début du traitement, le vecteur d'état étant initialisé à $Z_{0/0}$=[00 ... 0], l'estimation de mesure $\hat{y}$ est nulle. La première innovation est donc égale au premier échantillon de mesure qui, s'il est pris directement de l'extérieur par le bus IN_Bus, évite une écriture préalable des échantillons en mémoire I. Dans le chapitre 6, nous expliciterons les performances et simulations de cette structure.

5.3.3 La structure de bus

(Voir la figure 17). Nous avons jugé utile de présenter séparément la structure de bus qui sous-tend tous les transferts de données à l'intérieur et à l'extérieur du processeur. Pour cela, il faut recenser tous les transferts qui ont lieu pendant les deux phases: initialisation et traitement.

Phase d'initialisation: Cette phase est celle pendant laquelle les mémoires du processeur sont chargées (vecteurs et paramètres de reconstitution). Toutes les données sont issues de l'extérieur et y arrivent sous forme de mots de 16 bits, d'où la présence d'un port de données de 16 bits. Intérieurement, ces données sont portées par le bus **IN_Bus**. En phase de traitement, ces données sont démultiplexées vers leur destination adéquate.

Phase de traitement: Cette phase est celle pendant laquelle l'algorithme est en exécution; le bus IN_Bus sert alors à l'entrée de la mesure issue d'un convertisseur A/N (ayant en entrée la sortie du conditionneur du capteur). Le second bus, **ALU_Bus,** a pour rôle de convoyer toute donnée en provenance de l'UAL, qu'il s'agisse de la mise à jour de la mémoire Z, de l'innovation ou de la reconstitution.

À chaque mémoire, on a dédié un registre plutôt qu'un bus; le registre a pour mission de collecter la donnée au bon endroit. Cela est possible étant donné que l'échange entre mémoire est nul lors de l'exécution de l'algorithme. Des six bus de la première version du processeur, il n'en reste donc que deux; et cela constitue une des changements majeurs. Les données évoluent selon deux sens: Des mémoires (ou de l'extérieur du processeur) vers l'UAL, et de l'UAL vers les mémoires (ou vers l'extérieur du processeur). Donc, l'algorithme est donc réalisable avec une structure de bus minimale dédiée. Évidemment, la perte de flexibilité que cela engendre est contre-balancée par une réduction importante de surface.

5.3.4. Les unités de générations d'adresse

(Voir figure 18). Le rôle d'une unité de génération d'adresse (UGA) est de calculer une adresse qui sert à accéder une mémoire. Ceci a pour avantage de décharger le microséquenceur de cette tâche et donc de réduire la taille de la mémoire du microprogramme. Pour bien choisir la structure d'une UGA, il faut savoir le type d'adressage à effectuer et la fréquence de ceux-ci. Les quatre mémoires nécessiteraient quatre UGA. Dans le cas de l'adressage d'une table inscrite en mémoire (telle K et H), l'UGA se comporte comme un simple compteur s'incrémentant de 1 à chaque adressage. Or les mémoires K et H sont adressées de la case 1 à la case 128, au fur et à mesure que les boucles 1 et 2 s'exécutent (revoir l'algorithme parallélisé à la figure 3). On utilise des compteurs, 1 et 2, (dont la présence est obligatoire) pour effectuer le double rôle de compteur des boucles 1 et 2 et d'UGA pour les mémoires K et H.

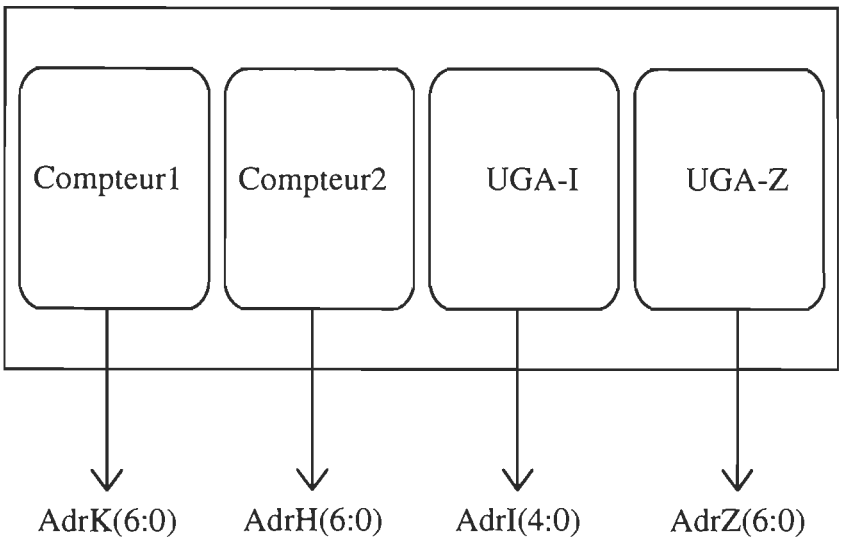


Figure 18 : Bloc UGA et Compteurs

En ce qui concerne les mémoires I et Z, nous leur avons attribué à chacune une UGA dont la structure est fonction du type d'adressage qui y est fait.

a) L'UGA à un pointeur

La mémoire de paramètre I est accédée le plus souvent pour y lire ou écrire l'innovation I. En début de traitement, elle est accédée pour y lire les paramètres de commande des décaleurs et les paramètres d'initialisation des compteurs. Vu le caractère aléatoire des données qui y sont stockées, une UGA à un pointeur (registre de base et registre de déplacement) est suffisante pour l'adressage.

La simulation de cette structure est présentée au chapitre 6.

b) L'UGA à deux pointeurs

L'algorithme parallélisé montre que la mémoire du vecteur d'état Z est celle qui requiert le plus d'accès mémoire. De plus, ces accès alternent sans cesse d'une lecture (lecture de Z_i pour alimenter le multiplieur1) à une écriture (écriture de la mise à jour $\hat{Z}_i$);

Cette alternance d'adressage est effectuée par une UGA à deux pointeurs. Sa structure comporte deux pointeurs identiques à celui de l'UGA à un pointeur.

Les registres de base et de déplacement des deux pointeurs étant préchargés, les écritures et lectures peuvent alterner sans avoir à recharger les pointeurs. Le premier pointeur peut être dédié à la lecture et le second à l'écriture. De plus, la lecture commence avec la boucle 1. Vu le pipelinage des multiplieurs (quatre étages), les écritures se font avec un délai. Les deux pointeurs vont finalement générer des adresses décalées de 4 et ils fonctionnent en parallèle, tel que suggéré par le diagramme bloc de la figure 20. Remarquons que chaque pointeur est doté de son propre additionneur, de sorte à garantir l'indépendance mutuelle de ceux-ci.

5.3.5. Les compteurs

Il y en a deux qui jouent tous les deux un double rôle dans le fonctionnement du processeur.

- compter les boucles 1 et 2;
- adresser les mémoires K et H.

La figure 3 a montré l'évolution parallèle des deux boucles effectuant les équations (3.12) et (3.14). Ces deux boucles évoluent parallèlement avec les adressages des mémoires K_i (boucle1) et h_i (boucle 2). La structure des compteurs est identique à celle d'une UGA à un pointeur, la seule différence est la présence d'un circuit logique ayant pour rôle d'indiquer au microséquenceur une valeur nulle en sortie du compteur. Tout comme les UGA, tous les registres et multiplexeurs sont pilotés par des signaux de commande issus de la mémoire du microséquenceur. En retour, le microséquenceur est informé de la fin d'une boucle par un niveau bas sur la broche Indic_0. Ce double rôle, assumé par les compteurs permet de faire l'économie de deux pointeurs. Le fonctionnement interne, qui est identique à celui explicité au chapitre des UGA sera détaillé à l'aide de simulations, plus loin dans ce mémoire .

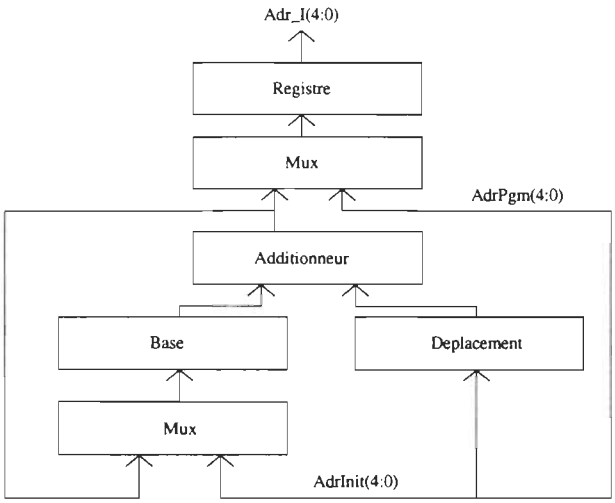


Figure 19 : UGA de la mémoire I (un pointeur)

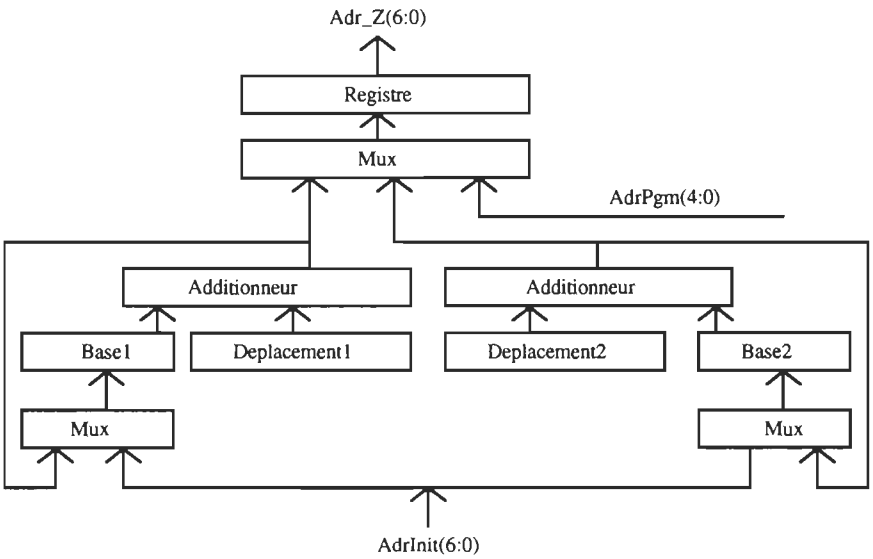


Figure 20 : UGA de la mémoire Z (deux pointeurs)

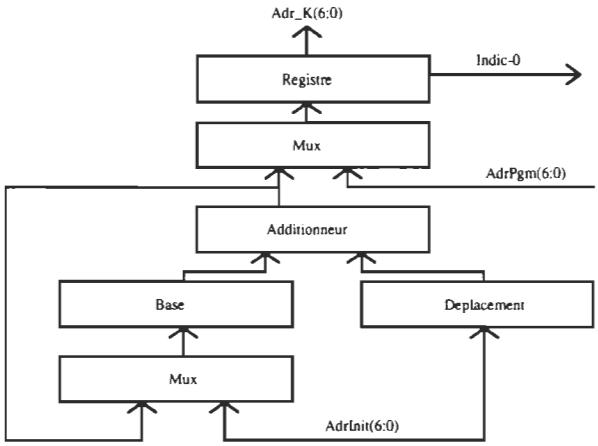
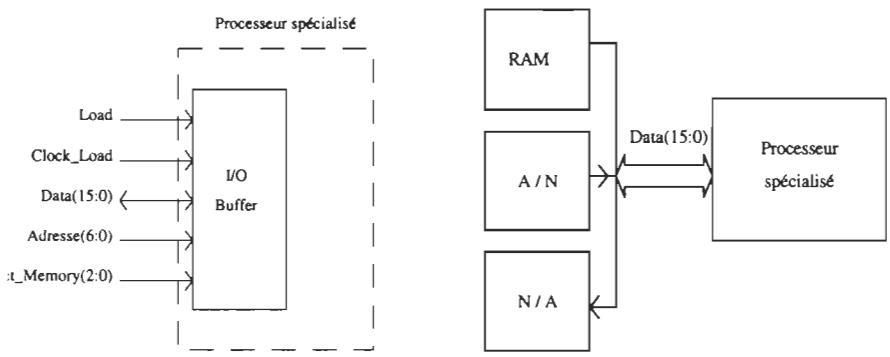


Figure 21 : Structure d'un compteur

5.3.6. Les entrées/sorties

Les entrées/sorties entre le processeur et l'extérieur sont de plusieurs sortes. **Pendant la phase d'initialisation** qui consiste à remplir les mémoires avec les données appropriées, on effectue essentiellement des transferts d'adresse et de données, d'où la nécessité d'avoir un bus de données 16 bits , Data<15:0>, et un bus d'adresse Adress<6:0>. **Pendant la phase de traitement**, il s'agit principalement d'échanges entre le processeur et les deux convertisseurs auxquels il est rattaché: A/N et N/A. Le bus de données est alors utilisé pour acheminer les échantillons de mesure et leur reconstitution.



(a) (b)

Figure 22 : Entrées/Sorties

Analysons plus en détail la phase d'initialisation. Une fois les adresses et données stables sur les broches d'entrée, un autre problème se pose; celui de l'acheminement de ces données vers la mémoire appropriée. Nous avons résolu cela par un démultiplexage piloté par une commande externe, SelMem<2:0> (Select Memory). Selon le mot de commande que l'on applique aux broches externes SelMem<2:0>, on cible une mémoire en particulier, selon la correspondance suivante:

Tableau 1 : Adressage externe des mémoires

SelMem(2:0)	Mémoire adressée
000	K
001	H
010	I
011	J
100	Microprogramme

Ainsi, on a en entrée un démultiplexeur d'adresse et un démultiplexeur de données, tels que présentés aux figures 23 et 24.

Une fois l'adresse démultiplexée, il faut la différencier d'avec l'adresse générée intérieurement par les UGA. On utilise pour cela une série de multiplexeurs commandés par le signal externe LOAD:

LOAD = "0" $\Rightarrow$ phase d'initialisation $\Rightarrow$ adressage externe

LOAD = "1" $\Rightarrow$ phase de traitement $\Rightarrow$ adressage interne

Le tampon en entrée est constitué de 4 registres à décalage de 16 bits chacun, de sorte à constituer séquantiellement le mot de 64 bits devant être chargé dans la mémoire de microprogramme.

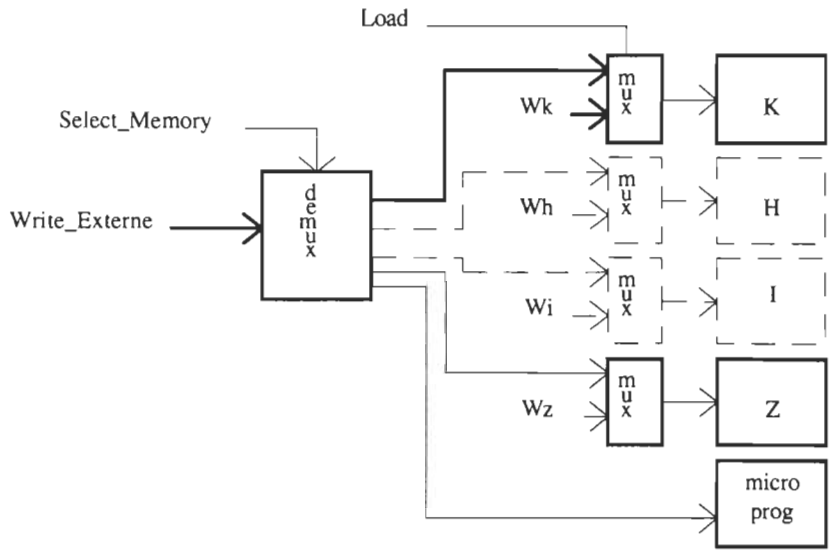


Figure 23 : Écriture interne/externe des mémoires

Cette logique reste la même pour les signaux d'écriture (write). Les ordres d'écriture interne sont multiplexés avec l'ordre (unique pour toutes les mémoires) d'écriture externe.

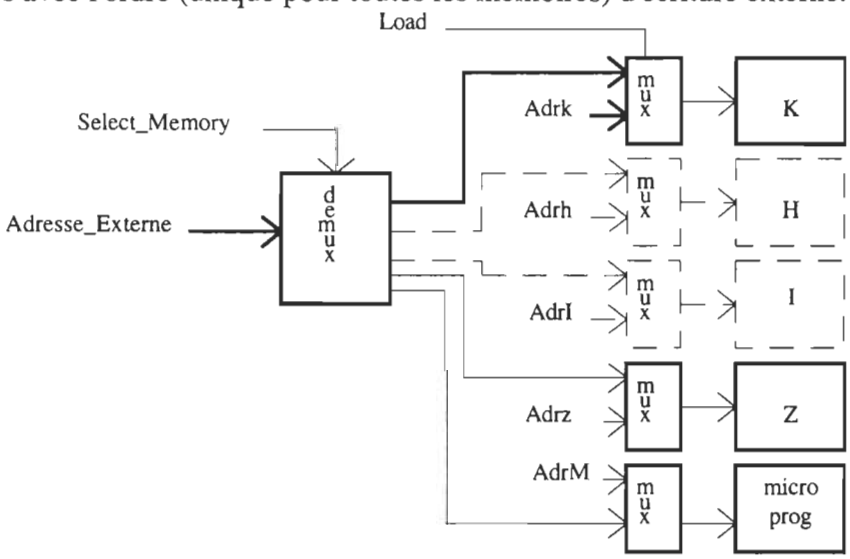


Figure 24 : Adressage interne/externe des mémoires

Cet ensemble permet de piloter aisément les mémoires à partir de l'extérieur quand il le faut. Une fois l'adresse et la donnée stabilisées, cela permet d'orienter celles-ci vers la mémoire adéquate.

5.3.7. Le microséquenceur

a) Architecture du microséquenceur.

Étant donné la variété de "microactions" devant être exécutées dans l'ensemble de l'architecture, compte tenu du fait que le processeur doit être programmable, la meilleure option en ce qui concerne le pilotage interne du processeur est une architecture à programmation horizontale, similaire à celle du AM2910 de la société Advanced Micro Devices [18]. Le rôle du microséquenceur est de piloter les éléments de l'architecture, tandis que celui du circuit d'horloge est de rythmer ceux-ci. Sa structure est à la figure 23. Pour bien comprendre le séquencement des opérations effectuées lors du traitement d'échantillons, (cet aspect sera abordé à la section suivante) il est important de bien connaître la structure du microséquenceur. Nous avons adopté une structure de commande à microséquenceur programmable pour les raisons suivantes:

- assurer une flexibilité logicielle (prévoyant éventuellement d'autres types d'opérations pouvant être effectuées par la même structure);

- répondre aux besoins de l'algorithme à implanter: celui-ci exige en effet un comportement séquentiel ordonné avec des séries de boucles et de tests;

Le microséquenceur programmable a les caractéristiques intéressantes suivantes:

- il peut exécuter deux boucles imbriquées de longueurs quelconques, (les longueurs de boucle initialisent les compteurs 1 et 2);

- il s'auto-adresse et il peut à l'issu d'un test programmé par logiciel décider de trois types d'adresse:

* Adresse du compteur ordinal (adresse précédente + 1)

* Adresse de saut (instruction JUMP)

* Adresse de retour de sous-programme (en fin de boucle).

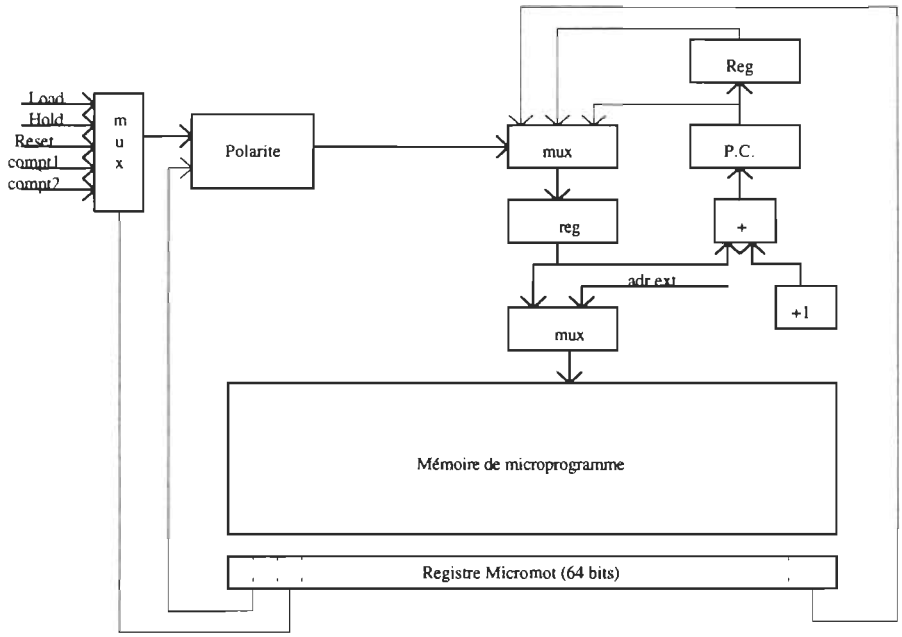


Figure 25 : Architecture du microséquenceur

L'auto-adressage est effectué par le multiplexeur d'adressage; ce multiplexeur est piloté par une logique chargée de commander la polarité du bit issu du multiplexeur de condition. Ainsi, le choix du type d'adressage est fonction de la donnée qui est scrutée en entrée du multiplexeur de condition. Un dernier multiplexeur permet éventuellement d'accepter un adressage externe, lors de la phase d'initialisation par exemple. Les données scrutées en entrée sont:

- hold: sert à suspendre le programme;
- reset: ramène le programme à l'adresse 00_h;
- load: Indique par un niveau bas que l'on est à la phase d'initialisation;
- compt1: Un niveau bas indique que le compteur 1 a fini sa boucle;
- compt2: Un niveau bas indique que le compteur 2 a fini sa boucle.

Le microséquenceur possède une mémoire de 128 mots de 64 bits chacun. Son lien avec le reste du processeur se situe au niveau du registre de sortie de sa mémoire (64 bits).

Les bits 63 à 52 servent à son fonctionnement interne (pilotage du multiplexeur de condition et contrôle de polarité) tandis que les bits 51 à 0 servent à piloter l'ensemble du processeur. La fonction de chaque bit du microséquenceur est indiquée à l'annexe B.

b) Microprogrammation

Les bits 51 à 0 sont chargés de piloter l'ensemble de l'architecture. Ils peuvent être divisés en deux catégories. La premiere catégorie regroupe les bits qui commandent des multiplexeurs, l'initialisation des UGA et compteurs et les signaux d'entrées sortie (convertisseurs); ils sont actifs dès la phase 1. La seconde concerne les bits qui commandent les registres du processeur; ils sont actifs à la phase 2. Le détail du micromot ainsi qu'une proposition de microprogramme (en langage machine avec explications) sont à l'annexe B. L'objectif de cette section est de fournir au futur utilisateur un microprogramme de sorte à pouvoir programmer le processeur. Nous avons dans un premier temps découpé l'exécution de l'algorithme en ses plus petites composantes, lesquelles ont été établies en fonction de l'architecture proposée. De manière générale, le séquencement des opérations comporte deux parties:

- l'initialisation: c'est l'étape pendant laquelle on charge les mémoires internes.
- le traitement: l'étape qui exécute l'algorithme de reconstitution.

Il est important de déterminer le nombre d'instructions minimal nécessaire pour effectuer une reconstitution, de sorte à en déduire le temps de reconstitution d'un échantillon. Pour cela, on doit se rappeler que la structure du processeur (en particulier celle des unités de génération d'adresse) ne permet pas une lecture et une écriture dans une même mémoire en un cycle machine (une microinstruction); aussi, le pipelinage du microséquenceur entraîne un délai d'une microinstruction dans toute boucle. Une fois le chargement des mémoires terminé, on passe en phase de traitement en mettant la broche

load au niveau haut "1". Une explication détaillée de la phase de traitement est fournie à l'annexe B.

c) Évaluation du temps de reconstitution

Nous devons tenir compte du fait que:

- la structure du processeur (en particulier celle des unités de génération d'adresse) ne permet pas une lecture et une écriture dans une même mémoire dans un cycle machine (une microinstruction),

- le pipelinage du microséquenceur entraîne un délai d'une microinstruction dans toute boucle. L'analyse suivante nous permets de déduire le nombre de mots (ou microinstructions) nécessaires pour effectuer la reconstitution d'un échantillon. Les étapes sont donc:

Acquisition

1 mot + temps d'acquisition

Temps de calcul du soustracteur (innovation I)

1 mot

Écriture de l'innovation I en mémoire

1 mot

remplissage du multiplieur 1

6 mots

Sortie de la reconstitution (Z(1))

2 mots + temps de sortie

Remplissage du multiplieur2

6 mots

Fonctionnement parallèle des deux multiplieurs

2 mots * 122

le multiplieur 1 se vide

2 mots * 5

le multiplieur 2 se vide

1 mot * 5

Écriture de l'estimé dans le registre de sortie de l'UAL

1 mot

réinitialisation des compteurs et UGA

5 mots

Retour

1 mot

TOTAL : 278 cycles machines
+ temps de conversion A/N
+ temps de conversion N/A

On en tire le tableau 2, qui donne le temps de reconstitution en fonction de la fréquence de l'horloge du processeur:

Tableau 2 : Temps de reconstitution vs fréquence

Fréquence interne (Mhz)	temps de reconstitution (µs)
100	3
80	3.75
60	5
40	7.5
20	15

Le chemin critique dans l'architecture se retrouve dans les multiplieurs pipelinés, soit 10 ns (cela sera explicité dans la section sur les multiplieurs). En ajoutant 20% de délai du au routage, on peut estimer que la fréquence maximale d'opération sera de l'ordre 70 à 80 MHz. De là, nous estimons que le temps de reconstitution est de 3.75 µs en plus des temps requis par les deux convertisseurs.

5.3.8 L'Unité Arithmétique et Logique (UAL)

L'unité arithmétique et logique reflète toutes les opérations arithmétiques et logiques ayant lieu lors de l'exécution de l'algorithme. La figure 3 représente la parallélisation des deux équations (3.12) et (3.14). Comme nous l'avons vu, ces équations sont exécutables par deux multiplieurs emboîtés (additionneur et accumulateur). Pour que l'UAL soit complète, elle doit contenir les éléments nécessaires à l'exécution de toutes les opérations qui ont lieu lors du déroulement de l'algorithme. ce sont:

- le calcul de l'innovation: $I_{k+1} = \tilde{y}_{k+1} - \hat{y}_{k+1}$;
- la contrainte de positivité; elle est effectué par un décalage vers la droite;
- la remise à l'échelle de l'estimé $\hat{y}_{k+1}$: décaleur gauche/droite.

Ainsi, les éléments nécessaires à l'exécution de toutes les opérations sont:

- un multiplieur/additionneur 16 bits	:	$(D) \Leftarrow (A)\times(B)+(C)$	pour la boucle 1;
- un multiplieur/accumulateur 16 bits	:	$(C) \Leftarrow (A)\times(B)+(C)$	pour la boucle 2;
- un soustracteur 16 bits	:	$(C) \Leftarrow (A)-(B)$	pour l'innovation;
- un décaleur droite 16 bits	:	pour la contrainte de positivité;	
- un décaleur gauche-droite	:	pour la mise à l'échelle de l'estimé de mesure.	

Les multiplieurs feront l'objet d'une étude particulière à la section 5.3.8. Dans cette partie nous ne nous intéresserons qu'au soustracteur et aux décaleurs. La figure 26 donne néanmoins un aspect de l'ensemble de l'UAL.

a) Le soustracteur à anticipation de retenue (CLA)

Comme son nom l'indique, c'est en fait un additionneur à anticipation de retenue 16 bits. On exploite un des avantages de la notation complément à deux qui permet à une même structure d'effectuer des additions et des soustractions. L'additionneur à anticipation de

retenue exploite le fait que la retenue c_i peut ne pas dépendre explicitement de la retenue c_{i-1} mais, peut être exprimée comme une fonction du cumulande, du cumulateur et de quelques retenues d'ordre inférieur. Cette approche permet une augmentation considérable de la vitesse. Les retenues entrant dans l'additionneur peuvent être générées simultanément par un générateur à anticipation de retenue. Ceci résulte en un temps d'addition croissant lentement avec la longueur des opérandes. Il y a néanmoins un problème: plus n devient grand, plus le nombre d'entrées des portes dans la logique de génération de la retenue devient grand. La retenue finale, c_{n-1}, requiert donc un ET à n+1 entrées. Cela pose un problème lié au nombre

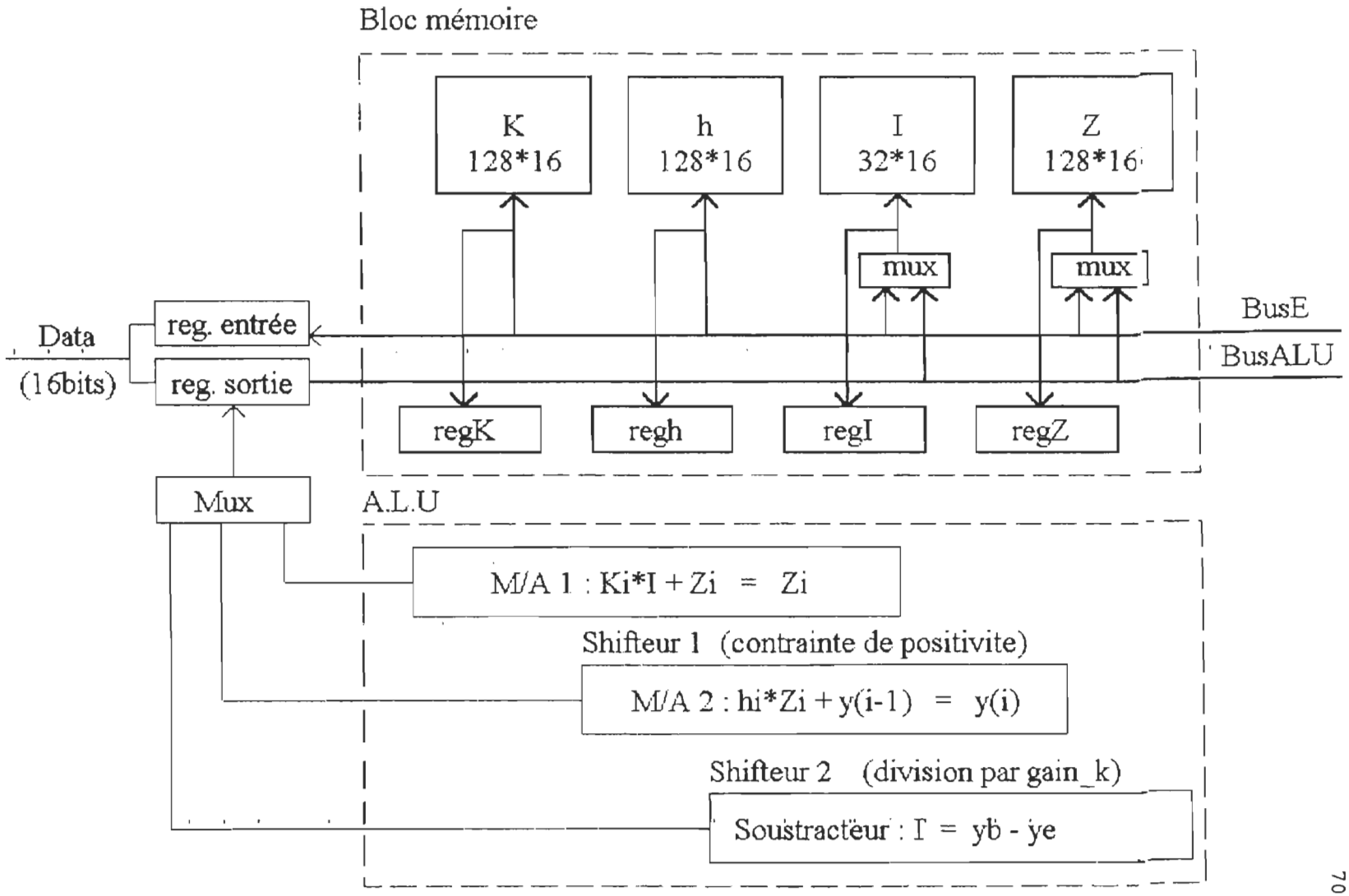


Figure 26: Ensemble Bloc mémoire et UAL

d'entrées maximal pour les technologies actuellement disponibles (fan-in). Pour les grandes opérandes, une solution efficace et pratique consiste à constituer des groupes de génération et de propagation. Il s'agit de considérer des groupes d'additionneurs (4 groupes de 4 bits) comme des unités élémentaires. La retenue issue du plus haut rang, i+3, peut être créée dans le groupe même, où provenir d'une propagation du groupe précédent. Les performances de l'additionneur qui en découle seront discutés au chapitre 6.

b) Les décaleurs

Quel que soit le sens dans lequel le décalage se fait, une des exigences du décalage est la suivante. La notation complément à deux doit être respectée en ce qui concerne le résultat du décalage. Pratiquement, pour que le résultat des décalages dans un sens comme dans l'autre, corresponde aux opérations mathématiques que l'on veut effectuer, les règles suivantes de décalage doivent être respectées:

règle 1 : Décalage à droite : On décale à droite tous les bits et le bit de poids le plus fort (MSB) reste inchangé; ceci assure que le signe est conservé.

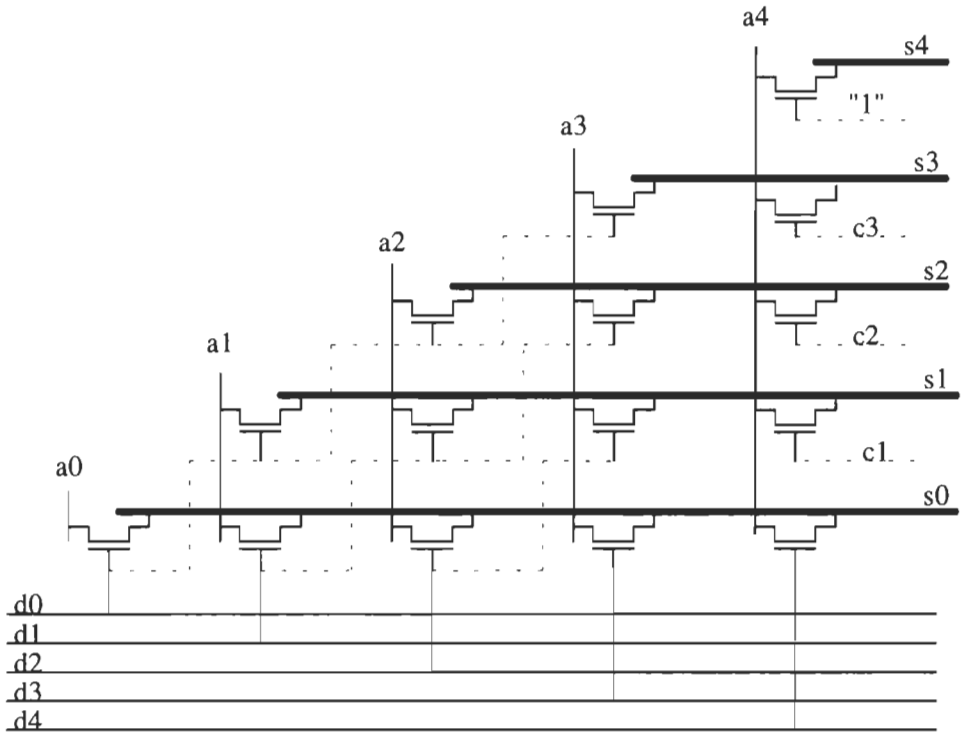
règle 2 : Décalage à gauche : On décale à gauche tous les bits et le bit de poids le plus faible (LSB) est mis à zéro (0).

Les deux décaleurs sont des variantes du décaleur barillet, tel qu'utilisé dans le projet OM de CALTECH [19].

- *Le décaleur de contrainte de positivité*

Cet élément de l'UAL a pour rôle d'effectuer la division d'une opérande 16 bits par un multiple de 2. Pratiquement, cela se fait en décalant l'opérande vers la droite du nombre de positions désiré. Contrairement à un décaleur barillet, qui exige une matrice de n^2 transistors, vu le nombre réduit de décalages prévu par le décaleur (vers la droite seulement) celui-ci n'en utilise que la moitié. La commande est un mot de 16 bits d(15:0). La logique du

décaleur a été concue de sorte qu'un seul des bits de commande doit être à "1". La position de ce bit indique le nombre de décalages à effectuer vers la droite. Cette règle nous a permis de réduire le nombre de signaux de commande interne à calculer à n-1. Un tel décaleur de grandeur 5 bits est présenté à la figure 27.



(Les signaux c3, c2 et c1 sont calcules par une logique interne)

Figure 27 : Structure du décaleur de contrainte de positivité (cas cinq bits)

Pour un décaleur 16 bits, le délai de décalage est égal au délai de calcul du plus long signal de commande, soit celui de 3 portes NOR (se trouvant dans la logique interne de commande) pour le signal c1, avec en plus le temps de transit dans un transistor. Le décaleur a un circuit logique interne servant à calculer les signaux de commande de la colonne de droite de la matrice de transistors (pour l'insertion du signe à la gauche du mot). La simula-

tion de cet élément est présenté au chapitre 6. Nous décrirons un décaleur gauche-droite dans ce qui suit.

- *Le décaleur de réajustement d'échelle*

Son rôle est d'effectuer l'opération suivante sur l'estimé de la mesure $\hat{y}$:

$$\hat{y}_{k\_h} = \hat{y} / \text{gain\_K}$$

où gain_K est un paramètre ayant servi à réajuster le vecteur gain K pour avoir le maximum d'amplitude. Pour faciliter cette opération, gain_K est un multiple de 2. Selon le cas, l'opération à effectuer sur l'estimé de la mesure $\hat{y}$ sera soit une division (décalage à gauche) ou une multiplication (décalage à droite).

Le décaleur que nous proposons est identique à celui présenté à la section précédente, avec en plus la deuxième moitié supérieure de la matrice de transistors. Celle-ci est chargée du décalage à gauche. Une colonne supplémentaire sert à l'insertion éventuelle de "0" à la gauche du mot pour respecter la notation complément à deux. Le bloc du décaleur se présente comme suit:

Figure 28 : Décaleur gauche-droite

Select<1:0> sert à indiquer au décaleur la direction du décalage: gauche, droite ou direct selon le tableau 3:

Tableau 3 : commande du décaleur gauche/droite

Select<1:0>		direction
0	0	direct
0	1	droite
1	0	gauche

La commande Select est directement issue du microséquenceur. La matrice des transistors pour un décaleur gauche-droite de 4 bits est illustrée à la figure 29.

La moitié inférieure effectue le décalage à droite et l'autre, le décalage à gauche. Contrairement au décaleur de contrainte de positivité, les commandes d<15:0> sont validées

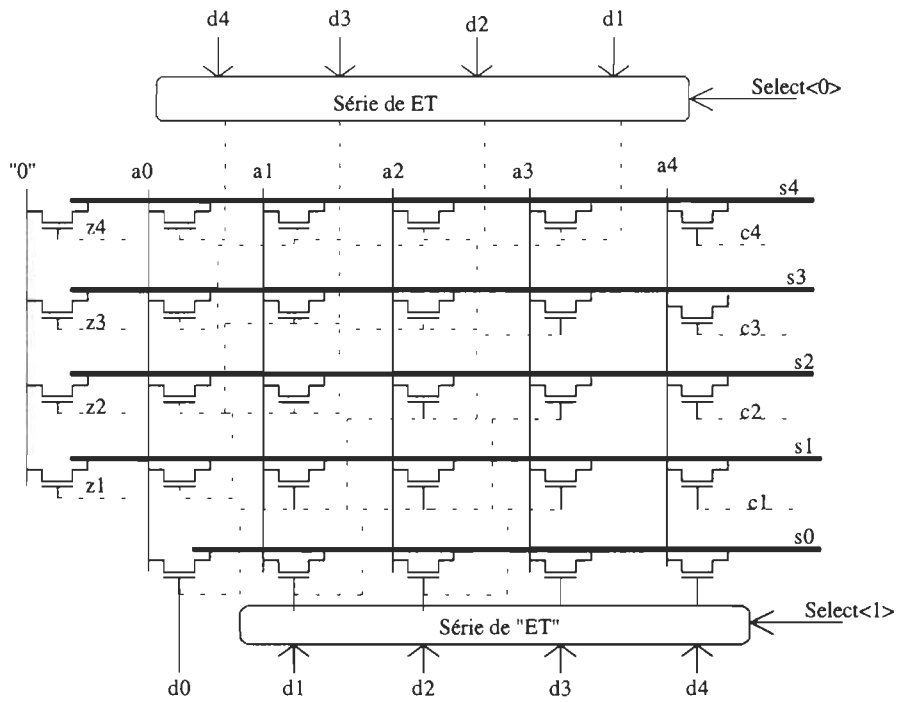


Figure 29 : Matrice de transistors pour le décaleur gauche-droite

par les bits 1 et 0 de l'entrée Select<1:0>. Si ce n'est pas le cas, les transistors de toute une moitié de la matrice restent bloqués. Il y a donc trois groupes de transistors ne devant pas être en conduction simultanément:

- matrice inférieure (décalage à droite);
- matrice supérieure (décalage à gauche);
- diagonale (direct).

La simulation de ce décaleur est présentée au chapitre 6 où sont caractérisés les temps de décalage. Évidemment, ceux-ci sont égaux à ceux du décaleur de positivité. Il est à noter que la logique de commande de cette matrice est double (comparée au décaleur de contrainte de positivité). En effet, il y a deux colonnes à piloter: celle de droite pour l'insertion du bit de poids fort dans le cas d'un décalage à droite (signaux c(i)), et celle de gauche dans le cas de l'insertion d'un "0" pour un décalage à gauche (signaux z(i)).

5.3.9. Le multiplieur/accumulateur 16×16 bits pipeliné à 100 MHz

La densité de calculs à effectuer étant énorme, les performances du processeur sont en grande partie déterminées par celles de son unité arithmétique et logique. En effet, les paramètres les plus importants lors de la reconstitution sont l'erreur et la vitesse de reconstitution. Si le premier est rapidement résolu en augmentant le nombre de bit des opérandes (16 bits), le second exige un design optimisé et surtout une solution offrant des coûts de production acceptables. Dans le contexte du prototypage d'un circuit aux fins de recherche, ce n'est pas tant le coût de production qui compte, mais la faisabilité du prototype; les deux sont cependant directement liés à la surface de silicium utilisée. Dans ce chapitre, nous nous concentrerons plus longuement sur la structure des multiplieurs qui déterminent la taille et la performance de l'unité arithmétique et logique.

a) Algorithme de multiplication

Les données étant en notation complément à deux, nous utilisons une technique classique de multiplication d'opérandes dans une telle notation. Ceci accélère la multiplication en évitant les délais dus aux reconversions signe-magnitude. Plutôt que de traiter l'opérande comme un ensemble de *positions* de bits, (le poids binaire dépend de la position du bit dans le vecteur), une autre approche consiste à considérer l'opérande comme un ensemble de symboles binaires de poids différents, avec un bit de signe négatif, lui aussi pondéré[20]. Takagi a proposé un algorithme utilisant une représentation binaire redondante mais dont les caractéristiques (surface et temps de calcul) sont équivalentes à celles d'un multiplieur avec arbre de Wallace [21].

Soit le nombre en notation complément à deux:

$N=(a_{n-1}\ a_{n-2}\ \ldots\ \ldots\ a_1\ a_0)$ où a_{n-1}est le bit de signe désigné. La valeur de N peut s'exprimer sous la forme:

$$N = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \qquad (5.1)$$

L'algorithme de multiplication directe en complément à deux introduit par Baugh et Wooley [22] a pour principal avantage que les signes de tous les termes sommés sont positifs, permettant ainsi une construction à partir d'additionneurs de un bit uniquement. La régularité d'une telle structure est très intéressante pour une réalisation VLSI.

Considérons deux opérandes entières, en notation complément à deux, le multiplicande $A = (a_{m-1}\ a_{m-2}\ \ldots\ a_1\ a_0)$ et le multiplieur $B = (b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)$. Les valeurs de A et B peuvent s'écrire selon l'équation :

$$A = -b_{m-1}2^{m-1} + \sum_{j=0}^{m-2} b_j 2^j \qquad (5.2)$$

$$B = -b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j \tag{5.3}$$

La valeur du produit P est :

$$P = A*B = (p_{m+n-1}\ p_{m+n-2} \cdot \cdot \cdot \cdot \cdot \cdot p_1\ p_0)$$

Le produit peut s'écrire en notation complément à deux sous forme de produit des a_i et des b_j, pondérés par des facteurs appropriés comme suit :

$$P = a_{m-1}b_{n-1}2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} - \sum_{i=0}^{n-2} a_{m-1}b_i 2^{m-1+i} - \sum_{i=0}^{m-2} a_i b_{n-1} 2^{n-1+i} \tag{5.4}$$

Dans l'équation précédente, les troisième et quatrième termes sont négatifs; Les signes de tous les termes $a_{m-1}b_i$ pour i=0, ... n-2 et $a_i b_{n-1}$ pour i=0, ... m-2 sont tous négatifs. Il est préférable d'avoir tous les signes positifs, car cela permettra une addition directe dans chaque colonne des matrices de produits partiels. En effet, dans ce cas, il ne sera pas nécessaire de faire une soustraction supplémentaire pour les éléments négatifs. Nous cherchons à ajouter l'opposé des termes négatifs, tout en gardant une expression mathématiquement correcte pour le produit. À ce point, il est avantageux de mettre l'équation (5.1) sous une forme qui aidera à générer une matrice de sous-produits tous négatifs. En développant l'équation (5.4) dans ce sens, on arrive à la formulation suivante du produit [20]:

$$\begin{aligned} \mathbf{P} \quad = \quad & a_{m-1}b_{n-1}2^{m+n-2} + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \qquad (5.5) \\ & + 2^{m-1}(-2^n + 2^{n-1} + \bar{a}_{m-1}2^{n-1} + a_{m-1} + \sum_{i=0}^{n-2} a_{m-1}\bar{b}_i 2^i) \\ & + 2^{n-1}(-2^m + 2^{m-1} + \bar{b}_{n-1}2^{m-1} + b_{n-1} + \sum_{i=0}^{m-2} b_{n-1}\bar{a}_i 2^i) \end{aligned}$$

Par exemple, pour un multiplieur 4×4 bits, la matrice de sommation est la suivante :

				$a_3\bar{b}_0$	a_2b_0	a_1b_0	a_0b_0
			$a_3\bar{b}_1$	a_2b_1	a_1b_1	a_0b_1	
		$a_3\bar{b}_2$	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	$\bar{a}_2b_3$	$\bar{a}_1b_3$	$\bar{a}_0b_3$			
	$\bar{a}_3$			a_3			
1	$\bar{a}_3$			a_3			
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Dans la section suivante, nous nous intéresserons à la manière la plus adéquate pour réaliser une telle matrice en dimension 16 par 16 bits.

b) Mise en oeuvre

Comme nous l'avons discuté dans l'étude architecturale, l'objectif principal avec les multiplieurs (accumulateurs ou additionneurs) est d'atteindre une période de 10ns minimum. Ce critère, ajouté à des considérations liées au coût d'un multiplieur (fonction de sa surface) nous conduisent à une catégorie précise d'algorithmes. En effet, le nombre d'étages pipeline va déterminer la latence du multiplieur. Dans le cas du pire échantillonnage (en termes de précision) avec un transopérateur de 64 éléments, quatre étages pipelines donneront une latence de 8 périodes, soit 11% du temps de reconstitution. Un étage supplémentaire à chaque multiplieur nous donnerait une latence de près de 13.5%, ce que nous jugeons comme trop élevé, préférant rester dans l'ordre des 10%. Quatre étages pipelines ayant en moyenne 10ns chacun nous amènent à rechercher un algorithme de multiplication 16 bits ayant un temps total de calcul de l'ordre de 40ns. Sachant qu'en technologie 1.2µm, le temps de propagation classique d'un additionneur de un bit est de 1.2ns, il nous a fallu pipeliner à chaque six additionneurs de un bit. Cette considération exclut donc toutes les techniques de

pipelinage rapides telles que proposées dans [23] et [24]. Les performances qui en découleraient seraient inutiles, vue la cadence qui est essentiellement imposée par les délais de lecture et écriture des mémoires et leurs circuits adresseurs; comme nous l'avons mentionné dans l'étude architecturale, une étude plus poussée pourrait proposer une structure de mémoire plus adaptée.

L'algorithme de multiplication fait donc apparaître la matrice de sous-produits figure présentés à la figure 30. Cette matrice a été découpée en différents secteurs qui font apparaître que le produit final d'une multiplication est la sommation de produits partiels qui, plutôt que d'être considérés comme le résultat d'un "ET logique" entre deux bits des opérandes, sont les résultats des produits 4×4bits. En effet, en divisant les opérandes (16 bits) A et B en 4 sous-opérandes de 4 bits chacun, le résultat de ses multiplications partielles se présente sous une forme plus intéressante pour un arbre de Wallace qui en fera les sommations. Pour respecter la formulation exacte de la matrice de sous-produits, certains secteurs ne correspondent pas exactement à des multiplications 4×4bits. Les secteurs notés "1" (figure 31.a) sont de véritables produits 4×4bits entre des parties des opérandes A et B. Les types d'opérations dans les secteurs notés "2", "3" et "4" sont présentées à la figure 31.b, 31.c et 31.d. Cette méthode peut être utilisée pour n'importe quelle longueur d'opérande. L'opération est donc effectuée en modules correspondant aux 16 produits partiels qui sont obtenus indépendamment les uns des autres.

Une fois les produits partiels obtenus, il faut décider de la manière d'effectuer la sommation des bits de même poids. Un "ripple-adder" entraîne une trop grande lenteur dans la transmission de la retenue, chaque étage devenant dépendant du précédent : pour un multiplieur 16 bits, le chemin critique serait alors de 32 additionneurs de un bit. Un "carry-save-adder" quant à lui ne propage pas la retenue mais la sauvegarde jusqu'à ce que toutes les additions soient complétées. L'utilisation d'un additionneur à anticipation de retenue

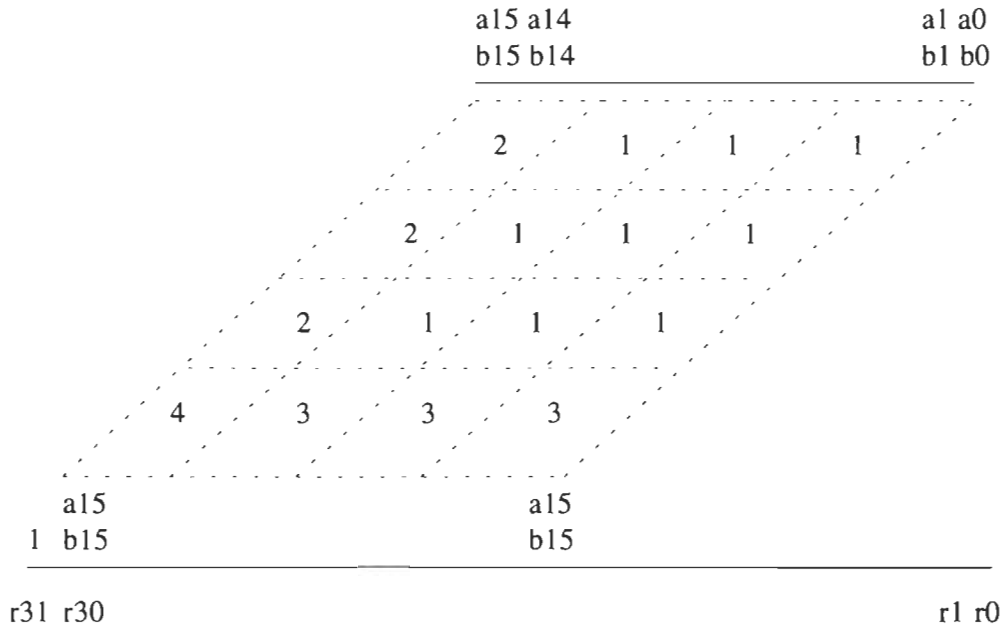


Figure 30 : Produit 16 bits

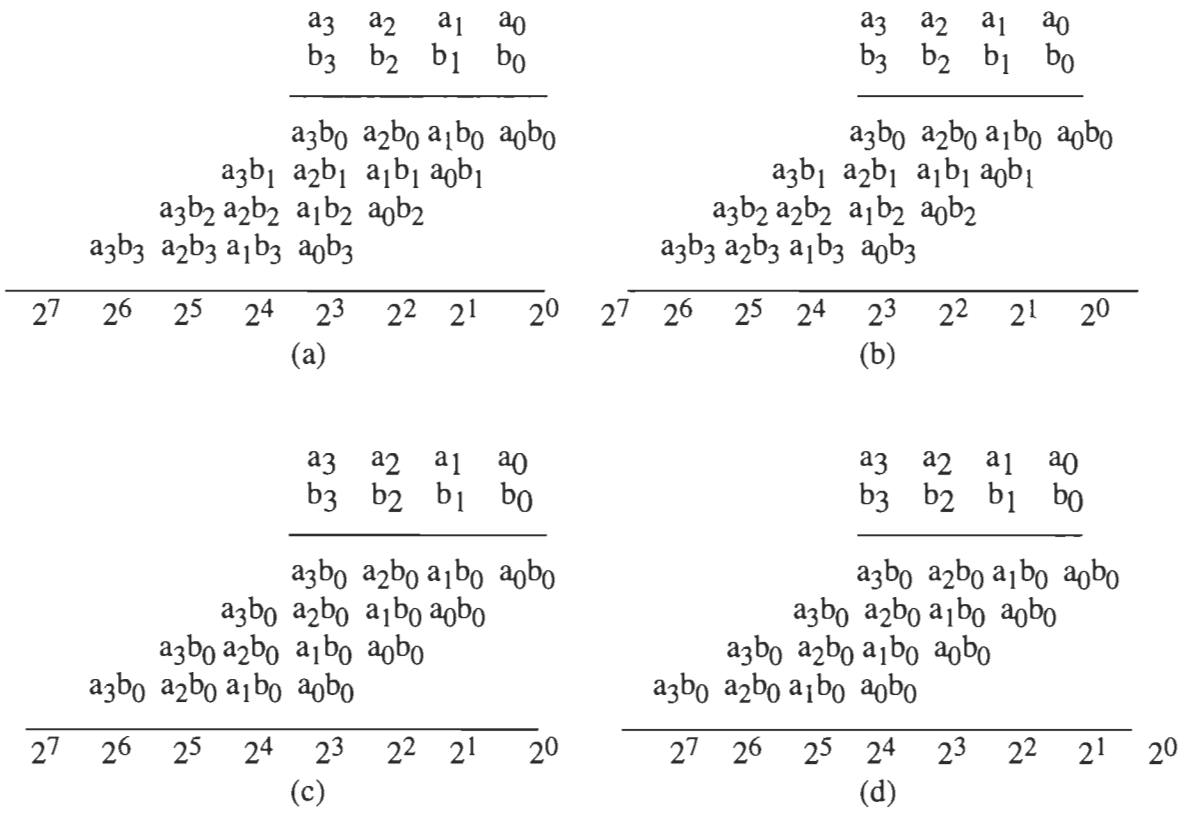


Figure 31 : Les 4 types de sous-produits 4×4 bits

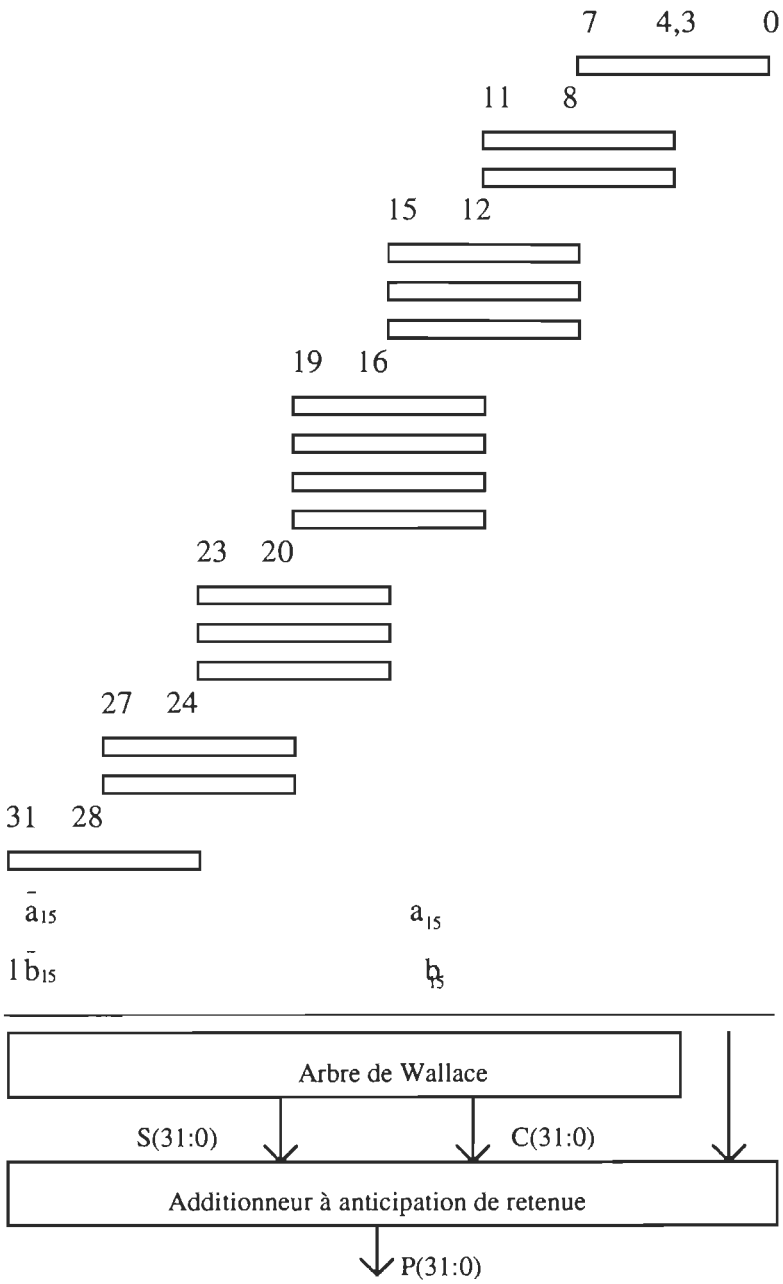


Figure 32: Structure du multiplieur 16×16 bits

(CLA) permet de présenter d'abord le résultat sous une forme redondante, c'est-à-dire, somme (S) et carry (C) pour chaque puissance de 2, pour ensuite propager la retenue dans un temps constant. Le calcul des signaux S et C est effectué par un arbre de Wallace et a un chemin critique relativement court. Les sous produits partiels se présentent ainsi en entrée de

l'arbre de Wallace (voir figure 32). Un WT à k entrées (Wk) est un circuit additionneur qui produit la somme des k entrées de même poids. L'alignement des sous-produits est réalisé par le regroupement des entrées dans un WT approprié. La colonne d'alignement est importante afin de produire le résultat. Les éléments notés W3 sont des additionneurs de un bit. (le chiffre 3 désigne le nombre de bits d'entrée de même poids). Les Modules W7 et W5 sont un agencement d'additionneurs de un bit fournissant 3 bits de sortie de poids progressifs croissants. La structure de l'arbre de Wallace est présentée aux figures 34, 35, 36, 37, 38, 39 et 40. Ces figures montrent comment les signaux issus des 16 sous-produits partiels sont regroupés selon leur poids et sommés pour fournir, pour chaque puissance de 2, deux signaux uniquement qui seront les entrées de l'additionneur à anticipation de retenue.

c) Architecture pipelinée

Pour atteindre les performances décrites dans le cahier de charge, il est utile d'imposer un pipelinage adéquat aux multiplieurs. En effet, la performance du processeur est essentiellement fonction de celle de son UAL, et en particulier des multiplieurs.

Comme on l'a mentionné précédemment, les 16 produits partiels s'effectuent parallèlement; le délai de cette partie du multiplieur est alors celui d'un produit partiel 4*4 bits. Le chemin critique (délai total) pour une multiplication est donc :

$$\mathbf{D_{tot} = D_{4*4bits} + D_{W7+W3+halh\text{-}adder} + D_{CLA} + D_{accu}}$$

avec

$D_{4*4bits}$: délai d'un produit 4*4 bits : 8.5ns

$D_{W7+W3+halh\text{-}adder}$: délai dans un arbre de Wallace à 7 entrées : 8ns

D_{CLA} : Délai de l'additionneur à anticipation de retenue : 6.5 ns

D_{accu} : délai dans l'accumulateur (CLA) : 6.5 ns

$\Rightarrow$ **D_{tot} = 29.5ns, sans les delais d'interconnections.**

Pour atteindre un débit de 100MHz (10ns) en tenant compte des délais dus aux fils qui s'ajouteront après le routage, ce chemin critique doit être subdivisé en au moins quatre étages. Nous avons donc adopté une subdivision "naturelle", **intercalant une série de registres entre les principales composantes du multiplieur** : le bloc des produits partiels 4*4 bits, l'arbre de Wallace, le CLA en sortie de l'arbre de Wallace, le CLA pour l'accumulation (multiplieur 1) ou pour l'addition avec l'entrée c(15:0) (multiplieur 2). De ces quatre étages, le premier est celui offrant le délai le plus grand : 8.5ns; il reste néanmoins inférieur à 10ns. La figure 33 donne la structure pipelinée des deux multiplieurs.

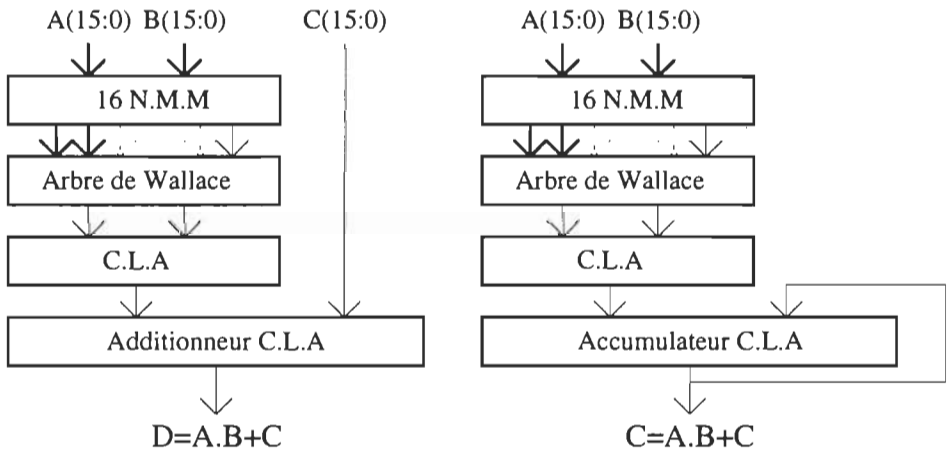


Figure 33 : Structure pipelinée des multiplieurs

(a) multiplieur/additionneur

(b) multiplieur/accumulateur

Les figures qui suivent exposent la structure de l'arbre de Wallace. On y montre comment les signaux issus des produits partiels sont sommés (selon leur poids respectif), de sorte à produire pour chaque poids allant de 0 à 31, deux bits (S et C) qui seront ensuite additionnés par un additionneur à anticipation de retenue.

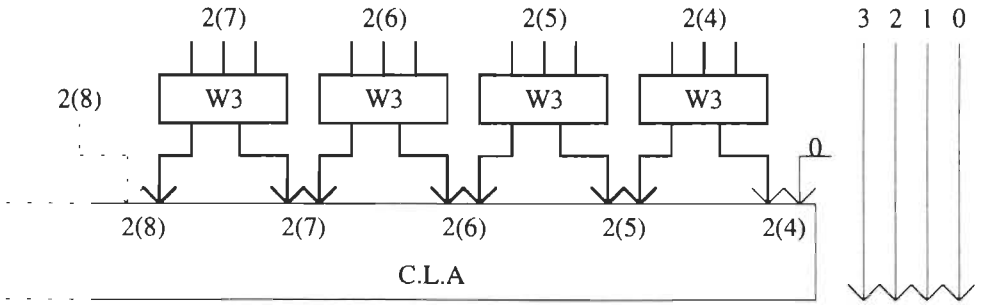


Figure 34 : Premier groupe des quatre W3

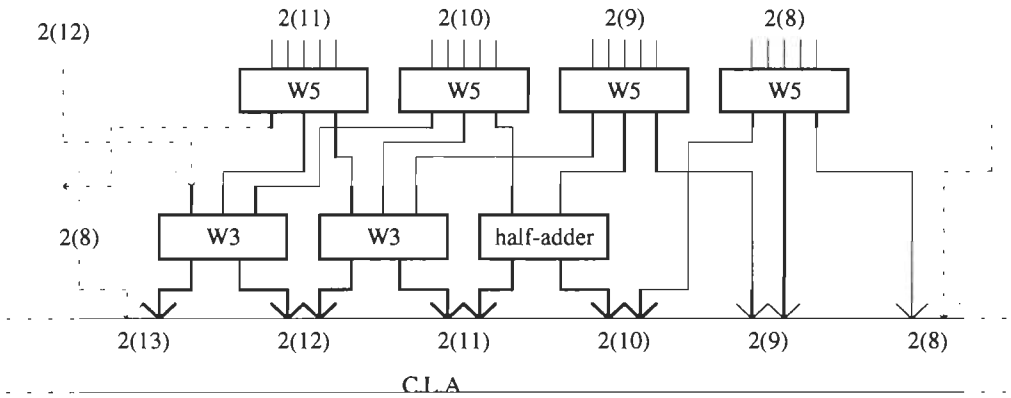


Figure 35 : Premier groupe des quatre W5

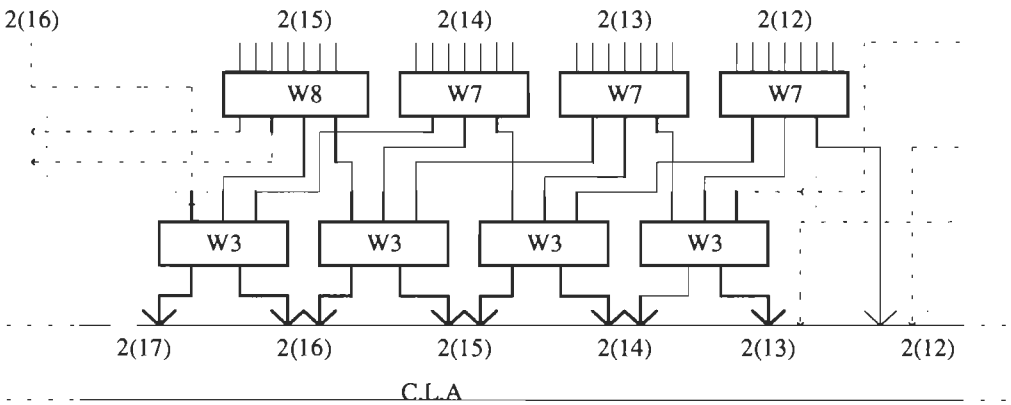


Figure 36 : Premier groupe des quatre W7

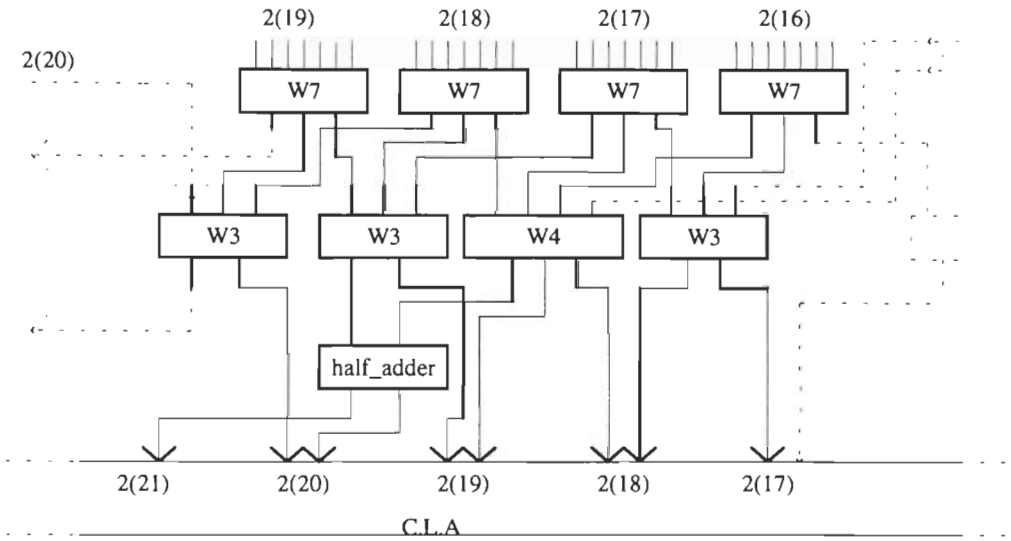


Figure 37 : Deuxième groupe des quatre W7

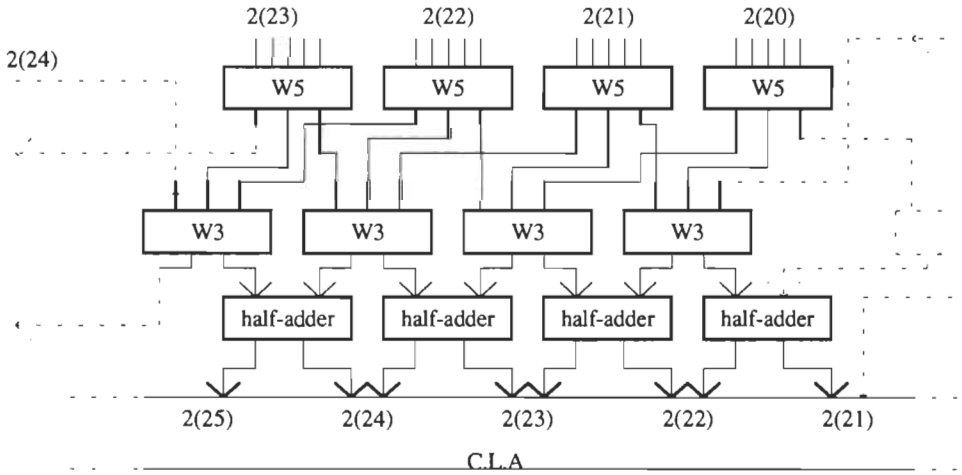


Figure 38 : Deuxième groupe des quatre W5

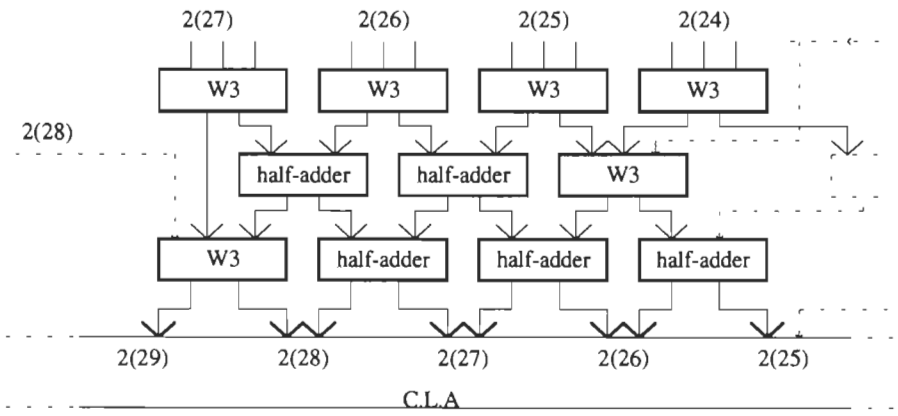


Figure 39 : Deuxième groupe des quatre W3

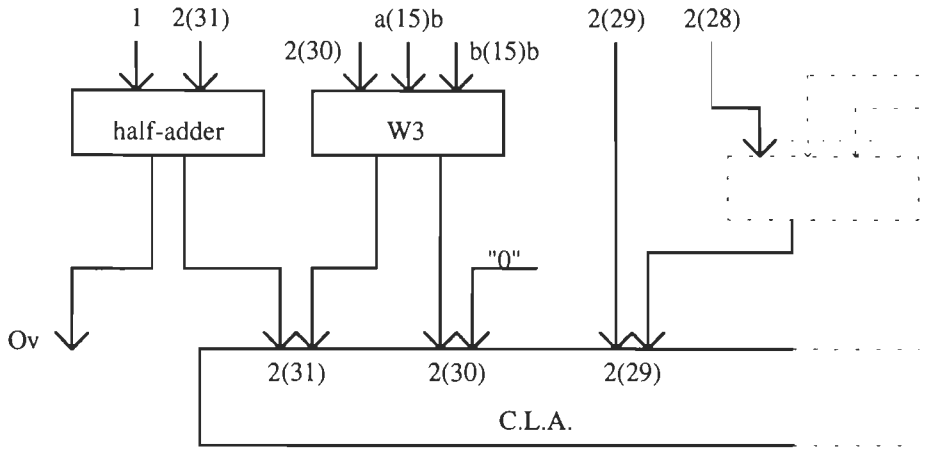


Figure 40 : Arbre de Wallace : bits 28 à 31

6. RÉSULTATS

Dans ce chapitre, nous exposerons les résultats des simulations effectuées, de sorte à valider les choix architecturaux explicités aux chapitres précédents. La performance du processeur est non-seulement fonction de l'architecture choisie, mais aussi du séquencement interne des microactions; ces deux aspects sont en fait étroitement liés. Pour mieux comprendre les résultats de simulation, nous avons jugé utile de présenter le séquencement des microactions. Dans ce chapitre, nous étudierons le séquencement des opérations et présenterons des simulations des principales composantes de l'architecture. Nous nous intéresserons donc dans un premier temps au rythme des microactions commandées par le microprogramme, puis, aux résultats de simulations.

6.1. LE SÉQUENCEMENT DES OPÉRATIONS

Au chapitre 4, nous nous sommes interessés au fonctionnement de l'architecture en expliquant les "macro-opérations" effectuées localement dans celle-ci (multiplication, décalage, lecture, écriture, ... etc ... etc). Dans cette section, nous réviserons le fonctionnement interne sous un autre aspect : le rythme des microactions commandées par le microprogramme.

6.1.1. Le circuit d'horloge.

L'architecture est dessinée pour fonctionner avec une horloge à deux phases sans recouvrements : ψ_1 et ψ_2. Ces deux phases sont générées intérieurement à partir d'une horloge externe de 100MHz.

Dans tout cette section, nous appellerons F1 et F2, les instants où les signaux ψ_1 et ψ_2 sont à leur front montant. On dira par exemple qu'un registre est actionné à F1, c'est-à-dire au front montant de ψ_1, même si son horloge est encore active quand elle présente un front descendant. F1 et F2 ne désignent que des instants et non des comportements, et ces instants sont les front montants des phases ψ_1 et ψ_2.

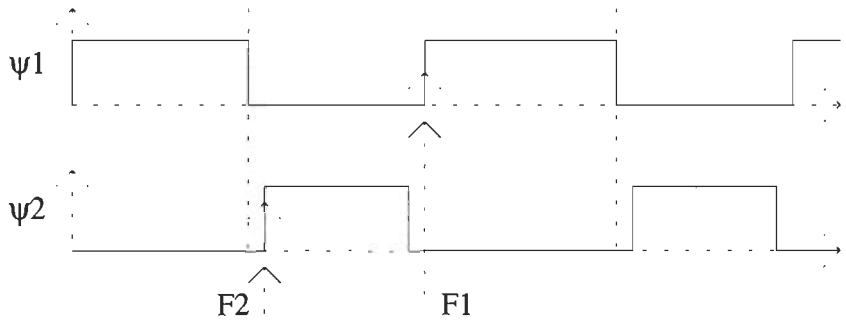


Figure 41 : Les deux phases sans recouvrements

6.1.2. Le séquencement des UGA, du bloc mémoire et de l'UAL

Dans le but de réduire la taille de la mémoire de microprogramme, on pourrait penser enlever le registre d'adresse mémoire (MAR) en sortie des UGA. Ceci en effet devrait respecter un des principes de resynchronisation qui veut que lorsque la logique entre deux registres est stable, un des registres peut être enlevé [26]. Mais, ceci n'est pas le cas, et de plus, il est important que l'adresse émise reste stable en attendant que l'additionneur recalcule l'adresse suivante : cela est vital, vu les délais de lecture/écriture des mémoires. **Le séquencement des UGA et des compteurs se réduit à celui de deux types d'éléments : les registres et les multiplexeurs. Les registres sont pilotés à F_2 et les multiplexeurs à F_1.** Finalement, les adresses et les états des compteurs sont stables à F_2. La figure 42 donne

un schéma d'une UGA à un pointeur en spécifiant les temps où ils reçoivent leur commandes respectives : descente de l'horloge pour un registre ou signal de commande pour un multiplexeur. Ainsi, et cela sera valable pour presque l'ensemble du processeur, le chemin des données (multiplexeurs) est spécifié à ψ_1 tandis que les registres qui mémorisent les données (front descendant des horloges de commande) sont actionnés à ψ_2.

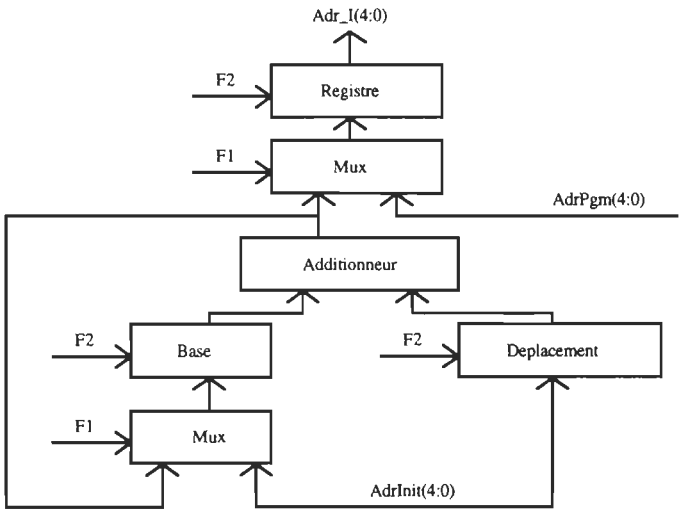


Figure 42 : UGA de la mémoire I (un pointeur)

L'objectif de cette section est d'établir le séquencement le plus important du processeur. Il concerne le transfert des données:

- des mémoires vers l'UAL;

- de l'UAL vers les mémoires;

- et aussi la commande du cheminement de ces données.

Les deux phases doivent permettre l'adressage des mémoires et le fonctionnement pipeliné de l'UAL. L'opération la plus répétée lors du fonctionnement (et qui doit donc être la plus optimisée en termes de temps) consiste en:

- les adressages mémoire pour extraire les données devant alimenter les registres;

    * commande multiplexeur UGA

* commande registre UGA

- lecture (ou écriture) en mémoire

* signal "Read" ou "Write"

- commande des registres en entrée ou en sortie de l'UAL ainsi que des registres pipelines.

La figure 43 explique la synchronisation de ces microactions aux temps F1 et F2. En la parcourant de haut en bas, il apparait que l'ensemble UGA-Bloc mémoire-UAL peut fonctionner avec deux horloges sans recouvrements. La chaîne de déplacement des données montre que l'on peut alterner, le long de ce trajet, les horloges ψ_1 et ψ_2 aux multiplexeurs et aux registres respectivement.

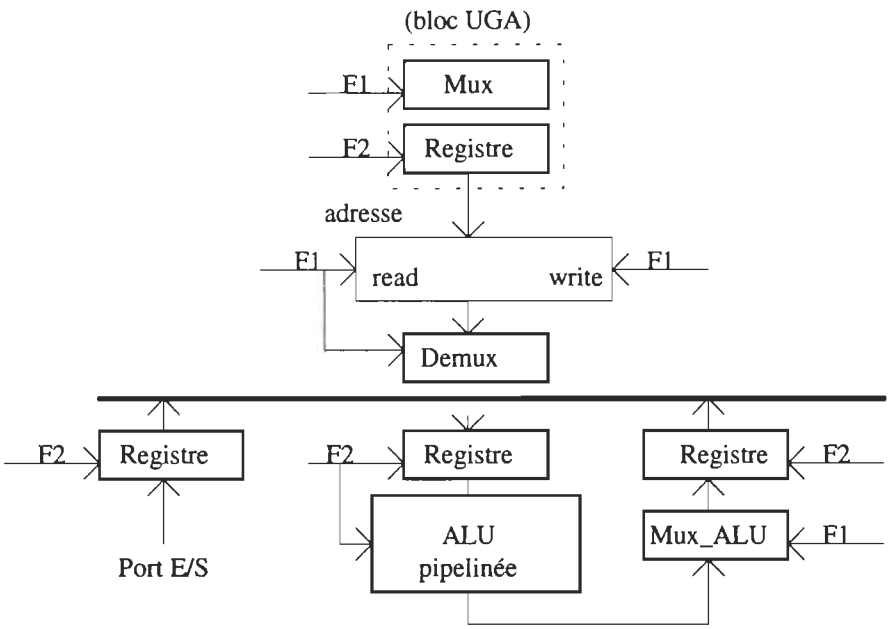


Figure 43 : Synchronisation du bloc mémoire et de L'UAL

On y remarque certaines choses qui ont permis de définir le séquencement le plus optimal.

1. il y a deux principaux sens dans lesquels cheminent les données: un sens descendant (L'UGA fournit l'adresse aux mémoires qui fournissent les données à l'UAL), et un sens ascendant (c'est le retour des données de l'UAL vers les mémoires ou vers le port de sortie). Le premier sens équivaut à la lecture (de K_i, I, Z_i et H_i), le second à l'écriture de Z_i.

2. Ces deux sens sont constitués d'une suite de multiplexeurs (ou démultiplexeurs) et de registres, intercalés quelquefois d'une logique (les mémoires par exemples); sauf dans le cas de l'UAL où les registres pipelines sont intercalés de logique de calcul seulement.

La remarque 1 implique que l'algorithme, pour son bon fonctionnement, est constamment en train d'effectuer une série de lecture/écriture, les deux sens devant évoluer en parallèle.

Étant donné que :

- l'UGA ne peut générer qu'une seule adresse par période (les deux phases ψ_1 et ψ_2 sont en effet nécessaires comme on l'a montré),

- l'écriture en mémoire ne se fait pas à la même case où la lecture a eu lieu (dans le cas du vecteur d'état, ces deux cases sont espacées d'un délai du au pipelinage de l'UAL);

il faudra donc deux périodes d'horloge pour effectuer une lecture et une écriture correspondant à cet élément de l'algorithme, l'écriture correspondant à la mise en mémoire Z. La remarque 2 indique que l'on peut conserver le principe suivant :

- chemin (mux et demux) spécifié à ψ_1;

- registres actionnés à ψ_2.

La figure 44 explique le séquencement du bloc mémoire et de l'UAL. **Cette chronologie impose que les signaux "Read" et "Write" des mémoires sont tous les deux actionnés à ψ_1, mais cela ne pose pas de problèmes, car les lectures et les écritures ont lieu alternativement, à des périodes différentes.**

Le séquencement proposé à la figure 44 est celui du cycle de lecture/écriture de la mémoire Z. Tandis que le multiplexeur de sortie de l'UGA (celle de la mémoire Z a deux

pointeurs) doit être sans cesse redirigé à ψ_1 (pointeur1 pour la lecture et pointeur 2 pour l'écriture), l'adresse est disponible (stable) seulement à ψ_2. READ peut être actionné par un

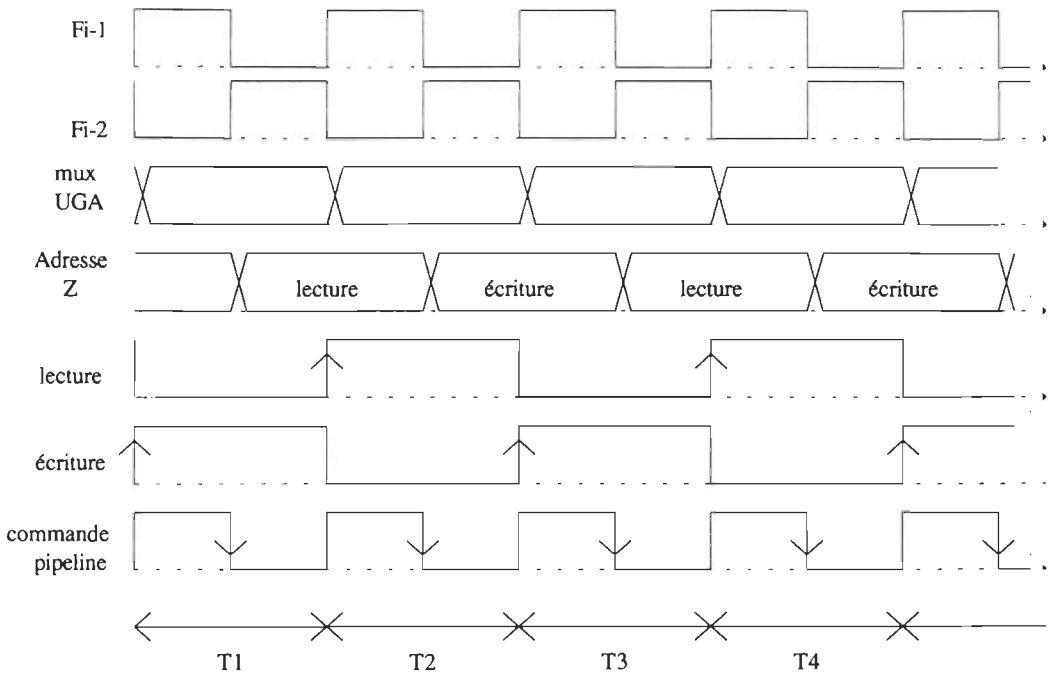


Figure 44 : Séquencement du bloc mémoire

front montant à ψ_1. La même situation se répète à la période T2, sauf que l'adresse générée est celle de l'écriture.

- T1 : ψ_1 : écriture + Mux-UGA(lecture)

  ψ_2 : adressage lecture + commande pipeline

- T2 : ψ_1 : lecture + Mux-UGA(écriture)

  ψ_2 : adressage écriture + commande pipeline

Finalement, l'UAL sort un résultat toutes les deux périodes.

6.1.3 Le séquencement du microséquenceur.

Nous montrerons le séquencement du microséquenceur avec une horloge à deux phases sans recouvrements. Celui-ci est essentiellement influencé par son chemin critique. Il

est constitué de la chaîne de rétroaction allant du multiplexeur de condition, passant par le contrôleur de polarité, le multiplexeur d'adresse, le registre d'adresse et le multiplexeur d'adresse interne/externe (Voir figure 45). Cette rétroaction qui sert à décider du choix de l'adresse suivante (program counter, return ou jump) en fonction du signal que l'on décide de contrôler (indicateur de retour de boucle par exemple) doit "idéalement" se faire en un cycle, ce qui permettrait au programmeur de programmer dès le mot suivant le test adéquat; mais la chaîne de rétroaction étant évidemment assez longue, une telle solution n'est réalisable que si l'on réduit la vitesse de l'horloge, ce qui réduirait considérablement les performances (vitesse) du processeur. Pour garder la vitesse de reconstitution intacte, nous proposons que l'ensemble de la boucle de rétroaction soit exécutée non pas en un cycle, mais en deux; il y a deux conséquences à un tel séquencement:

- l'apparition d'un délai d'un cycle à l'exécution d'une instruction du microprogramme;

- l'exécution d'un mot effectuant un test microprogrammé doit être programmé un cycle après le mot qui sélectionnera le bit à tester. C'est ce délai qui est rapporté dans la section réservée à la microprogrammation en annexe. Le chemin critique est mis en évidence dans la figure 45.

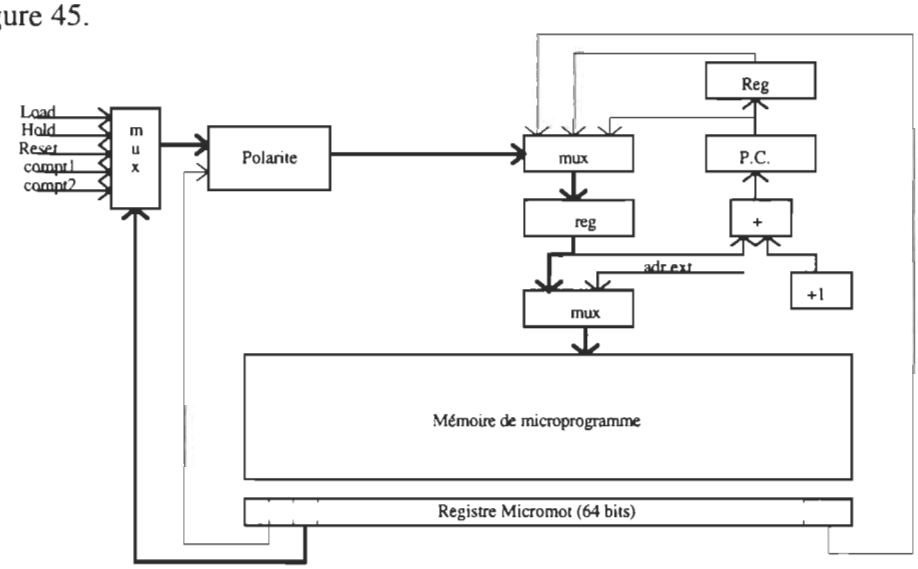


Figure 45 : Chemin critique du microséquenceur

Dans cette partie, nous avons été amené à faire un choix : nous avons préféré déplacer le problème du côté de la programmation, plutôt que de réduire la vitesse de l'horloge interne.

Le séquencement apparaît alors comme suit :

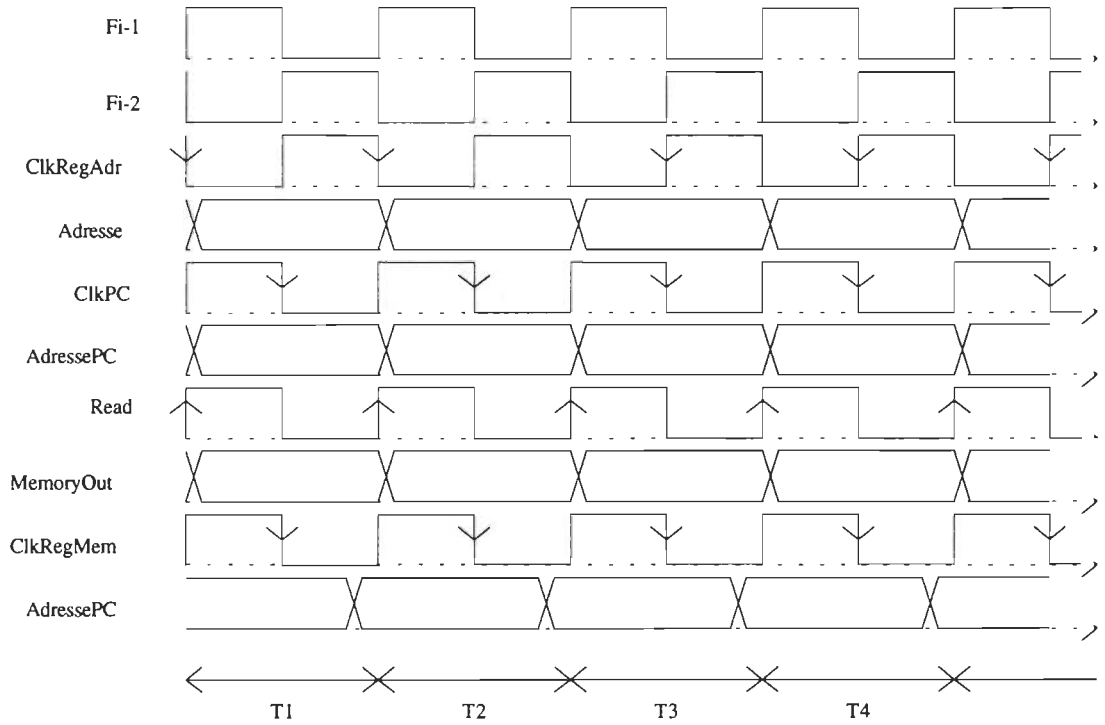


Figure 46 : Séquencement du microséquenceur

Rappelons que F1 et F2 sont les instants où les signaux ψ_1 et ψ_2 montent.

Si le registre d'adressage de la mémoire est actionné à F1, l'adresse est alors immédiatement disponible. L'entrée du compteur ordinal est stable (PC+1) après un délai dû à l'additionneur de 5 bits chargé d'effectuer l'incrémentation du PC (Voir figure 45); on actionne le PC à la fin de la demi-période, soit à F2. On peut appliquer un "Read" (front montant) en misant sur le non-recouvrement des horloges pour que le Read se fasse avec un

retard minime, mais toujours dans la première demi-période. Le micromot (MemoryOut) est alors disponible après un délai de lecture et on peut actionner le registre de micromot (ClkRegMem) à F2. Puis, vient la boucle de rétroaction, mux-polarité-mux qui se stabilise après un certain délai. Lorsque l'adresse de "Retour" est stable, et que les trois adresses éventuelles sont stables,le même séquencement reprend. On remarque alors que le mot qui a été lu ne peut comporter les éléments de test correspondant aux trois adresses stabilisées; c'est le mot suivant qui s'en chargera, d'où la notion de délai d'un mot.

6.1.4 Conclusion sur le séquencement des opérations

L'architecture est dessinée pour fonctionner avec deux phases sans recouvrements, ψ_1 et ψ_2, générées intérieurement à partir d'une horloge externe de 100MHz. Chacune des deux phases correspond à un comportement spécifique de l'architecture. **Le séquencement des UGA et des compteurs se réduit à celui de deux types d'éléments : les registres et les multiplexeurs. Les registres sont pilotés à F_2 et les chemins (commande des multiplexeurs) sont spécifiés à F_1.** Le microséquenceur ajoute une période supplémentaire lors de tests (en début et fin de boucle).

6.2. SIMULATIONS

6.2.1 Méthodologie

Nous avons procédé à un certain nombre de simulations. Celles-ci portent sur les blocs essentiels de l'architecture. Les simulations ont été choisies à cause de leur implication sur les performances de vitesse du processeur. Ainsi, le microséquenceur et la chaîne UGA-mémoire-registres-multiplieur ont été entièrement testés blocs par blocs, puisqu'ils sont constamment sollicités lors du fonctionnement. La méthodologie adoptée est donc la suivante:

- identification des blocs devant être testés : cela s'est fait en évaluant l'influence de ceux-ci sur la vitesse de reconstitution;

- choix de vecteurs de test : ceux-ci ont été choisis dans la plupart des cas, de sorte à exprimer soit les delais les plus longs (chemins critiques), soit les cas d'opérations logiques pertinents permettant de s'assurer de la justesse de la conception, soit des cas de fonctionnement faisant mieux ressortir (au lecteur) le fonctionnement espéré.

Nous expliciterons la simulation du bloc mémoire à titre d'exemple pour ensuite résumer dans un tableau les résultats de simulation effectuées sur toute l'architecture.

6.2.2 Simulation du bloc mémoire

Pour ne pas avoir à écrire les vecteurs de test de tout le bloc-mémoire au grand complet, nous avons dessiné et simulé seulement le chemin critique de celui-ci. Quand on observe le schéma du bloc mémoire, on observe que le chemin le plus long emprunté par les données issues des mémoires est celui allant de la mémoire de paramètre (I) aux registres des décaleurs 1 et 2 ou à celui de l'innovation. Ce chemin critique est mis en évidence à la figure 47.

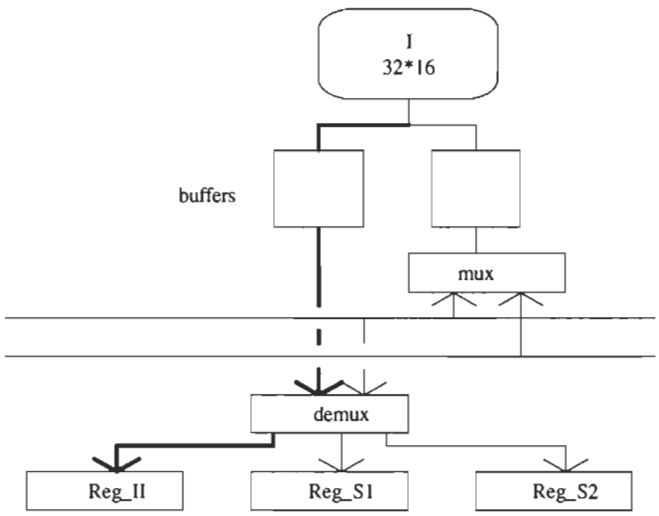


Figure 47 : Chemin critique du bloc mémoire

Le chemin critique mis en évidence dans la figure 47 fait apparaître les délais suivants:

- délai de lecture en mémoire;

- délai à travers les buffers de mémoire (qui sert à gérer les niveaux de haute impédance sur les entrées/sorties de la mémoire);

- délai à travers le démultiplexeur.

Pour la simulation, on a procédé à l'écriture d'une donnée ($FAFA_h$) dans une case mémoire (adresse 01_h) pour ensuite relire la même donnée. C'est en effet après ces trois délais que la donnée issue de la mémoire est stable en entrée du registre auquel elle est destinée. La simulation de cet ensemble fait apparaître le délai total de 7.637 ns. **Ce délai total nous confirme la possibilité de fonctionner à 100mhz, à condition que le placement et routage soit optimisé de sorte à garder ce delai en deça des 10ns. 10ns sont en effet suffisantes pour générer une adresse et effectuer une lecture parallèlement.**

6.2.3 Résultats de simulation

Le projet a été conçu essentiellement dans l'environnement Cadence EDGE en ciblant une technologie 1.2 microns de Northern Telecom et en utilisant les outils de simulation HSPICE et SILOS II; les simulations se sont effectuées au niveau schématique. Compte tenu de l'installation primaire de SYNOPSYS au moment des simulations, la représentation VHDL du processeur n'a été ni simulée ni synthétisée. Le module GVAN de SYNOPSYS a tout de même servi à valider la syntaxe de la représentation VHDL du processeur qui est présentée en annexe A; elle peut donc être utilisée comme version de référence pour des réalisations futures. L'ensemble des résultats de simulation est résumé dans le tableau 4; les principales simulations sont celles du chemin critique du bloc mémoire et des multiplieurs. Elles confirment le bon fonctionnement logique de ces ensembles et la

vitesse de reconstitution que l'on peut en espérer. Le tableau 5 donne une comparaison entre les estimations des temps de reconstitution pour un DSP5600, la première version du processeur spécialisée et la version que nous proposons.

Tableau 4 : Résultats de simulations

BLOC	Description du test	Résultats
L'UGA à un pointeur	Chargement de la base et du déplacement, commande des multiplexeurs.	Adresses générées avec un débit de 100MHz.
L'UGA à deux pointeurs	Chargement des deux bases et des deux déplacements, commande des multiplexeurs.	Deux adresses générées parallèlement avec un débit de 100MHz
Les compteurs de boucle	Reset, initialisation, test de l'indicateur de fin de boucle	Fonctionnement à 100MHz et indicateur de zéro.
Le bloc-mémoire	Test du chemin critique allant de la mémoire aux registres en entrée des multiplieurs	Delai 7.637ns pour obtenir la donnée issue de la mémoire à l'entrée du registre
L'additionneur de 1 bit	Toutes les opérations logiques possibles	delai = 1.646ns
Multiplieur 4×4 (NMM)	Test du 1er étage dans le pipelinage du multiplieur	delai = 8.148ns
Multiplieur 16bits	produit de deux opérandes complément à deux	Latence de 4 périodes et débit de 100MHz
Additionneur à anticipation de retenue 32 bits	Accumulateur du multiplieur. Somme de deux mots de 32 bits	delai = 6.4506ns
Soustracteur 16 bits	Calcul de l'innovation	delai = 5.42ns
Décaleur de contrainte de positivité	Décalage d'un mot vers la droite	delai = 1.7ns
Décaleur de remise à l'échelle	Décalage commandé vers la gauche puis vers la droite	delai = 1.7ns

Tableau 5 : Comparaison des temps de reconstitution

Dispositif	Fréquence ou temps de reconstitution
DSP 56000	5.6 ms
Première version	45 MHz
Deuxième version	100 Mhz / 3µs

6.2.4 Testabilité

La testabilité d'un tel dispositif est rendue très ardue pour plusieurs raisons :

- la présence de plusieurs mémoires dans l'architecture; les mémoires sont en effet difficilement testables.

- la taille du dispositif, qui est déjà rendue critique à cause de celle des mots (16bits), de celle des mémoires et à cause de l'insertion de deux multiplieurs 16 bits pipelinés, limite quelquepeu la quantité de circuits à ajouter pour permettre un test de production.

La testabilité est sans aucun doute un sujet important. Mais, **notre objectif principal étant de proposer une architecture pour un prototype académique, notre propos ne portera que sur les tests fonctionnels. Les sujets plus généraux liés aux tests de production ne seront que partiellement abordés.**

Nous avons donc opté pour une méthode de test réduite. Un test par un microprogramme approprié permettrait d'évaluer partiellement l'état de l'architecture. Il est en effet possible d'écrire un microprogramme qui aura un rôle réduit, orienté vers le test des mémoires. Il pourrait servir au préchargement de la mémoire visée pour ensuite amener le contenu d'une case mémoire vers l'UAL, effectuer une opération simple dont le résultat sera

recueilli en sortie par le port d'entrée/sortie. L'avantage des structures microprogrammée est justement le fait que celles-ci autorisent l'installation de tels programmes.

On peut donc, sans implanter de structures dédiées au test, tester certains chemins de données dont :

- les registres d'entrée/sortie;
- toutes les mémoires incluant celle du microséquenceur;
- l'UAL et toutes ses composantes.

Remarquons néanmoins que de tels tests ne seront pas aussi précis sur l'élément en faute où la nature de la faute.

D'autre part, le placement et routage automatique du processeur indique une surface de 58.54 mm^2 pour le coeur du circuit, soit un carré de 7.65mm de coté.

7. CONCLUSION

Notre but étant d'étudier tous les aspects liés à l'intégration du filtre de Kalman dans un processeur, nous avons proposé une architecture microprogrammable qui lui est dédiée et destinée aux signaux stationnaires. Ayant pour référence principale la première version de ce processeur telle qu'explicitée à la section 5.1, nous avons procédé à plusieurs modifications majeures dans le but d'améliorer les qualités principales d'un tel processeur : la vitesse et la qualité de reconstitution ainsi que le coût (surface) d'une telle architecture.

La qualité de reconstitution a été nettement améliorée par:

- la largeur des mots (16 bits);

- l'ajout dans l'algorithme d'une équation supplémentaire permettant l'utilisation optimale du nombre de bits. En effet, la notation adoptée a permis une meilleure numérisation grâce à l'incorporation d'un décaleur aidant à la mise à l'échelle des vecteurs de trop petites amplitudes. Le prétraitement effectué sur H et K permet d'avoir une reconstitution plus exacte, en leur attribuant une amplitude maximale. D'autre part la notation employée évite les débordements.

La vitesse de reconstitution est améliorée par:

- la présence de deux multiplieurs pipelinés; cette structure systolique a été simulée à 100MHz.

- le lien entre les mémoires et l'UAL qui est assuré par une série de registres dédiés à chacune des mémoires; ceci permet entre autres d'avoir à chaque instant la donnée qu'il faut à l'entrée désirée de l'UAL. Les multiplieurs sont cadencés au même rythme que les accès

mémoires à un débit de 10ns, ce qui permet d'effectuer une reconstitution toutes les 3µs, en plus des temps requis par les deux convertisseurs.

- Cette structure est dédiée au parallèlisme des équations les plus importantes en nombre d'opérations dans l'algorithme; ceci a été rendu possible grâce à l'agencement particulier des deux multiplieurs, l'un étant pourvoyeur de données à l'autre.

On a réussi à faire une **relative économie de surface**:

- en attribuant un double rôle à certains modules; ceci entraîne une réduction du nombre de signaux issus du microséquenceur et donc la taille de la mémoire de microprogramme qui lui est attribuée. Par exemple, les deux compteurs de boucles effectuent parallèlement le rôle d'adresseur des mémoires dont l'adressage est parallèle à l'évolution des boucles.

- en réduisant de beaucoup la structure de bus; celle-ci a été limitée à deux bus correspondant aux principaux flux de données : vers ou en provenance de l'UAL.

L'architecture que nous proposons regroupe les éléments de mémoires correspondant aux différents vecteurs entrant en jeu dans la reconstitution de signaux; ces mémoires sont pilotables de l'extérieur à l'initialisation. Par la suite, elles sont gérées par un microséquenceur programmable interne. Cette structure à programmation horizontale est adéquate pour de telles applications.

Le projet a été conçu essentiellement dans l'environnement Cadence EDGE en ciblant une technologie 1.2 microns de Northern Telecom et en utilisant les outils de simulation HSPICE et SILOS II; les simulations se sont effectuées au niveau schématique. Le module GVAN de SYNOPSYS a servi à valider la représentation VHDL. Le dessin de masque n'a pas été effectué; nous nous sommes arrêtés au placement automatique (sans optimisation) pour l'estimation de surface.

Le présent projet constitue une étape dans le développement de structures intégrées pour des applications métrologiques. Il m'a permis de me familiariser avec les principaux algorithmes de reconstitution de mesurandes, ainsi qu'aux outils de conception VLSI.

Les perspectives qui s'offrent à ce projet seraient la mise en commun de celui-ci avec d'autres champs de recherche, de sorte à mettre au point des systèmes de mesures compacts et autonomes. Un tel projet pourrait être utile dans tous les domaines où l'on a besoin de faire un suivi de données en temps réel (le contrôle environnemental par exemple). Ce processeur a besoin d'un environnement électronique dont la conception pourrait faire l'objet d'un projet d'ingénierie. Ce projet fait partie intégrante du programme de recherche du laboratoire de systèmes de mesure de l'Université du Québec à Trois-Rivières, avec la collaboration de l'École polytechnique de Montréal. Je considère cette collaboration comme souhaitable dans un environnement universitaire où seules les grandes coopérations donnent les meilleurs idées. J'espére que ce mémoire en est une des plus modestes démonstrations.

BIBLIOGRAPHIE

[1] : A. Barwicz, "System Approach to Electric Measurement/, Conference Record, IEEE, IMTC'93, pp. 397-402, Irvin Ca., 18-20 Mai 93

[2] : Signal processing toolbox for use with MATLAB, T., P., Krauss, L., Shure, J., N., Little, The Math Works Inc., February 1994.

[3] : D., Massicotte, Reconstitution de signaux basée sur le filtrage de Kalman, implantation sur la carte DSP56000, Rapport interne, Laboratoire de systèmes de mesures, Université du Québec à Trois-Rivières, 1991, 33 pages.

[4] : A., Barwicz, D., Massicotte, Y., Savaria, M., A., Santerre, An Integrated Structure for Kalman-Filter-based Measurand Reconstruction, IEEE Transactions on Instrumentation and Measurement, vol. 43 No.3, pp. 403-410.

[5] : M. Kunt, Techniques modernes de traitement numérique des signaux, Presses Polytechniques et universitaires romandes, Paris, 1991, p. v.

[6] : R. Z. Morawski, Unified Approach to Mesaurement Signal Reconstruction", invited paper, Conference Record IEEE IMTC/93, Irvine, California, May 18-20, 1993

[7] : R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", ASME, Journal of Basic Engineering, Vol. 82-D, pp.34-45, 1960.

[8] : R. E. Kalman, R. S. Bucy, "New Results in Linear Filtering and Prediction Theory", Trans. ASME, Series D Journal of Basic Engineering, Vol. 38, pp. 95-101, 1960.

[9] : N. Wiener : The Extrapolation, Interpolation and Smoothing of Stationnary Time Series with Engineering Applications. J. Wiley, New-York, 1949.

[10] : P.Rao and M. Bayoumi, "An Algorithm Specific VLSI Parallel Architecture for Kalman Filter", IEEE Press : VLSI Signal Processor, IV, 1991, pp. 264-273.

[11] : D. Lawrie, P. Fleming, "Parallel Processing of the Kalman Filter", VLSI System for DSP and Control, Ed. R. Woods, pp. 15-20.

[12] : S. Y. Kung, and J. N. Hwang, "Systollic Array Design for Kalman Filtering", IEEE Transactions on Signal Processing, Vol. 39, No. 1, January 1991, pp. 171-182.

[13] : F. M. F. Gaston and G. W. Irwin, "Systollic Approach to Square root Information Kalman Filtering", International Journal of Control, Vol. 50, No. 1, 1989, pp.225-248.

[14] : M. R. Azimi-Sadjadi, T. Lu and E. M. Nebot, "Parallel and Sequential Block Kalman Filtering and Their Implementation Using Systollic Arrays", IEEE Transactions on signals Processing, Vol. 39, No. 1, January 1991, pp. 137-147.

[15] : D., Massicotte, M., A., Santerre, A., Barwicz, Structure de calculs parallèles pour le filtrage de Kalman dans la reconstitution de signaux, Congrès Canadien en génie électrique et informatique, Toronto, Ontario, Septembre 92.

[16] : D. Massicotte, R. Z. Morawski, A. Barwicz, "Incorporation of a Positivity Constraint into a Kalman-Filter-Based Algorithm for Correction of Spectrometric Data", Conference Record IEEE IMTC/92, New-York, May 1992, pp. 590-593

[17] : M., A., Santerre, D., Massicotte, A., Barwicz, Y. Savaria, "Architecture of Specialized Processor for Kalman-Filter-Based Reconstruction", Canadian Conference on Electrical and Computer Engineering, Toronto, Ontario, Septembre 92.

[18] : J. Mick, J. Brick, Bit-Slice Microprocessor Design, Mc Graw-Hill, McGraw-Hill, 1980.

[19] : C. Mead, L. Conway, Introduction aux Systèmes VLSI, InterEditions, 1983, p. 168-169.

[20] : J. Cavanagh, Digital computer arithmetic: design and implementation. chap. 3.6, 1984, page 203-213.

[21] : N. Takagi, H. Yasuura and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", IEEE Transactions on Computers, Vol. C-34, No. 9, September 1985

[22] : C. R. Baugh, B. A. Wooley, "A two's Complement Parallel Array Multiplication Algotithm", IEEE TRansactions on Computers, Vol. C-22, No. 12, Dec. 1973, pp. 1045-1047.

[23] : F. Lu, H. Samueli, /A 140 MHz bit-level pipelined Multiplier/Accumulator using a New Dynamic Full-Adder Cell Design", IEEE Symposium on VLSI Circuits, 1990, pp. 123-124.

[24] : F. Lu, H. Samueli, J. Yuan and C. Svensson, "A 700 MHz 24 bits pipelined Accumulator in 1.2μm CMOS for Application as a Numerically Controlled Oscillator", IEEE Journal of Solid-State Circuits, Vol 28, No. 8, August 1993.

[25] : H., B., Bakoglu, Cicuits, interconnections and packaging for VLSI, chap. 8.6, pp. 358-9

[26] : Glasser, L. A., Dobberpuhl D., The design and analysis of VLSI circuits , Addison Wesley, 1985 473 p.

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

[28] : D. Massicotte, R. Z. Morawski, A. Barwicz, "Incorporation of a Positivity Constraint into a Kalman-Filter-Based Algorithm for Correction of Spectrometric Data", IEEE Transactions on Instrumentation and Measurements, Vol. 44, No. 1, Février 1995, pp. 2-7,

[29] : D. Massicotte, R. Z. Morawski, A. Barwicz, "Efficiency of Constraining the Set of Feasible Solutions in Kalman-Filter-Based Algorithms of Spectrometric Data Correction", Instrumentation and Measurements technology Conference (IMTC/93), Irwin, California, 18-20 May, pp. 496-499,

[30]: A. Barwicz, D. Massicotte, Y. Savaria, P. Pango, R. Z. Morawski, "An Application-Specific Processor Dedicated to Kalman-Filter-Based Correction of Spectrometric Data ", IEEE Transactions on Instrumentation and Measurement, accepté en Septembre 1994, sera publié en Juin 1995.

ANNEXE A : DESCRIPTION VHDL DU PROCESSEUR

```
------------------------------------------------------------------------
--
--              Partie du programme de description VHDL du
--              specialise pour le filtrage de Kalman
--
--              Package reunissant divers parametres :
--                      _ types de vecteurs
--
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
PACKAGE buses IS
        TYPE bus64 IS ARRAY(63 downto 0) OF std_ulogic;
        TYPE bus52 IS ARRAY(51 downto 0) OF std_ulogic ;
        TYPE bus16 IS ARRAY(15 downto 0) OF std_ulogic ;
        TYPE bus12 IS ARRAY(11 downto 0) OF std_ulogic ;
        TYPE bus7 IS ARRAY(6 downto 0) OF std_ulogic ;
        TYPE bus5 IS ARRAY(4 downto 0) OF std_ulogic;
        TYPE bus3 IS ARRAY(2 downto 0) OF std_ulogic;
        TYPE bus2 IS ARRAY(1 downto 0) OF std_ulogic;
end buses;


-----------------------------------------------------------------
-----       Description des signaux generaux du processeur
-----------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
PACKAGE vddgnd IS
        SIGNAL vdd: std_logic := '1';
        SIGNAL gnd: std_logic := '0';
END vddgnd;


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

```
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                  _ Tous les circuits speciaus (particuliers)
--                  _ Des fonctions de conversion
--                  _ Un package qui les reunit tous
--
--------------------------------------------------------------------------


--------------------------------------------------------------------
--      Package definissant des circuits speciaux et les
--      fonctions utilises dans l'architecture
--
--      Ce sont :
--
--      * Logique de test de polarite dans le microsequenceur
--      * Additionneur 7 bits
--      * UGA-I (un pointeur )
--      * UGA-Z (deux pointeurs )
--      * Compteur 7 bits (compteurs 1 et 2)
--      * Compteur 16 bits (compteur d'echantillons)
--      * Les fonctions de conversion INTEGER/ BitARRAY
--
--      Les structures de ceux-ci sont en fin de fichier
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
PACKAGE circuits IS
        COMPONENT polarlogic PORT
                (Pin,Ptest,Bit4:IN   std_logic;
                ClkRetour:OUT   std_logic;
                SelMuxAdr:OUT  std_logic_vector(1 downto 0));
        END COMPONENT;
        COMPONENT addition7 PORT
                (in0,in1:IN  std_logic_vector(6 downto 0);
                out0:OUT  std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT addition16 PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                out0:OUT std_logic_vector(15 downto 0));
        END COMPONENT;
```

```
        COMPONENT addition5 PORT
                (in0,in1:IN  std_logic_vector(4 downto 0);
                out0:OUT  std_logic_vector(4 downto 0));
        END COMPONENT;
        COMPONENT Compteur12 PORT
                (AdrInit:IN  std_logic_vector(6 downto 0);
                SelMuxPtr:IN   std_logic;
                RB:IN   std_logic;
                Indic0:OUT   std_logic;
                Adresse:OUT  std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT UGA_1_pointeur PORT
                (AdrInit:IN  std_logic_vector(4 downto 0);
                SelMuxPtr:IN   std_logic;
                RB:IN   std_logic;
                RD:IN   std_logic;
                SelMuxPtrPgm:IN   std_logic;
                ClkMAR:IN      std_logic;
                Adresse:OUT  std_logic_vector(4 downto 0));
        END COMPONENT;
        COMPONENT UGA_2_pointeur PORT
                (AdrInit:IN  std_logic_vector(6 downto 0);
                SelMuxPtr1:IN   std_logic;
                RB1:IN   std_logic;
                RD1:IN   std_logic;
                SelMuxPtr2:IN   std_logic;
                RB2:IN   std_logic;
                RD2:IN   std_logic;
                SelMuxPtrPgm:IN  std_logic_vector(1 downto 0);
                ClkMAR:IN      std_logic;
                Adresse:OUT  std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT Indic0_7bit PORT
                (in0:IN std_logic_vector(6 downto 0);
                out0:OUT std_logic);
        END COMPONENT;

        FUNCTION Bit16_to_Int( bitarray : std_logic_vector(15 downto 0))   RETURN
integer;
        FUNCTION Bit7_to_Int( bitarray :  std_logic_vector(6 downto 0))   RETURN
integer;
        FUNCTION Bit5_to_Int( bitarray :  std_logic_vector(4 downto 0))   RETURN
integer;
        FUNCTION Int_to_Bit16(intg : INTEGER) RETURN std_logic_vector;
        FUNCTION Int_to_Bit7(intg : INTEGER) RETURN std_logic_vector;
```

```
        FUNCTION Int_to_Bit5(intg : INTEGER) RETURN std_logic_vector;
END circuits;

-- use work.circuits.all;
PACKAGE BODY circuits IS

        FUNCTION Bit16_to_Int( bitarray : std_logic_vector(15 downto 0)) RETURN
integer IS
        VARIABLE result :INTEGER := 0;
        VARIABLE sum :INTEGER := 1;
        BEGIN
                FOR i IN 1 TO bitarray'LENGTH LOOP
                        sum := sum*2;
                        IF bitarray(i) = '1' THEN
                                result := result + sum/2;
                        END IF;
                END LOOP;
                RETURN result;
        END Bit16_to_Int;

        FUNCTION Bit7_to_Int( bitarray :  std_logic_vector(6 downto 0)) RETURN
integer IS
        VARIABLE result :INTEGER := 0;
        VARIABLE sum :INTEGER := 1;
        BEGIN
                FOR i IN 1 TO bitarray'LENGTH LOOP
                        sum := sum*2;
                        IF bitarray(i) = '1' THEN
                                result := result + sum/2;
                        END IF;
                END LOOP;
                RETURN result;
        END Bit7_to_Int;

        FUNCTION Bit5_to_Int( bitarray :  std_logic_vector(4 downto 0)) RETURN
integer IS
        VARIABLE result :INTEGER := 0;
        VARIABLE sum :INTEGER := 1;
        BEGIN
                FOR i IN 1 TO bitarray'LENGTH LOOP
                        sum := sum*2;
                        IF bitarray(i) = '1' THEN
                                result := result + sum/2;
                        END IF;
                END LOOP;
```

```
        RETURN result;
END Bit5_to_Int;

FUNCTION Int_to_Bit16(intg: INTEGER) RETURN std_logic_vector IS
VARIABLE Nb_Bit: INTEGER := 16;
VARIABLE result: std_logic_vector(15 downto 0);
VARIABLE digit: INTEGER := 2**15;
VARIABLE local:INTEGER;
BEGIN
        local := intg;
        FOR i IN Nb_Bit-1 DOWNTO 0 LOOP
                IF local/digit >= 1 THEN
                        result(i) := '1';
                        local :=  local - digit;
                ELSE
                        result(i) := '0';
                END IF;
                digit := digit/2;
        END LOOP;
        RETURN result;
END Int_to_Bit16;

FUNCTION Int_to_Bit7(intg: INTEGER) RETURN std_logic_vector IS
VARIABLE Nb_Bit: INTEGER := 7;
VARIABLE result:  std_logic_vector(6 downto 0);
VARIABLE digit: INTEGER := 2**6;
VARIABLE local:INTEGER;
BEGIN
        local := intg;
        FOR i IN Nb_Bit-1 DOWNTO 0 LOOP
                IF local/digit >= 1 THEN
                        result(i) := '1';
                        local :=  local - digit;
                ELSE
                        result(i) := '0';
                END IF;
                digit := digit/2;
        END LOOP;
        RETURN result;
END Int_to_Bit7;

FUNCTION Int_to_Bit5(intg: INTEGER) RETURN  std_logic_vector IS
VARIABLE Nb_Bit: INTEGER := 5;
VARIABLE result:  std_logic_vector(4 downto 0);
VARIABLE digit: INTEGER := 2**4;
```

```
        VARIABLE local:INTEGER;
        BEGIN
                local := intg;
                FOR i IN Nb_Bit-1 DOWNTO 0 LOOP
                        IF local/digit >= 1 THEN
                                result(i) := '1';
                                local :=  local - digit;
                        ELSE
                                result(i) := '0';
                        END IF;
                        digit := digit/2;
                END LOOP;
                RETURN result;
        END Int_to_Bit5;
  END circuits;


library IEEE;
use IEEE.std_logic_1164.all;
use work.muxdemux.all;
use work.delays.all;
        ENTITY Indic0_7bit IS
        PORT
                (in0:IN std_logic_vector(6 downto 0);
                out0:OUT std_logic);
        END Indic0_7bit;
        ARCHITECTURE behave OF Indic0_7bit IS
        BEGIN
                Indic0_7bit_proc : PROCESS(in0)
                BEGIN
                        IF in0="0000000" THEN
                                out0 <= '1';
                        ELSE
                                out0 <= '0';
                        END IF;
                END PROCESS Indic0_7bit_proc;
        END behave;


------------------ Les circuits speciaux --------------------

        ----- Controle de polarite
library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
```

```
        ENTITY polarlogic IS
        PORT
                (Pin,Ptest,Bit4:IN   std_logic;
                ClkRetour:OUT   std_logic;
                SelMuxAdr:OUT  std_logic_vector(1 downto 0));
        END polarlogic;
        ARCHITECTURE behave OF polarlogic IS
        BEGIN
                polarlogic_proc : PROCESS( Pin,Ptest,Bit4 )
                BEGIN
                        --      Calcul de ClkRetour
                        IF (Pin='0' and Ptest='0' and Bit4='0') or (Pin='1' and Ptest='1' and
Bit4='0') THEN
                                ClkRetour <= '0' after DelayClkRetour;
                                SelMuxAdr <= "10" after DelaySelMux;
                        ELSE
                                ClkRetour <= '1' after DelayClkRetour;
                                IF Pin='0' and Ptest='0' and Bit4='1' THEN
                                        SelMuxAdr <= "01" after DelaySelMux;
                                ELSE
                                SelMuxAdr <= "00" after DelaySelMux;
                                END IF;
                        END IF;
                END PROCESS polarlogic_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.muxdemux.all;
use work.delays.all;
use work.registres.all;
use work.circuits.all;
        ------- UGA et pointeurs
        ENTITY Compteur12 IS
        PORT
                (AdrInit:IN  std_logic_vector(6 downto 0);
                SelMuxPtr:IN   std_logic;
                RB:IN   std_logic;
                ClkMAR: IN   std_logic;
                Out_zero:OUT   std_logic;
                Adresse:OUT  std_logic_vector(6 downto 0));
        END Compteur12;
        ARCHITECTURE behave OF Compteur12 IS
                SIGNAL PtrOut          : std_logic_vector(6 downto 0);
                SIGNAL Reg_Base        : std_logic_vector(6 downto 0);
```

```
        SIGNAL Mux_Out      : std_logic_vector(6 downto 0);
        SIGNAL Reg_Depl     : std_logic_vector(6 downto 0);
        SIGNAL Indic_in     : std_logic_vector(6 downto 0);
        BEGIN
        ----------    Pour chacun des mux, verifier l'ordre des entreees
        -- Structure of pointer 1
        MuxPtr1: Mux2071 PORT MAP(PtrOut, AdrInit,SelMuxPtr,Mux_Out);
        Reg_Depl <= "0000001";
        RegBase1: registre7 PORT MAP(RB, Mux_Out, Reg_Base);
        Addition1: addition7 PORT MAP( Reg_Base, Reg_Depl, PtrOut);
        -- Memory Adress Register
        MAR_UGA_2Ptr: registre7 PORT MAP(ClkMAR, PtrOut, Adresse);
        -- Indicateur de zero

                -- a faire plus tard

        Out_zero <= '0';
    END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
use work.muxdemux.all;
use work.registres.all;
use work.circuits.all;
    ENTITY UGA_1_pointeur IS
    PORT
        (AdrInit:IN  std_logic_vector(4 downto 0);
        SelMuxPtr:IN  std_logic;
        RB:IN  std_logic;
        RD:IN  std_logic;
        SelMuxPtrPgm:IN  std_logic;
        ClkMAR: IN  std_logic;
        Adresse:OUT  std_logic_vector(4 downto 0));
    END UGA_1_pointeur;
    ARCHITECTURE behave OF UGA_1_pointeur IS
        SIGNAL PtrOut       : std_logic_vector(4 downto 0);
        SIGNAL Ptr          : std_logic_vector(4 downto 0);
        SIGNAL Reg_Base : std_logic_vector(4 downto 0);
        SIGNAL Reg_Depl : std_logic_vector(4 downto 0);
        SIGNAL Mux_Out      : std_logic_vector(4 downto 0);
    BEGIN
    ----------    Pour chacun des mux, verifier l'ordre des entrees    ----------
    -- Structure of pointer 1
        MuxPtr1: Mux2051 PORT MAP(Ptr, AdrInit,SelMuxPtr,Mux_Out);
```

```
        RegBase1: registre5 PORT MAP(RB, Mux_Out, Reg_Base);
        RegDepl1: registre5 PORT MAP(RD, AdrInit, Reg_Depl);
        Addition1: addition5 PORT MAP( Reg_Base, Reg_Depl, Ptr);
    -- Selection of the memory pointer
        MuxAdr: Mux2051 PORT MAP(Ptr, AdrInit, SelMuxPtrPgm, PtrOut);
    -- Memory Adress Register
        MAR_UGA_2Ptr: registre5 PORT MAP(ClkMAR, PtrOut, Adresse);
    END behave;


library IEEE;
 use IEEE.std_logic_1164.all;
 use work.delays.all;
 use work.muxdemux.all;
use work.registres.all;
use work.circuits.all;
    ENTITY UGA_2_pointeur IS
    PORT
        (AdrInit:IN  std_logic_vector(6 downto 0);
        SelMuxPtr1:IN   std_logic;
        RB1:IN   std_logic;
        RD1:IN   std_logic;
        SelMuxPtr2:IN   std_logic;
        RB2:IN   std_logic;
        RD2:IN   std_logic;
        SelMuxPtrPgm:IN  std_logic_vector(1 downto 0);
        ClkMAR: IN   std_logic;
        Adresse:OUT  std_logic_vector(6 downto 0));
    END UGA_2_pointeur;
    ARCHITECTURE behave OF UGA_2_pointeur IS
        SIGNAL PTR          :     std_logic_vector(6 downto 0);

        SIGNAL Ptr1         :     std_logic_vector(6 downto 0);
        SIGNAL Reg_Base_1   :     std_logic_vector(6 downto 0);
        SIGNAL Reg_Depl_1   :     std_logic_vector(6 downto 0);
        SIGNAL Mux_Out_1    :     std_logic_vector(6 downto 0);

        SIGNAL Ptr2         :     std_logic_vector(6 downto 0);
        SIGNAL Reg_Base_2   :     std_logic_vector(6 downto 0);
        SIGNAL Reg_Depl_2   :     std_logic_vector(6 downto 0);
        SIGNAL Mux_Out_2    :     std_logic_vector(6 downto 0);
    BEGIN
    ----------    Pour chacun des mux, verifier l'ordre des entrees    ----------
    -- Structure of pointer 1
        MuxPtr1: Mux2071 PORT MAP(Ptr1, AdrInit,SelMuxPtr1,Mux_Out_1);
```

```
            RegBase1: registre7 PORT MAP(RB1, Mux_Out_1, Reg_Base_1);
            RegDepl1: registre7 PORT MAP(RD1, AdrInit, Reg_Depl_1);
            Addition1: addition7 PORT MAP( Reg_Base_1, Reg_Depl_1, Ptr1);
        -- Structure of pointer 2
            MuxPtr2: Mux2071 PORT MAP(Ptr2, AdrInit,SelMuxPtr2,Mux_Out_2);
            RegBase2: registre7 PORT MAP(RB2, Mux_Out_2, Reg_Base_2);
            RegDepl2: registre7 PORT MAP(RD2, AdrInit, Reg_Depl_2);
            Addition2: addition7 PORT MAP( Reg_Base_2, Reg_Depl_2, Ptr2);
        -- Selection of the memory pointer
            MuxAdr: Mux3071 PORT MAP(Ptr1, Ptr2, AdrInit, SelMuxPtrPgm, Ptr);
        -- Memory Adress Register
            MAR_UGA_2Ptr: registre7 PORT MAP(ClkMAR, Ptr, Adresse);
        END behave;

---    Ligne 164
library IEEE;

------- Additionneurs 5, 7 et 16 bits;

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
use work.circuits.all;
        ENTITY addition5 IS
        PORT
            (in0,in1:IN  std_logic_vector(4 downto 0);
            out0:OUT  std_logic_vector(4 downto 0));
        END addition5;
        ARCHITECTURE behave OF addition5 IS
        BEGIN
            addition5_proc : PROCESS(in0,in1)
            BEGIN
                out0 <= Int_to_bit5( Bit5_to_Int(in0)+Bit5_to_Int(in1) )
                AFTER DelayAddition5;
            END PROCESS addition5_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
use work.circuits.all;
        ENTITY addition7 IS
        PORT
            (in0,in1:IN  std_logic_vector(6 downto 0);
            out0:OUT  std_logic_vector(6 downto 0));
```

```
        END addition7;
        ARCHITECTURE behave OF addition7 IS
        BEGIN
                addition7_proc : PROCESS(in0,in1)
                BEGIN
                        out0 <= Int_to_bit7( Bit7_to_Int(in0)+Bit7_to_Int(in1) )
                        AFTER DelayAddition7;
                END PROCESS addition7_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
use work.circuits.all;
        ENTITY addition16 IS
        PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                out0:OUT std_logic_vector(15 downto 0));
        END addition16;
        ARCHITECTURE behave OF addition16 IS
        BEGIN
                addition16_proc : PROCESS(in0,in1)
                BEGIN
                        out0 <= Int_to_bit16( Bit16_to_Int(in0)+Bit16_to_Int(in1) )
                        AFTER DelayAddition16;
                END PROCESS addition16_proc;
        END behave;


------------------------------------------------------------
--      Entites servant a la creation et a la repartition
--      des horloges
------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
ENTITY Clock_Generator IS
        PORT(clock1, clock1b, clock2, clock2b:OUT std_logic);
END Clock_Generator;
ARCHITECTURE Clock_Generator_Behave OF Clock_Generator IS
BEGIN
        clock_proc : PROCESS
        BEGIN
                clock1 <= '1','0' AFTER phaseLag;
```

```
                clock1b <= '0','1' AFTER phaseLag;
                clock2 <= '1' AFTER phaseLag, '0' AFTER on_time;
                clock2b <= '0' AFTER phaseLag, '0' AFTER on_time;
                WAIT FOR period;
        END PROCESS;
END Clock_Generator_Behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.delays.all;
PACKAGE clocks IS
        COMPONENT Clock_Generator
                PORT(clock1, clock1b, clock2, clock2b:OUT std_logic);
        END COMPONENT;
END clocks;
--------------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                _ Tpus les delais
--
--------------------------------------------------------------------------


--------------------------------------------------------------------------
----    Package specifiant les delais des constituants de base  -------

library IEEE;
use IEEE.std_logic_1164.all;
PACKAGE delays IS
        TYPE clock_level IS(Low,Rising,High,Falling);
        --      Caracteristiques de l'horloge
        CONSTANT period:TIME          := 20 Ns;
        CONSTANT duty_cycle:REAL      := 0.5;
        CONSTANT phaseLag:TIME        := 5 Ns;
        CONSTANT on_time:TIME         := period * duty_cycle;
        CONSTANT off_time:TIME        := period - on_time;
        --      Delais des multiplexeurs
        CONSTANT DelayMux2011:TIME:=1.2 Ns;
        CONSTANT DelayMux5011:TIME:=1.2 Ns;
        CONSTANT DelayMux2051:TIME:=1.2 Ns;
        CONSTANT DelayMux2071:TIME:=1.2 Ns;
        CONSTANT DelayMux3071:TIME:=1.2 Ns;
```

```
        CONSTANT DelayMux2161:TIME:=1.2 Ns;
        CONSTANT DelayMux3161:TIME:=1.2 Ns;
        CONSTANT DelayDemux1015:TIME:=1.2 Ns;
        CONSTANT DelayDemux1075:TIME:=1.2 Ns;
        CONSTANT DelayDemux2163:TIME:=1.2 Ns;
        --      Delais dans les registres
        CONSTANT DelayReg64:TIME:=1.2 Ns;
        CONSTANT DelayReg16:TIME:=1.2 Ns;
        CONSTANT DelayReg7:TIME:=1.2 Ns;
        CONSTANT DelayReg5:TIME:=1.2 Ns;
        --              Delais des décaleurs
        CONSTANT DelayShift1:TIME:=1.2 Ns;
        CONSTANT DelayShift2:TIME:=1.2 Ns;
        --              Delais dans les circuits speciaux
        CONSTANT DelayClkRetour:TIME:=1.2 Ns;
        CONSTANT DelaySelMux:TIME:=1.2 Ns;
        CONSTANT DelayAddition5:TIME:=1.2 Ns;
        CONSTANT DelayAddition7:TIME:=1.2 Ns;
        CONSTANT DelayAddition16:TIME:=1.2 Ns;
        CONSTANT DelayIndic0_7bits:TIME:=1.2 Ns;
        --              Delais dans les memoires
        CONSTANT WriteDelay32:TIME:=3 Ns;
        CONSTANT ReadDelay32:TIME:=3 Ns;
        CONSTANT WriteDelay128:TIME:=4 Ns;
        CONSTANT ReadDelay128:TIME:=4 Ns;
        CONSTANT WriteDelayMicro:TIME:=5 Ns;
        CONSTANT ReadDelayMicro:TIME:=5 Ns;
        CONSTANT DelayBuffer16:TIME:=1.2 NS;
        CONSTANT DelayBuffer64:TIME:=1.2 NS;
END delays;

------------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                  _ Toutes les memoires
--                  _ Un package qui les reunit toutes
--
------------------------------------------------------------------------


------------------------------------------------------------------
--      Package definissant les memoires et les buffers
```

```
--      de memoire.
--
--      Ce sont :
--
--      * Ram32_16, Ram128_16, Ram128_64
--      * buffer16, buffer64
--
--      Les structures de ceux-ci sont en fin de fichier
--
library IEEE;
use IEEE.std_logic_1164.all;

PACKAGE Memoires IS

        TYPE t_ram_dataK IS ARRAY(127 downto 0) of std_logic_vector(15 downto 0);
        TYPE t_ram_dataP IS ARRAY(31 downto 0) of std_logic_vector(15 downto 0);
        TYPE t_ram_dataM IS ARRAY(31 downto 0) of std_logic_vector(63 downto 0);
        --      Parametres des memoires

        CONSTANT WordLength:Integer:=16;
        CONSTANT WordLengthMicro:Integer:=64;

        COMPONENT Ram32_16 PORT
                (adresse:IN std_logic_vector(4 downto 0);
                data:INOUT std_logic_vector(15 downto 0);
                read, write:IN std_logic);
        END COMPONENT;
        COMPONENT Ram128_16 PORT
                (adresse:IN std_logic_vector(6 downto 0);
                data:INOUT std_logic_vector(15 downto 0);
                read, write:IN std_logic);
        END COMPONENT;
        COMPONENT Ram128_64 PORT
                (adresse:IN std_logic_vector(6 downto 0);
                data:INOUT std_logic_vector(63 downto 0);
                read, write:IN std_logic);
        END COMPONENT;
        COMPONENT Buffer16 PORT
                (in0:IN std_logic_vector(15 downto 0);
                out0:OUT std_logic_vector(15 downto 0);
                en:IN std_logic);
        END COMPONENT;
        COMPONENT Buffer64 PORT
                (in0:IN std_logic_vector(63 downto 0);
                out0:OUT std_logic_vector(63 downto 0);
```

```
                en:IN std_logic);
        END COMPONENT;
END Memoires;

---------- Les memoires  ----------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.Memoires.all;
use work.circuits.all;
use work.delays.all;
        ENTITY Ram32_16 IS
        PORT
                (adresse:IN std_logic_vector(4 downto 0);
                data:INOUT std_logic_vector(15 downto 0);
                read, write:IN std_logic);
        END Ram32_16;
        ARCHITECTURE behave OF Ram32_16 IS

        BEGIN
                main_proc : PROCESS (adresse, data, read, write)
                        VARIABLE ram_data : t_ram_dataP;
                        VARIABLE ram_init : BOOLEAN := FALSE;
                BEGIN
                ------- Initialisation de la memoire  ------------
                IF NOT (ram_init) THEN
                        FOR i IN ram_data'LOW TO ram_data'HIGH
                                LOOP
                                        ram_data(i) :=  "0000000000000000";
                                        ram_data(i) :=  "0000000000000000";
                                END LOOP;
                        ram_init := TRUE;
                END IF;
                -------    Fonctionnement    -----------
                IF (write'EVENT and write='1') THEN
                        ram_data( Bit5_to_Int(adresse) ) := data; -- AFTER
WriteDelayMicro;
                END IF;
                IF (read'EVENT and read='1') THEN
                        data <= ram_data( Bit5_to_Int(adresse) ); --AFTER
ReadDelayMicro;
                END IF;
                END PROCESS main_proc;
        END behave;

library IEEE;
```

```
use IEEE.std_logic_1164.all;
use work.Memoires.all;
use work.circuits.all;
use work.delays.all;
        ENTITY Ram128_16 IS
        PORT
                (adresse:IN std_logic_vector(6 downto 0);
                data:INOUT std_logic_vector(15 downto 0);
                read, write:IN std_logic);
        END Ram128_16;
        ARCHITECTURE behave OF Ram128_16 IS
        BEGIN
                main_proc : PROCESS (adresse, data, read, write)
                        VARIABLE ram_data : t_ram_dataK;
                        VARIABLE ram_init : BOOLEAN := FALSE;
                BEGIN
                ----   Initialisation de la memoire   ------------
                IF NOT (ram_init) THEN
                        FOR i IN ram_data'LOW TO ram_data'HIGH
                                LOOP
                                        ram_data(i) :=  "0000000000000000";
                                        ram_data(i) :=  "0000000000000000";
                                END LOOP;
                        ram_init := TRUE;
                END IF;
                -------    Fonctionnement     -----------
                IF (write'EVENT and write='1') THEN
                        ram_data( Bit7_to_Int(adresse) ) := data; -- AFTER WriteDelayK;
                END IF;
                IF (read'EVENT and read='1') THEN
                        data <= ram_data( Bit7_to_Int(adresse) ); -- AFTER ReadDelayK;
                END IF;
                END PROCESS main_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.Memoires.all;
use work.circuits.all;
use work.delays.all;
        ENTITY Ram128_64 IS
        PORT
                (adresse:IN std_logic_vector(6 downto 0);
                data:INOUT std_logic_vector(63 downto 0);
                read, write:IN std_logic);
```

```
        END Ram128_64;
        ARCHITECTURE behave OF Ram128_64 IS
        BEGIN
                main_proc : PROCESS (adresse, data, read, write)
                        VARIABLE ram_data : t_ram_dataM;
                        VARIABLE ram_init : BOOLEAN := FALSE;
                BEGIN
                ------- Initialisation de la memoire    ------------
                IF NOT (ram_init) THEN
                        FOR i IN ram_data'LOW TO ram_data'HIGH
                                LOOP
                                        ram_data(i) :=
"00000000000000000000000000000000000000000000000000000000000000000000";
                                        ram_data(i) :=
"00000000000000000000000000000000000000000000000000000000000000000000";
                                END LOOP;
                        ram_init := TRUE;
                END IF;
                -------     Fonctionnement      ------------
                IF (write'EVENT and write='1') THEN
                        ram_data( Bit7_to_Int(adresse) ) := data; -- AFTER
WriteDelayMicro;
                END IF;
                IF (read'EVENT and read='1') THEN
                        data <= ram_data( Bit7_to_Int(adresse) ); -- AFTER
ReadDelayMicro;
                END IF;
                END PROCESS main_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.Memoires.all;
use work.circuits.all;
use work.delays.all;
        ENTITY Buffer16 IS
        PORT
                (in0: IN std_logic_vector(15 downto 0);
                en: IN std_logic;
                out0: OUT std_logic_vector(15 downto 0));
        END Buffer16;
        ARCHITECTURE behave OF Buffer16 IS
        BEGIN
                Buffer16_proc : PROCESS(en)
                VARIABLE BufferInit :BOOLEAN := FALSE;
```

```
                BEGIN
                        IF BufferInit = FALSE THEN
                                out0 <= "ZZZZZZZZZZZZZZZZ";
                                BufferInit := TRUE;
                        END IF;
                        IF en='1' THEN
                                out0 <= in0 AFTER DelayBuffer16;
                        ELSE
                                out0 <= "ZZZZZZZZZZZZZZZZ" AFTER DelayBuffer16;
                        END IF;
                END PROCESS Buffer16_proc;
        END behave;

library IEEE;
use IEEE.std_logic_1164.all;
use work.Memoires.all;
use work.circuits.all;
use work.delays.all;
        ENTITY Buffer64 IS
        PORT
                (in0: IN std_logic_vector(63 downto 0);
                en: IN std_logic;
                out0: OUT std_logic_vector(63 downto 0));
        END Buffer64;
        ARCHITECTURE behave OF Buffer64 IS
        BEGIN
                Buffer64_proc : PROCESS(en)
                VARIABLE BufferInit :BOOLEAN := FALSE;
                BEGIN
                        IF BufferInit = FALSE THEN
                                out0 <=
"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZ";
                                BufferInit := TRUE;
                        END IF;
                        IF en='1' THEN
                                out0 <= in0 AFTER DelayBuffer16;
                        ELSE
                                out0 <=
"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZ";
                        END IF;
                END PROCESS Buffer64_proc;
        END behave;
```

```
--------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                  _ Tous les multiplexeurs et leurs architecture
--                  _ Un package qui les reunit tous
--
--------------------------------------------------------------------

library IEEE;
use work.delays.all;
use IEEE.std_logic_1164.all;
PACKAGE muxdemux IS
        COMPONENT mux2011 PORT
                (in0,in1:IN std_logic;
                Selection:IN std_logic;
                out0:OUT std_logic);
        END COMPONENT;
        COMPONENT mux2051 PORT
                (in0,in1:IN std_logic_vector(4 downto 0);
                Selection:IN std_logic;
                out0:OUT std_logic_vector(4 downto 0));
        END COMPONENT;
        COMPONENT mux2071 PORT
                (in0,in1:IN std_logic_vector(6 downto 0);
                Selection:IN std_logic;
                out0:OUT std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT mux2161 PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                Selection:IN std_logic;
                out0:OUT std_logic_vector(15 downto 0));
        END COMPONENT;
        COMPONENT mux3071 PORT
                (in0,in1,In2:IN std_logic_vector(6 downto 0);
                Selection:IN std_logic_vector(1 downto 0);
                out0:OUT std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT mux3161 PORT
                (in0,in1,In2:IN std_logic_vector(15 downto 0);
                Selection:IN std_logic_vector(1 downto 0);
                out0:OUT std_logic_vector(15 downto 0));
        END COMPONENT;
```

```
        COMPONENT mux5011 PORT
                (in0,in1,In2,in3,in4:IN std_logic;
                Selection:IN std_logic_vector(2 downto 0);
                out0:OUT std_logic);
        END COMPONENT;
        COMPONENT demux2163 PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                Selection:IN std_logic_vector(1 downto 0);
                out0,out1,out2:OUT std_logic_vector(15 downto 0));
        END COMPONENT;
        COMPONENT demux1015 PORT
                (in0:IN std_logic;
                Selection:IN std_logic_vector(2 downto 0);
                out0,out1,out2,out3,out4:OUT std_logic);
        END COMPONENT;
        COMPONENT demux1075 PORT
                (in0,in1:IN std_logic_vector(6 downto 0);
                Selection:IN std_logic_vector(2 downto 0);
                out0,out1,out2,out3,out4:OUT std_logic_vector(6 downto 0));
        END COMPONENT;

END muxdemux;



----------      Les multiplexeurs et demux ----------
library IEEE;
use IEEE.std_logic_1164.all;
        use work.delays.all;
        ENTITY mux2011 IS
                PORT
                (in0,in1:IN std_logic;
                Select0:IN std_logic;
                out0:OUT std_logic);
        END Mux2011;
        ARCHITECTURE behave OF Mux2011 IS
        BEGIN
                mux2011_proc : PROCESS(in0,in1,Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN '0'=>
                                out0<=in0 AFTER DelayMux2011;
                        WHEN '1' =>
                                out0<=in1 AFTER DelayMux2011;
                        WHEN OTHERS=>
```

```
                END CASE;
            END PROCESS mux2011_proc;
        END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux2051 IS
            PORT
            (in0,in1:IN std_logic_vector(4 downto 0);
            Select0:IN std_logic;
            out0:OUT std_logic_vector(4 downto 0));
        END mux2051;
        ARCHITECTURE behave OF Mux2051 IS
        BEGIN
            mux2051_proc : PROCESS(in0,in1,Select0)
            BEGIN
            CASE Select0 IS
            WHEN '0' =>
                out0<=in0 AFTER DelayMux2051;
                WHEN '1' =>
                out0<=in1 AFTER DelayMux2051;
                WHEN OTHERS=>
                END CASE;
                END PROCESS mux2051_proc;


                                END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux2071 IS
            PORT
            (in0,in1:IN std_logic_vector(6 downto 0);
            Select0:IN std_logic;
            out0:OUT std_logic_vector(6 downto 0));
        END mux2071;
        ARCHITECTURE behave OF Mux2071 IS
        BEGIN
            mux2071_proc : PROCESS(in0,in1,Select0)
            BEGIN
                CASE Select0 IS
                WHEN '0' =>
                    out0<=in0 AFTER DelayMux2071;
```

```
                WHEN '1' =>
                        out0<=in1 AFTER DelayMux2071;
                WHEN OTHERS=>
                END CASE;
        END PROCESS mux2071_proc;
END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux2161 IS
        PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                Select0:IN std_logic;
                out0:OUT std_logic_vector(15 downto 0));
        END mux2161;
        ARCHITECTURE behave OF Mux2161 IS
        BEGIN
                mux2161_proc : PROCESS(in0,in1,Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN '0' =>
                                out0<=in0 AFTER DelayMux2161;
                        WHEN '1' =>
                                out0<=in1 AFTER DelayMux2161;
                        WHEN OTHERS=>
                        END CASE;
                END PROCESS mux2161_proc;
        END behave;


use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux3071 IS
        PORT
                (in0,in1,in2:IN std_logic_vector(6 downto 0);
                Select0: IN std_logic_vector(1 downto 0);
                out0:OUT std_logic_vector(6 downto 0));
        END mux3071;
        ARCHITECTURE behave OF Mux3071 IS
        BEGIN
                mux3071_proc : PROCESS(in0,in1,in2,Select0)
                BEGIN
                        CASE Select0 IS
```

```
                    WHEN  "00" =>          -- ligne 183
                        out0<=in0 AFTER DelayMux3071;
                    WHEN "01" =>
                        out0<=in1 AFTER DelayMux3071;
                    WHEN "10" =>
                        out0<=in2 AFTER DelayMux3071;
                    WHEN OTHERS=> NULL;
                    END CASE;
                END PROCESS mux3071_proc;
        END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux3161 IS
        PORT
                (in0,in1,in2:IN std_logic_vector(15 downto 0);
                Select0:IN std_logic_vector(1 downto 0);
                out0:OUT std_logic_vector(15 downto 0));
        END mux3161;
        ARCHITECTURE behave OF Mux3161 IS
        BEGIN
                mux3161_proc : PROCESS(in0,in1,in2,Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN  "00" =>
                                out0<=in0 AFTER DelayMux3161;
                        WHEN  "01" =>
                                out0<=in1 AFTER DelayMux3161;
                        WHEN  "10"  =>
                                out0<=in2 AFTER DelayMux3161;
                        WHEN OTHERS=>
                        END CASE;
                END PROCESS mux3161_proc;
        END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY mux5011 IS
        PORT
                (in0,in1,in2,in3,in4:IN std_logic;
                Select0:IN std_logic_vector(2 downto 0);
                out0:OUT std_logic);
        END mux5011;
```

```
        ARCHITECTURE behave OF Mux5011 IS
        BEGIN
                mux5011_proc : PROCESS(in0,in1,in2,in3,in4,Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN "000" =>
                                out0<=in0 AFTER DelayMux5011;
                        WHEN "001" =>
                                out0<=in1 AFTER DelayMux5011;
                        WHEN "010" =>
                                out0<=in2 AFTER DelayMux5011;
                        WHEN "011" =>
                                out0<=in2 AFTER DelayMux5011;
                        WHEN "100" =>
                                out0<=in2 AFTER DelayMux5011;
                        WHEN OTHERS=>
                        END CASE;
                END PROCESS mux5011_proc;
        END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY demux2163 IS
        PORT
                (in0,in1:IN std_logic_vector(15 downto 0);
                Select0:IN std_logic_vector(1 downto 0);
                out0,out1,out2:OUT std_logic_vector(15 downto 0));
        END demux2163;
        ARCHITECTURE behave OF demux2163 IS
        BEGIN
                demux2163_proc : PROCESS(in0,in1,Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN "00"  =>
                                out0<=in1 AFTER DelayDemux2163;
                        WHEN "01"  =>
                                out1<=in1 AFTER DelayDemux2163;
                        WHEN "10"  =>
                                out2<=in1 AFTER DelayDemux2163;
                        WHEN "11"  =>
                                out0<=in0 AFTER DelayDemux2163;
                        WHEN OTHERS=>
                        END CASE;
        END PROCESS demux2163_proc;
```

```
        END behave;

use work.buses.all;
use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
ENTITY demux1015 IS
        PORT
                (in0:IN std_logic;
                Select0:IN std_logic_vector(2 downto 0);
                out0,out1,out2,out3,out4:OUT std_logic);
        END demux1015;
        ARCHITECTURE behave OF demux1015 IS
        BEGIN
                demux1015_proc : PROCESS(in0, Select0)
                BEGIN
                        CASE Select0 IS
                        WHEN "000" =>
                                out0<=in0 AFTER DelayDemux1015;
                        WHEN "001" =>
                                out1<=in0 AFTER DelayDemux1015;
                        WHEN "010" =>
                                out2<=in0 AFTER DelayDemux1015;
                        WHEN "011" =>
                                out3<=in0 AFTER DelayDemux1015;
                        WHEN "100" =>
                                out4<=in0 AFTER DelayDemux1015;
                        WHEN OTHERS=>
                        END CASE;
                END PROCESS demux1015_proc;
        END behave;

use work.delays.all;
library IEEE;
use IEEE.std_logic_1164.all;
        ENTITY demux1075 IS
        PORT
                (in0,in1:IN std_logic_vector(6 downto 0);
                Select0:IN std_logic_vector(2 downto 0);
                out0,out1,out2,out3,out4:OUT std_logic_vector(6 downto 0));
        END demux1075;
        ARCHITECTURE behave OF demux1075 IS
        BEGIN
                demux1075_proc : PROCESS(in0, Select0)
                BEGIN
```

```
                CASE Select0 IS
                WHEN "000"   =>
                        out0<=in0 AFTER DelayDemux1075;
                WHEN "001"   =>
                        out1<=in0 AFTER DelayDemux1075;
                WHEN "010"   =>
                        out2<=in0 AFTER DelayDemux1075;
                WHEN "011"   =>
                        out3<=in0 AFTER DelayDemux1075;
                WHEN "100"   =>
                        out4<=in0 AFTER DelayDemux1075;
                WHEN OTHERS=>
                END CASE;
                END PROCESS demux1075_proc;
        END behave;


-------------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                  _ Tous les registres et leurs architectures
--                  _ Un package qui les reunit tous
--
-------------------------------------------------------------------------


-------------------------------------------------------------------------
------- Package specifiant les types de registres ----------------
--
--    Les structures de ceux-ci sont en fin de fichier
--

library IEEE;
use IEEE.std_logic_1164.all;
PACKAGE registres IS
        COMPONENT registre64 PORT
                (clk:IN  std_logic;
                in0:IN  std_logic_vector(63 downto 0);
                out0:OUT  std_logic_vector(63 downto 0));
        END COMPONENT;
        COMPONENT registre16 PORT
                (clk:IN  std_logic;
```

```
            in0:IN  std_logic_vector(15 downto 0);
            out0:OUT  std_logic_vector(15 downto 0));
        END COMPONENT;
        COMPONENT registre7 PORT
            (clk:IN  std_logic;
            in0:IN  std_logic_vector(6 downto 0);
            out0:OUT  std_logic_vector(6 downto 0));
        END COMPONENT;
        COMPONENT registre5 PORT
            (clk:IN  std_logic;
            in0:IN  std_logic_vector(4 downto 0);
            out0:OUT  std_logic_vector(4 downto 0));
        END COMPONENT;
END registres;


---------- Les registres et leurs architectures ----------
 library IEEE;
use IEEE.std_logic_1164.all;
 use work.delays.all;
        ENTITY registre64 IS
        PORT
            (clk:IN  std_logic;
            in0:IN  std_logic_vector(63 downto 0);
            out0:OUT  std_logic_vector(63 downto 0));
        END registre64;
        ARCHITECTURE behave OF registre64 IS
        BEGIN
            registre64_proc : PROCESS
            BEGIN
                WAIT UNTIL (clk'EVENT and clk = '0');
                out0 <= in0 AFTER DelayReg64;
            END PROCESS registre64_proc;
        END behave;
 library IEEE;
use IEEE.std_logic_1164.all;
 use work.delays.all;
        ENTITY registre16 IS
        PORT
            (clk:IN  std_logic;
            in0:IN  std_logic_vector(15 downto 0);
            out0:OUT  std_logic_vector(15 downto 0));
        END registre16;
        ARCHITECTURE behave OF registre16 IS
        BEGIN
```

```
                registre16_proc : PROCESS
                BEGIN
                        WAIT UNTIL (clk'EVENT and clk = '0');
                        out0 <= in0 AFTER DelayReg16;
                END PROCESS registre16_proc;
        END behave;
 library IEEE;
use IEEE.std_logic_1164.all;
 use work.delays.all;
        ENTITY registre7 IS
        PORT
                (clk:IN  std_logic;
                in0:IN  std_logic_vector(6 downto 0);
                out0:OUT  std_logic_vector(6 downto 0));
        END registre7;
        ARCHITECTURE behave OF registre7 IS
        BEGIN
                registre7_proc : PROCESS
                BEGIN
                        WAIT UNTIL (clk'EVENT and clk = '0');
                        out0 <= in0 AFTER DelayReg7;
                END PROCESS registre7_proc;
        END behave;
 library IEEE;
use IEEE.std_logic_1164.all;
 use work.delays.all;
        ENTITY registre5 IS
        PORT
                (clk:IN  std_logic;
                in0:IN  std_logic_vector(4 downto 0);
                out0:OUT  std_logic_vector(4 downto 0));
        END registre5;
        ARCHITECTURE behave OF registre5 IS
        BEGIN
                registre5_proc : PROCESS
                BEGIN
                        WAIT UNTIL (clk'EVENT and clk = '0');
                        out0 <= in0 AFTER DelayReg5;
                END PROCESS registre5_proc;
        END behave;


--------------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
```

```
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                  _ Toutes les grands composants du sequenceur
--                      et leurs architectures
--                  _ Un package qui les reunit tous
--
------------------------------------------------------------------------


------------------------------------------------------
--          Package definissant les super-blocs utilises par
--          le top-level-entity.
--
library IEEE;
use IEEE.std_logic_1164.all;
PACKAGE topcomp IS

        COMPONENT Microsequenceur
                PORT(   MicroMot        :IN std_logic_vector(63 downto 0);
                        WriteM          :IN std_logic;
                        AdresseExterne  :IN std_logic_vector(6 downto 0);
                        Load            :IN std_logic;
                        compt1_0        :IN std_logic;
                        compt2_0        :IN std_logic;
                        Reset_Micro     :IN std_logic;
                        Hold            :IN std_logic;
                        MicroCommande   :OUT std_logic_vector(51 downto 0);
                        Fi_1            :IN std_logic;
                        Fi_1b           :IN std_logic;
                        Fi_2            :IN std_logic;
                        Fi_2b           :IN std_logic);
        END COMPONENT;

        COMPONENT BlocCompteurUGA
                PORT(   SelMuxCompteur1 :IN std_logic;
                        SelMuxCompteur2 :IN std_logic;
                        ClkCompteur1    :IN std_logic;
                        ClkCompteur2    :IN std_logic;
                        RB_I            :IN std_logic;
                        RD_I            :IN std_logic;
                        SelMuxPtr_I     :IN std_logic;
                        SelMuxPtrPgmI   :IN std_logic;
                        RB1_Z           :IN std_logic;
                        RB2_Z           :IN std_logic;
```

```
                RD1_Z          :IN std_logic;
                RD2_Z          :IN std_logic;
                SelMuxPtr_1_Z  :IN std_logic;
                SelMuxPtr_2_Z  :IN std_logic;
                ClkMAR_I       :IN std_logic;
                ClkMAR_Z       :IN std_logic;
                SelMuxPtrPgm_Z :IN std_logic_vector(1 downto 0);
                AdresseInterneK :OUT std_logic_vector(6 downto 0);
                AdresseInterneH :OUT std_logic_vector(6 downto 0);
                AdresseInterneI :OUT std_logic_vector(4 downto 0);
                AdresseInterneZ :OUT std_logic_vector(6 downto 0);
                Indic_C1_0     :OUT std_logic;
                Indic_C2_0     :OUT std_logic);
END COMPONENT;

COMPONENT BlocMemoire
        PORT(  AdresseK       :IN std_logic_vector(6 downto 0);
                AdresseH       :IN std_logic_vector(6 downto 0);
                AdresseI       :IN std_logic_vector(4 downto 0);
                AdresseZ       :IN std_logic_vector(6 downto 0);
                WriteK         :IN std_logic;
                WriteH         :IN std_logic;
                WriteI         :IN std_logic;
                WriteZ         :IN std_logic;
                ReadK          :IN std_logic;
                ReadH          :IN std_logic;
                ReadI          :IN std_logic;
                ReadZ          :IN std_logic;
                Load           :IN std_logic;
                Cde_DemuxI     :IN std_logic_vector(1 downto 0);
                ClkPipeline    :IN std_logic;
                ClkRegShift1   :IN std_logic;
                ClkRegShift2   :IN std_logic;
                IN_Bus         :IN std_logic_vector(15 downto 0);
                ALU_Bus        :IN std_logic_vector(15 downto 0);
                Out_K          :OUT std_logic_vector(15 downto 0);
                Out_H          :OUT std_logic_vector(15 downto 0);
                Out_I          :OUT std_logic_vector(15 downto 0);
                Out_Z          :OUT std_logic_vector(15 downto 0);
                Out_Shift1     :OUT std_logic_vector(15 downto 0);
                Out_Shift2     :OUT std_logic_vector(15 downto 0));
END COMPONENT;

COMPONENT ALU
        PORT(  In_K           :IN std_logic_vector(15 downto 0);
```

```
                In_H          :IN std_logic_vector(15 downto 0);
                In_I          :IN std_logic_vector(15 downto 0);
                In_Z          :IN std_logic_vector(15 downto 0);
                In_S1         :IN std_logic_vector(15 downto 0);
                In_S2         :IN std_logic_vector(15 downto 0);
                ClkPipeline   :IN std_logic;
                ResetPipeline :IN std_logic;
                SelectMuxALU  :IN std_logic_vector(1 downto 0);
                SelectShift2  :IN std_logic_vector(1 downto 0);
                ClkRegALU     :IN std_logic;
                ALU_Out       :OUT std_logic_vector(15 downto 0));
END COMPONENT;

COMPONENT IOBuffer
        PORT(   Load          :IN std_logic;
                Clock_Load    :IN std_logic;
                ClkRegIn_Micro :IN std_logic;
                Data          :INOUT std_logic_vector(15 downto 0);
                BusALU        :IN std_logic_vector(15 downto 0);
                MicroMot      :OUT std_logic_vector(63 downto 0);
                INBus         :OUT std_logic_vector(15 downto 0));
END COMPONENT;

COMPONENT Pilotage_Memoire
        PORT(   Load          :IN std_logic;
                SelectMemory  :IN std_logic_vector(2 downto 0);
                AdresseExterne :IN std_logic_vector(6 downto 0);
                AdresseInterneK :IN std_logic_vector(6 downto 0);
                AdresseInterneH :IN std_logic_vector(6 downto 0);
                AdresseInterneI :IN std_logic_vector(4 downto 0);
                AdresseInterneZ :IN std_logic_vector(6 downto 0);
                WriteExterne  :IN std_logic;
                WriteInterneI :IN std_logic;
                WriteInterneZ :IN std_logic;
                WriteK        :OUT std_logic;
                WriteH        :OUT std_logic;
                WriteI        :OUT std_logic;
                WriteZ        :OUT std_logic;
                WriteM        :OUT std_logic;
                AdresseK      :OUT std_logic_vector(6 downto 0);
                AdresseH      :OUT std_logic_vector(6 downto 0);
                AdresseI      :OUT std_logic_vector(4 downto 0);
                AdresseZ      :OUT std_logic_vector(6 downto 0);
                AdresseM      :OUT std_logic_vector(6 downto 0));
END COMPONENT;
```

```
END topcomp;

------------------------------------------------------------
--    Declaration des entites constituant l'architecture
--    du processeur et leurs architecture.
--          ( Ces composants sont dans le package TopComp )
--
--          Ces entites sont :
--
--    * Le microsequenceur
--    * le bloc des compteurs et UGA
--    * Le bloc memoire
--    * Les registres d'entrees sorties ( IObuffer )
--    * Le bloc de pilotage interne\externe des memoires
------------------------------------------------------------

---------------- Entite microsequenceur ---------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.registres.all;
use work.muxdemux.all;
use work.circuits.all;
use work.Memoires.all;
use work.vddgnd.all;
use work.vddgnd.all;
ENTITY Microsequenceur IS
                PORT(   MicroMot       :IN std_logic_vector(63 downto 0);
                        WriteM         :IN std_logic;
                        AdresseExterne  :IN std_logic_vector(6 downto 0);
                        Load           :IN std_logic;
                        compt1_0       :IN std_logic;
                        compt2_0       :IN std_logic;
                        Reset_Micro    :IN std_logic;
                        Hold           :IN std_logic;
                        MicroCommande   :INOUT std_logic_vector(63 downto 0);
                        Fi_1           :IN std_logic;
                        Fi_1b          :IN std_logic;
                        Fi_2           :IN std_logic;
                        Fi_2b          :IN std_logic);
        END Microsequenceur;
ARCHITECTURE MicrosequenceurArch OF Microsequenceur IS
signal MemData, DataOut: std_logic_vector(63 downto 0);
signal MuxCondOut, ClkRetour: std_logic;
signal SelMuxAdr: std_logic_vector(1 downto 0);
```

```
signal AdresseRetour, AdressePC, MuxAdrOut,RegAdrOut,AdresseMicro:
std_logic_vector(6 downto 0);
signal pc_add : std_logic_vector(6 downto 0) := "0000001";
BEGIN
        Memoire : Ram128_64
        PORT MAP (AdresseMicro, MemData, Fi_2, WriteM);
        BufferIn: Buffer64
        PORT MAP (MicroMot, MemData, Fi_2);
        BufferOut: Buffer64
        PORT MAP (MemData, DataOut, Fi_2);
        RegistreMicroCommande: registre64
        PORT MAP (Fi_1, DataOut, MicroCommande);
        MuxDeCondition: Mux5011
        PORT MAP(Hold, Reset_Micro, Compt1_0, Compt2_0, gnd, MicroCommande(63
downto 61), MuxCondOut);
        PolarityControl:PolarLogic
        PORT MAP (MuxCondOut, MicroCommande(60), MicroCommande(59),
ClkRetour, SelMuxAdr);
        MuxAdressage: Mux3071
        PORT MAP (MicroCommande(58 downto 52), AdresseRetour, AdressePC,
SelMuxAdr, MuxAdrOut);
        RegistreAdressage: registre7
        PORT MAP (Fi_1b, MuxAdrOut, RegAdrOut);
        MuxAdrIntExt: Mux2071
        PORT MAP (AdresseExterne, RegAdrOut, Load, AdresseMicro);
        AdditionneurPC: addition7
        PORT MAP (RegAdrOut, pc_add, AdressePC);
        RegRetour: registre7
        PORT MAP (ClkRetour, AdressePC, AdresseRetour);
 END MicrosequenceurArch;


--------- Entite Bloc des compteurs et UGA ----------
--
--      Ce bloc contient quatre sous-groupe
--      UGA-I
--      UGA-Z
--      Compteur1
--      Compteur2
--      Compteur d'echantillons
--

library IEEE;
use IEEE.std_logic_1164.all;
use work.registres.all;
```

```
use work.muxdemux.all;
use work.circuits.all;
ENTITY BlocCompteurUGA IS
        PORT(  SelMuxCompteur1 :IN std_logic;
               SelMuxCompteur2 :IN std_logic;
               ClkCompteur1    :IN std_logic;
               ClkCompteur2    :IN std_logic;
               RB_I            :IN std_logic;
               RD_I            :IN std_logic;
               SelMuxPtr_I     :IN std_logic;
               SelMuxPtrPgmI   :IN std_logic;
               RB1_Z           :IN std_logic;
               RB2_Z           :IN std_logic;
               RD1_Z           :IN std_logic;
               RD2_Z           :IN std_logic;
               SelMuxPtr_1_Z   :IN std_logic;
               SelMuxPtr_2_Z   :IN std_logic;
               ClkMAR_I        :IN std_logic;
               ClkMAR_Z        :IN std_logic;
               AdressePgm      :IN std_logic_vector(6 downto 0);
               SelMuxPtrPgm_Z  :IN std_logic_vector(1 downto 0);
               AdresseInterneK :OUT std_logic_vector(6 downto 0);
               AdresseInterneH :OUT std_logic_vector(6 downto 0);
               AdresseInterneI :OUT std_logic_vector(4 downto 0);
               AdresseInterneZ :OUT std_logic_vector(6 downto 0);
               Indic_C1_0      :OUT std_logic;
               Indic_C2_0      :OUT std_logic);
    END BlocCompteurUGA;
ARCHITECTURE BlocCompteurUGAArch OF BlocCompteurUGA IS
BEGIN
    UGA_I:UGA_1_pointeur
    PORT MAP(AdressePgm(4 downto 0), SelMuxPtr_I, RB_I, RD_I, SelMuxPtrPgmI,
ClkMAR_I, AdresseInterneI);
    UGA_Z:UGA_2_pointeur
    PORT MAP(AdressePgm, SelMuxPtr_1_Z, RB1_Z, RD1_Z, SelMuxPtr_2_Z,
RB2_Z, RD2_Z, SelMuxPtrPgm_Z, ClkMAR_Z, AdresseInterneZ);
    Compteur1:Compteur12
    PORT MAP(AdressePgm, SelMuxCompteur1, ClkCompteur1, Indic_C1_0,
AdresseInterneK);
    Compteur2:Compteur12
    PORT MAP(AdressePgm, SelMuxCompteur2, ClkCompteur2, Indic_C2_0,
AdresseInterneH);
END BlocCompteurUGAArch;

---------- Entite BlocMemoire ----------
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.registres.all;
use work.muxdemux.all;
use work.Memoires.all;
ENTITY BlocMemoire IS
        PORT(  AdresseK      :IN std_logic_vector(6 downto 0);
               AdresseH      :IN std_logic_vector(6 downto 0);
               AdresseI      :IN std_logic_vector(4 downto 0);
               AdresseZ      :IN std_logic_vector(6 downto 0);
               WriteK        :IN std_logic;
               WriteH        :IN std_logic;
               WriteI        :IN std_logic;
               WriteZ        :IN std_logic;
               ReadK         :IN std_logic;
               ReadH         :IN std_logic;
               ReadI         :IN std_logic;
               ReadZ         :IN std_logic;
               Load          :IN std_logic;
               Cde_DemuxI    :IN std_logic_vector(1 downto 0);
               ClkPipeline   :IN std_logic;
               ClkRegShift1  :IN std_logic;
               ClkRegShift2  :IN std_logic;
               IN_Bus        :IN std_logic_vector(15 downto 0);
               ALU_Bus       :IN std_logic_vector(15 downto 0);
               Out_K         :OUT std_logic_vector(15 downto 0);
               Out_H         :OUT std_logic_vector(15 downto 0);
               Out_I         :OUT std_logic_vector(15 downto 0);
               Out_Z         :OUT std_logic_vector(15 downto 0);
               Out_Shift1    :OUT std_logic_vector(15 downto 0);
               Out_Shift2    :OUT std_logic_vector(15 downto 0));
END BlocMemoire;
ARCHITECTURE BlocMemoireArch OF BlocMemoire IS
SIGNAL MemKData, MemHData,  MemIData, MemZData: std_logic_vector(15 downto
0);
SIGNAL BufKOut,  BufHOut, BufIOut, BufZOut: std_logic_vector(15 downto 0);
SIGNAL MuxIOut, MuxZOut: std_logic_vector(15 downto 0);
SIGNAL RegI, RegS1, RegS2:std_logic_vector(15 downto 0);
BEGIN
        Mem_K: Ram128_16
        PORT MAP (AdresseK,MemKData,ReadK,WriteK);
        Mem_H: Ram128_16
        PORT MAP (AdresseH,MemHData,ReadH,WriteH);
        Mem_I: Ram32_16
        PORT MAP (AdresseI,MemIData,ReadI,WriteI);
```

```
        Mem_Z: Ram128_16
        PORT MAP (AdresseZ,MemZData,ReadZ,WriteZ);

        BufferOutK: Buffer16 PORT MAP (MemKData, BufKOut,ReadK);
        BufferInK: Buffer16 PORT MAP (IN_Bus,MemKData,WriteK);
        BufferOutH: Buffer16 PORT MAP (MemHData, BufHOut,ReadH);
        BufferInH: Buffer16 PORT MAP (IN_Bus,MemHData,WriteH);
        BufferOutI: Buffer16 PORT MAP (MemIData, BufIOut,ReadI);
        BufferInI: Buffer16 PORT MAP (MuxIOut,MemIData,WriteI);
        BufferOutZ: Buffer16 PORT MAP (MemZData, BufZOut,ReadZ);
        BufferInZ: Buffer16 PORT MAP (MuxZOut,MemZData,WriteZ);

        MuxI: Mux2161 PORT MAP (IN_Bus, ALU_Bus, Load, MuxIOut);
        MuxZ: Mux2161 PORT MAP (IN_Bus, ALU_Bus, Load, MuxZOut);
        DemuxI: Demux2163 PORT MAP (IN_Bus, ALU_Bus, Cde_DemuxI,RegI,RegS1,
RegS2);

        RegistreK: registre16 PORT MAP (ClkPipeline, BufKOut, Out_K);
        RegistreH: registre16 PORT MAP (ClKPipeline, BufHOut, Out_H);
        RegistreI: registre16 PORT MAP (ClkPipeline, RegI, Out_I);
        RegistreS1: registre16 PORT MAP (ClkRegShift1, RegS1, Out_Shift1);
        RegistreS2: registre16 PORT MAP (ClkRegShift2, RegS2, Out_Shift2);
        RegistreZ: registre16 PORT MAP (ClkPipeline, BufKOut, Out_K);

END BlocMemoireArch;

---------- Entite ALU ----------
library IEEE;
use IEEE.std_logic_1164.all;
use work.registres.all;
use work.muxdemux.all;
ENTITY ALU IS
                PORT(  In_K          :IN std_logic_vector(15 downto 0);
                       In_H          :IN std_logic_vector(15 downto 0);
                       In_I          :IN std_logic_vector(15 downto 0);
                       In_Z          :IN std_logic_vector(15 downto 0);
                       In_S1         :IN std_logic_vector(15 downto 0);
                       In_S2         :IN std_logic_vector(15 downto 0);
                       ClkPipeline   :IN std_logic;
                       ResetPipeline :IN std_logic;
                       ClkReg_ALU    :IN std_logic;
                       SelectMuxALU  :IN std_logic_vector(1 downto 0);
                       SelectShift2  :IN std_logic_vector(1 downto 0);
                       ClkRegALU     :IN std_logic;
                       ALU_Out       :OUT std_logic_vector(15 downto 0));
```

```
        END ALU;
--ARCHITECTURE ALUArch OF ALU IS

--END ALUArch;


---------------- Entite IOBuffer ---------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.registres.all;
use work.muxdemux.all;
ENTITY IOBuffer IS
                PORT(   Load          :IN std_logic;
                        Clock_Load    :IN std_logic;
                        ClkRegIn_Micro  :IN std_logic;
                        ClkRegOut_Micro :IN std_logic;
                        Data          :INOUT std_logic_vector(15 downto 0);
                        ALU_Bus       :IN std_logic_vector(15 downto 0);
                        MicroMot      :OUT std_logic_vector(63 downto 0);
                        IN_Bus        :OUT std_logic_vector(15 downto 0));
        END IOBuffer;
ARCHITECTURE IOBufferArch OF IOBuffer IS
SIGNAL Reg1Out, Reg2Out, Reg3Out, Reg4Out: std_logic_vector(15 downto 0);
SIGNAL MuxOut: std_logic;
BEGIN
        MuxClkReg1: Mux2011 PORT MAP (Clock_Load,
ClkRegIn_Micro,Load,MuxOut);
        Registre1: registre16 PORT MAP (MuxOut, Data, Reg1Out);
        Registre2: registre16 PORT MAP (Clock_Load, Reg1Out,Reg2Out);
        Registre3: registre16 PORT MAP (Clock_Load, Reg2Out,Reg3Out);
        Registre4: registre16 PORT MAP (Clock_Load, Reg3Out,Reg4Out);

        RegistreOut: registre16 PORT MAP (ClkRegOut_Micro, ALU_Bus, Data);

        IN_Bus <= Reg1Out;

        MicroMot(15 downto 0) <= Reg1Out;
        MicroMot(31 downto 16) <= Reg2Out;
        MicroMot(47 downto 32) <= Reg3Out;
        MicroMot(63 downto 48) <= Reg4Out;
END IOBufferArch;


---------------- Entite Pilotage_Memoire ---------------
library IEEE;
```

```
use IEEE.std_logic_1164.all;
use work.buses.all;
use work.registres.all;
use work.muxdemux.all;

ENTITY Pilotage_Memoire IS
        PORT(  Load          :IN std_logic;
               SelectMemory  :IN std_logic_vector(2 downto 0);
               AdresseExterne :IN std_logic_vector(6 downto 0);
               AdresseInterneK :IN std_logic_vector(6 downto 0);
               AdresseInterneH :IN std_logic_vector(6 downto 0);
               AdresseInterneI :IN std_logic_vector(4 downto 0);
               AdresseInterneZ :IN std_logic_vector(6 downto 0);
               WriteExterne  :IN std_logic;
               WriteInterneI :IN std_logic;
               WriteInterneZ :IN std_logic;
               WriteK        :OUT std_logic;
               WriteH        :OUT std_logic;
               WriteI        :OUT std_logic;
               WriteZ        :OUT std_logic;
               WriteM        :OUT std_logic;
               AdresseK      :OUT std_logic_vector(6 downto 0);
               AdresseH      :OUT std_logic_vector(6 downto 0);
               AdresseI      :OUT std_logic_vector(4 downto 0);
               AdresseZ      :OUT std_logic_vector(6 downto 0);
               AdresseM      :OUT std_logic_vector(6 downto 0));
END Pilotage_Memoire;
ARCHITECTURE Pilotage_MemoireArch OF Pilotage_Memoire IS
SIGNAL WrI, WrZ:std_logic;
SIGNAL AdrK, AdrH, AdrZ:std_logic_vector(6 downto 0);
SIGNAL AdrI : std_logic_vector(4 downto 0);
BEGIN
        DemuxWrite: Demux1015
        PORT MAP (WriteExterne, SelectMemory, WriteK, WriteH, WrI, WrZ, WriteM);
        MuxWriteI: Mux2011
        PORT MAP (WrI, WriteInterneI, Load, WriteI);
        MuxWriteZ: Mux2011
        PORT MAP (WrZ, WriteInterneZ, Load, WriteZ);

        DemuxAdr: Demux1075
        PORT MAP (AdresseExterne, SelectMemory, AdrK, AdrH, AdrI, AdrZ,
AdresseM);
        MuxAdrK: Mux2071
        PORT MAP (AdrK, AdresseInterneK, Load, AdresseK);
        MuxAdrH: Mux2071
```

```
        PORT MAP (AdrH, AdresseInterneH, Load, AdresseH);
        MuxAdrI: Mux2051
        PORT MAP (AdrI(4 downto 0), AdresseInterneI, Load, AdresseI);
        MuxAdrZ: Mux2071
        PORT MAP (AdrZ, AdresseInterneZ, Load, AdresseZ);
END Pilotage_MemoireArch;

------------------------------------------------------------------------
--
--          Partie du programme de description VHDL du
--          specialise pour le filtrage de Kalman
--
--          Package reunissant divers parametres :
--                _ le Top-level entity
--                      et son architectures
--
------------------------------------------------------------------------

--      TOP LEVEL ENTITY
library IEEE;
use IEEE.std_logic_1164.all;
use work.vddgnd.all;
use work.delays.all;
use work.muxdemux.all;
use work.circuits.all;
use work.Memoires.all;
use work.registres.all;
use work.topcomp.all;
use work.clocks.all;

ENTITY Kalman_Processor IS
PORT(       Data          : INOUT std_logic_vector(15 downto 0);  -- Bus de donnees
            Adresse       : IN std_logic_vector(6 downto 0);      -- Bus d'adresse
            SelectMemory  : IN std_logic_vector(2 downto 0);      -- Selection de
memoires
            WriteExterne  : IN std_logic;       -- "Write" externe
            Load          : IN std_logic;
            Hold          : IN std_logic;
            Reset         : IN std_logic;
            ClockLoad     : IN std_logic;       -- Horloge externe
            AN1           : OUT std_logic;      -- Signaux de commande
            AN2           : OUT std_logic;      -- du convertisseur AN
            AN3           : OUT std_logic;
            AN4           : OUT std_logic;
            NA1           : OUT std_logic;      -- Signaux de commande
```

```
                NA2         : OUT std_logic;     -- du convertisseur NA
                NA3         : OUT std_logic;
                NA4         : OUT std_logic);
END Kalman_Processor;


--      L'architecture du processeur
ARCHITECTURE Kalman_Processor_Arch OF Kalman_Processor IS


        ---------------------------------------------------------------
        --      Declaration des signaux reliant les composants entre eux -
        ---------------------------------------------------------------

        --      Signaux issus du circuit d'horloge
        signal Fi_1, Fi_1b, Fi_2, Fi_2b: std_logic;

        --      Signaux issus du microsequenceur
        signal MicroCode:std_logic_vector(51 downto 0);

        --      Signaux issus des compteurs et UGA
        signal AdresseInterneK, AdresseInterneH, AdresseInterneZ:std_logic_vector(6
downto 0);
        signal AdresseInterneI:std_logic_vector(4 downto 0);
        signal Indic_C1_0, Indic_C2_0:std_logic;

        --      Signaux issus du bloc memoire
        signal Out_K, Out_H, Out_I, Out_Z, Out_Shift1, Out_Shift2:std_logic_vector(15
downto 0);

        --      Signaux issus de l'ALU
        signal ALU_Bus :std_logic_vector(15 downto 0);

        --      Signaux issus du bloc de pilotage externe des memoires
        signal AdresseK,AdresseH,AdresseZ,AdresseM:std_logic_vector(6 downto 0);
        signal AdresseI:std_logic_vector(4 downto 0);
        signal WriteK,WriteH,WriteI,WriteZ,WriteM:std_logic;

        --      Signaux issus du buffer entree\sorties
        signal IN_Bus:std_logic_vector(15 downto 0);
        signal MicroMot64:std_logic_vector(63 downto 0);

        ---------------------------------------------------------------
        --      Fin de la declaration des signaux internes du processeur -
        ---------------------------------------------------------------
```

```
BEGIN

--    Description de l'architecture du processeur

DeuxPhases:Clock_Generator PORT MAP(Fi_1, Fi_1b, Fi_2, Fi_2b);

Micro:Microsequenceur PORT MAP(MicroMot64, WriteM, AdresseM, Load, Indic_C1_0,
Indic_C2_0, Reset, Hold, MicroCode(51 downto 0), Fi_1, Fi_1b, Fi_2, Fi_2b);

PilotMemory:Pilotage_Memoire PORT MAP(Load, SelectMemory, Adresse,
AdresseInterneK, AdresseInterneH, AdresseInterneI, AdresseInterneZ, WriteExterne,
MicroCode(26), MicroCode(25), WriteK, WriteH, WriteI, WriteZ, WriteM, AdresseK,
AdresseH, AdresseI, AdresseZ, AdresseM);

Adresseurs:BlocCompteurUGA PORT MAP(MicroCode(51), MicroCode(50),
MicroCode(49), MicroCode(48), MicroCode(47), MicroCode(46), MicroCode(44),
MicroCode(43), MicroCode(42), MicroCode(40), MicroCode(41), MicroCode(39),
MicroCode(37), MicroCode(36), MicroCode(45), MicroCode(38), MicroCode(35 downto
34), AdresseInterneK, AdresseInterneH, AdresseInterneI, AdresseInterneZ, Indic_C1_0,
Indic_C2_0);

Memoire:BlocMemoire PORT MAP(AdresseK, AdresseH, AdresseI, AdresseZ, WriteK,
WriteH, WriteI, WriteZ, MicroCode(24), MicroCode(23), MicroCode(22), MicroCode(21),
Load, MicroCode(20 downto 19), MicroCode(11), MicroCode(13), MicroCode(12),
IN_Bus, ALU_Bus, Out_K, Out_H, Out_I, Out_Z,  Out_Shift1, Out_Shift2);

A_L_U:ALU PORT MAP(Out_K, Out_H, Out_I, Out_Z, Out_Shift1, Out_Shift2,
MicroCode(11), MicroCode(14), MicroCode(16 downto 15), MicroCode(18 downto 17),
MicroCode(10), ALU_Bus);


Registres_IO:IObuffer PORT MAP(Load, ClockLoad, MicroCode(9), Data, ALU_Bus,
MicroMot64, IN_Bus);

Convertisseur_AN_Proc: PROCESS( MicroCode(7 downto 4) )
        BEGIN
                AN1 <= MicroCode(7);
                AN2 <= MicroCode(6);
                AN3 <= MicroCode(5);
                AN4 <= MicroCode(4);
        END PROCESS Convertisseur_AN_Proc;

Convertisseur_NA_Proc: PROCESS( MicroCode(3 downto 0) )
        BEGIN
```

```
            NA1 <=MicroCode(3);
            NA2 <=MicroCode(2);
            NA3 <=MicroCode(1);
            NA4 <=MicroCode(0);
        END PROCESS Convertisseur_NA_Proc;

END Kalman_Processor_Arch;

-----------------------------------------------------------------
--      Fin de la description de niveau 1 du processeur
-----------------------------------------------------------------
```

ANNEXE B : PROGRAMMATION DU MICROSÉQUENCEUR

Microprogrammation

Les bits 51 à 0, chargés de piloter l'ensemble de l'architecture peuvent être divisés en deux catégories; la premiere catégorie regroupe les bits qui commandent des multiplexeurs, l'initialisation des UGA et compteurs et les signaux d'entrées sortie (convertisseurs); ils sont actifs dès la phase 1. La seconde concerne les bits qui commandent les registres du processeur; ils sont actifs à la phase 2.

- *L'Initialisation*

Les opérations d'initialisation sont les suivantes; elles sont écrites "symboliquement" pour représenter la séquence des opérations.

Rq : les termes load, select_memory, adresse_externe, data, clock_load et write_externe désignent des s du processeur.

Le mot du microséquenceur

63	commande du muiltiplexeur de condition du microséquenceur	(bit 2)
62	" " "	(bit 1)
61	" " "	(bit 0)
60	donnée de comparaison pour test de polarité	
59	choix d'adressage : JUMP / RETOUR	
58	adresse de saut (bit 6)	
57	adresse de saut (bit 5)	
56	"	
55	"	
54	"	

53	"			
52	"		(bit 0)	
51	commande du multiplexeur du compteur 1			
50	commande du multiplexeur du compteur 2			
49	clock compteur 1			
48	clock compteur2			
47	clock du registre de base de l'UGA_I			
46	clock du registre de déplacement de l'UGA_I			
45	clock du registre de sortie de l'UGA_I			
44	commande du multiplexeur du pointeur de l'UGA_I			
43	commande du multiplexeur de sortie de l'UGA_I			
42	clock du registre de base 1 de l'UGA_Z			
41	clock du registre de déplacement 1 de l'UGA_Z			
40	clock du registre de base 2 de l'UGA_Z			
39	clock du registre de déplacement 2 de l'UGA_Z			
38	clock du registre de sortie de l'UGA_Z			
37	commande du multiplexeur du pointeur 1 de l'UGA_Z			
36	commande du multiplexeur du pointeur 2 de l'UGA_Z			
35	commande du multiplexeur de sortie de l'UGA_Z			(bit 1)
34	"	"	"	(bit 0)
33	donnée d'initialisation des UGA et compteurs			(bit 6)
32	"	"	"	(bit 5)

Mot du microséquenceur : suite

31	donnée d'initialisation des UGA et compteurs			(bit 4)
30	"	"	"	
29	"	"	"	
28	"	"	"	
27	"	"	"	(bit 0)
26	Write Mémoire	I		
25	"	Z		
24	Read Mémoire	K		
23	"	H		
22	"	I		
21	"	Z		
20	commande du démultiplexeur de la mémoire I			(bit 1)

19	" " "	(bit 0)
18	commande du décaleur 2	(bit 1)
17	" "	(bit 0)
16	commande du multiplexeur de l'A.L.U	(bit 1)
15	" " "	(bit 0)
14	reset pipeline, compteurs, UGA et registre A.L.U	
13	clock registre décaleur 1	
12	clock registre décaleur 2	
11	clock pipeline	
10	clock registre A.L.U	
9	clock registre d'entrée (mesure)	
8	clock registre de sortie (reconstitution)	
7	convertisseur A/N	(bit 3)
6	convertisseur A/N	(bit 2)
5	convertisseur A/N	(bit 1)
4	convertisseur A/N	(bit 0)
3	convertisseur N/A	(bit 3)
2	convertisseur N/A	(bit 2)
1	convertisseur N/A	(bit 1)
0	convertisseur N/A	(bit 0)

Le séquencement des opérations

Il se subdivise en deux parties:

- l'initialisation;

- le traitement.

- *L'Initialisation*

Les opérations d'initialisation sont les suivantes; elles sont écrites "symboliquement" pour représenter la séquence des opérations.

Rq : les termes load, select_memory, adresse_externe, data, clock_load et write_externe désignent des broches du processeur.

1_ *mettre le load au niveau "0"*

2_ *chargement de la mémoire K*

select_memory = "000"

for i=0 to M-1 (M=taille du transopérateur)

adresse_externe = a_i

data = d_i

clock_load (front descendant)

write_externe (front montant)

end

3_ *chargement de la mémoire H*

select_memory = "001"

for i=0 to M-1 (M=taille du transopérateur)

adresse_externe = a_i

data = d_i

clock_load (front descendant)

write_externe (front montant)

end

4_ *chargement de la mémoire I*

select_memory = "010"

for i=0 to N-1 (N=nombre de paramètres)

adresse_externe = a_i

data = d_i

clock_load (front descendant)

write_externe (front montant)

end

5_ *chargement de la mémoire Z*

select_memory = "011"

for i=0 to M-1 (M=taille du transopérateur)

adresse_externe = a_i

data = "0000H" (le vecteur d'`tat est initialisé à 0)

clock_load (front descendant)

write_externe (front montant)

end

6_ *chargement de la mémoire du microséquenceur M*

select_memory = "100"

for i=0 to F-1 (F=nombre de mots du microprogramme)

adresse_externe = a_i

for j=0 à 3 (boucle de remplissage du registre à décalage)

data = d_j

clock_load (front descendant)

end

write_externe (front montant)

end

- *Le traitement*

Une fois le chargement des mémoires terminé, on passe en phase de traitement en mettant la broche load au niveau haut "1".

1_ *Le pré-traitement*

Cette étape initialise les UGA, les compteurs, Reset l'accumulateur du multiplieur/ accumulateur:

Reset Accu et registreALU	:	1 mot
Charger RB_I	:	"
Charger RD_I	:	"
Charger RB1_Z	:	"
Charger RD1_Z	:	"
Charger RB2_Z	:	"
Charger RD2_Z	:	"
Charger compteur 1	:	"
Charger compteur 2	:	"

total : 9 mots

2_ *Chargement des paramètres dans leurs registres*

Cette étape charge les registres des deux décaleurs, RegS1 et RegS2. En supposant que les données correspondantes se trouvent aux cases 00H et 01H et que les registres de base et déplacement de l'UGA_I ont été préchargés en conséquent (11111 et 00001) pour fournir ces adresses successivement.

adressage mémoire I
lecture mémoire
clock RegS1
adressage mémoire I
lecture
clock RegS2 total : 3 mots

3_ *Traitement de l'échantillon*

Cette séquence est répétée autant de fois qu'il y a d'échantillons à traiter. Elle comporte des parties correspondant à des boucles internes (série d'instructions répétitives se terminant par un saut conditionnel).

3_a *Reset pipeline et reset sur le registre de l'A.L.U*

La mise à zéro du registre de sortie de l'A.L.U donne l'illusion d'un calcul d'estimé nul. Ceci est le point de départ de l'algorithme, puisque:

$Z_{(0/0)}=0$, $Z_{(0/1)}=0$ et $y_0=h^t \cdot Z_{(0/1)}=0$.

Le Bus ALU (ALU_Bus) contient une valeur nulle

total : 1 mot

3_b *Acquisition*

Lecture du convertisseur analogique/numérique: le nombre de microinstructions nécessaires à cette opération n'a pas été évalué; appelons *temps d'acquisition* ce temps.

Clock registre d'entrée (1 mot): IN_Bus = mesure.

total : 1 mot + temps d'acquisition

3_c *Temps de calcul du soustracteur (innovation I)*

Il s'agit du temps de calcul du soustracteur qui est en fait un additionneur à anticipation de retenue (CLA) 16 bits (identique à celui utilisé dans l'accumulateur du multiplieur 2) ayant un temps de calcul inférieur à 10 ns.

cette étape est effectuée à la phase 1

clock registre ALU. (phase 2)

préadressage de la mémoire I

total : 1 mot

3_d *Écriture de l'innovation I en mémoire*

_ pilotage du multiplexeur de l'ALU

_ écriture de I (la mémoire a déjà été préadressée)

total : 1 mot

3_e *remplissage du multiplieur 1*

i de 0 à 4

_ clock compteur 1 (adressage de la mémoire K)

_ Lecture parallèle de Ki, I, Zi

_ clock pipeline

total : 6 mots

3_f *Sortie de la reconstitution (Z(1))*

_ Pilotage du multiplexeur de l'ALU

_ clock registre ALU

_ clock registre de sortie

_ temps de conversion N/A; appelons temps de sortie le temps nécessaire pour piloter le convertisseur numérique/analogique.

total : 2 mots + temps de sortie.

3_g *Remplissage du multiplieur 2*

i de 5 à 9

_ clock compteur 1 (adressage de la mémoire K)

_ Lecture parallèle de Ki, I, Zi

_ clock pipeline

total : 6 mots

3_h *Fonctionnement parallèle des deux multiplieurs*

i de 10 à 127

(pilotage du multiplexeur de l'ALU vers le multiplieur1)

_ écriture de Z(i-5)

_ lecture de Ki, Zi, h(i-5)

_ clock pipeline / clock des deux pointeurs de l'UGA_Z / clock registre ALU

_ test de fin de la boucle 1

total : 2 mots

3_i *le multiplieur 1 se vide*

i de 128 à 132

_ écriture de Z(i-5)

_ lecture de h(i-5)

_ clock pipeline

_ clock registre ALU

total : 2 mots

3_j *le multiplieur 2 se vide*

i de 133 à 137

(pas de lecture mémoire)

_ pilotage du multiplexeur de l'ALU vers le multiplieur 2

_ clock registre ALU

_ test de fin de boucle 2

total : 1 mot

3_k *Écriture de l'estimé dans le registre de sortie de l'ALU*

_ clock du registre ALU (ALU_Bus = y_estimé)

total : 1 mot

3_l *réinitialisation des compteurs et UGA*

_ chargement compteur 1

_ chargement compteur 2

_ Charger RB_I Rq: on ne réinitialise que les registres de base.

_ Charger RB1_Z

_ Charger RB2_Z

total : 5 mots

3_m *Retour*

À cette étape, on a terminé le traitement d'un échantillon; Les adresseurs (UGA et compteurs) sont repositionnés pour effectuer de nouveau les deux boucles l'estimé de l'échantillon à venir se trouve déjà sur le ALU_Bus et l'on s'apprête à faire une acquisition; Il s'agit donc de reprendre toute la séquence décrite précédemment, mais à partir de l'étape b.

total = 1 mot : (JUMP inconditionnel).

ANNEXE C : LISTE DES PROGRAMMES DE SIMULATIONS MATLAB

C1 :	ana.m	Fonction restituant la valeur decimale d'un mot de Nb bits representé en virgule fixe complement a 2.
C2 :	bin.m	Programme de conversion decimale / binaire virgule fixe complement a 2 des vecteurs H, X, Y et K
C3 :	bruitfct.m	Programme ajoutant le bruit sur un signal function [y] = bruitfct(x,sigma_b,poid), par Daniel MASSICOTTE
C4 :	convfct.m	Convolution des signaux
C5 :	decalage.m	Fonction qui effectue le decalage vers la gauche d'un vecteur (de Ndec elements)
C6 :	errqdfct.m	Calcule l'erreur quadratique entre deux vecteurs [ErrQuad] = errqdfct(x,y,n_debut,n_fin)
C7 :	gaussoid.m	Fonction de creation d'une gaussoide gaussoid(T,Mu,Sigma)
C8 :	kalman2.m	Programme effectuant la reconstitution des echantillons en utilisant l'algorithme rapide parallelisé
C9 :	kalmanbi.m	Algorithme de Kalman parallelise Programme effectuant la reconstitution des echantillons en tenant compte de la resolution due au nombre de bits
C10 :	prog4.m	Programme de création de signaux, convolution et reconstitution gain sur la mesure et reechelonnement de h et K au choix
C11 :	prog4n.m	Suite de prog4.m mais avec les valeurs numerisées des vecteurs
C12 :	rk3a1fct.m	Création des signaux (entrée et transopérateur) fonction phi moitie d'une gaussoide; [x,h,y,yb,tx] = rk3a1fct (Nb_pt_x, Nb_pt_h,sigma_b,poid);
C13 :	vecana.m	Programme reajustant les valeurs de y, h et K, par une troncature en fonction du nombre de bits.

C14 : vecbin.m Programme de conversion numerique des vecteurs de reconstitution x, h, y et K.

C1

```
%
%    Fonction restituant la valeur decimale
%    d'un mot de Nb bits representé en virgule fixe complement a 2
%
%    par Philippe A. PANGO
%
function c=ana(x, Nb_bit);
for i=1:Nb_bit
        a(i) = x(Nb_bit-i+1);
end
x = a;
seuil = 2^(-Nb_bit-1);
c = -x(Nb_bit)*2^(Nb_bit-1);
for i=1:Nb_bit-1
        c = c + x(i)*2^(i-1);
end
%c = c / (2^(Nb_bit-1)-1) - seuil;
return
```

C2

```
%
%   Programme de conversion decimale / binaire virgule fixe
%   complement a 2 des vecteurs H, X, Y et K
%

function c = bin(x, Nb_bit);
seuil = 0.5;
N_Max = 2^(Nb_bit-1)-1;
%x = x*N_Max;

% Le signe
if x >= 0
  c(Nb_bit) = 0;
  else
  c(Nb_bit) = 1;
end

if x < 0
        x = N_Max-abs(x) + 1;
        %x = abs(x) + 1;
```

```
end

i = Nb_bit-1;
while i > 0
        if  x >= 2^(i-1);
        c(i) = 1;
                x = x - 2^(i-1);
        else
        c(i) = 0;
        end
        i = i-1;
end
for i=1:Nb_bit
        a(i) = c(Nb_bit-i+1);
end
c = a;
return
```

C3

```
%
%    Programme ajoutant le bruit sur un signal
%    function [y] = bruitfct(x,sigma_b,poid)
%       par Daniel MASSICOTTE
%

function [y] = bruitfct(x,sigma_b,poid);
Nb_pt_y = max(size(x));
%
%  Bruit sur la mesure
%
randn('seed',poid)
bruit = sigma_b * rand(Nb_pt_y,1);


if bruit ~= 0

bruit = sigma_b/std(bruit) *bruit;

i = 0;
while i == 1

%
%  Il faut s'assurer que la valeur moyenne du bruit
%  est inférieure à 1e-3 ou à la valeur minimale qu'il
```

```
% est possible de réaliser avec matlab
%
k = poid;
moy_bruit_min = 10;
while abs( mean(bruit) ) >= 1e-3 & k <= 1e4
  k = k+1
  rand('seed',k);
  bruit = sigma_b * randn(Nb_pt_y,1);
  bruit = sigma_b/std(bruit) *bruit;
  if moy_bruit_min > abs( mean(bruit) )
    moy_bruit_min = abs(mean(bruit));
    poid = k;
  end
end

if k > 1e4
  disp('**** ATTENTION, la valeur moyenne du bruit est supérieure ... 1e-5')
  rand('seed',poid)
  bruit = sigma_b*randn(Nb_pt_y,1);
  bruit = sigma_b/std(bruit) *bruit;
end

end
end
y = x + bruit;
```

C4

```
%
%     CONVOLUTION DES SIGNAUX
%

function [y,SNR] = convfct(x,h,dt,sigma_b,poid);
Nb_pt_x = max(size(x));
Nb_pt_h = max(size(h));

y = conv(x,h)*dt;
y = y(1:Nb_pt_x);
```

C5

```
%
%      Fonction qui effectue le decalage vers la gauche
%      d'un vecteur ( de Ndec elements )
%
```

```
%      z = decalage(x,Ndec)
%
function z=decalage(x,Ndec)
N = max(size(x));
for k=1:N-Ndec
   z(k) = x(k+Ndec);
end
for k=N-Ndec+1:N
   z(k) = 0;
end
z = z';
```

C6

```
%
%    Calcule l'erreur quadratique entre deux vecteurs
%
%          [ErrQuad] = errqdfct(x,y,n_debut,n_fin)
%
function [ErrQuad] = errqdfct(x,y,n_debut,n_fin)

ErrQuad = 0;
for i=n_debut:n_fin
        ErrQuad = ErrQuad + (x(i)-y(i))^2;
end
ErrQuad = sqrt(ErrQuad/(n_fin-n_debut+1));
```

C7

```
%
%   Fonction de creation d'une gaussoide
%   gaussoid(T,Mu,Sigma)
%
%              T = vecteur temps
%              Mu = valeur ou la gaussoide atteint son maximum
%              Sigma = largeur de la gaussoide
%
%


function c = gaussoid(t,mu,sigma);
m = max(size(t));

for i=1:m
     c(i) = 1/sqrt(2*pi)/sigma * exp( -(t(i)-mu)^2 / 2/sigma^2 );
```

```
end
return
```

C8

```
%
%
%          Programme effectuant la reconstitution
%               des echantillons
%       en utilisant l'algorithme rapide parallelisé
%
%                  par Philippe A. PANGO
%

Z = zeros(Nb_pt_h,1);
Max_Z = 2;
for k=1:Nb_pt_x
%k
        ye(k) = ordre*h(1)*Z(1);
        for i=1:Nb_pt_h-1
                ye(k) = ye(k) + h(i+1)*Z(i);
                if ye(k)>Max_Z & (0)
                        disp('depassement ye')
                        [k i]
                        ye(k)
                        pause
                end
        end
        if echelonnement_K == 'oui'
                ye(k) = ye(k) / gain_K;
        end

        I(k) = yb(k) - ye(k);

        for i=Nb_pt_h-1:-1:2
                Z(i) = Z(i-1) + K(i)*I(k);
                if Z(i)>Max_Z & (0)
                        disp('depassement Zi')
                        [k i]
                        Z(i)
                        pause
                end

        end
        Z(1) = Z(1) + K(1)*I(k);
```

```
        if positivite == 'oui'
                for i=1:Nb_pt_h
                        if Z(i) < 0
                                Z(i) = C * Z(i);
                        end
                end
        end
        mz(k) = max(Z);
        yr(k) = Z(retard);


end
ye = ye';
yr = yr';
```

C9

```
%
%               Algorithme de Kalman parallelise
%          Programme effectuant la reconstitution
%               des echantillons
%      en tenant compte de la resolution due au nombre
%                   de bits
%
%       Les vecteurs sont tronqués par les procédures ana.m et bin.m
%
%              par Philippe A. PANGO
%

Z = zeros(Nb_pt_h,1);
Max_Z = 2;

for k=1:Nb_pt_x
k
   yen(k) = ordre * ana( bin( hn(1)*Z(1), Nb_bit_mult), Nb_bit_mult);
   for i=1:Nb_pt_h-1
      yen(k) = yen(k) + ana( bin( hn(i+1)*Z(i), Nb_bit_z ), Nb_bit_z);
                if yen(k)>Max_Z & (0)
                        disp('depassement ye')
                        [k i]
                        yen(k)
                        pause
                end
   end
   yen(k) = ana(bin(yen(k)/gain_K, Nb_bit_z), Nb_bit_z);
```

```
In(k) = yb(k) - yen(k);

for i=Nb_pt_h-1:-1:2
  Z(i) = Z(i-1) + ana(bin( Kn(i)*In(k), Nb_bit_z ), Nb_bit_z);
        if Z(i)>Max_Z & (0)
              disp('depassement Zi')
              [k i]
              Z(i)
              pause
        end
end
Z(1) = Z(1) + ana(bin( Kn(1)*In(k), Nb_bit_z), Nb_bit_z);
    if positivite == 'oui'
        for i=1:Nb_pt_h
        if Z(i) < 0
              Z(i) = ana(bin( C * Z(i), Nb_bit_z ), Nb_bit_z);
        end
        end
    end
mnz(k) = max(Z);
yrn(k) = Z(retard);

if k>=25        % visualisation
hold off
axis([0 k min([min(x(1:k)) min(yr(1:k)) min(yrn(1:k))/gain_yb/dt*gain_h/gain_K])
max([max(x(1:k)) max(yr(1:k)) max(yrn(1:k))/gain_yb/dt*gain_h/gain_K]) ]);
plot(x(1:k))
hold on
plot(yr(1:k),'r')
plot(yrn(1:k)/gain_yb/dt*gain_h/gain_K,'g')
hold off
pause
end

end
yrn = yrn';
```

C10

```
%       Prog4.m
%     Signaux de la fonction RK3A1FCT.M
%       Programme de création de signaux, convolution et reconstitution
%        gain sur la mesure et reechelonnement de h et K au choix
%        par Philippe A. PANGO
```

```
%

clear
clc
clg

%%%  Definition des variables

Nb_pt_x = 128;              % Nombre de points de x
Nb_pt_h = 64;          % Nombre de points du transoperateur
sigma_b = 1e-4;             % Deviation standard du bruit
poid = 1;              %
Nb_pt = Nb_pt_h;       % Nombre max d'iterations pour le calcul du gain
Q_R = 1e5;             % Q_R si h multiplie par 1
R = 1;                 % R
ordre = 1;             % F(1,1)
retard = 1;            % Element 1 du vecteur d'état á extraire
%retard = Nb_pt_h/2;% Element milieu du vecteur d'état á extraire
echelonnement_h = 'oun';% Indique si oui ou non le vecteur h
%                       sera re-echelonneé
echelonnement_K = 'oun';% Indique si oui ou non le vecteur K
%                       sera re-echelonneé
echelonnement_max='oui';% Indique que le max des vecteurs sera la ref.
ampli_mesure = 'oun'; % Indique si le signal de mesure sera reechelonné
positivite = 'oun';    % Autorise la contrainte de positivite
C = 1/2;               % Parametre de la contrainte de positivite
if positivite == 'non'
        C = 0.8;
end


%%%  Creation des signaux
[x,h,tx,th,dt] = rk3a1fct(Nb_pt_x,Nb_pt_h);
%[x,h,tx,th,dt] = rk1a6fct(Nb_pt_x,Nb_pt_h);
disp(' ')
disp('Signaux x et h crees')

%%%  Convolution et bruit
[y,SNR] = convfct(x,h,dt);
[yb] = bruitfct(y,sigma_b,poid);
ybi = yb;
if ampli_mesure == 'oui'
        gain_yb = 1/max(yb);
        yb = yb * gain_yb;
  else
```

```
        gain_yb = 1;
end

disp(' ')
disp('Convolution terminee et bruit ajoute')
save signau4

%%% Calcul du gain
% echelonnement du transoperateur
if echelonnement_h == 'oui'
        disp(' ')
        disp('Echelonnement du transoperateur')
        gain_h = 1 / max(h);
        hi = h;
        h = h * gain_h;
  else
        gain_h = 1;
end
% calcul du gain
disp(' ')
disp('Calcul du gain en cours')
[K,Err_K] = gain_k(h,Nb_pt,Q_R,R,dt,ordre);
if echelonnement_K == 'oui'
        disp(' ')
        disp('Reechelonnement du vecteur K')
        % Reechelonnement du gain K
        Ki = K;
      %gain_K = 2^(fix(log10(2/max(Ki))/log10(2)))
        gain_K = 1 / 2^(ceil(log(max(K))/log(2)));
        K = K * gain_K;
  else
        gain_K = 1;
end


%%% Deconvolution
disp(' ')
disp('Deconvolution')
%       L'indice de dépassement
Id = Nb_pt_h * max(x) * gain_yb * dt / gain_h * gain_K;
if Id > 1 & (0)
        disp(' ');
        disp('L`indice de dépassement est > 1');
        Id
        pause
```

```
end
%kalman
kalman2
%[yr] = kalmanma(yb,h,K,dt,retard);
yr = decalage(yr,retard);
yr = yr / gain_yb / dt * gain_h / gain_K;
ErrQuad = errqdfct(x,yr,1,Nb_pt_x)
save recons4
%plotxyr
plotrec

end
```

C11

```
%
%     Suite de prog4.m mais avec les valeurs
%     numerisees des vecteurs
%
clear
clc
clg

load recons4

if echelonnement_max == 'oui'
        Nb_bit = 8;
        gain_max = (2^(Nb_bit-1)-1) / max([max(yb),max(h),max(K)]);
        ybi = yb;
        yb = yb * gain_max;
        hi = h;
        h = h*gain_max;
        Ki = K;
        K = K*gain_max;
end
disp('bhjk')
kalman2
disp('dfdghfg')
pause

Nb_bit_mult = Nb_bit;
Nb_bit_x = Nb_bit;
Nb_bit_y = Nb_bit;
Nb_bit_z = Nb_bit;
Nb_bit_h = Nb_bit;
```

```
Nb_bit_k = Nb_bit;

numerisation = 'oui';   % plotte les vecteurs et leur valeur numerisee.
visualisation = 'oui';   % indique si l'on désire visualiser les vecteurs.

if numerisation == 'oui'
disp(' ')
disp('Numerisation des vecteurs')
vecana
save vecana4
disp(' ')
disp('Numerisation terminee')
end

if visualisation == 'oui'
hold off
plot(yb)
hold on
plot(ybn,'g')
title('Le signal de mesure')
hold off
pause

hold off
plot(h)
hold on
plot(hn,'g')
title('Le transoperateur')
hold off
pause

hold off
plot(K)
hold on
plot(Kn,'g')
title('Le vecteur GAIN')
hold off
pause

end     % fin de la visualisation

disp(' ')
disp('Kalman numerique en cours')
disp(' ')
load vecana4;
```

```
kalmanbi

yrn = ((( yrn/gain_m )/ dt) * gain_h )/ gain_K;
yrn = decalage(yrn,retard);
ErrQuadNum = errqdfct(x,yrn,1,Nb_pt_x);
disp(' ')
disp('L`erreur quadratique point flottant est : ');
ErrQuad;
disp('L`erreur quadratique numerique est      : ');
ErrQuadNum;
save recons4n
plotan
```

C12

```
%  par Daniel MASSICOTTE
%  modifications : Philippe A. PANGO
%
%  Création des signaux ressemblant à ceux de Crilly
%
%  fonction phi : moitie d'une gaussoid
%
% [x,h,y,yb,tx] = rk3a1fct(Nb_pt_x,Nb_pt_h,sigma_b,poid);

function [x,h,tx,th,dt] = rk3a1fct(Nb_pt_x,Nb_pt_h);

dt = 25/Nb_pt_x;
tx = 0:dt:25-(25/Nb_pt_x);
th = 0:dt:(Nb_pt_h-1)*dt;
tx = tx';
th = th';

x = 2*exp(-6*(tx-5.5).^2) + 6*exp(-6*(tx-8).^2);

p = 0.7; , q = 0.9;
for k=1:Nb_pt_h
  h(k) = 6*( exp(-p*tx(k)) - exp(-q*tx(k)) );
end
h = h';
```

C13

```
%
%   Programme reajustant les valeurs de y, h et K
%
```

```
for i=1:Nb_pt_x

        Vec_yb(i,1:Nb_bit_y) = bin(yb(i), Nb_bit_y);
        ybn(i) = ana(Vec_yb(i,:) , Nb_bit_y);
end
for i=1:Nb_pt_h

        Vec_h(i,1:Nb_bit_h) = bin(h(i), Nb_bit_h);
        hn(i) = ana(Vec_h(i,:) , Nb_bit_h);

        Vec_K(i,1:Nb_bit_k) = bin(K(i), Nb_bit_k);
        Kn(i) = ana(Vec_K(i,:) , Nb_bit_k);

        Vec_Z(i,1:Nb_bit_z) = bin(0, Nb_bit_z);
        Zn(i) = ana(Vec_Z(i,:) , Nb_bit_z);

end
```

C14

```
%
%       Programme de conversion analogique numerique
%       des vecteurs de reconstitution x, h, y et K.
%
%         par Philippe A. PANGO
%

clear;
load recons4;
Nb_bit_z = 16;
Nb_bit_k = 16;
Nb_bit_h = 16;
Nb_bit_y = 16;

%  Conversion des vecteurs en binaire
for k=1:Nb_pt_x
   Vecyb_s(k,1:Nb_bit_y) = bin(yb(k), Nb_bit_y);
end
for k=1:Nb_pt_h
   Vech_s(k,1:Nb_bit_h) = bin(h(k), Nb_bit_h);
   VecK_s(k,1:Nb_bit_k) = bin(K(k), Nb_bit_k);
   VecZ_s(k,1:Nb_bit_z) = bin(0, Nb_bit_z);
end
```

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