UNIVERSITÉ DU QUÉBEC

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN ÉLECTRONIQUE INDUSTRIELLE

PAR

DIOMANDÉ ABOU BAKARI

ÉTUDE DE L'IMPLANTATION EN TECHNOLOGIE ITGE D'UN ALGORITHME ITÉRATIF BASÉ SUR LE FILTRE DE KALMAN POUR LA RECONSTITUTION DE SIGNAUX

SEPTEMBRE 1998

À mes parents
À mes frères et mes sœurs
À mes amis

RÉSUMÉ

Le reconstitution de mesurandes constitue un problème fondamental dans la métrologie. Elle consiste à estimer un signal *x(t)*, qui n'est pas mesurable directement, à partir des résultats de mesure d'un autre signal $\tilde{y}(t)$ qui est lié avec le premier de façon causale. Les algorithmes itératifs de reconstitution de mesurandes les plus courants, comme ceux de Jansson, Van-Cittert et Gold, ont entre elles en commun le stockage de toutes les données $\{\tilde{y}_n\}$pour effectuer la reconstitution. La conséquence de ce stockage est la quantité de mémoire importante que cela demande à une architecture pour effectuer la reconstitution qui entraîne une structure non régulière et des délais de communication très longs. Pour éviter ces inconvenients, dans le mémoire qui suit, nous utiliserons un algorithme itératif de reconstitution basé sur le filtre de Kalman, avec une contrainte de positivité, dénommé ITERKAL+. Ce dernier apporte l'avantage d'un traitement en temps réel, car pour chaque donnée mesurée $\tilde{y}_n$, une reconstitution $\hat{x}_n$ est réalisée en K itération avant la lecture de la donnée $\tilde{y}_{n+1}$. Avec ITERKAL+, on obtient également des résultats de reconstitution meilleurs que ceux de Kalman., dans le cas où la réponse impulsionnelle du système est symétrique, telle que rencontrée en spectrométrie, où elle est presqu'une gaussoide [MAS95].

L'objectif principal de ce mémoire est d'apporter les améliorations nécessaires pour une estimation optimale du nombre d'itération, et de faire une étude de l'implantation en

Grande Échelle) de l'algorithme ITERKAL+ en vue d'élaborer une proposition d'une architecture capable de recevoir l'algorithme ITERKAL+.

Dans ce travail nous proposons deux critères d'arrêt du nombre d'itération K. Ces critères sont étudiées, comparés et le choix sera fixé par la qualité des résultats et la possibilité de l'intégrer à un processeur VLSI. De plus, trois architectures basées sur une architecture semi-systolique linéaire à structure en anneau nommée SYSKAL sont proposées ($SYSKALI_1$, $SYSKALI_2$, $SYSKALI_3$). Ces architectures répondent aux différentes contraintes en VLSI telle que vitesse et surface d'intégration. La fonctionnalité des architectures proposées sera validée par des résultats de simulation de leur modèle VHDL en utilisant des données synthétiques en spectrométrie.

Les résultats de ce travail constituent une contribution à l'implantation en technologie VLSI d'algorithmes de reconstitution de signaux.

REMERCIEMENT

Je tiens à remercier fortement mon directeur de recherche, le Docteur Daniel Massicotte, qui m'a apporté tout au long de ce travail de précieux conseils. Mes remerciements vont également à mon co-directeur de recherche, le Docteur Andrejz Barwicz, qui m'a initié aux outils de recherches dès mon arrivée à l'Université du Québec à Trois Rivières.

Il serait injuste de passer sous silence l'aide précieuse que m'ont aussi apporté les Docteurs Mohamed Ben Slima, Leszec Szczecinski et l'étudiant au doctorat Michal Wisniewski. Le partage de leur connaissance dans le domaine des systèmes de mesure a représenté un apport considérable tout au cours de ma recherche.

Je tiens également à remercier tous mes amis du laboratoire de micro-électronique et de systèmes de mesure pour l'encouragement et le support qu'ils m'ont apporté.

Enfin, je remercie très sincèrement le gouvernement de la République de Côte d'Ivoire pour le support financier qu'il m'a apporté tout au long de mon parcours scolaire.

TABLE DES MATIÈRES

Liste des tableaux

LISTE DES FIGURES

LISTE DES SYMBOLES

b vecteur de commande d'une représentation;

C condition d'ensemble;

$\mathbf{C_y}$ cellule de l'architecture SYSKAL pour le calcul de $\hat{y}_n$;

$\mathbf{C_z}$ cellule de l'architecture SYSKAL pour le calcul de $\hat{x}_n$ utilisant le vecteur $\hat{z}_{n/n-1}$;

dim dimension d'un vecteur ou d'une matrice;

g réponse impulsionnelle du système de conversion;

h réponse impulsionnelle modifiée;

I innovation;

J critère d'optimisation;

J_{LCh} critère d'arrêt optimal du nombre d'itérations basé sur la méthode L-COURBE;

J_x critère d'arrêt optimal du nombre d'itération basé sur l'erreur d'estimation du résultat de mesure;

k valeur entière de la $k^{ième}$ itération dans l'algorithme Iterkal+;

K vecteur de gain de Kalman;

k_∞ vecteur de gain stationnaire de Kalman;

K nombre total d'itérations dans l'algorithme Iterkal+;

lim valeur limite de;

L nombre de processeurs élémentaires dans l'architecture SYSKAL;

m pas d'échantillonnage de la réponse impulsionnelle g, m=1,2,3…,m;

min valeur minimale;

M module de mémoire;

n pas d'échantillonnage de la mesure $\hat{y}_n$, n=1,2,…,*N*;

N nombre total d'échantillon de la mesure $\tilde{y}_n$;

r variance du bruit;

t temps;

t_{PE} temps de calcul d'un PE;

x valeur exacte du mesurande;

y valeur du signal mesuré à la sortie du système de conversion;

$\mathbf{z}$ vecteur d'état de la représentation des données $\{\tilde{y}_n\}$;

$\hat{v}_{n/i,m}$ valeur estimée à l'instant n de l'élément m du vecteur v compte tenu des informations disponibles à l'instant i;

$\Delta\lambda$ pas de numérotation de la longueur d'onde;

λ scalaire représentant la longueur d'onde;

η_n valeur du bruit aléatoire s'ajoutant à la mesure de y_n;

Φ matrice d'état de la représentation des données $\{\tilde{y}_n\}$;

$\sum$ matrice de covariance normalisée;

$\sum'$ matrice de covariance;

$\cong$ approximé à;

$\equiv$ équivalent à;

$\| \|_2$ norme dans l'espace de Hilbert des séquences L_2;

$*$ convolution de.

ε erreur relative moyenne du moindre carré

ε_{opt} erreur relative moyenne du moindre carré optimale

$\varepsilon_{\min}$ erreur relative moyenne du moindre carré minimale

K_{opt} nombre d'itération optimale

β paramètre de regularisation de la classe d'algorithmes pour minimiser l'erreur de reonstitution

ω représente la fréquence

V_k vecteur de bruit de mesure

B_k vecteur de commande de la représentation d'état

H_k Matrice de mesure

R_k Variance du bruit de mesure

W_k vecteur générateur du bruit d'entrée

γ paramètre de régularisation

CHAPITRE 1

INTRODUCTION

Le reconstitution de mesurandes constitue un problème fondamental dans la métrologie. Elle consiste à estimer un signal *x(t)*, qui n'est pas mesurable directement, à partir des résultats de mesure d'un autre signal $\tilde{y}(t)$ qui est lié avec le premier de façon causale. Le traitement numérique des signaux joue un rôle de plus en plus important dans la reconstitution de mesurandes. Cette dernière pour être une opération relativement complexe utilisée pour l'interprétation de données séismiques[MEN86], [GUE89], l'amélioration de la résolution de l'analyse spectrométrique et chromatologique [JAN84], la correction dynamique de capteurs, la mesure de la thermocinétique de réactions chimiques, l'accélération des mesures quasi-statique, l'égalisation de canaux de télécommunication[GOD89], le diagnostique médical basé sur les techniques d'échographie ultra-sonore[HER91] et dans le domaine de la robotique comme moyen de l'étalonnage cinétique et de l'estimation de paramètres dynamiques de manipulateurs[CHA90], etc.

Les résultats de mesure obtenus ne sont pas toujours exacts, car lorsqu'une mesure est obtenue au travers d'une chaîne de conversion de mesure, les perturbations indésirables, telles que la température, les bruits de fond, les bruits d'amplification, les vibrations etc,

entraînent des imperfections au niveau du résultat obtenu. La présence inévitable de ces erreurs de conversion affecte non seulement l'acquisition des résultats de mesure d'observations, mais aussi ne facilite pas leur interprétation et ne permet pas toujours d'avoir la valeur exacte que nous recherchons. A cause de ces limitations et des imperfections des éléments du système de mesure, le mesurande subit, au cours de son traitement, des dégradations. Dans ce cas, pour extraire la valeur originale, nous sommes obligés de nous baser sur des observations, et des informations que nous connaissons a priori sur le système de conversion.

Nous savons qu'en absence de ces bruits, il existe diverses solutions formelles utilisant des transformations linéaires intégrales, mais dans la pratique, on ne peut envisager de solutions à ces problèmes qu'en ajoutant des contraintes physiques externes. Ce sont des problèmes mal-posés [MAS95a], et leur résolution n'est pas du tout évidente. Il faut donc traiter ces signaux à l'aide d'algorithmes, si nécessaire, tel que ceux de Tikonov[CRI91], de Jansson[CRI91], de Gold, de Kalman [CRI91] etc.

Notons que jusqu'à très récemment, les filtres numériques étaient réalisés, pour la plupart d'entre eux, par programmation des systèmes à base de processeurs tels que les DSP. Avec l'avènement des circuits VLSI, qui deviennent de plus en plus rentables compte tenu des coûts de fabrication de plus en plus bas, nous pouvons envisager l'implantation de filtres de correction numériques dédiées en utilisant du silicium.

Les objectifs visés dans ce mémoire peuvent se résumer en deux grands volets. Le premier volet consiste à apporter une amélioration nécessaire à un algorithme itératif basé sur le filtre de Kalman avec une contrainte de positivité dénommé ITERKAL+[MAS95a]. Pour

cela, nous proposerons des critères d'arrêts du nombre d'itération afin de maximiser la qualité de la reconstitution, dont la détermination en temps réel pose des difficultés dans les méthodes itératives de reconstitution [MAS95a]. Quand au second volet, il sera consacré à l'étude d'architecture en technologie VLSI dédiée à ITERKAL+, à l'aide d'outils et méthodes pour la conception assistée par ordinateur.

Compte tenu du fait que ITERKAL+ est basé sur le filtre de Kalman, il apporte l'avantage d'un traitement en temps réel, et par l'utilisation de la version stationnaire de Kalman, il profite des qualités algorithmiques tels que la simplicité de calcul, la récursivité, le nombre relativement faible d'opérations élémentaires par cycle et la possibilité d'ordonnancement des opérations en parallèle [MAS95b].

Ce mémoire est organisé en trois chapitres. Dans le chapitre 2, nous faisons un rappel sur les systèmes de mesure et les techniques de traitement de signaux de mesure, puis nous élaborons la modélisation mathématique du problème de reconstitution des mesurandes. Dans le chapitre 3, nous explicitons le filtre de Kalman ainsi que l'algorithme itératif de reconstitution basé sur ce filtre, puis nous présentons des critères ayant permis la détermination optimale du nombre d'itération de la méthode itérative et les résultats de reconstitution obtenus sous ces critères. Dans le chapitre 4, nous proposons trois architectures VLSI dédiée à la méthode itérative du filtre de Kalman, et le résultat de leur évaluation sera présenté.

Finalement, nous terminons l'étude avec une conclusion concernant l'algorithme itératif basé sur le filtre de Kalman, du critère proposé du nombre d'itérations optimal en vue d'une implantation en technologie VLSI et des architectures proposées.

CHAPITRE 2

LA RECONSTITUTION DU MESURANDE DANS LES SYSTÈMES DE MESURE

2.1 Définitions des caractéristiques d'un système de mesure

2.1.1 La mesure

Par définition, la mesure est un procédé opérationnel d'application des nombres aux éléments d'une certaine classe d'aspects ou de caractéristiques de l'univers suivant des règles bien définies [MAS95a]. C'est elle qui permet de définir quantitativement, les propriétés des objets, et de vérifier numériquement des lois, ou d'en établir empiriquement la forme. La figure 2.1 représente le schéma général d'un système de mesure, où x est le mesurande, $\hat{x}$ le résultat final de mesure et $\tilde{y}$ le résultat fourni par l'instrument de mesure, soit le résultat de conversion.

La transformation de l'information de mesure du mesurande au résultat final de mesure, soit de $\{x\} \rightarrow \{\hat{x}\}$, peut être logiquement décomposée en deux parties:
la conversion et la reconstitution.

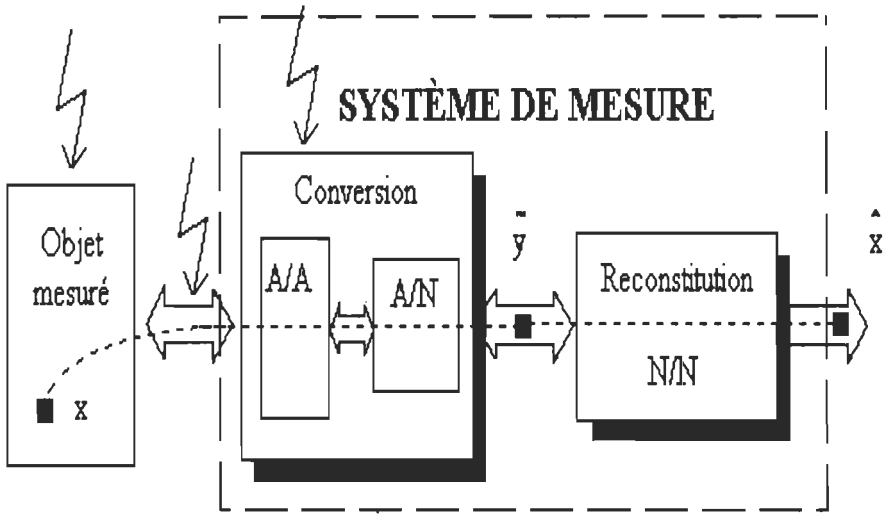


Figure. 2.1 Schéma général d'un système de mesure

2.1.2 La conversion

La conversion est une opération qui consiste à transmettre l'information de mesure dans des domaines de phénomènes physiques facilement interprétables, soit de $\{x_n\} \rightarrow \{\tilde{y}_n$.

Ainsi, dans un système de mesure, le bloc de conversion sera une série de transformations de signaux électriques comme :

- *La conversion A/A* : Il s'agit là d'une conversion analogique - analogique (A/A) des grandeurs de différentes natures physiques en grandeurs électriques. Cette étape de la conversion est le premier élément de la mesure. Sa fonction est assurée par un capteur ou

transducteur approprié, qui traduit le mesurande en un signal interprétable suivant une loi, une propriété connue, ou même suivant une condition physique que l'on veut déterminer.

- *La conversion A/N et N/A de l'amplitude* :L'utilisation d'un ordinateur requiert toujours une conversion préalable de tous les signaux analogiques en digitaux (conversion A/N des tensions électriques dans le cas d'une mesure électrique).

Le convertisseur A/N fait correspondre à une grandeur d'entrée continue, variable et de valeur bien définie à chaque instant, une succession de messages numériques, c'est à dire, faire correspondre à chaque code numérique, une amplitude discrète valable à l'instant d'échantillonnage.

- *La conversion A/N et N/A de l'intervalle de temps* : Le bloc de conversion analogique - numérique du temps, assure la discrétisation de l'échelle du temps en des intervalles égaux. Il constitue la base de la mesure numérique du temps, de la fréquence, et de leurs dérivées. Il en est de même de la conversion numérique - analogique du temps; du discret au continu. Elle est surtout utilisée dans la programmation des valeurs standards du temps.

- *La conversion N/N des données* : Ce dernier bloc occupe une place très importante dans la chaîne de mesures électriques. Il assure le traitement numérique des données issues des blocs de conversion A/N. On peut lui assigner des tâches de filtrage, de compensation des non linéarités des capteurs, d'analyse statistique de données mesurées, etc.

2.1.3 La reconstitution

Le bloc de reconstitution, permet l'estimation du mesurande, en se basant sur le résultat de conversion, soit de $\{\tilde{y}_n\} \rightarrow \{\hat{x}_n$. À partir des résultats de mesure $\tilde{y}$, et de la connaissance

d'un modèle mathématique de la relation entre le mesurande x et $\tilde{y}$, l'utilisateur sera amené à estimer la valeur de x. La relation mathématique entre x_n et y_n se présente sous l'une des deux formes suivantes:

$$\{y_n\} = \mathbf{G}[\{x_n\}] \qquad (2.1) \text{ ou}$$

$$\{\hat{x}_n = \mathbf{R}[\{\tilde{y}_n\}] \qquad (2.2)$$

où G représente un opérateur qui transforme le mesurande $\{x_n\}$ en données de mesure sans erreur $\{y_n\}$ et R est un opérateur à identifier à l'aide des données $\{\hat{x}_n\}$ et $\{\tilde{y}_n\}$. La figure 2.2 montre bien les différentes étapes explicitées dans une sonde spectrométrique.

Il faut noter qu'il existe plusieurs méthodes de reconstitution et les plus courantes sont les suivantes [SLI91]:

- *La méthode directe de reconstitution*: Elle consiste à trouver la solution directe des équations algébriques résultant de la discrétisation des équations (2.1) ou (2.2).

- *La méthode variationnelle de reconstitution*: Elle consiste à contraindre l'ensemble des solutions de signaux, qui minimisent un critère d'erreur donné définissant la qualité de la reconstitution.

- *La méthode probabiliste de reconstitution*: Elle consiste à contraindre l'ensemble des solutions en faisant certaines d'entre elles plus probables que d'autres. Cette méthode utilise des informations a priori statistiques sur les signaux et les bruits qui les affectent.

- *La méthode paramétrique de reconstitution*: Elle consiste à modéliser le signal x (t) avec une fonction connue et paramétrée. On essaie donc de déterminer les paramètres P, en minimisant l'erreur entre la mesure réelle et le résultat de mesure obtenu.

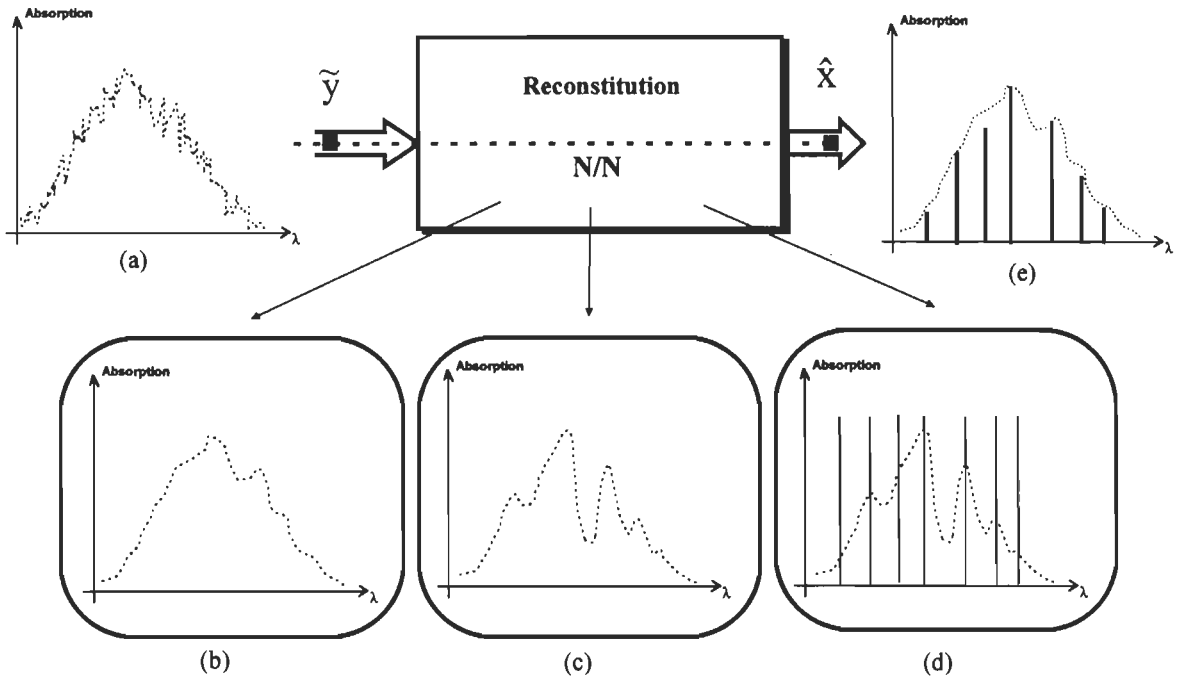


Figure. 2.2 Exemple d'une étape de traitement numérique dans une sonde spectrométrique (a) Signal brut à la sortie d'un capteur spectrométrique, (b) résultat de traitement par le filtrage d'entrée, (c) résultat de la déconvolution, (d) positions des pics, (e) désignant le résultat final de la mesure de la sonde spectrométrique.

- *La méthode itérative de reconstitution*: Elle consiste à appliquer un processus itératif, jusqu'à la convergence à la solution optimale.

- *La méthode de transformée*: Elle consiste à la formulation des contraintes en utilisant différents domaines de transformation tel que le domaine spectral et le domaine cepstral.

2.1.4 Filtres et filtrages numériques

En général, on utilise un filtre pour effectuer trois opérations de base pour le traitement des signaux :

- *Filtrage*: extraction de l'information utile d'un signal à l'instant n en utilisant les données mesurées jusqu'à l'instant n inclusivement

- *Lissage*: l'information utile peut ne pas être disponible à l'instant n et les données mesurées après l'instant n peuvent être utilisées pour obtenir cette information. Ainsi, dans le cas de lissage, il y a un délai pour produire le résultat à l'instant n. Puisque le processus de lissage utilise non seulement des données mesurées jusqu'à l'instant n mais aussi après cet instant, on peut prévoir que ce processus sera plus précis que celui du filtrage.

- *Prédiction* : concerne le traitement prévisionnel de l'information. Il s'agit de prévoir la valeur de l'information utile à un instant n+M dans le futur, pour un M>0, en utilisant les données mesurées jusqu'à l'instant n inclusivement.

2.1.5 Caractéristiques métrologiques des systèmes de mesure

Les performances d'un système de mesure sont liées aux caractéristiques métrologiques des éléments et des instruments de mesure qui le constitue. Ces caractéristiques sont nombreuses, et les principales sont [SLI91]:

- *Exactitude des mesures*: elle se définie comme la valeur exacte à mesurer.

- *Rapidité* : un système est dit rapide s'il est capable de suivre l'évolution de la grandeur d'entrée.

- *Sensibilité* : elle est définie comme le quotient de l'accroissement de la réponse par l'accroissement correspondant du signal d'entrée.

- *Mobilité* : c'est l'aptitude d'un instrument de mesure à réagir aux petites variations du signal d'entrée.

- *Justesse*: c'est l'aptitude d'un instrument à donner des indications conventionnellement vraie de la grandeur mesurée.

- *Fidélité* : c'est l'aptitude d'un instrument à donner la même indication pour une même valeur de la grandeur mesurée.

- *Finesse (discrétion)* : elle caractérise l'aptitude d'un instrument à ne pas modifier la valeur de la grandeur mesurée.

2.2 Modélisation et identification d'un système

2.2.1 La modélisation

La modélisation des relations entre les entrées et les sorties d'un système par une analyse graphique ou statistique, consiste à déterminer des modèles de représentation ou modèles de type boîte noire. La démarche de modélisation comporte une phase de caractérisation, c'est à dire une phase qui consiste à déterminer une classe de modèle, à savoir le choix des variables, la nature des relations qui lient ces variables etc.

Il est illusoire de rechercher un modèle parfait, qui, s'il existait, nécessiterait de toute façon l'utilisation d'un nombre trop grand de paramètres pour être exploitable en système de mesure. Dans le cas de la reconstitution, les incertitudes qui affectent le modèle sont multiples. En plus de perturbations, il existe par ailleurs de nombreuses sources de non-

linéarités, dont certaines sont difficilement modélisables. Elles peuvent être dues au système lui-même, ou à des phénomènes parasites. Il n'y a donc pas, en général, un seul modèle possible du système de mesure mais plusieurs modèles dont chacun restitue une ou plusieurs caractéristiques importantes du système. On a donc une représentation multi-modèles d'un même système, d'où, la difficulté de limiter leur nombre pour ne retenir que les plus représentatifs.

Une autre façon de percevoir le problème de modélisation, plus souvent utilisée car plus générale dans sa formulation, consiste à associer à un modèle élémentaire un ensemble d'incertitudes, celles-ci étant dues à l'impossibilité de décrire complètement le système. On peut toujours, en partant d'un modèle nominal et des incertitudes qui l'affectent, exprimer plusieurs modèles du même système. Inversement, il est possible d'extrapoler une structure d'incertitude à partir de différents modèles. Une approche idéale de modélisation doit donc permettre de construire une représentation nominale et un ensemble de perturbations, pour aboutir à un ensemble de modèles simples le plus réduit possible et le plus proche de la physique du système. Malheureusement, si le premier point est souvent réalisable, il n'existe pas réellement de méthode générale pour déterminer et exprimer les incertitudes. C'est donc ici à l'expérimentateur d'utiliser au mieux sa connaissance, du système et de l'approche de modélisation qui a conduit à la représentation nominale, pour identifier les perturbations, les formuler, et les rendre exploitables.

2.2.2 L'identification

La phase d'identification est une étape intermédiaire voire indispensable, dont les objectifs finaux peuvent être :

- d'appréhender le fonctionnement du système,

- d'élaborer une stratégie de commande ou de contrôle,

- de valider des données,

- de diagnostiquer l'état de fonctionnement d'un système,

- de prédire ou de maîtriser le comportement d'un système,

- de minimiser ou de prendre en compte les effets de perturbations.

Il arrive, par exemple, que l'identification complète d'un système multi-variable ne soit pas possible, soit parce que certaines entrées ou sorties ne peuvent être mesurées ou parce que les conditions d'essai sont différentes des conditions d'utilisation. Une des approches que l'on peut alors utiliser consiste, en partant d'un modèle approché, à essayer de le recaler, même partiellement, à partir des données de l'identification. Dans un second temps, on utilise ce modèle pour estimer les variables non mesurables et prédire le comportement du système dans d'autres conditions d'essai. En somme, l'identification consiste à déterminer les paramètres numériques des paramètres du modèle choisi, ce qui permettra de faire des tests sur des données autres que celles qui ont servie à l'étape d'identification.

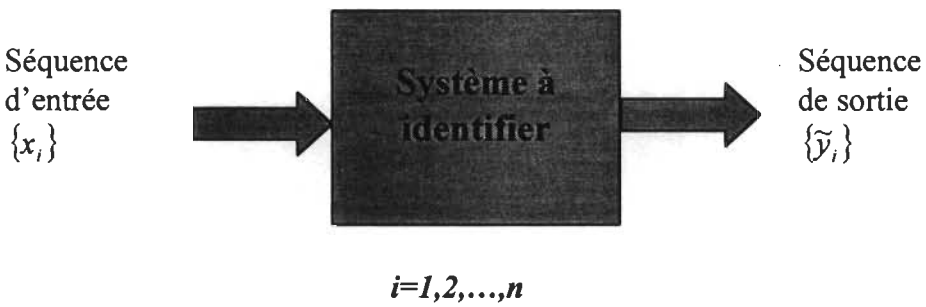


Figure. 2.3 Exemple d'une étape de traitement numérique dans un système à identifier où N est le nombre d'observation.

2.3 Formulation du problème de reconstitution

Sous sa forme la plus simple, un système de mesure peut se réduire à un instrument de mesure qui, si il était parfait, devrait fournir une information liée strictement à la grandeur à mesurer ou au mesurande.

Or, nous savons qu'un instrument de mesure est toujours caractérisé par sa réponse impulsionnelle g(t), qui dans le cas où le système de mesure est linéaire et invariant, affecte le mesurande x(λ) par convolution [BEL90]:

$$y(\lambda) = x(\lambda) * g(\lambda) = \int_{-\infty}^{+\infty} g(\lambda - \lambda')x(\lambda')d\lambda' \qquad (2.3)$$

Ainsi, la première étape pour l'analyse du problème de reconstitution sera la modélisation des données de mesure qui n'est pas une tâche facile. Elles sont généralement discrètes, en nombre fini et souvent entachées d'erreurs de mesure ou bruits. Elles peuvent être représentées sous forme d'un vecteur:

$$\tilde{\mathbf{y}} = \left[\tilde{y}_0 | \tilde{y}_1 | ... | \tilde{y}_{N-1}\right]^T = \dot{\mathbf{y}} + \tilde{\mathbf{y}}_n \qquad (2.4)$$

avec $\tilde{y}_n = y(\lambda_n) + \eta_n$

et $\eta = \left[\eta_0 | \eta_1 | ... | \eta_{N-1}\right]$ (2.5)

$$\dot{y} = \left[y_0 | y_1 | ... | y_{N-1}\right] \qquad (2.6)$$

avec η_n représentant les erreurs ou bruits, liés à la connaissance expérimentale de $\tilde{\mathbf{y}}$.

2.3.1 Sources d'erreur de reconstitution

Nous distinguons deux catégories de sources d'erreur qui peuvent influencer la résolution et l'exactitude de l'opération de reconstitution du mesurande:

- Erreurs causées suite à l'acquisition des données de mesure qui sont les conséquences d'un bruit aléatoire affectant les résultats de mesure.
- Erreurs causées par le bloc de reconstitution qui sont les conséquences de la discrétisation et de la quantification nécessaire à un traitement numérique de l'opération de reconstitution.

Cette dernière catégorie d'erreur est due à la discrétisation du modèle de mesure, par exemple, approximation d'une intégrale par une somme finie, et aux erreurs de quantification dues à la représentation des données par un nombre de bit fini, nécessaire à un traitement numérique de l'opération de reconstitution. Il faut dire que ces deux grandes classes d'erreurs sont très préoccupantes, ce qui fait que leur résolution est souvent abordée dans la littérature des traitements de signaux [BEL90].

2.3.2 Mal - conditionnement numérique du problème de reconstitution

En fait, un problème est dit mal posé, s'il ne satisfait pas à au moins une des aux exigences suivantes [SLI96]:

- la solution existe;
- la solution est définie de façon unique;

- la solution est stable.

Par exemple, nous admettons que $x(\lambda)$, $y(\lambda)$ et $g(\lambda)$ possèdent une transformée de Fourrier, alors nous pouvons écrire l'équivalent de l'équation (2) dans le domaine fréquentiel sous la forme:

$$Y(j\omega) = X(j\omega).G(j\omega) + N(j\omega), \tag{2.5}$$

où $X(j\omega) = F\{x(\lambda)\}$, $Y(j\omega) = F\{y(\lambda)\}$, $G(j\omega) = F\{g(\lambda)\}$, et $N(j\omega) = F\{\eta(t)\}$.

Ainsi, en procédant par filtrage inverse [ALB84]; on obtient:

$$X(j\omega) = \frac{Y(j\omega)}{G(j\omega)} - \frac{N(j\omega)}{G(j\omega)} \tag{2.6}$$

Pour qu'une solution à (2.6) existe, il faut que $\frac{1}{G(j\omega)}$ existe, et que $G(j\omega)$ ne doit pas s'annuler pour aucune valeur de fréquence ω et qu'il ne tende pas vers zéro à l'infini plus vite qu'une puissance de $\frac{1}{\omega}$, pour que nous ayons une solution unique [SLI96].

Dans la pratique, $G(j\omega)$ présentera des passages par zéro et la division $\frac{1}{G(\omega)}$ est pratiquement impossible.

Et d'autre part, le rapport $\frac{\mathrm{N}(\omega)}{G(j\omega)}$ peut ne pas admettre de transformée de Fourrier inverse à cause de l'influence des hautes fréquences ω de la fonction aléatoire $N(j\omega)$.

En conclusion, il ressort que la reconstitution n'est pas seulement un problème inverse particulier, et les difficultés rencontrées sont liées à la résolution d'une équation intégrale de première espèce du bruit de mesure. Même s'il existe une solution par filtrage inverse,

les fluctuations des données initiales risquent d'amener de grandes variations sur la solution, ce qui entraîne une instabilité vis-à-vis des faibles fluctuations sur les données.

CHAPITRE 3

ALGORITHME ITÉRATIF DE RECONSTITUTION BASÉ SUR LE FILTRE DE KALMAN

3.1 Le filtre de Kalman

Cette section a pour but principal de familiariser le lecteur au filtre de Kalman avant d'aborder l'algorithme itératif de reconstitution.

3.1.1 Équations du filtre de Kalman

Soit $\mathbf{x}_n$ un vecteur d'état de variables aléatoires de dimension M d'un système dynamique linéaire à temps discret, et soit $\mathbf{y}_n$ un vecteur de dimension N qui représente les données d'observation de ce système. Le modèle du système peut être ainsi décrit dans l'espace des états sous forme de deux équations:

Équation du système ou du processus:

$$\mathbf{z}_{k+1} = \phi_n \mathbf{z}_n + B_n \mathbf{w}_n \qquad (3.1)$$

Équation d'observation ou de la mesure:

$$\mathbf{y}_{n+1} = H_n \mathbf{z}_{n+1} + \mathbf{v}_{n+1} \qquad (3.2)$$

où ϕ est la matrice de transition de dimension $M \times M$ et $\mathbf{w}_k$ est le vecteur générateur du processus ou bruit d'entrée modélisé comme un bruit blanc de moyenne nulle et dont la matrice de corrélation est définie par:

$$\mathbf{E}\left[\mathbf{w}_i \, \mathbf{w}_j^T\right] = \begin{cases} Q_k & \text{si } i = j \\ 0 & \text{si } j \neq i \end{cases} \tag{3.3}$$

$\mathbf{H}$ est la matrice de mesure de dimension $N \times M$ et $\mathbf{v}_k$ est le vecteur de bruit de mesure modélisé comme un bruit blanc de moyenne nulle et dont la matrice de corrélation est:

$$\mathbf{E}\left[\mathbf{v}_i \, \mathbf{v}_j^T\right] = \begin{cases} R_k & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases} \tag{3.4}$$

Les deux bruits $\mathbf{w}_n$ et $\mathbf{v}_n$ sont supposés être statistiquement indépendant, on a:

$$\mathbf{E} = \left[\mathbf{v}_k \, \mathbf{w}_n^T\right] = 0 \;\; \forall \, n,k \tag{3.5}$$

$\mathbf{B}_k$ est le vecteur de commande de la représentation d'état.

L'état initial du système x_0 est supposé de moyenne nulle, de matrice de corrélation P_0 et non corrélé avec $\mathbf{w}_k$ et $\mathbf{v}_k$,

$$\mathbf{E}\left[\mathbf{z}_0\right] = 0, \mathbf{E}\left[\mathbf{z}_0 \, \mathbf{z}_0^T\right] = P_0 \tag{3.6}$$

$$\mathbf{E}\left[\mathbf{z}_0 \, \mathbf{z}_k^T\right] = 0, \mathbf{E}\left[\mathbf{z}_0 \, \mathbf{z}_k^T\right] = 0 \tag{3.7}$$

Le problème consiste à utiliser les vecteurs des données d'observation $\mathbf{y}_1$, $\mathbf{y}_2$,...,$\mathbf{y}_n$, pour trouver pour chaque $n \geq 1$ les estimés au sens quadratique moyenne minimal (QMM) des composantes de l'état $\mathbf{z}_i$ où pour i = n , on a un problème de filtrage, pour i>n on a un

problème de prédiction et pour 1<i<n on a un problème de lissage. Nous avons joint en annexe plus de détail sur la démonstration des équations du filtre de Kalman.

En résumé: Les étapes de calcul du filtre de Kalman sont données par les équations suivantes:

- modèle du système:

$$\hat{\mathbf{z}}_{n+1} = \Phi_n \hat{\mathbf{z}}_n + \mathbf{w}_n \tag{3.8}$$

$$\hat{\mathbf{z}}_{n+1} = \mathbf{H}_{n+1} \hat{\mathbf{x}}_{n+1} + \mathbf{v}_{n+1} \tag{3.9}$$

- la prédiction:

$$\hat{\mathbf{z}}_{k+1/k} = \Phi_n \hat{\mathbf{z}}_{n/n} \ , \hat{\mathbf{x}}_{0/0} = \mathbf{0} \tag{3.10}$$

$$\hat{\mathbf{y}}_{n+1/n} = \mathbf{H}_{n+1} \hat{\mathbf{x}}_{n+1/n} \tag{3.11}$$

où la notation $\hat{\mathbf{v}}_{k+1/k}$ est l'estimé linéaire au sens quadratique moyenne minimale de $\mathbf{v}$ à l'instant k+1 compte tenu des données à l'instant k.

- la correction:

$$\hat{\mathbf{x}}_{k+1/k+1} = \hat{\mathbf{x}}_{k+1/k} + \boldsymbol{K}_{K+1} \ \Delta \mathbf{y}_{k+1/k} \tag{3.12}$$

$$\Delta \mathbf{y}_{k+1/k} = y_{k+1} - y_{k+1/k} \tag{3.13}$$

- le gain de Kalman:

$$\boldsymbol{K}_{k+1} = \mathbf{P}_{k+1} \mathbf{H}_{k+1}^{\mathrm{T}} [\mathbf{H}_{k+1} \mathbf{P}_{k+1/k} \mathbf{H}_{k+1}^{\mathrm{T}} + \mathbf{R}_{k+1}]^{-1} \tag{3.14}$$

- la matrice de covariance d'erreur de prédiction:

$$\mathbf{P}_{k+1/k} = {}_{k}\mathbf{P}_{k/k} \ {}_{k}^{\mathrm{T}} + \mathbf{B}_k \ \mathbf{Q}_k \mathbf{B}_k^{\mathrm{T}}, \ \mathbf{P}_{0/0} = \mathbf{P}_0 \tag{3.15}$$

- la matrice de covariance d'erreur d'estimation:

$$\mathbf{P}_{k+1/k+1} = [\mathrm{I} - \mathrm{K}_{k+1}\mathrm{H}_{k+1}]\mathrm{P}_{k+1/k} \tag{3.16}$$

Avec $\mathbf{R}_k = \mathbf{E}[\mathbf{w}_k \mathbf{w}_k^T]$ la matrice d'auto-corrélation du bruit de mesure.

On peut remarquer que les équations (3.15) et (3.16) sont indépendantes des mesures $\{\tilde{y}_K\}$, donc elles peuvent être calculées avant même que ceux-ci ne soient traités par le filtre. En effet, ces trois équations dépendent uniquement des paramètres du modèle du système et des caractéristiques statistiques des processus du bruit $\mathbf{v}_k$ et $\mathbf{w}_k$.

3.1.2 Filtre de Kalman stationnaire

Si le modèle stochastique du système est invariant dans le temps, alors on a:

$$\mathbf{z}_{k+1} = \Phi_k \mathbf{z}_k + \mathbf{w}_k \tag{3.17}$$

$$\tilde{\mathbf{y}}_{k+1} = \mathbf{H}_{k+1}\hat{\mathbf{x}}_{k+1} + \mathbf{v}_{k+1} \tag{3.18}$$

$$\mathbf{z}_{k+1/k+1} = \mathbf{z}_{k+1/k} + \boldsymbol{K}_\infty (\mathbf{y}_{k+1/k+1} - \boldsymbol{H}^T \Phi \ \hat{\mathbf{x}}_{k+1/k}) \tag{3.19}$$

où $\boldsymbol{K}_\infty$ représente le vecteur de gain stationnaire de Kalman de l'équation 3.14, tel que :

$$\boldsymbol{K}_\infty = \boldsymbol{K}_i \quad i \to \infty \tag{3.20}$$

obtenu selon la formule suivante [MAS95]:

$$\sum^{(0/0)} = \boldsymbol{I} \tag{3.21}$$

$$\sum^{(i/i-1)} = \sum^{(i-1/i-1)} {}^{T} + b \quad b^{T} \tag{3.22}$$

$$k_i = \sum^{(i/i-1)} H(1 + H^{T} \sum^{(i/i-1)} H)^{-1} \tag{3.23}$$

$$\sum^{(i/i)} = (I - k_i h^{T}) \sum^{(i/i-1)} \tag{3.24}$$

avec i=1, 2, 3, …

3.1.3 Filtrage de Kalman avec contrainte dans la reconstitution de signaux

En supposant que les mesures sont perturbées par un bruit supposé gaussien et de moyenne nulle, dans le cas de données discrets, nous avons le modèle de données $\{\tilde{y}_n\}$ suivant :

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{b}\,\mathbf{u}_k, \quad \mathbf{z}_0 = \mathbf{0} \tag{3.25}$$

$$\tilde{\mathbf{y}}_k = \mathbf{h}^{T}\,\mathbf{z}_k + \eta_n \tag{3.26}$$

où h est la réponse impulsionnelle normalisée du système

$h = c^{-1}\left[g_1 | g_2 | \ldots | g_{M-1} | g_M\right]^{T}$, $dim(h)=dim(g)=M \times 1$ avec $c = \sum_{m=1}^{M} |g_m|$ et g la réponse impulsionnelle du système de conversion.

La matrice Φ et le vecteur **b** sont invariants et de formes canoniques commandables

$$\Phi = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{bmatrix}, \quad \dim \Phi = M \times M$$

$b = \left[0 | 0 | \ldots | 0 | 0 | 1 |\right]^{T}$, $dim\ (b)=M$

Dans [MAS95a] nous retrouvons l'application d'une contrainte de positivité au filtrage de Kalman pour la reconstitution de signaux. Cette approche est basée sur le fait que dans les analyses spectrométriques, le spectre d'absorption est toujours positif, ce qui fait que l'incorporation d'une contrainte de positivité dans la classe des solutions admissibles entraîne une précision des résultats de reconstitution. La contrainte de positivité appliquée à l'algorithme de reconstitution consiste en l'ajout de l'équation supplémentaire suivante:

$$\hat{z}_{n+1/n+1,m} = \begin{cases} \alpha \hat{z}_{n+1/n+1,im} & \text{si } \hat{z}_{n+1/n+1,m} \leq 0 \\ \hat{z}_{n+1/n+1,im} & \text{si } \hat{z}_{n+1/n+1,m} > 0 \end{cases} \qquad \text{pour m=1,2,...,M} \qquad (3.27)$$

où $\hat{z}_{n+1/n+1,m}$ est le m^{ième} élément du vecteur $\hat{z}_{n+1/n+1}$, et α est une constante empitique qui permet l'amélioration des estimations. Le vecteur d'état $\hat{z}_n$ contient les échantillons $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n$ et le mesurande est reconstitué à partir de la connaissance à priori que l'on a sur la réponse impulsionnelle g_n.

> [MAS95] : '' *En pratique, nous devons tronquer g_n pour diminuer la taille des vecteurs et matrices. Ainsi, le vecteur de gain stationnaire de Kalman k_∞ est dans ce cas de même dimension que la réponse impulsionnelle. Dans le cas où la réponse impulsionnelle est symétrique et se rapproche d'une gaussoïde, la méthode de reconstitution de Kalman apporte des oscillations sur le résultat de reconstitution même sans bruit (η_n = 0). Dans ces conditions, l'algorithme de reconstitution basé sur le filtre de Kalman avec une contrainte " rigide " cause une divergence du vecteur $\hat{z}_n$ et nous faisons face à un problème d'instabilité. On a également observé que cette divergence, lors de l'application de la contrainte " rigide ", semble être*

associée à la symétrie de g_n, à sa largeur de bande en fréquence trop limitée et à sa troncature qui peut être trop sévère''.

Une des solutions à ce problème a été proposée dans [MAS95] et consiste à augmenter la largeur de bande en fréquence de la réponse impulsionnelle et de la réduire progressivement à chaque itération de l'application de Kalman avec une contrainte " rigide " pour une réponse impulsionnelle g_n. Cet algorithme est nommé ITERKAL+ c'est à dire un algorithme itératif de reconstitution basé sur le filtre de Kalman avec une contrainte de positivité.

3.2 Algorithmes itératifs de reconstitution de signaux

Compte tenu de leur simplicité, les algorithmes de reconstitution itératifs sont beaucoup utilisés par les chercheurs dans le domaine de la reconstitution de signaux. Ces algorithmes consistent à contraindre l'ensemble des solutions admissibles à celui des solutions qui peuvent être générées par une procédure itérative:

$$\mathbf{x}^{k+1} = J(\mathbf{x}^k) \tag{3.28}$$

où k est le nombre d'itération et J un opérateur qui peut être déterminé à partir du modèle de données. Mais l'on est confronté à des problèmes tels que la forte quantité de calcul dû au très grand nombre d'itérations demandés pour la convergence à la solution optimale. Or en utilisant une méthode itérative pour la résolution d'un problème, une règle d'arrêt doit être fixée pour terminer les itérations. En générale, cette règle d'arrêt est basée sur l'évolution de l'erreur d'estimation déterminée par la norme :

$$\left\|\mathbf{x}^{k+1} - \mathbf{x}^k\right\| \leq \varepsilon, \tag{3.29}$$

où l'évolution de l'erreur résiduelle est évaluée par la norme:

$$\left\| \mathbf{y} - \mathbf{M}\left[\mathbf{x}^{k}\right] \right\| \leq \gamma , \qquad (3.30)$$

où $\varepsilon > 0$ et $\gamma > 0$ sont des seuils d'arrêt choisis au début des itérations de façon empirique [BIL92]. Notons qu'on ne peut étudier d'une façon théorique que la dernière phase de la convergence lorsque $\hat{\mathbf{x}}^{k+1}$ est voisin de $\hat{\mathbf{x}}^{k}$. Ainsi, de nombreux algorithmes itératifs, tel que celui de Gold [CRI91], Van Cittert [CRI91], Jansson [MOR97] etc, ont été élaborés dans le domaine des traitements de signaux pour déterminer $\hat{\mathbf{x}}$. Nous présenterons brièvement les méthodes de Van Cittert et Jansson qui sont davantage sujet à l'intégration en VLSI et plus près de l'algorithme que nous allons étudier.

3.2.1 Méthode itérative de Van Cittert

L'algorithme de Van Cittert permet une bonne estimation du signal à reconstituer compte tenu de sa simplicité. Ainsi, d'une manière itérative nous améliorons l'estimation à chaque itération k jusqu'à atteindre une estimation satisfaisante.

L'algorithme de Van Cittert est donc régie par les deux équations suivantes :

$$\hat{\mathbf{x}}^{0} = \tilde{\mathbf{y}} \qquad (3.31)$$

$$\hat{\mathbf{x}}^{k+1} = \hat{\mathbf{x}}^{k} + [\tilde{\mathbf{y}} - \mathbf{g} * \hat{\mathbf{x}}^{k}] \qquad (3.32)$$

Le signal observé $\tilde{\mathbf{y}}_{n}$ est pris comme la première approximation du signal estimé $\hat{\mathbf{x}}^{0}$. La correction sur l'estimation $\hat{\mathbf{x}}^{k}$ est obtenue à chaque itération k par l'erreur d'estimation sur le signal observé :

$$\tilde{\mathbf{y}} - \hat{\mathbf{y}} \text{ où } \hat{\mathbf{y}} = \mathbf{g} * \hat{\mathbf{x}}^{k}$$

3.2.2 Méthode itérative de Jansson

À la différence de l'algorithme de Van Cittert, celui de Jansson utilise une fonction de relaxation **r** et se définie par les équations suivantes :

$$\hat{\mathbf{x}}^0 = \tilde{\mathbf{y}} \tag{3.33}$$

$$\hat{\mathbf{x}}^{k+1} = \hat{\mathbf{x}}^k + \mathbf{r}(\hat{\mathbf{x}}^k)(\tilde{\mathbf{y}} - \mathbf{g} * \hat{\mathbf{x}}^k) \tag{3.34}$$

$$r(\hat{x}^k) = c(1 - (2 / (a + b)\left|\hat{x}^k - (a + b) / 2\right|) \tag{3.35}$$

avec $\hat{x}^k$ la valeur estimée de $\dot{x}$ à l'itération k, c une constante empirique, a et b désignant respectivement la valeur minimale et maximale du signal recherché $\dot{x}$. Il faut noter que la fonction de relaxation **r** a pour but de contraindre l'estimation de $\dot{x}$ à rester dans ses limites comprises entre a et b. Elle a aussi pour but de tenir compte des bruits qui entachent le signal afin d'obtenir une meilleure reconstitution de $\hat{x}$.

3.3 Méthode itérative basée sur le filtre de Kalman

3.3.1 Formulation du problème

L'algorithme itératif basé sur le filtre de Kalman proposé dans [MAS97],[MAS95a], [PAN95], [SAN92], se distingue par sa réponse impulsionnelle h(λ), qui est définie de telle sorte que nous considérons la réponse impulsionnelle g(λ):

$$g(\lambda) \cong h^1(\lambda) * h^2(\lambda) * .... * h^K(\lambda) \tag{3.36}$$

où K désigne le nombre d'itération . La réponse impulsionnelle g(λ) est donc considérée comme une convolution factorielle de signaux $h^k(\lambda)$, où les $h^k(\lambda)$ pour k=1,2,...,K, sont des fonctions connues.

Ainsi, le problème de résolution de l'équation (2.3) est transformé en un problème de résolution d'une série de K équations:

$$\tilde{y}(\lambda) = \left(h^1\right)^T z_n^1 + \eta_n \tag{3.37}$$

$$z_{n+1}^k(\lambda) = \Phi z_n^k + b u_n^k \text{ pour k=1,2,...,K} \tag{3.38}$$

$$y_n^k = (h^k)^T zk_n \tag{3.39}$$

pour $k=2,3,\ldots,K$ et n=1,2,...,N

Cette approche peut être modélisée comme suit :

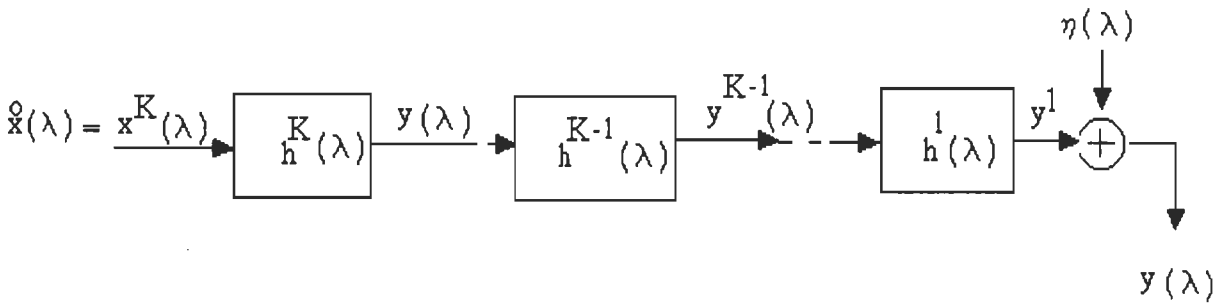


Figure 3.1 exemple de correction de donnée par une architecture d'ITERKAL+

À chaque itération, nous effectuons K fois l'équation:

$$z_{n+1/n+1}^k = z_{n+1/n}^k + K_\infty \left[(y_{n+1}^k - (h^k)^T z_{n+1/n}^k \right. \tag{3.40}$$

Il faut noter qu'avec l'algorithme itératif basé sur le filtre de Kalman, nous obtenons de meilleurs résultats de reconstitution comparativement à celui du filtre de Kalman. La figure. 3.5, illustre bien cet état de fait.

Comme nous pouvons le constater, l'algorithme de Kalman nous donne, dans le cas où la réponse impulsionnelle est symétrique, de mauvais résultats de reconstitution, ce qui n'est pas le cas du filtre itératif de Kalman. Avec ITERKAL+, l'erreur de reconstitution est réduit d'environ 50% par rapport à Kalman+ pour une complexité de calcul équivalente dans certaines applications [MAS95a].

Disons que la performance de l'algorithme itératif basé sur l e filtre de Kalman dépend de trois paramètres qui permettent de minimiser l'erreur de reconstitution au sens quadratique du terme. Ces paramètres sont :

- la longueur de la réponse impulsionnelle h(λ),
- le paramètre de régularisation β définit comme le rapport de la variance du processus générateur σ_u^2 et de la variance du bruit σ_η^2, et
- le nombre d'itération *K*.

En fixant deux des précédents paramètres, le troisième doit être optimisé afin de minimiser l'erreur de reconstitution [MAS95a]. Dans les applications en spectrométrie, nous pouvons fixer la réponse impulsionelle h(λ) en fonction de la connaissance à priori que nous avons de g(λ). Ainsi, nous pouvons fixer la période d'échantillonnage de h(λ) à quatre fois celle de g(λ). Par la suite, pour chaque signaux à traîter, nous fixons β et devons maximiser la qualité de reconstitution en limitant le nombre d'itération K.

3.3.2 Signaux de mesure synthétiques

Notons que tout au long de ce chapitre, nous utiliserons trois principaux signaux synthétiques pour illustrer l'influence de certains paramètres sur les résultats de reconstitution, et faire des comparaisons avec d'autres algorithmes itératifs. Les trois signaux synthétiques choisis, présentent chacun différentes difficultés au niveau de la position des pics et de leur amplitude, ce qui va bien avec les cas pratiques dans le domaine de la spectrométrie. Ces signaux synthétiques sont les suivants:

$$\dot{x}_1(\lambda) = 4\,gaussoid(\lambda - 50;2.25) + gaussoid(\lambda - 70;2.25) \tag{3.41}$$

$$\dot{x}_2(\lambda) = gaussoid(\lambda - 25;2.25) + 4\,gaussoid(\lambda - 70;2.25) + 3\,gaussoid(\lambda - 90;2.25) \tag{3.42}$$

$$\begin{aligned}\dot{x}_3(\lambda) = {} & 8\,gaussoid(\lambda - 25;4) + 4\,gaussoid(\lambda - 30;4) + 6.5\,gaussoid(\lambda - 80;4) \\ & + 2\,gaussoid(\lambda - 100;4) + 3\,gaussoid(\lambda - 150;4) + gaussoid(\lambda - 160;4) \\ & + 4\,gaussoid(\lambda - 170;4) + gaussoid(\lambda - 230;4)\end{aligned} \tag{3.43}$$

$$g(\lambda) = 0.0525\,gaussoid(\lambda;8)$$

$$\text{et} \quad h(\lambda) = gaussoid(\lambda;2) \tag{3.44}$$

$$\text{avec} \quad gaussoid(\lambda,\sigma) = \exp\left[-0.5\left[\frac{\lambda}{\sigma}\right]^2\right]$$

$$y_i(\lambda) = g(\lambda) * x_i(\lambda)\text{, i=1,2,3...} \tag{3.45}$$

$$\text{et} \quad \tilde{y}_{i,n} = \dot{y}_{i,n} + \eta_n \qquad \text{n} = 1,2,3,\ldots,\text{N}$$

La qualité de reconstitution sera évaluée suivant l'erreur relative ε définie par:

$$\varepsilon = \frac{\left\|\{\hat{x}_n\} - \{\dot{x}_n\}\right\|_2}{\left\|\{\dot{x}_n\}\right\|_2} \tag{3.46}$$

3.3.3 Influence de la largeur de la réponse impulsionelle $h(\lambda)$sur l'erreur relative

Dans l'algorithme ITERKAL+, l'utilisation de différentes valeurs de la réponses impulsionnelles $h(\lambda)$, influe énorment sur la qualité de reconstitution. Comme le montre la figure 3.3, plus $h(\lambda)$est étroit, plus K^{opt} augmente, et plus l'erreur optimale ε^{opt} diminue. Mais, plus $h(\lambda)$ est large, plus K^{opt} diminue, et plus la méthode de reconstitution devient sensible au bruit η. D'où, il faut trouver un compromis sur la qualité de la reconstitution versus le nombre d'itération, et la robustesse au bruit.

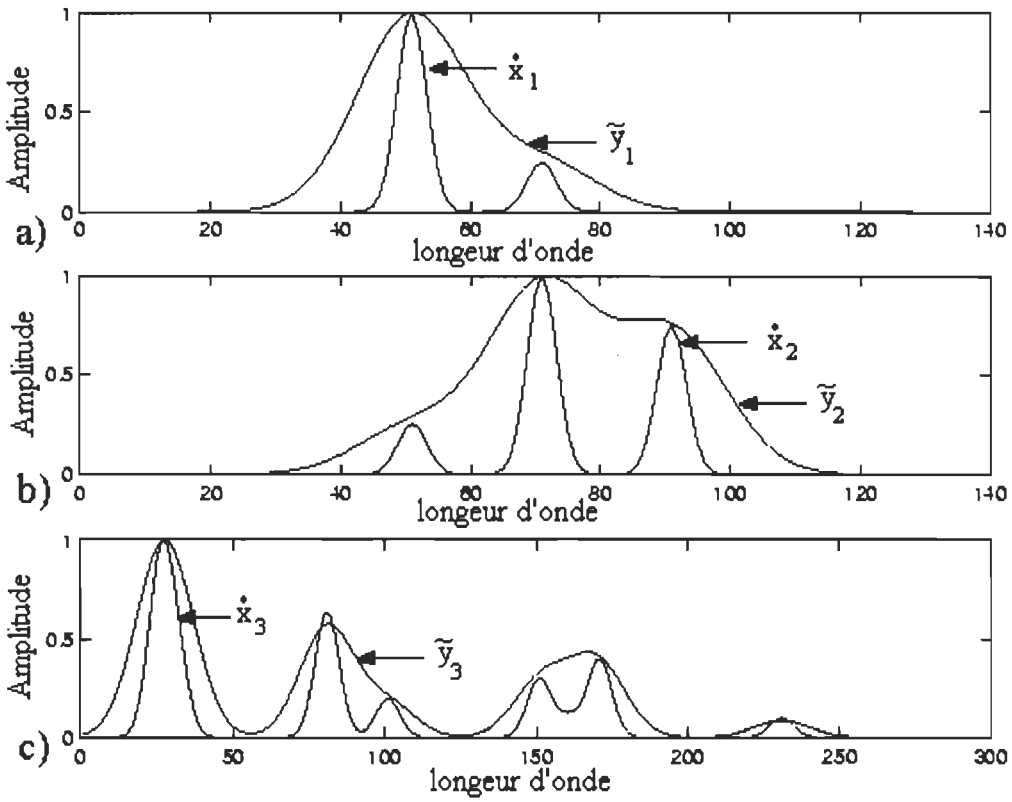


Figure. 3.2 Signaux de mesures synthétiques x_1, x_2, x_3 définis par Eq.(3.34), Eq.(3.35) et Eq.(3.36), ainsi que leur résultat de mesure respective y_1, y_2, y_3.

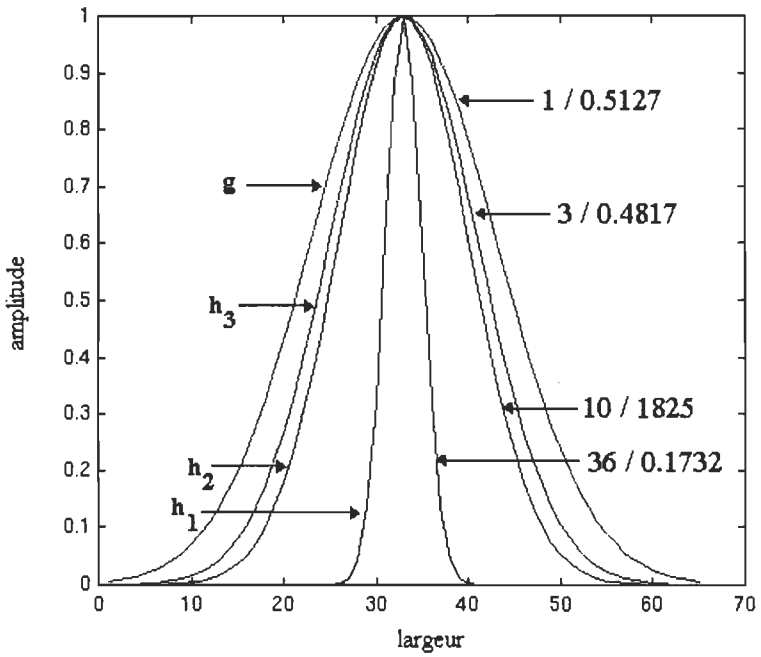


Figure. 3.3 Influence de la largeur de la reponse impulsionelle h sur la qualité de reconstitution ($K^{opt}/\varepsilon^{opt}$) du signal x_1

3.3.4 Influence du paramètre de régularisation β sur l'erreur relative et le nombre d'itération.

Le paramètre de régularisation β défini précédemment, influe énormément sur l'erreur relative définie à l'équation 3.39 et le nombre d'itération . Cela est bien illustré à la figure ci-après, où pour une variation de β au millième près, on observe différentes valeurs de l'erreur relative et du nombre d 'itération dans le cas du signal x_1. D'où pour un β fixe, comme c'est le cas en pratique, il nous est possible de déterminer une itération optimale permettant d'avoir une erreur relative optimale [MAS95a].

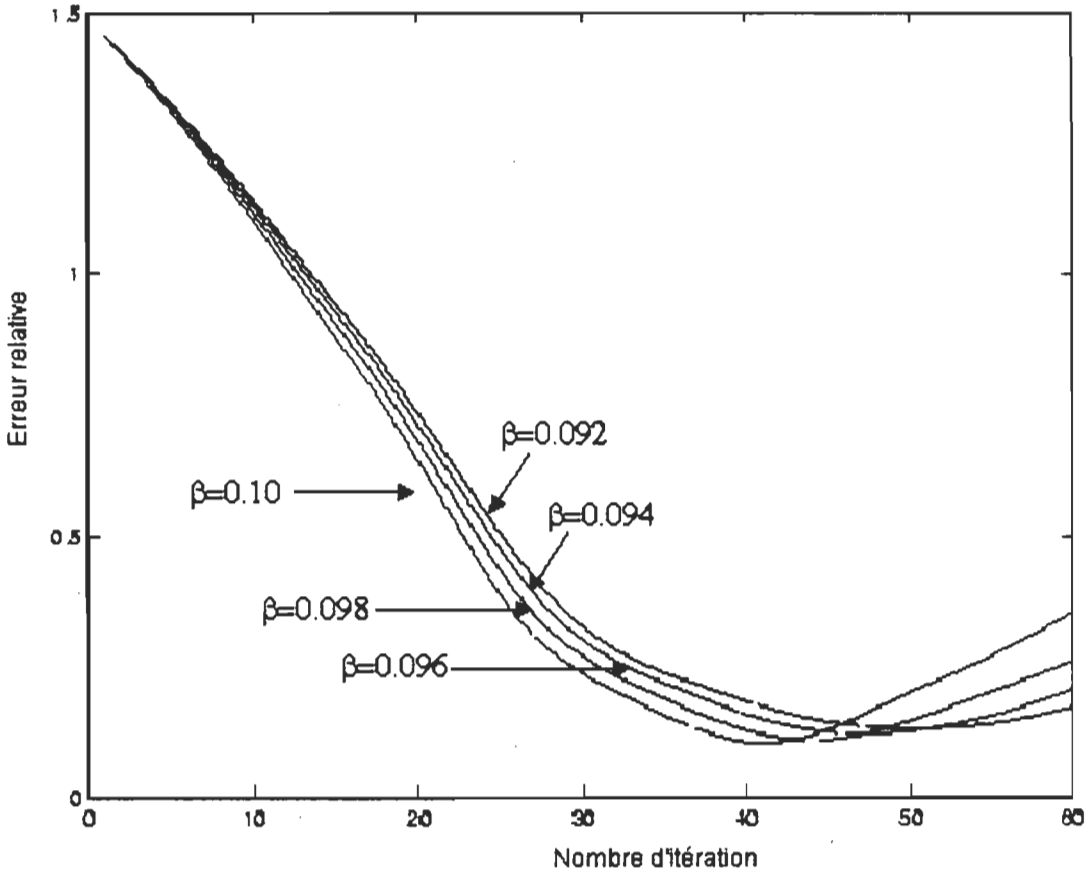


Figure. 3.4 Influence du nombre d'itération et du paramètre β sur l'erreur relative pour un niveau de bruit $\sigma_\eta^2 = 10^{-5}$

La figure 3.4 est une application du signal x_1, mais nous obtenons des courbes similaires pour les autres signaux, à la différence que le minimum des courbes ne se situe pas aux mêmes nombres d'itérations. Les tableaux 3.1 à 3.2 et les figures 3.5 et 3.6 illustrent bien les effets du paramètre de régularisation β sur la qualité de reconstitution et le nombre d'itération.

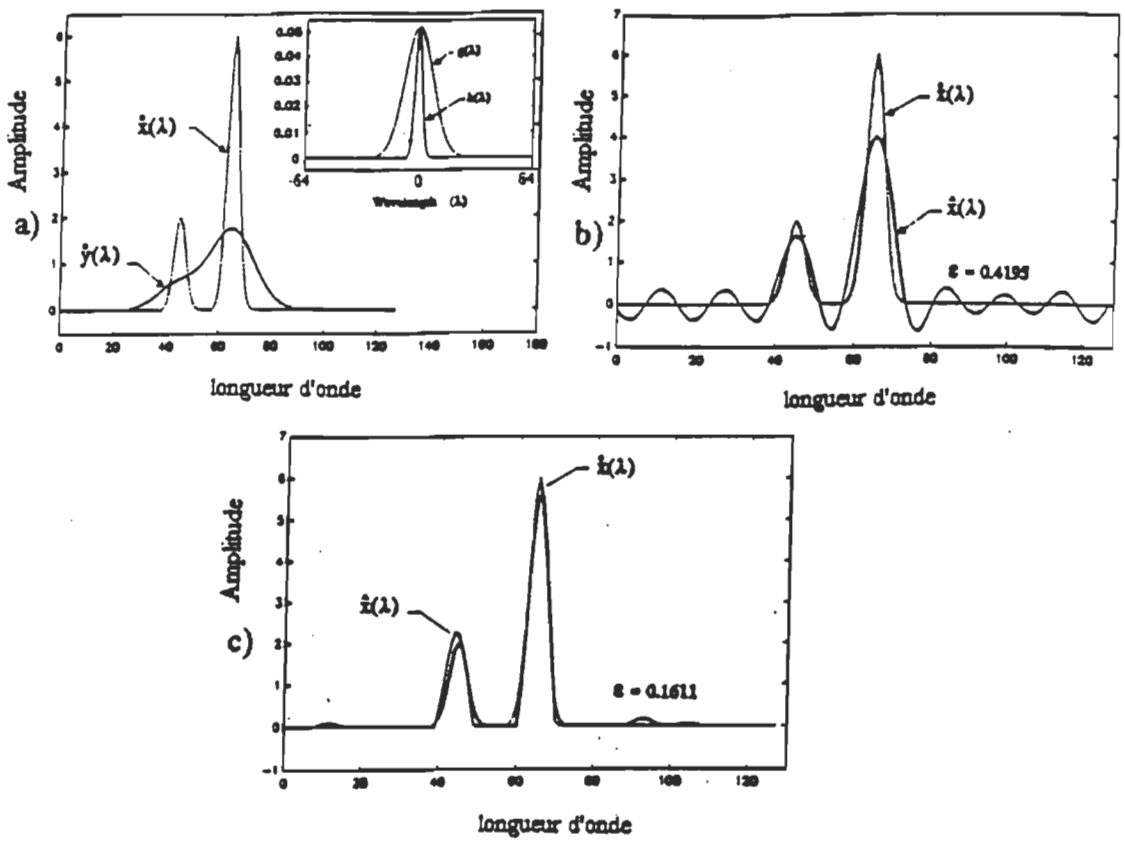


Figure. 3.5 (a) Signaux x, $\tilde{y}$, g et h; (b) résultats de reconstitution à partir de l'algorithme de Kalman; (c) résultats de reconstitution à partir de l'algorithme itératif de Kalman, ces résultats sont tirés de [MAS97].

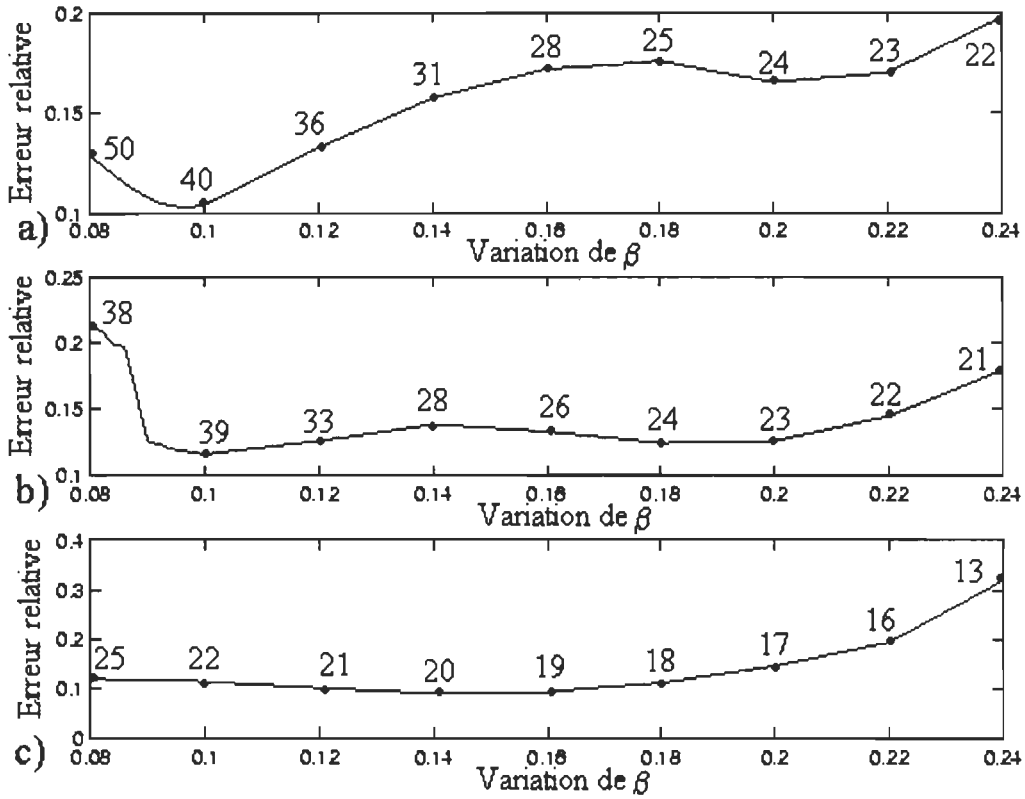


Figure 3.6 Variation du nombre d'itération K_{opt} en fonction de l'erreur relative ε et de β pour $\sigma^2_\eta=10^{-4}$, a) Signal x_1 avec $\sigma^2_\eta=10^{-4}$, b) signal x_2 avec $\sigma^2_\eta=10^{-4}$, c) signal x_3 avec $\sigma^2_\eta=10^{-4}$

Tableau3.1 Variation de l'erreur ε et du nombre d'itération K_{opt} en fonction de β pou les signaux synthétiques x_1, x_2, x_3 définis par Eq.(3.34), Eq.(3.35) et Eq.(3.36), avec $\sigma_\eta^2 = 0$.

β	x_1		x_2		x_3	
	ε	K_{opt}	ε	K_{opt}	ε	K_{opt}
0.080	0.1285	50	0.2131	38	0.1173	25
0.082	0.1237	48	0.2098	37	0.1180	24
0.084	0.1188	47	0.1991	37	0.1168	24
0.086	0.1142	46	0.1978	36	0.1164	24
0.090	0.1065	44	0.1261	43	0.1164	23
0.092	0.1040	43	0.1231	42	0.1151	23
0.094	0.1021	42	0.1202	41	0.1145	23
0.096	0.1011	42	0.1183	40	0.1143	23
0.098	0.1010	41	0.1164	40	0.1145	22
0.10	0.1017	40	0.1152	39	0.1128	22
0.12	0.1287	36	0.1247	33	0.1007	21
0.14	0.1522	31	0.1442	28	0.0897	20
0.16	0.1666	29	0.1500	26	0.0913	19
0.18	0.1777	27	0.1513	24	0.1094	18
0.20	0.1832	25	0.1521	23	0.1442	17
0.22	0.1828	23	0.1517	22	0.1922	17
0.24	0.1773	23	0.1669	21	0.3175	13

On constate bien qu'il existe un nombre d'itération optimal K_{opt} pour une erreur de reconstitution ε minimale correspondant à un β donné. Il faut noter qu'une augmentation du nombre d'itération n'améliore pas indéfiniment l'estimation du mesurande $x(\lambda)$ contrairement aux méthodes itératives usuelles de la section 3.2. [MAS95a].

Tableau 3.2 Variation de l'erreur ε et du nombre d'itération K_{opt} en fonction de β pour les signaux synthétiques x_1, x_2, x_3 définis par Eq.(3.34), Eq.(3.35) et Eq.(3.36), avec $\sigma_\eta^2 = 10^{-4}$.

β	x_1		x_2		x_3	
	ε	K_{opt}	ε	K_{opt}	ε	K_{opt}
0.080	0.1288	50	0.2125	38	0.1173	25
0.082	0.1242	48	0.2091	37	0.1179	24
0.084	0.1194	47	0.1981	37	0.1167	24
0.086	0.1149	46	0.1969	36	0.1164	24
0.090	0.1076	44	0.1262	43	0.1162	23
0.092	0.1052	43	0.1234	42	0.1150	23
0.094	0.1036	42	0.1205	41	0.1143	23
0.096	0.1028	42	0.1188	40	0.1141	23
0.098	0.1029	41	0.1173	40	0.1143	22
0.10	0.1038	40	0.1161	39	0.1126	22
0.12	0.1327	36	0.1252	33	0.1003	21
0.14	0.1574	31	0.1382	28	0.0899	20
0.16	0.1718	28	0.1328	26	0.0928	19
0.18	0.1758	25	0.1242	24	0.1111	18
0.20	0.1652	24	0.1258	23	0.1451	17
0.22	0.1701	23	0.1441	22	0.1936	16
0.24	0.1974	22	0.1785	21	0.3189	13

3.4 Critère d'arrêt optimal du nombre d'itération

Par définition, un critère est une expression mathématique d'une distance entre un point (modèle) et un autre (système) dans l'espace engendré par les paramètres à déterminer. Un critère est également considéré comme une fonction de coût, dont la phase d'estimation des paramètres se réduit bien souvent à la minimisation de cette fonction de coût. Il existe

quatre grandes familles de critères, différenciées par la nature de la distance qu'ils minimisent.

Ces distances sont [KUN91] :

- la distance d'état basée sur l'erreur de sortie (sortie - modèle)
- la distance de prédiction basée sur l'erreur d'équation (procédé - modèle de prédiction)
- la distance de structure cherchant à minimiser la différence entre les paramètres du système et ceux du modèle
- la distance visant à minimiser une erreur d'entrée; c'est une distance qui est utilisée lorsque les mesures des entrées sont entachées d'erreurs.

3.4.1 Critère d'arrêt optimal du nombre d'itération basé sur la méthode L-COURBE

Dans cette section, nous ferons l'étude d'un critère basé sur la méthode du *L-COURBE* [SZC96], [PER93] habituellement appelé L-CURVE. Il sera utilisé pour déterminer automatiquement l'itération optimale donnant la meilleure qualité de reconstitution et la position des pics de données spectrométriques, en utilisant l'algorithme itératif basé sur le filtre de Kalman.

Une étude de la méthode du *L-COURBE* simplifiée a été proposée dans [PER93] pour la détermination de la valeur optimale du paramètre $\alpha_{optimal}$ de régularisation de l'algorithme de Thikonov. Pour cela, on génère une séquence de $\{\alpha_i = 2^i | i = 0,1,2,\ldots\}$ et la séquence correspondante des solutions $\{\hat{x}(\alpha_i)\}$.

La valeur de $\alpha_{optimal}$ sera la valeur de α_i qui satisfera les conditions:

$$\frac{\|\hat{x}(\alpha_i)\|_2}{\|\hat{x}(\alpha_{i-1})\|_2} \geq \frac{\|\tilde{y} - g * \hat{x}(\alpha_{i-1})\|_2}{\|\tilde{y} - g * \hat{x}(\alpha_i)\|_2} \tag{3.47}$$

$$\frac{\|\hat{x}(\alpha_{i-1})\|_2}{\|\hat{x}(\alpha_{i-2})\|_2} \geq \frac{\|\tilde{y} - g * \hat{x}(\alpha_{i-2})\|_2}{\|\tilde{y} - g * \hat{x}(\alpha_{i-1})\|_2} \tag{3.48}$$

avec $\tilde{y}$ correspondant au résultat de mesure, g la réponse impulsionnelle et $\hat{x}(\alpha_i)$ le signal reconstitué avec α_i comme paramètre de régularisation.

L'article propose une procédure qui consiste à trouver une valeur de α pour laquelle nous aurons une croissance relative de $\|\hat{x}(\alpha)\|_2$ égale à une décroissance relative de $\|\Delta\tilde{y}(\alpha)\|_2$ soit pour laquelle la dérivée de la fonction:

$$f(\alpha) = \frac{\partial \log\|\hat{x}(\alpha)\|_2}{\partial \log\|\Delta\tilde{y}(\alpha)\|_2} \tag{3.49}$$

se rapproche de la valeur -1, avec $\Delta\tilde{y}(\alpha) = \tilde{y} - g * \hat{x}(\alpha_{i-1})$

Une solution approximative de l'équation précédente est obtenue en utilisant la méthode de Euler :

$$f(\alpha) \cong \hat{f}(\alpha_i) = \frac{\log\|\hat{x}(\alpha_i)\|_2 - \log\|\hat{x}(\alpha_{i-1})\|_2}{\log\|\Delta\tilde{y}(\alpha_i)\|_2 - \log\|\Delta\tilde{y}(\alpha_{i-1})\|_2} \tag{3.50}$$

et la séquence des estimés $\{\hat{x}(\alpha_i)\}$ correspond à la séquence $\{\alpha_i = 2^i | i = 0,1,2,\ldots\}$, et l'index de la valeur optimale α_i est déterminé selon la règle heuristique :

$$\hat{i} = \arg_i \{\hat{f}(\alpha_{i-1}) \geq -1 \text{ et } f(\alpha_i) \leq -1 \,|\, i = 2,3,\ldots\} \tag{3.51}$$

ce qui est équivalent aux inéquations (3.40) et (3.41). Par similitude, nous pouvons remplacer $\hat{x}(\alpha_i)$ par $\hat{x}^k$ et $G \bullet \hat{x}(\alpha_i)$ par $h * \hat{x}^k$ avec h désignant la réponse impulsionnelle utilisée dans l'algorithme itératif basé sur le filtre de Kalman. Et puisque $\hat{y}^k = h * \hat{x}^k$ par définition, et que $y_n - \hat{y}_n^k$ correspond à l'erreur de modélisation appelée dans le filtre de Kalman l'innovation, on obtient ainsi les inéquations:

$$\frac{\left\|\hat{\mathbf{x}}^{k}\right\|_2}{\left\|\hat{\mathbf{x}}^{k-1}\right\|_2} \geq \frac{\left\|\tilde{\mathbf{y}}_n - \hat{\mathbf{y}}_n^{\,k-1}\right\|_2}{\left\|\tilde{\mathbf{y}}_n - \hat{\mathbf{y}}_n^{\,k}\right\|_2} \qquad (3.52)$$

$$\frac{\left\|\hat{\mathbf{x}}^{k-1}\right\|_2}{\left\|\hat{\mathbf{x}}^{k-2}\right\|_2} \leq \frac{\left\|\tilde{\mathbf{y}}_n - \hat{\mathbf{y}}_n^{\,k-2}\right\|_2}{\left\|\tilde{\mathbf{y}}_n - \hat{\mathbf{y}}_n^{\,k-1}\right\|_2} \qquad (3.53)$$

Nous désignerons ces deux inéquations par le critère J_{LCy}. Par soucis de faciliter l'implantation de ce critère en technologie VLSI, nous proposons une méthode en remplaçant le rapport $\frac{\left\|\mathbf{y}_n - \mathbf{y}_n^{\,k-1}\right\|_2}{\left\|\mathbf{y}_n - \mathbf{y}_n^{\,k}\right\|_2}$ par $\frac{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k-1}\right\|_2}{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k}\right\|_2}$. Ainsi, à partir des inéquations (3.52) et (3.53), nous pouvons déduire deux nouvelles inéquations :

$$\frac{\left\|\hat{\mathbf{x}}^{k}\right\|_2}{\left\|\hat{\mathbf{x}}^{k-1}\right\|_2} \geq \frac{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k-1}\right\|_2}{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k}\right\|_2} \qquad (3.54)$$

$$\frac{\left\|\hat{\mathbf{x}}^{k-1}\right\|_2}{\left\|\hat{\mathbf{x}}^{k-2}\right\|_2} \geq \frac{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k-2}\right\|_2}{\left\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * \ldots * \mathbf{h}^{k-1}\right\|_2} \qquad (3.55)$$

Nous désignerons également ces deux dernières inéquations par J_{LCh}.

On constate bien que les J_{LCy} et J_{LCh} ont été obtenus en substituant le rapport $\frac{\|\tilde{y} - G \bullet \hat{x}(\alpha_{i-1})\|_2}{\|\tilde{y} - G \bullet \hat{x}(\alpha_i)\|_2}$ utilisée[LZC96] , respectivement par $\frac{\|\tilde{y} - y^{k-1}\|_2}{\|\tilde{y} - y^k\|_2}$ dans le critère J_{LCy} et par $\frac{\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * ... * \mathbf{h}^{k-1}\|_2}{\|\mathbf{g} - \mathbf{h}^1 * \mathbf{h}^2 * ... * \mathbf{h}^k\|_2}$ dans le critère J_{LCh}, où g désigne la réponse impulsionelle réelle du système et h^k une réponse impulsionnelle variable à chaque itération k .

3.4.2 Critère d'arrêt optimal du nombre d'itération basé sur l'erreur de modélisation

Une étude sur un critère d'optimisation a été proposée dans [MIL91], et consiste à déterminer le nombre d'itération optimal de l'algorithme de Jansson. Le deuxième critère d'arrêt optimal que nous avons utilisé pour déterminer l'itération optimal donnant le meilleur estimé du signal reconstitué, est basé sur cette méthode. La condition d'arrêt de l'itération est fonction du critère J_x défini par:

$$J_x = \arg_k \inf \left\{ \gamma . \|\hat{x}^k(\lambda)\|_2 + (1-\gamma) . \|\tilde{y}(\lambda) - g(\lambda) * \hat{x}^k(\lambda)\|_2 \,|\, k = 1,2,.. \right\} \qquad (3.56)$$

Où $\gamma \in [0,1]$ est un paramètre de régularisation permettant d'obtenir l'itération optimale donnant la plus petite erreur relative, $\hat{x}^k$ le signal reconstitué à la $k^{ième}$ itération, et $g(\lambda)$ la réponse impulsionnelle telle que définie à la section 3.3. Dans [MIL91], γ est déterminé de façon empirique, mais il est possible de le déterminer avec des algorithmes d'optimisation . Dans le cadre de nos applications, nous avons observé que l'intervalle de variation du paramètre de régularisation γ pouvait être réduit [0 1] à [0.60 1]. La figure 3.7 exprime bien la forte influence de γ sur les résultats de reconstitution des signaux x_1, x_2 et x_3.

La procédure pour déterminer J_x consiste à fixer γ et de minimiser l'équation 3.49 à la suite de plusieurs itérations. C'est ainsi que nous obtenons la valeur de l'itération K_{opt} ayant permis d'avoir l'erreur relative la plus précise. Un exemple sur l'impact de γ sur les signaux synthétiques x_1, x_2, x_3 est présenté à la figure 3.5, et nous constatons qu'il existe bien une valeur de γ permettant l'obtention de l'erreur minimale, ce qui correspond à la valeur de l'itération optimale K_{opt}, et $\varepsilon_\gamma = \dfrac{\|\{\hat{x}_n\} - \{\dot{x}_n\}\|_2}{\|\{\dot{x}_n\}\|_2}$. (3.57)

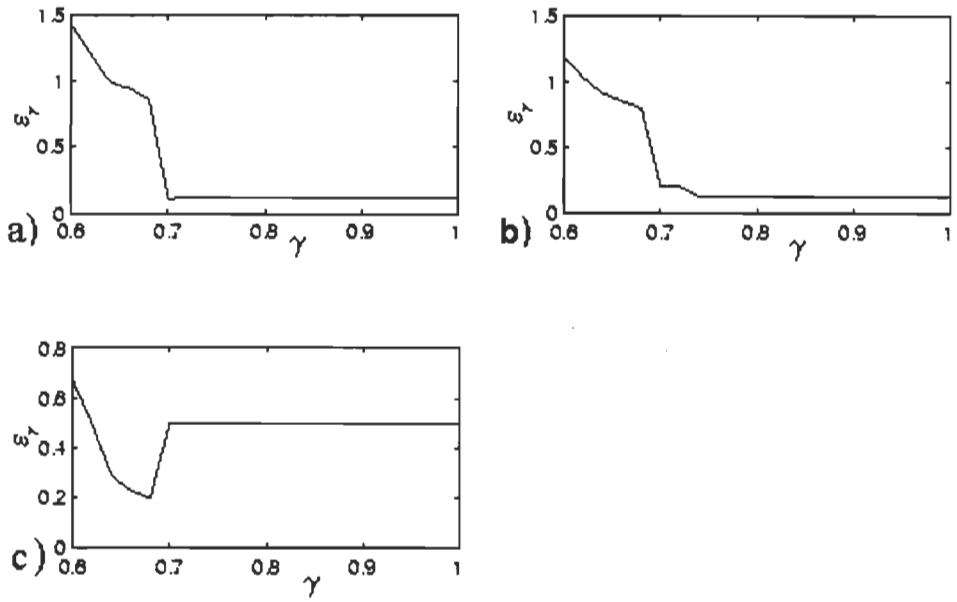


Figure.3.7 Erreur relative ε_γ en fonction de γ, avec $\sigma_\eta^2 = 0$ .a) Erreur relative ε_γ du signal x_1 en fonction de γ, b) Erreur relative ε_γ du signal x_2 en fonction de γ, c) Erreur relative ε_γ du signal x_3 en fonction de γ.

3.4.3 Résultats de reconstitution avec application du critère J_{LCh}

Les critères J_{LCy} et J_{LCh} ont donné pratiquement les mêmes résultats, mais dans un souci de faciliter l'implantation de l'algorithme Iterkal+, nous avons opté pour J_{LCh} , car les valeurs de $\left\|g - h^k\right\|_2$ sont connues d'avance, donc nous pouvons les calculer et les stocker en mémoire.

Notons que $k_{\min}$ correspond au nombre d'itération ayant donné la plus petite erreur relative $\varepsilon_{\min}$, k_{opt} correspond au nombre d'itération obtenu avec le critère J_{LCh}, et ε_{opt} est l'erreur relative associée à k_{opt}, et ε_γ, correspondant à l'erreur relative pour une valeur de γ donné.

Tableau 3.3 Résultats de reconstitution du signal x_1 avec J_{LCh}

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	K_{opt}	$K_{\min}$
0	0.1708	0.1064	36	44
10^{-6}	0.1729	0.1191	36	44
10^{-4}	0.3244	0.3018	41	48

Tableau 3.4 Résultats de reconstitution du signal x_2 avec J_{LCh}

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	K_{opt}	$K_{\min}$
0	0.1619	0.1260	37	43
10^{-6}	0.1761	0.1309	36	43
10^{-4}	0.4524	0.3369	52	32

Notons que pour fin d'étude, la réponse impulsionelle h a été élargie afin d'utiliser 10itérations, plutôt que 36 (tableau3.4).

Tableau 3.5 Résultats de reconstitution du signal x3 avec J_{LCh}

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	K_{opt}	$K_{\min}$
0	0.1871	0.1410	27	24
10^{-6}	0.1883	0.1412	27	23
10^{-4}	0.2402	0.1453	28	23

3.3.4 Résultats de reconstitution avec application du critère J_x

Dans cette section, γ_{opt} est le paramètre de régularisation ayant permis le calcul de k_{opt} dans le critère J_x. Comme précédemment, $k_{\min}$ correspond au nombre d'itération ayant donné la plus petite erreur relative $\varepsilon_{\min}$, k_{opt} correspond au nombre d'itération obtenu avec le critère J_x, et ε_{opt} est l'erreur relative associée à k_{opt}.

Tableau 3.6 Résultats de reconstitution du signal x_1 avec J_x

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	γ_{opt}	K_{opt}	$K_{\min}$
0	0.1077	0.1064	0.70	45	44
10^{-6}	0.1197	0.1191	0.70	46	44
10^{-4}	0.3468	0.3018	0.70	38	48

Tableau 3.7 Résultats de reconstitution du signal x_2 avec J_x

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	γ_{opt}	K_{opt}	$K_{\min}$
0	0.1281	0.1260	0.70	45	43
10^{-6}	0.1326	0.1309	0.69	44	43
10^{-4}	0.1419	0.3369	0.71	39	32

Tableau 3.8 Résultats de reconstitution du signal x_3 avec J_X

σ^2_η	ε_{opt}	$\varepsilon_{\min}$	γ_{opt}	K_{opt}	$K_{\min}$
0	0.1952	0.1410	0.68	19	24
10^{-6}	0.1970	0.1412	0.68	20	23
10^{-4}	0.2117	0.1653	0.66	20	23

Après une simple analyse des tableaux 3.1 à 3.6, nous pouvons conclure que les résultats obtenus avec le critère J_X sont nettement meilleur que celui du critère J_{LCh}, car il permet de déterminer l'itération optimale avec une erreur de plus ou moins une itération. Par contre, l'un de ses inconvénients majeurs est qu'il faut, en plus des itérations, faire plusieurs variations de γ avant d'obtenir K_{opt}. Nous pouvons observer sur les figures 3.6, 3.7 et 3.8, l'allure de la reconstitution des signaux de mesure synthétiques x_1, x_2, x_3 à l'aide du critère J_X.

3.4.5 Comparaison de l'algorithme Iterkal+, Jansson et Van-Cittert

Comme précédemment, nos comparaisons se feront sur les signaux de mesure synthétiques x_1, x_2, et x_3.

Tableau 3.9 Résultats de reconstitution du signal x_1 avec l'algorithme ITERKAL+, JANSSON et VAN-CITTERT

Algorithme	σ_η^2	k_{opt}	$\varepsilon_{\min}$
ITERKAL+	0	45	0.1077
	10^{-4}	38	0.3468
VAN_CITER	0	500	0.037
	10^{-4}	500	0.7659
JANSSON	0	500	$6.52268\ 10^{-4}$
	10^{-4}	500	0.7616

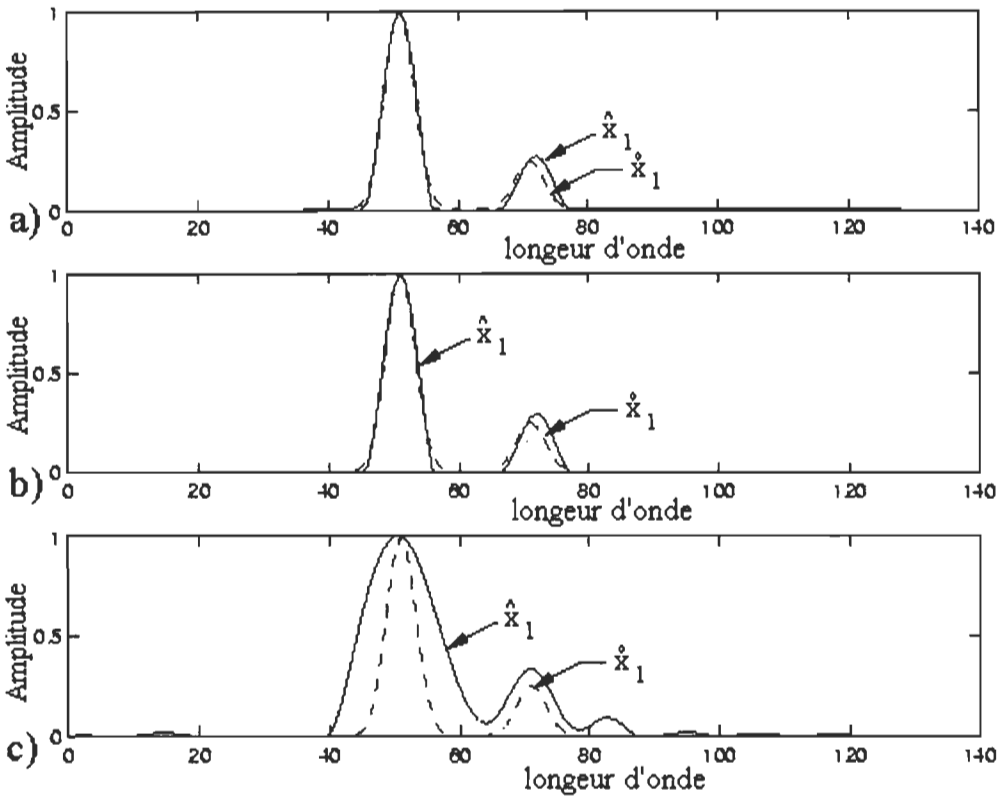


Figure. 3.8 Forme d'onde du signal x_1 avec ITERKAL+

a) pour σ^2_η=0, $\varepsilon_{opt} = 0.1077, \gamma_{opt} = 0.70$, k_{opt} =44;

b) pour σ^2_η=10^{-6}, $\varepsilon_{opt} = 0.1197, \gamma_{opt} = 0.70$, k_{opt} =44;

c) pour σ^2_η=10^{-4}, ε_{opt} =0.3468 , $\gamma_{opt} = 0.70$, $k_{opt} = 48$.

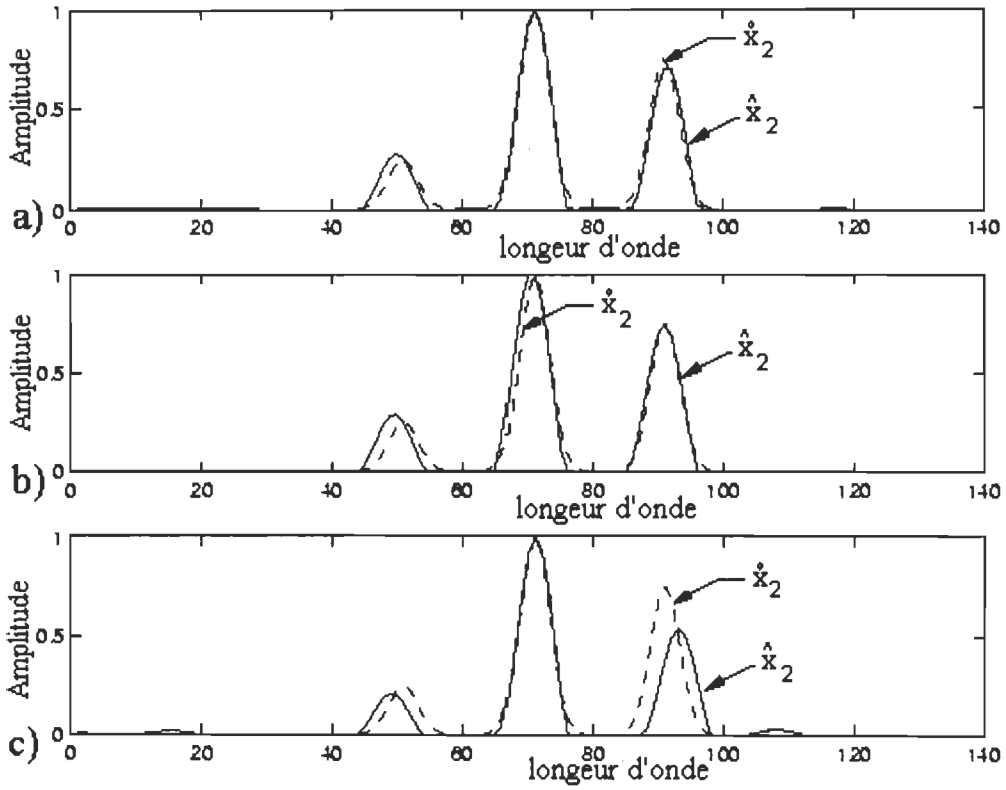


Figure. 3.9 Forme d'onde du signal x_2 avec ITERKAL+

a) pour σ^2_η=0, $\varepsilon_{opt} = 0.1281, \gamma_{opt} = 0.69$, k_{opt} =45;

b) pour σ^2_η=10^{-6}, $\varepsilon_{opt} = 0.1326, \gamma_{opt} = 0.44$, k_{opt} =44;

c) pour σ^2_η=10^{-4}, ε_{opt} =0.1419 , $\gamma_{opt} = 0.39$, $k_{opt} = 39$.

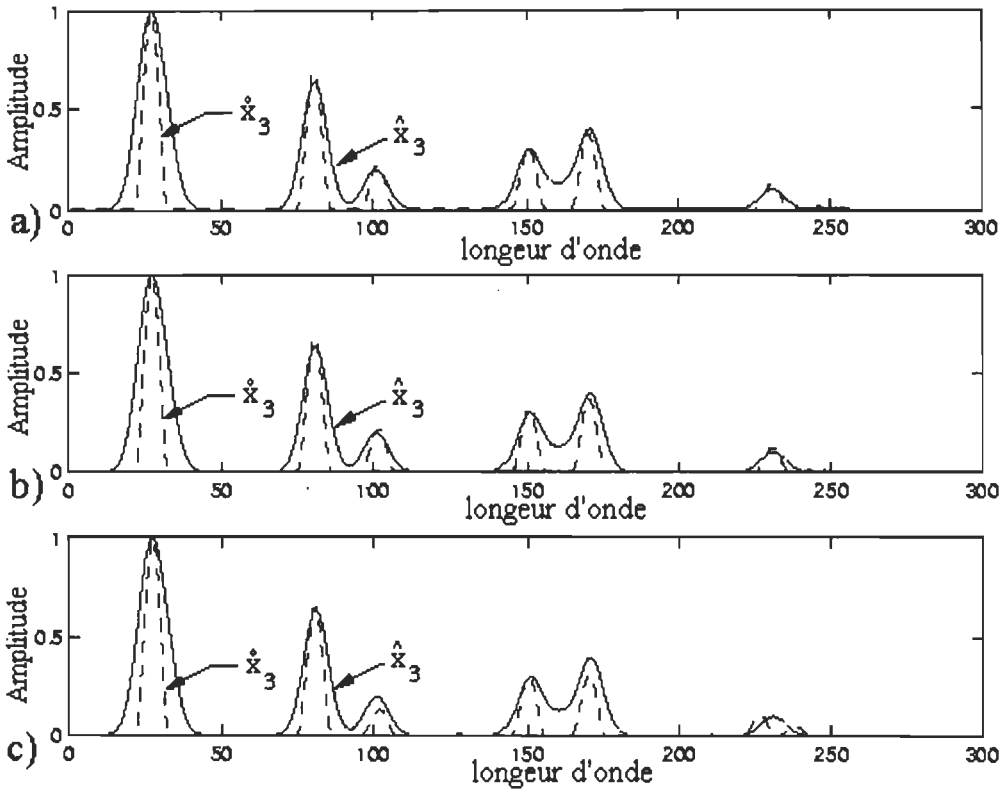


Figure. 3.10 Forme d'onde du signal x_3 avec ITERKAL+

a) pour σ^2_η=0, $\varepsilon_{opt} = 0.1952, \gamma_{opt} = 0.68$, k_{opt} =19;

b) pour σ^2_η=10^{-6}, $\varepsilon_{opt} = 0.1970, \gamma_{opt} = 0.68$, k_{opt} =20;

c) pour σ^2_η=10^{-4}, ε_{opt} =0.2117 , $\gamma_{opt} = 0.66$, $k_{opt} = 20$.

Tableau 3.10 Résultats de reconstitution du signal x_2 avec l'algorithme ITERKAL+, JANSSON et VAN-CITTERT

Algorithme	σ_η^2	k_{opt}	$\varepsilon_{\min}$
ITERKAL+	0	45	0.1281
	10^{-4}	39	0.1419
VAN_CITER	0	500	0.0035
	10^{-4}	500	0.7692
JANSSON	0	500	$3.0798\ 10^{-4}$
	10^{-4}	500	0.7817

Tableau 3.11 Résultats de reconstitution du signal x_3 avec l'algorithme ITERKAL+, JANSSON et VAN-CITTERT

Algorithme	σ_η^2	k_{opt}	$\varepsilon_{\min}$
ITERKAL+	0	19	0.1952
	10^{-4}	20	0.1917
VAN-CITER	0	500	0.0040
	10^{-4}	15	0.3607
JANSSON	0	500	$7.6083\ 10^{-4}$
	10^{-4}	13	0.3407

Sur les figures ci-dessous, nous avons appliqué le critère J_x pour la reconstitution des signaux synthétiques x_1, x_2, x_3, que nous comparons à deux algorithmes itératives de la littérature: Jansson et Van-Cittert . Nous constatons que lorsqu'il n'y a pas de bruit, $\sigma_\eta^2=0$, les algorithmes de Van-Citer et de Jansson donnent de meilleurs résultats que Iterkal+ , mais celui-ci devient plus efficace pour un niveau de bruit $\sigma_\eta^2=10^{-4}$.

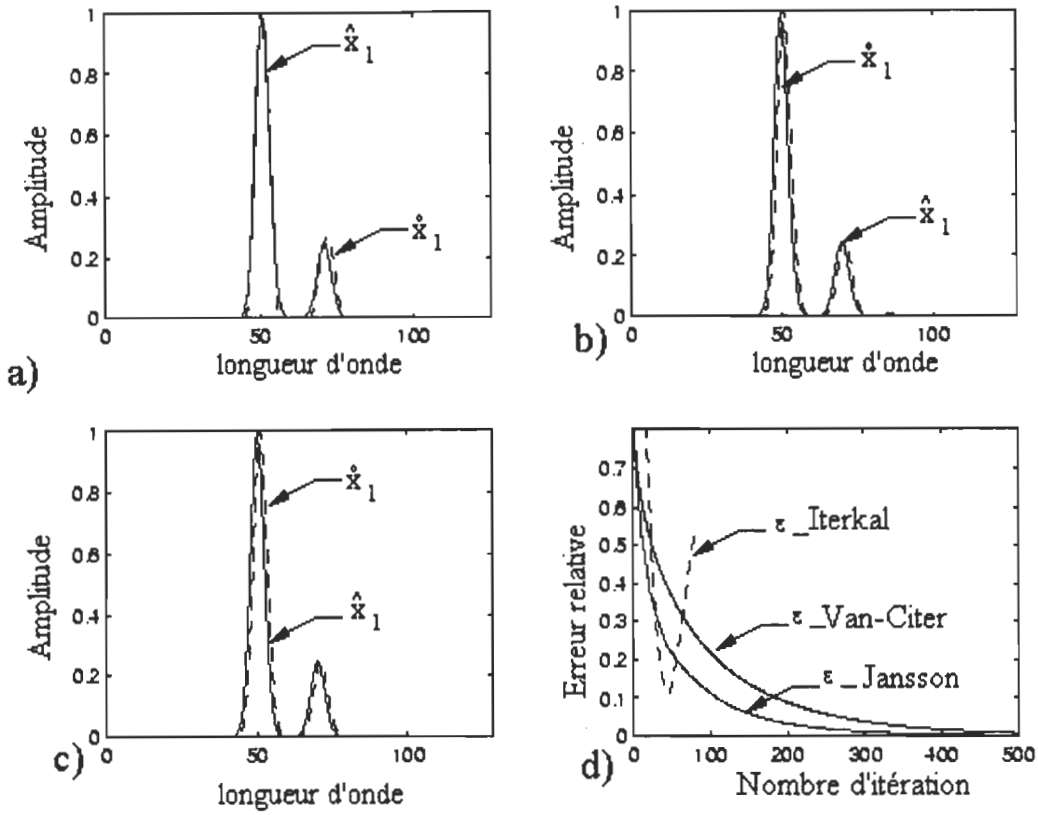


Figure. 3.11 Reconstitution du signal x_1 pour $\sigma_\eta^2 = 10^{-4}$, et N=128, a) algorithme Iterkal+ avec $\beta = 0.09$, k_{opt}=48, $\gamma = 0.7$ et $\varepsilon = 0.3468$, b) algorithme de Van-Cittert avec k_{opt}=500, et $\varepsilon = 0.7659$, c) algorithme de Jansson avec k_{opt}=500, et $\varepsilon = 0.7616$, d) courbe d'erreur relative en fonction du nombre d'itération des algorithmes Iterkal+, Van-Cittert, et Jansson.

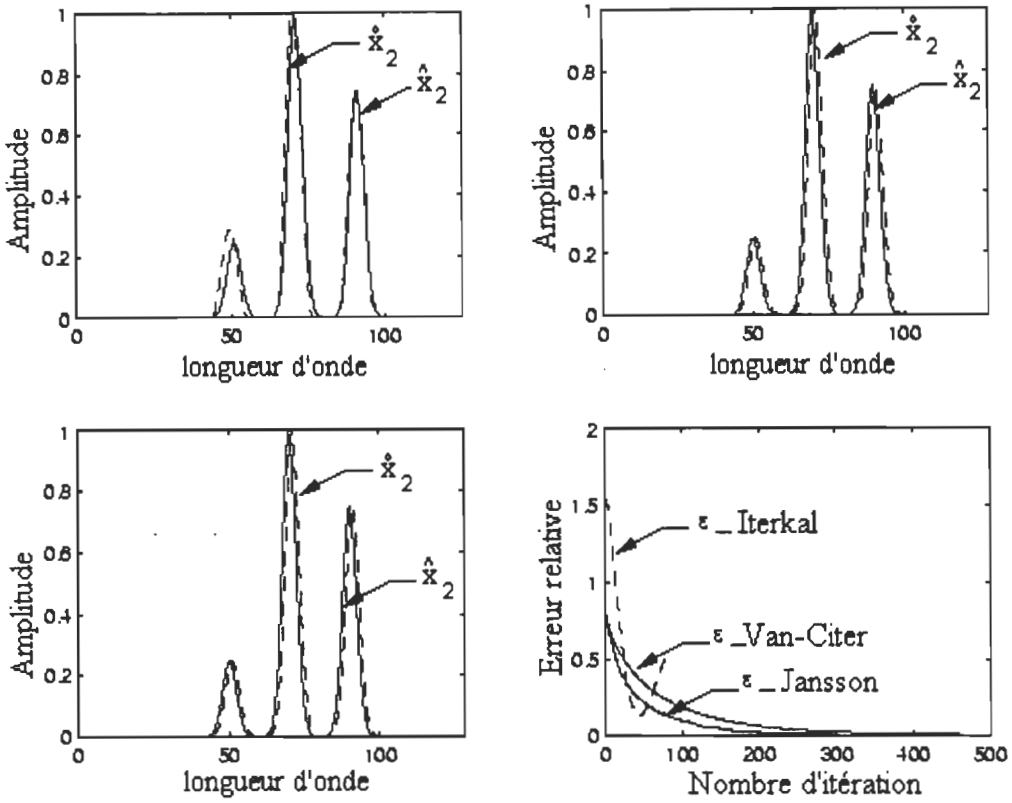


Figure. 3.12 Reconstitution du signal x_2 pour $\sigma_\eta^2 = 10^{-4}$, et N=128, a) Algorithme ITERKAL+ avec $\beta = 0.09$, k_{opt}=39, $\gamma = 0.7$ et $\varepsilon = 0.1419$, b) algorithme de VAN-CITTERT avec k_{opt}=500, et $\varepsilon = 0.7692$, c) algorithme de JANSSON avec k_{opt}=500, et $\varepsilon = 0.7817$, d) courbe d'erreur relative en fonction du nombre d'itération des algorithmes ITERKAL+, VAN-CITTERT, et JANSSON.

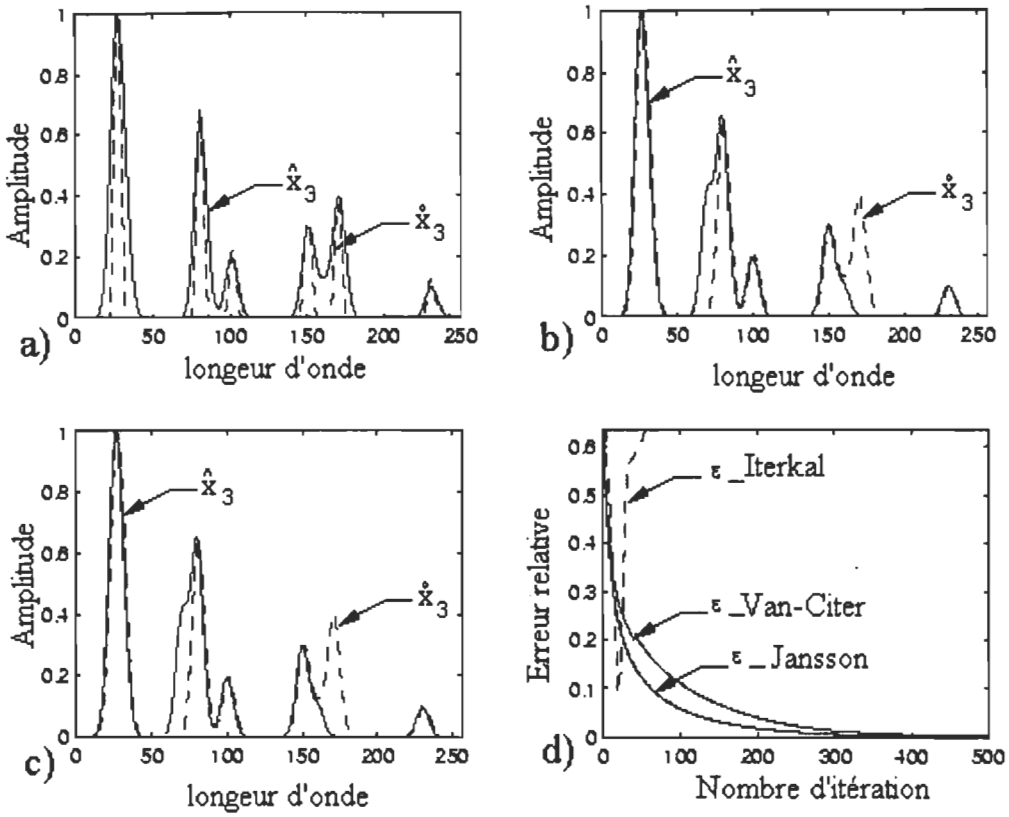


Figure. 3.13 Reconstitution du signal x3 pour $\sigma_\eta^2 = 10^{-4}$, et N=256, a) algorithme ITERKAL+ avec $\beta = 0.09$, k_{opt}=20, $\gamma = 0.7$ et $\varepsilon = 0.2117$, b) algorithme de VAN-CITTERT avec k_{opt}=500, et $\varepsilon = 0.3607$, c) algorithme de JANSSON avec k_{opt}=500, et $\varepsilon = 0.3407$, d) courbe d'erreur relative en fonction du nombre d'itération des algorithmes ITERKAL+, VAN-CITTERT, et JANSSON.

3.5 Analyse des résultats

Une analyse des tableaux et des figures de la section 3.4 montrent que pour un niveau de bruit $\sigma_\eta^2=10^{-4}$, les algorithmes de VAN-CITTERT et de JANSSON divergent, alors que l'algorithme ITERKAL+ nous donne des résultats appréciables au niveau de la qualité et de l'erreur relative lors de la reconstitution. Mais lorsque nous avons un niveau de bruit faible ($\sigma_\eta^2=0$), les algorithmes de Van-Citter et de JANSSON donnent de meilleurs résultats que ITERKAL+ à la suite de plusieurs itérations. Nous constatons également que ITERKAL+ converge plus rapidement que JANSSON et Van-Cittert comme on le voit sur les figures (3.9) à (3.11).

En ce qui concerne les critères J_X et J_{LCh} proposés pour le calcul de l'itération optimale, nous constatons que les résultats obtenus avec J_X sont légèrement meilleurs que ceux obtenus par J_{LCh}. Avec J_X , nous avons réussi à déterminer, pour certains signaux, k_{opt} avec une marge d'erreur de plus ou moins une itération. En revanche , le critère J_{LCh} s'implante plus facilement en technologie à grande échelle que J_X comme nous l'avons déjà mentionné.

En somme, la robustesse de l'algorithme ITERKAL+ face à celui de Van-cittert et JANSSON est bel et bien prouvé à travers nos résultats de simulation, et grâce aux critères proposés, le nombre d'itération optimal ne sera plus déterminé de façon empirique.

Dans le chapitre suivant, nous allons élaborer une étude de l'implantation d'ITERKAL+ en technologie à très grande échelle (VLSI).

CHAPITRE 4

PROPOSITION D'ARCHITECTURES VLSI SPÉCIALISÉE POUR LA MÉTHODE ITÉRATIVE BASÉE SUR LE FILTRE DE KALMAN

4.1 ARCHITECTURE SEMI-SYSTOLIQUE SYSKAL

4.1.1 Présentation de SYSKAL

L'architecture du processeur SYSKAL [MAS98a], dédié à l'algorithme reconstitution basé sur le filtre de Kalman, est déjà existant au laboratoire de micro-électronique de l'université du Québec à Trois-Rivières. Plusieurs variantes de cette architecture sont en étude [ELO98],[MAS98b] et elle à été développée par le professeur Daniel Massicotte [MAS95a]. SYSKAL est un réseau de processeurs en anneau, résultat des équations récurrentes et uniformes développées dans [MAS98a]. De ces équations, l'on a alloué les calculs à un réseau de processeurs, ce qui a permis de diviser le problème en L sous - problèmes. D'où l'utilisation de L processeurs élémentaires. Le réseau de processeurs sur la figure 4.1 est une architecture semi-systolique linéaire avec une structure en anneau (SYSKAL). Il est dit semi-systolique puisqu'une donnée I_n est commune à tous les processeurs, et à structure en anneau puisque le processeur PE_L reçoit des données de PE_1.

Tous ces processeurs sont synchronisés sur la même horloge et les données se déplacent au même rythme d'un processeur à l'autre via des registres internes aux processeurs.

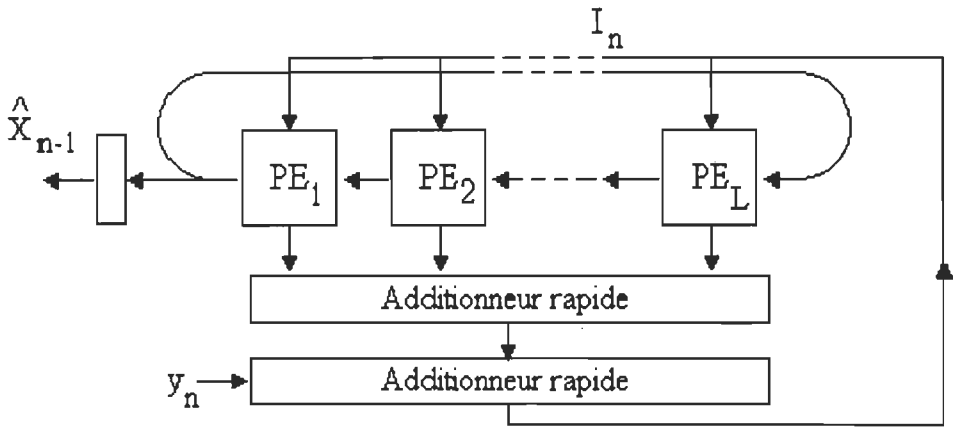


Figure. 4.1 SYSKAL : Architecture semi-systolique linéaire à structure en anneau.

Le bloc additionneur rapide [MAS95a], [PES95] permet de combiner les résultats des L sous- problèmes pour reconstruire la solution de l'estimation $\hat{y}_n$. Cette estimation permet d'obtenir l'innovation I_n pour reconstituer le prochain échantillon $\hat{x}_n$.

À l'annexe D, nous pouvons voir la structure interne d'un PE dans le cas d'un exemple pour 4 PE et *dim(h)=12* . Le dernier processeur, PE_L, diffère par l'ajout d'un multiplexeur afin de satisfaire des contraintes dues à certaines conditions limites découlant des équations du filtre de Kalman [MAS95a]. Dans cette version de l'architecture de SYSKAL, nous retrouvons deux cellules C_z et C_y. Les cellules C_z servent aux calculs de $\hat{z}_{n+1/n,l-1}$ et $\hat{z}_{n+1/n,l+(C-1)L}$ et les cellules C_y servent aux calculs partiels de $\hat{y}_n$. Un décaleur

barillet que l'on retrouve dans C_z sert à appliquer la contrainte de positivité selon l'équation Nous remarquons une légère modification du PE_1 qui diffère uniquement par un registre de sortie en moins à l'intérieur de la cellule C_z et d'un registre à valeur nulle à la cellule C_y pour satisfaire, comme nous l'avons déjà mentionné, des contraintes dues à certaines conditions limites découlant des équations du filtre de Kalman. L'additionneur rapide est réalisé sous la forme d'un arbre, t représente la fin du cycle d'horloge et n représente le numéro de l'échantillon en cours d'exécution.

4.2 Architecture semi-systolique $SYSKALI_1$

L'architecture semi-systolique linéaire à structure en anneau $SYSKALI_1$, est constituée comme SYSKAL de processeurs élémentaire PE, et chaque processeur élémentaire est formé de deux multiplieurs 16x16 bits et de trois sortes sorte de piles pour stocker le gain de Kalman K_∞, la réponse impulsionelle h, et les données $\hat{z}$ calculées. La princpale différence de $SYSKALI_1$ et SYSKAL est sa mémoire de stockage, dans laquelle les échantillons sont stocker pour effectuer plusieurs itérations.

Afin de saisir le fonctionnement complet de cette architecture, un exemple de fonctionnement est présenté à l'annexe D l'algorithme la reconstitution d'un échantillon de mesure.

4.2.1 Fonctionnement du processeur SYSKALI1

Notons que le processeur $SYSKALI_1$ effectue la reconstitution de mesurande à partir du filtre de Kalman itératif vu au chapitre 3. Son architecture est basée sur celle de SYSKAL, mais elle comporte en plus une mémoire pour stocker les données reconstituées, qui seront utilisées pour effectuer la seconde itération. De la même façon, les nouvelles données reconstituées seront stockées pour la troisième itération, et ainsi de suite, nous arriverons à effectuer le nombre d'itération voulue.

Le calcul pour la reconstitution d'un échantillon est effectué en un nombre de cycles égal à la profondeur de la pile du vecteur de gain K_∞. Disons que les piles du vecteur K_∞ sont d'une profondeur de trois éléments dans notre application, et le cycle de calcul est complété en cinq cycles. L'échantillon $\hat{x}_1^1$ est mis en mémoire, et l'on commence le calcul de l'échantillon $\hat{x}_1^2$. On notera que la contrainte de positivité est appliquée par le décaleur à barillet, qui multiplie le résultat du calcul par zéro s'il est négatif, et que l'arbre d'additionneur fait la somme des estimés $\hat{y}_n$ partiels qui sortent des processeurs. Ces valeurs de $\hat{y}_n$ partiels sont accumulés dans le registre y_acc pour former un estimé $\hat{y}_n$ complet à la fin du cycle de calcul.

Afin de calculer l'innovation I_n, le calcul de l'estimé du mesurande $\hat{x}_n$, est effectué en deux étapes. Premièrement, chaque PE effectue l'opération d'estimation de y selon le produit du vecteur ligne h^T avec le vecteur colonne $\hat{z}$ afin d'obtenir le scalaire $\hat{y}_n$. Ensuite, $\hat{y}_n$ partiels sont sommés au travers d'un arbre d'additionneurs. Lors du fonctionnement complet du processeur, plusieurs données se présentent à l'entrée du bus data_bus. Ainsi, ce bus voit

passer les différentes valeurs du gain de Kalman K_∞, de la réponse impulsionnelle h, de la contrainte de positivité et enfin les résultats de mesure $\tilde{y}_n$. Toutes ces valeurs sont représentées sous forme binaire avant d'être transférés au processeur.

Notons qu'un cycle complet du déplacement des données dans les bancs de registres des données $\hat{z}_{n/n+1}$, k_∞ et h pour réaliser un point de reconstitution se fait en $C = M/L$ cycles, sachant qu'il faut un cycle d'horloge pour créer un déplacement [MAS95a].

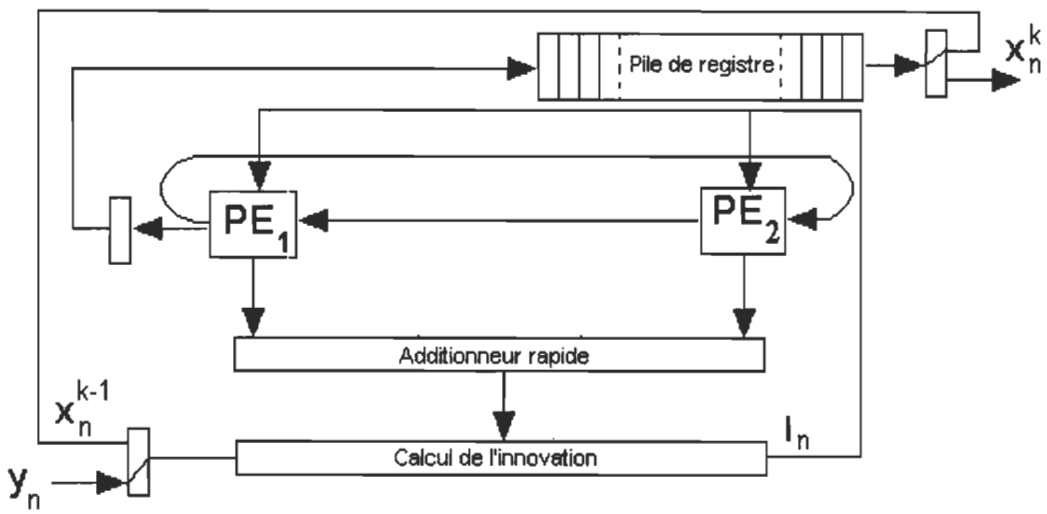


Figure. 4.2 Architecture semi-systolique du processeur $SYSKALI_1$.

Comme nous l'avons dit, le processeur $SYSKALI_1$ est muni des mêmes éléments composant SYSKAL, et possède en plus une mémoire constituée d'une empilation de pile FIFO (" First in-first out", première entrée-prémière sortie) pour emmagasiner les échantillons reconstitués.

Comme nous l'avons dit auparavant, le lecteur se référera à l'annexe D, où est préseté un exemple de déplacement des flots de données à travers l'architecture $SYSKALI_1$ avec quatres processeurs élémentaires pour la reconstitution du premier échantillon à différents cycles d'horloge.

4.3 Architecture semi-systolique $SYSKALI_2$

4.3.1 Fonctionnement du processeur $SYSKALI_2$

Le processeur $SYSKALI_2$ contient trois réseaux de processeurs $SYSKALI_2$ présenté à la figure 4.9. Ces derniers permettent d'accélérer la reconstitution, car $SYSKAL_2$ est en quelque sorte trois processeurs SYSKAL mis en cascade, et dont les résultats sont stockés en mémoire pour les prochaines itérations. La différence entre $SYSKALI_1$ et $SYSKALI_2$, est que dans le dernier, les résultats sont stockés en mémoire après trois itérations, ce qui rend la reconstitution très rapide par rapport à $SYSKALI_1$, mais est très gourmande en surface d'intégration. Sur la figure 4.3, nous avons un réseau de processeurs élémentaires dédié à $SYSKALI_2$ que nous nommons RPES2. Le bloc RPES2 sera donc utilisé k fois pour faire k itérations. Pour reconstituer un signal par exemple en trois itérations,. Nous utiliserons trios blocs de RPES2. Le fait que RPES2 contient des PE, cela rend la reconstitution plus rapide. Cette méthode est certes rapide, mais demande beaucoup de surface lors de son implantation. Pour palier à ce problème, l'on peut faire rapidement trois itérations par exemple, stocker le résultat, et continuer par la suite, comme l'indique le schéma synoptique de $SYSKALI_2$ avec une mémoire externe à la page suivante.

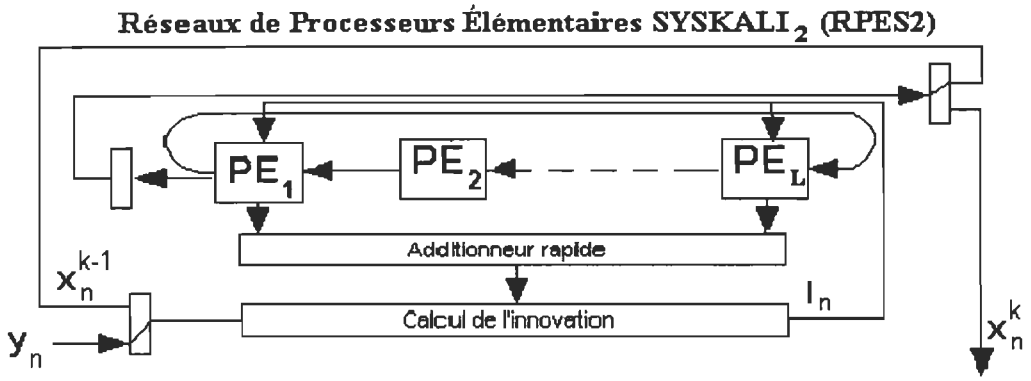


Figure. 4.3 Réseaux de Processeurs Élémentaires $SYSKALI_2$ ou RPES2.

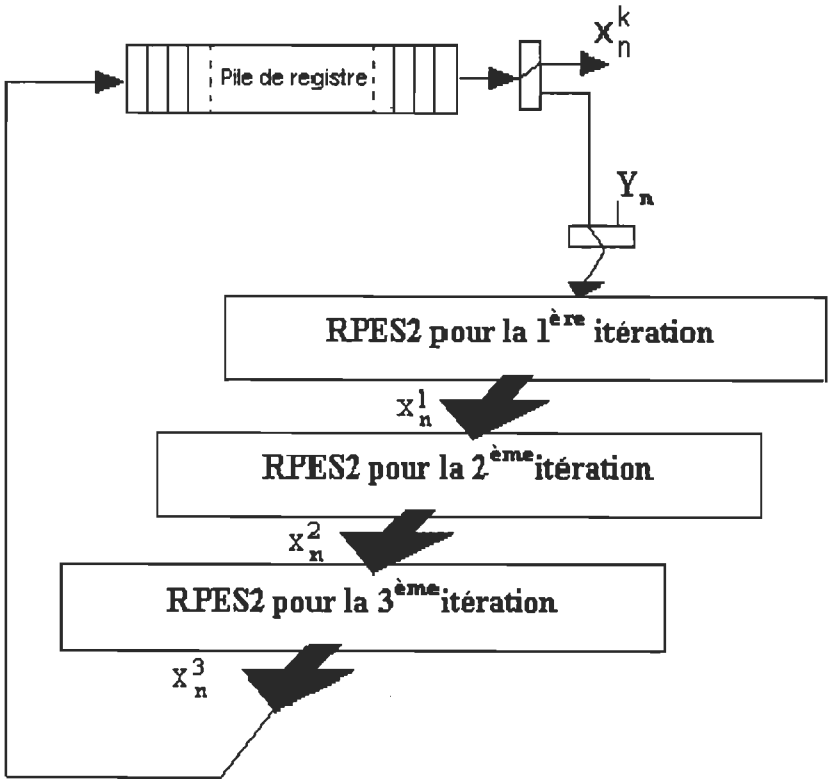


Figure. 4.4 Schéma synoptique de l'architecture semi-systolique du processeur $SYSKALI_2$.

Un avantage considérable du processeur SYSKALI$_2$ est que l'on peut l'adapter, tel queprésenté à la figure 4.5 pour qu'il effectue plusieurs itérations sans stockage d'échantillons.

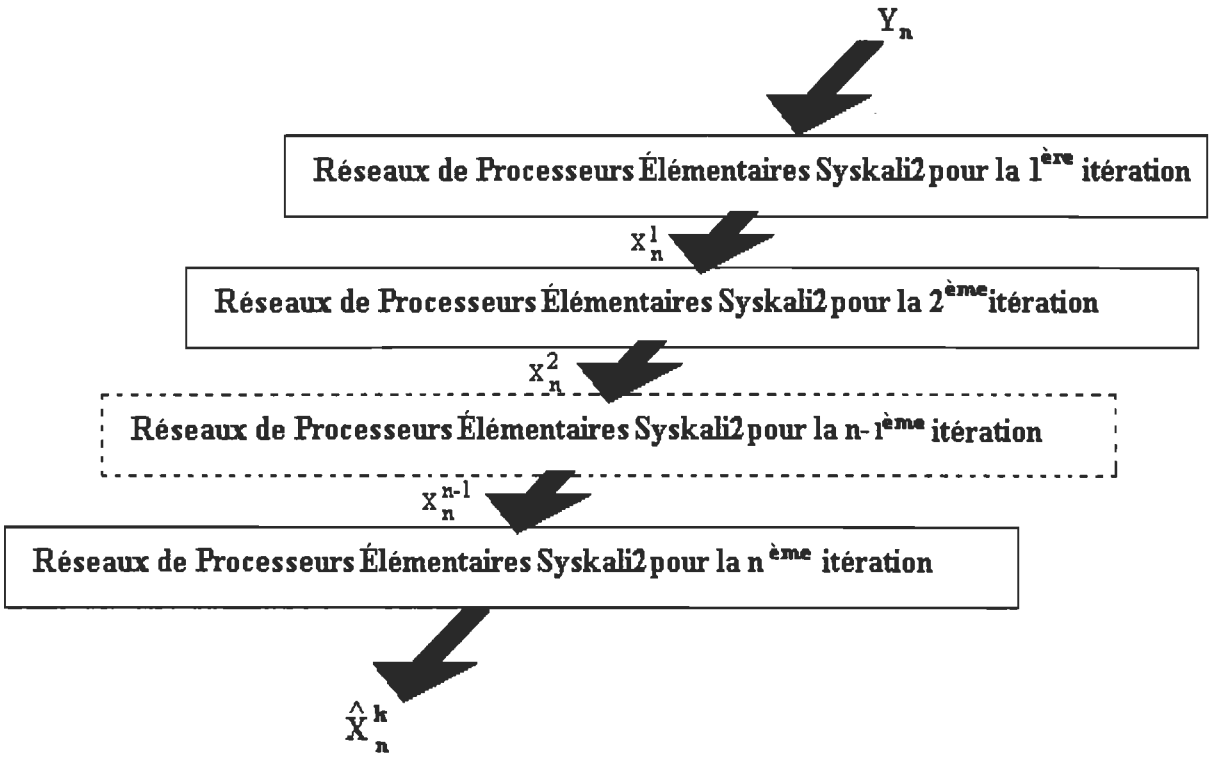


Figure. 4.5 Schéma synoptique de l'architecture du processeur SYSKALI$_2$ sans mémoire externe.

Nous présentons à l'annexe D le déplacement des flots de données à travers l'architecture SYSKALI$_2$ pour la reconstitution à t=0 et n=1, avec t correspondant au temps, et n le pas d'échantillonnage.

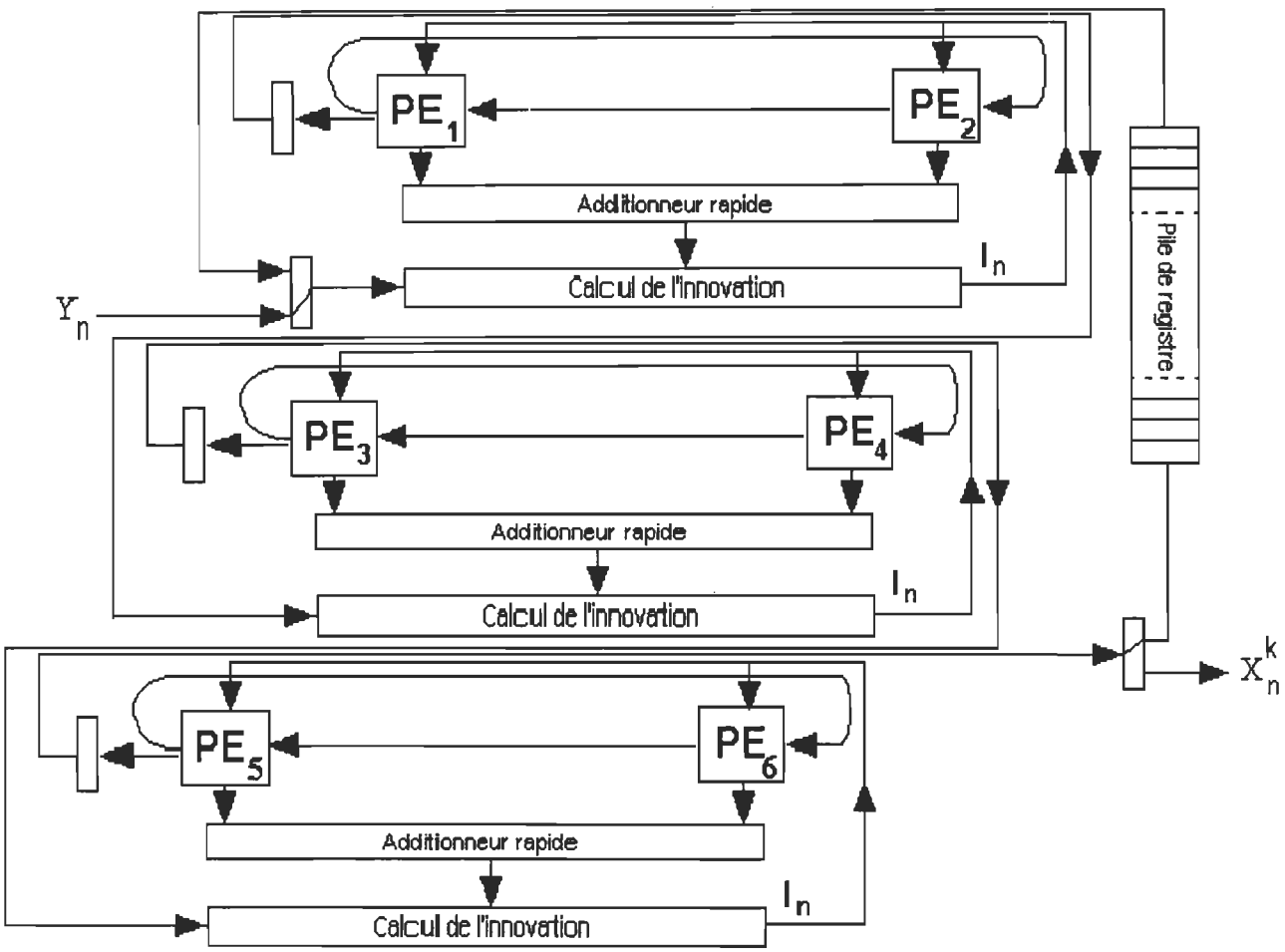


Figure. 4.6 Architecture semi-sytolique $SYSKALI_2$

4.4 Architecture semi-systolique $SYSKALI_3$

4.4.1 Fonctionnement du processeur $SYSKALI_3$

L'architecture du processeur $SYSKALI_3$ est composée des mêmes blocs fonctionnelles que celles de $SYSKALI_1$, à savoir un même nombre d'additionneurs, de soustracteurs et de multiplieurs, et possède en plus un nombre de piles de registre supplémentaires dans chaque processeur élémentaire, tel que présenté à la figure 4.14 destinées au stockage des estimés

du vecteur d'état Z_n pour le calcul des échantillons des itérations futures. Pour calculer le premier échantillon de la deuxième itération par exemple, les estimés du vecteur d'état Z_n ayant servis au calcul de $\hat{x}_1^1$ par exemple, seront stocké dans les piles de registre et seront réutilisés pour le calcul de $\hat{x}_1^2$. Notons que le nombre de registre dans chaque processeur élémentaire est lié au nombre d'itération à effectuer, soit $3k$, où k désigne le nombre d'itération et le chiffre 3, pour le nombre de registre dans chaque processeur élémentaire de $SYSKALI_1$.

Un exemple de déplacement des flots de données de $SYSKALI_3$ est présenté en annexe D pour t=0 et n=1 en ce qui concerne la reconstitution de $\hat{x}_1$. Une évaluation des différentes architecture est proposée au tableau 4.4. Cette évaluation tien compte de la dimension de la réponse impulsionnelle M, du nombre N de points d'échantillonnage de la mesure, du nombre K d'itération et du nombre L de processeurs élémentaires.

Pour une simplification du schéma de l'architecture $SYSKALI_3$, nous présenterons celle de deux processeurs élémentaires avec K=3.

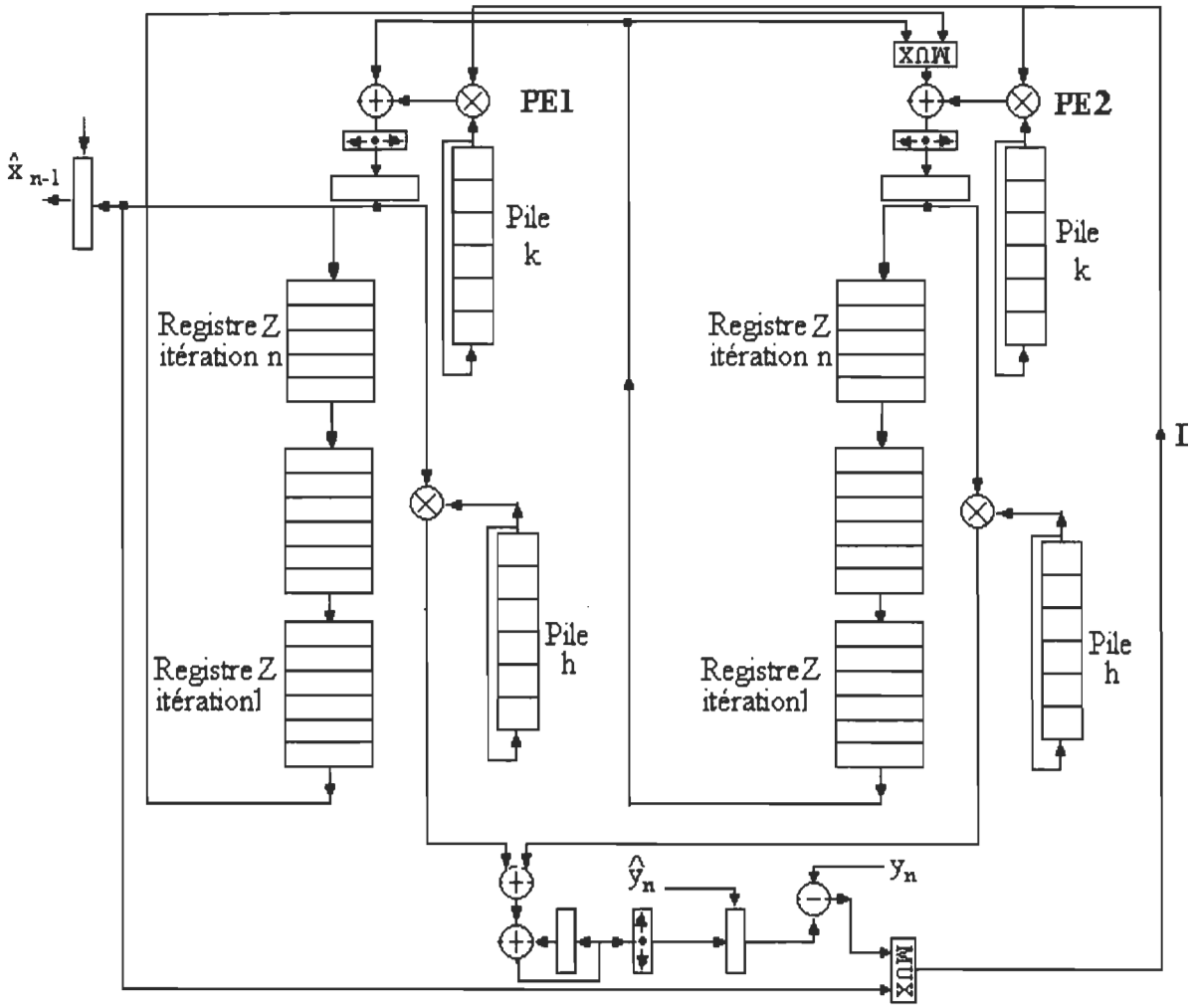


Figure. 4.7 Architecture semi-systolique linéaire SYSKALI3 pour K=3, et 2PE

4.5 Bloc d'évaluation du critère d'arrêt des itérations

Les architectures SYSKALI peuvent être munies d'un bloc pour le calcul du critère d'arrêt des itérations. Parmi les critères étudiés au chapitre 3, seul J_{LCh} est facilement implantable en technologie VLSI compte tenu de sa simplicité. La réalisation de ce critère basé sur la méthode du L-COURBE vue au chapitre3, nécessite deux additionneurs accumulateurs,

deux soustracteurs, deux registres, trois multiplieurs et une pile de registre pour stocker les valeurs de $\left\|g - h^1 * h^2 * \ldots * h^{k-2}\right\|_2^2$ à chaque itération. Comme explicité au chapitre 3, tant que les équations suivantes : $\frac{\left\|\hat{x}^k\right\|_2}{\left\|\hat{x}^{k-1}\right\|_2} \geq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2}{\left\|g - h^1 * h^2 * \ldots * h^{k-1} * h^k\right\|_2}$ et

$\frac{\left\|\hat{x}^{k-1}\right\|_2}{\left\|\hat{x}^{k-2}\right\|_2} \geq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-2}\right\|_2}{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2}$ sont satisfaites, on continue de faire des itérations, mais dès que nous aurons :

$\frac{\left\|\hat{x}^k\right\|_2}{\left\|\hat{x}^{k-1}\right\|_2} \geq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2}{\left\|g - h^1 * h^2 * \ldots * h^k\right\|_2}$ et

$\frac{\left\|\hat{x}^{k-1}\right\|_2}{\left\|\hat{x}^{k-2}\right\|_2} \leq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-2}\right\|_2}{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2}$, cela suppose selon la règle du L-COURBE simplifiée que nous avons atteint la valeur optimale du nombre d'itération. Ces deux conditions s'implantent facilement, car on a besoin uniquement des données présentes et de leurs valeurs précédentes comme $\hat{x}^k$ et $\hat{x}^{k-1}$, h^k et h^{k-1}. De plus, pour facilité et l'implantation de J_{LCh}, nous avons utilisé la norme au carré, soit :

$\frac{\left\|\hat{x}^k\right\|_2^2}{\left\|\hat{x}^{k-1}\right\|_2^2} \geq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2^2}{\left\|g - h^1 * h^2 * \ldots * h^k\right\|_2^2}$ et $\frac{\left\|\hat{x}^{k-1}\right\|_2}{\left\|\hat{x}^{k-2}\right\|_2} \geq \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-2}\right\|_2}{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2}$,

ce qui revient à juste arrêter l'itération au changement de signe de la quantité

$$\frac{\left\|\hat{x}^k\right\|_2^2}{\left\|\hat{x}^{k-1}\right\|_2^2} - \frac{\left\|g - h^1 * h^2 * \ldots * h^{k-1}\right\|_2^2}{\left\|g - h^1 * h^2 * \ldots * h^k\right\|_2^2}.$$

Rappelons que la substitution de la norme simple par la norme au carré nous a donné les mêmes résultats sur le logiciel matlab [KRA94].

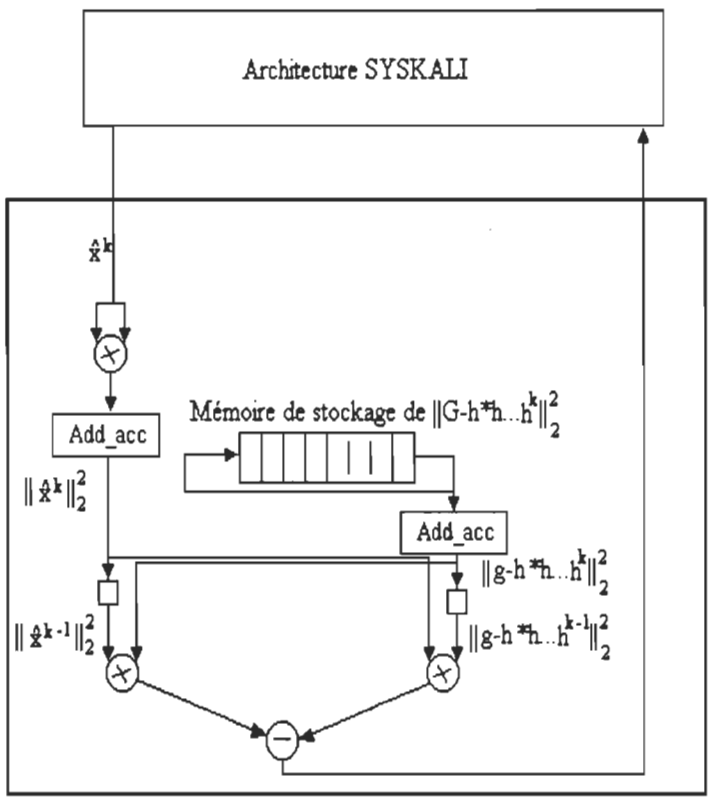


Figure. 4.8 Architecture du bloc d'évaluation de l'itération à effectuer.

4.6 Les entrées /sorties des processeurs SYSKALI dédié à l'algorithme ITERKAL+

Tableau 4.1 Description des entrées/sorties des processeurs SYSKALI

Nom	Type	Fonction
CLR	entrée	Initialise entièrement le Processeur SYSKALI
CLR_Z	entrée	Initialise les piles Z
CLR_S	entrée	Initialise le multiplexeur
CLR_ACC	entrée	Initialise le registre additionneur accumulateur
CLR_X	entrée	Initialise le registre de sortie du signal reconstitué
CLR_CALC	entrée	Initialise le registre de sortie du signal calculé y_cal
EN_S	entrée	valide la sortie du multiplexeur
EN_K	entrée	valide la sortie de la pile de stockage du gain K
EN_ACC	entrée	valide la sortie de l'additionneur accumulateur
EN_X	entrée	valide la sortie du registre de sortie du signal reconstitué
EN_CALC	entrée	valide la sortie du registre de sortie du signal y_cal
Y_MES	bus d'entrée 12 bits	échantillon du signal mesuré y à l'entrée de SYSKALI
DISTANCE	entrée	valeur de correction du signal mesuré
CHAIN	entrée	sélection du type de pile
S	entrée	commande du multiplexeur
PIL_INK	entrée	pile de stockage du gain K
PIL_INH	entrée	pile de stockage de la réponse impulsionnelle h
X_OUT	Bus de sortie 16 bits	échantillon du signal reconstitué par SYSKALI

Sur le tableau 4.1, les signaux de contrôles clk, en, clr, sont des signaux de 1 bit qui servent respectivement à activer le signal d'horloge sur front montant, à la validation de la pile, et à la remise à 0 des registres de pile.

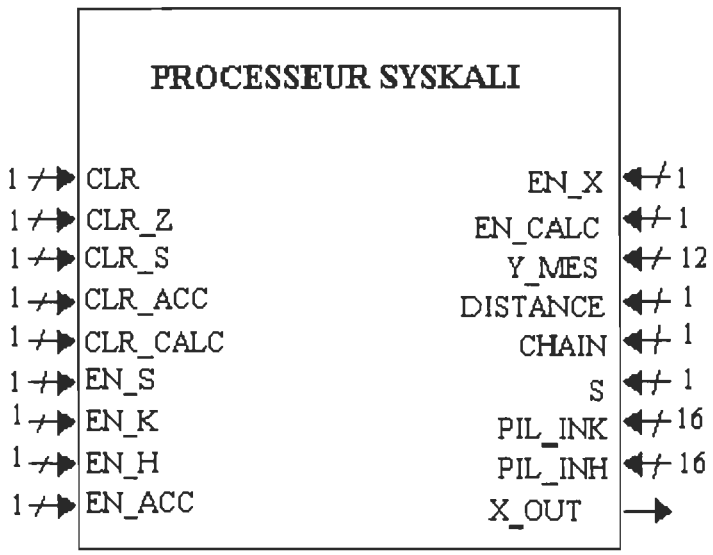


Figure. 4.9 Représentation du processeur SYSKALI

4.7 Modélisation des principaux blocs des architectures SYSKALI

4.7.1 Le langage VHDL et l'environnement de Mentor Graphic

Le langage de modélisation VHDL (Very High Speed Integrated Circuit) Hardware Descriptin Language, est un langage très modulaire, puissant et général. On ne va pas y écrire de longues descriptions, mais des unités plus petites et hiérarchisées. C'est également un langage moderne où l'on utilise une grande partie de l'expérience acquise en génie matériel et en génie logiciel. Notons également que les développements de descriptions VHDL sont prévus pour pouvoir être faits en parallèle par des équipes complètes. De plus, puisque les unités de compilations sont séparées, les modèles sont conçus pour être réutilisables [PES95], [MEA83].

Nous avons modélisé les processeurs SYSKALI grâce aux outils du logiciel Mentor Graphics

4.8 Résultats d'évaluation des architectures SYSKALI

4.8.1 Résultats de simulation

Nous avons simulé au complet les architectures des trois processeurs SYSKALI, car un des objectifs de ce mémoire est de proposer une architecture capable d'accepter l'algorithme itératif basé sur le filtre de Kalman. À la suite d'une analyse des trois architectures que nous présentons à la section 4.8.5, notre choix c'est porté sur celle du processeur $SYSKALI_3$, compte tenu de sa plus grande flexibilité par rapport aux autres. Dans cette section, nous présenterons un exemple de reconstitution d'un de nos signaux de référence, à savoir $\hat{x}_2$, obtenu respectivement de iterkal+ sur matlab, et du processeur $SYSKALI_3$ sur Mentor Graphic.

Rappelons que pour simuler les processeurs SYSKALI, il faut leur présenter les signaux qui stimuleront leurs entrées, tel que présentées sur les tableaux 4.1. Ces stimuli représentent donc les états de ses entrées au cours du temps. L'outil de simulation de Mentor graphics utilise deux méthodes pour charger les stimuli. La première méthode consiste à entrer à la main toutes les valeurs de tous les stimuli au cours du temps, et la seconde que nous avons adopté, consiste à écrire un programme (fichier force file en Annexe B), à partir duquel le processeur prendra ses données. Il faut dire qu'avant la compilation, certaines précautions sont à prendre. Il faut préciser le répertoire de travail, les options de compilation pour une synthèse et pour une simulation. On doit également vérifier, à partir du shell, le contenu du fichier mgc_location_map qui contient tous les chemins des librairies utiles lors de la

compilation de programme VHDL. Le résultat de la compilation est placé dans un répertoire de type mgc_component portant le nom de l'entité du programme VHDL compilé.

Sur la figure 4.10a, nous pouvons observer respectivement le signal reconstitué $\hat{x}_2$, de l'algorithme ITERKAL+ et du processeur qui lui est dédié, et à la figure 4.10b, la courbe d'erreur obtenue à la suite des deux reconstitutions.

La différence entre les résultats en virgule flottante dans Matlab et virgule fixe dans Mentor Graphic se justifie par l'effet de quantification de 16 bits.

4.8.2 Résultats d'évaluation des architectures SYSKALI

Les principales caractéristiques des architectures SYSKALI sont :

- un calcul rapide grâce à leur spécificité,
- une fréquence d'horloge relativement indépendante du nombre de processeurs élémentaires utilisés,
- une surface de silicium facile et rapide à estimer,
- une régularité et une modularité facilitant leur réalisation,
- une communication et un contrôle localisé,
- une programmation immédiate.

Dans les sous sections qui suivent, nous avons fait une évaluation des architectures de SYSKALI en prenant comme critères d'évaluation, leur performance, leur surface d'intégration, la facilité de conception de leur architecture, et la facilité d'application de leur architecture.

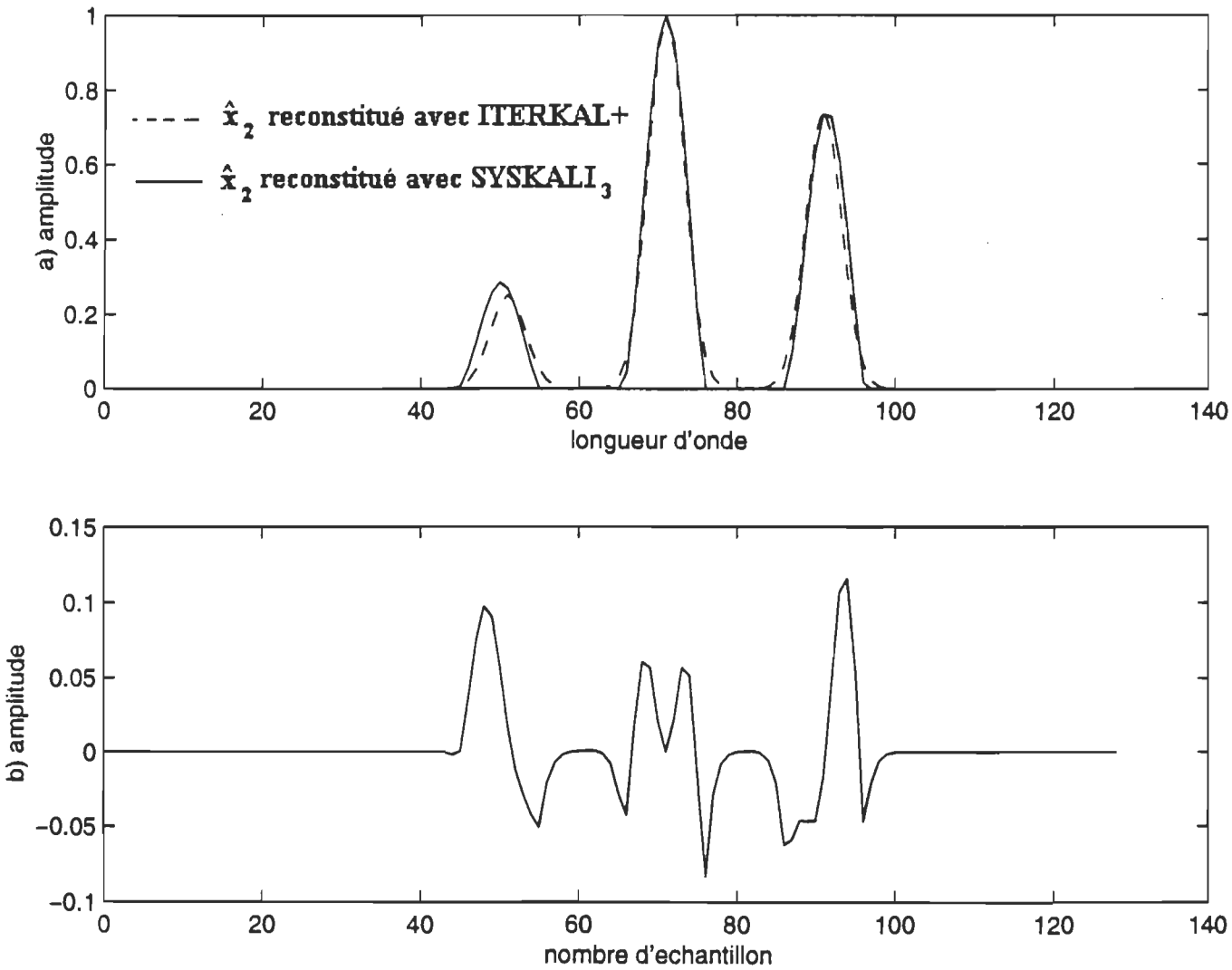


Figure 4.10 a) Résultat de reconstitution de x_2 par ITERKAL+ et SYSKALI$_3$ en10 itérations, avec β=0.09 , b) Courbe d'erreur de reconstitution de x_2 par ITERKAL+ et SYSKALI$_3$ en10 itérations, avec β=0.09

4.8.3 Performance

Pour l'étude de la performance, nous avons tenu compte du nombre de cycles et de la fréquence d'horloge pour reconstituer un échantillon. Partant de cette définition, nous pouvons dire que les architectures $SYSKALI_1$ et $SYSKALI_2$ sont plus performantes que $SYSKALI_3$, car elles nécessites moins de cycles d'horloge pour reconstituer un échantillon. Les flots de données dans $SYSKALI_3$ doivent traverser tous les registres permettant le stockage des valeurs du vecteurs z à chaque itération, comme précédemment mentionné dans l'exemple de SYSKAL.

4.8.4 La surface d'intégration

La surface d'intégration dépend de la largeur d'un mot binaire qui définit la dynamique des valeurs, de la quantité de mémoires et de son emplacement, du nombre d'unités opératives, du nombre de bus ainsi que de la complexité de la connectivité.

Rappelons que pour reconstituer un signal avec $SYSKALI_1$, les échantillons du vecteur $\tilde{\mathbf{y}}$ sont injectés dans le processeur, et l'on obtient un nouveau vecteur $\hat{\mathbf{x}}$ à la sortie du processeur. Les échantillons de ce vecteur représentent les résultats de la première itération et son stocker dans une mémoire externe. Et pour effectuer K itérations, nous répétons l'opération ci dessus K fois, en injectant à chaque fois le nouveau vecteur reconstitué à l'entrée du processeur. Quant à $SYSKALI_2$, il est constitué de K fois $SYSKALI_1$, à l'exception de la mémoire externe. Dès qu'un échantillon est reconstituer après la première itération, il est automatiquement injecté dans un autre processeur $SYSKALI_1$ pour

l'obtention de l'échantillon de la deuxième itération, puis ainsi de suite nous arrivons à l'échantillon de la $K^{ième}$ itération. Le processeur $SYSKALI_3$ quand à lui est constituer du mêmes nombre d'unités opératives que $SYSKALI_1$, mais possède en plus des registres afin de mémoriser les élément du vecteur $\hat{\mathbf{z}}_n$, qui seront réutiliser pour les itérations ultérieures. Il faut noter que les registres occupe très peu de place mémoire contrarient aux unités opératives, ce qui fait que $SYSKALI_2$ est de loin la plus gourmande des architecture en surface, suivie de $SYSKALI_3$.

4.8.5 Comparaison des architectures SYSKALI

- $SYSKALI_1$

Ces avantages sont qu'elle permet une reconstitution en temps réel, et sa surface d'intégration est indépendante du nombre d'itération.

Son principal inconvénient est qu'elle nécessite une mémoire externe pour le stockage des échantillons.

- $SYSKALI_2$

Ces avantages sont qu'elle permet une reconstitution très rapide et son principal inconvénient est sa très grande surface d'intégration liée au nombre d'itération à effectuer.

- $SYSKALI_3$

Ces avantages sont qu'elle permet une reconstitution en temps réel, et ne nécessite pas de mémoire externe pour stocker les données reconstituées. Son principal inconvénients est que sa surface d'intégration est dépendante du nombre d'itération.

La détermination du nombre de cycle des architectures $SYSKALI_1$, $SYSKALI_2$, $SYSKALI_3$ est régie respectivement par les équations suivantes :

$$k\,N\left(\frac{M}{N_{PE}}+2\right) \tag{4.1}$$

$$\frac{k}{N_{RPES2}}N\left(\frac{M}{L}+2\right)+2N_{RPES2} \tag{4.2}$$

$$N\,k\,\frac{M}{N_{PE}}+2 \tag{4.3}$$

où

N_{PE} est le nombre de processeurs élémentaires;

N_{RPES2} est le nombre de réseaux de processeurs RPES2.

En fait, la surface de $SYSKALI_3$ est $\frac{N}{M\,K}$ fois plus petite que $SYSKALI_1$.Cette constatation est basée sur la supposition que le stockage des données est réalisé à l'aide de registres. Un cas typique serait, M=64, K=10, et N=2048 alors la surface de $SYSKALI_3$ devient 3.2 fois plus petite. Si $N \rangle\rangle M\,K$, le processeur $SYSKALI_3$ devient plus avantageux que $SYSKALI_1$ d'autant plus qu'il reconstitue en temps réel.

Tableau 4.2: Évaluation des architectures proposées :

CRITÈRES D'ÉVALUATION	ARCHITECTURES PROPOSÉES		
	$SYSKALI_1$	$SYSKALI_2$	$SYSKALI_3$
Nombre de cycle	moyenne	élevée	moyenne
surface d'intégration	faible	élevée	moyenne
facilité de conception	très simple	moyenne	très simple
facilité d'application	très simple	très simple	très simple

4.9 Synthèse des résultats de recherche

Dans le cadre de cette recherche, nous avions pour but d'apporter dans un premier temps les adaptations nécessaires à l'algorithme itératif de reconstitution de signaux basé sur le filtre de Kalman, avec une contrainte de positivité, dénommé ITERKAL+, afin qu'il soit autonome pour le calcul optimal du nombre d'itération à effectuer.

ITERKAL+ est dorénavant muni de deux critères J_{LCh} et J_x, respectivement basé sur la méthode du L-courbe et d'un marqueur d'arrêt optimal du nombre d'itération basé sur l'erreur d'estimation du résultat de mesure, tel que vu au chapitre3 . La nouveauté par rapport à l'ancienne version de l'algorithme ITERKAL+, est que l'utilisation de J_{LCh} ou J_x, dans le cadre d'une application donnée évite la fixation du nombre d'itération k de façon empirique, ce qui permet l'obtention de meilleurs résultats de reconstitution par rapport à la

première version d''ITERKAL+. Nos résultats de recherche ont été testés en utilisant des signaux synthétiques parfois complexes, et cela a permis de confirmer que ITERKAL+ est plus résistant au bruit et qu'il converge plus rapidement comparativement aux autres algorithmes proposés dans la littérature [MAS95a].

Dans le second volet de notre recherche nous avons proposé trois architectures capables de recevoir l'algorithme ITERKAL+. En suite, une étude de son implantation à l'aide des outils et méthodes de conception assistées par ordinateur à été effectuée. Finalement, nous avons simulé entièrement l'architecture la plus intéressante, soit $SYSKALI_3$.

CHAPITRE 5

CONCLUSION

Dans les deux premiers chapitres de ce travail, nous avons présenté un survol des principales techniques utilisées dans le domaine de la reconstitution de signaux. Ensuite, nous avons apporté dans le chapitre 3, un ajout à l'algorithme itératif de Kalman à travers l'étude de deux critères pour déterminer de façon automatique, le nombre d'itération optimale, qui dans l'ancienne version de l'algorithme, se faisait de façon empirique. Effectuer la reconstitution avec un nombre d'itération optimale est d'une importance capitale, car nous obtenons ainsi la meilleure qualité de reconstitution et de positionnement des pics du signal reconstitué. Dans le chapitre 4, nous avons proposé trois architectures pour le filtre itératif de Kalman, et simulé la plus flexible d'entre elles en technologie VLSI.

Nos principaux objectifs ont été atteins, en l'occurrence :

- la proposition d'un critère d'optimisation du nombre d'itération assurant la qualité de reconstitution du signal, qui dans l'ancienne version d'ITERKAL+ se faisait de façon empirique ;

- la proposition de trois architectures $SYSKALI_1$, $SYSKALI_2$, $SYSKALI_3$ capables de recevoir l'algorithme itératif de Kalman ;

- La modélisation VHDL et la simulation dans l'environnement Mentor Graphic validant l'architecture SYSKALI$_3$ qui offre les meilleures performances parmi les architectures proposées en technologie ITGE (Intégration à très grande échelle).

En comparaison aux algorithmes couramment utilisés en reconstitution de signaux spectrometriques (JANSSON, VAN-CITTERT...), nous sommes arrivé à la conclusion que ITERKAL+ offre d'excellents résultats de reconstitution et une convergence rapide.

En somme, ce mémoire représente une contribution au développement d'un algorithme itératif basé sur le filtre de Kalman et d'architectures en technologie VLSI pour la reconstitution de mesurandes, et consiste essentiellement en :

- L'étude et la proposition de critères d'arrêt du nombre d'itérations en considérant les contraintes algorithmiques et architecturales en vues d'une implantation en technologie VLSI.
- La proposition d'architectures systoliques pour l'algorithme étudié selon un développement architectural progressif tenant compte des contraintes tel que la surface et le temps de calcul.

L'élaboration de ce mémoire m'a permis de me familiariser avec les principaux algorithmes de reconstitution de mesurandes, ainsi qu'aux différents outils de conceptions VLSI.

Les résultats obtenus ont été présentés au 65ème congrès de l'ACFAS (Association Canadienne Française pour l'Avancement de la Science).

RÉFÉRENCES

[AIR87] R. AIRIAN, J. –M. BERGÉ, V. OLIVE, J. ROUILLARD, "VHDL, du Langage à la Modélisation", Presses Polytechniques et Universitaire Romandes 1987.

[ALB84] D. ALBA et G. R. MEIRA, " Inverse optimal filtering method for the instrumental spreading correction in size exclusion chromatography", journal of liquid chromatography, vol. 7, N°14, 1984, pp.2833-2862.

[AZI91] 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.

[BAR93] A. Barwicz, "System Approch to Electric Measurement", Conference Record, IEEE, IMTC'93, pp. 397-402, Irvin Ca., 18-20 Mai 93.

[BEL90] M. BÉLLANGER, Traitement numérique du signal, théorie et pratique, Masson, 1990, 464 PAGES.

[BIL92] Génetic Algorithms, Bill P. Buckles and Frederick E. Petry, IEEE Computer Press. 1992.

[CAV84] J. Cavanagh, Digital computer arithmetic: design and implementation. Chap. 3.6, 1984, page 203-213.

[CHA90] L. P. CHARLES ET H. TROY NAGLE, Digital Control System Analysis and Design, Prentice Hall, 2ème édition, 1990.

[CRI91] P. B. CRILLY, A Quantitative Evaluation of Various Iterative Deconvolution Algorithms, IEEE-Transactions on Instrumentation & Measurement, vol.40, N°3, Juin 1991, pp 558-562.

[ELO98] M.B Elouafay and D. Massicotte, " Pipelined Systolic Architecture for Kalman-Filter-Based Signal Reconstruction Algorith " CCECE'98, 24-28 May 1998

[GAS89] 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. 255-248.

[GOD89] D. GODARD, " Channel equalization using a kalman filter for fast data transmission" , IBM Journal Res. Develop., vol. 18, 1989 , pp. 267-273.

[GUE89] A.GUERCHAOUI, J. C. BALLUET et J. L. LACOUME, " Etude comparative des principales méthodes de déconvolution sur des données de types sismiques",Traitement du signal, Vol.6 No 3, pp.187-203, 1989.

[HAD02] J. Hadamard, sur les problèmes aux dérivés partielles et leur signification phisique. Univ. Princeton, 1902.

[HER91] A. HERMAN, G. DEMOMENT et R. REYNAUD, " Apport de la déconvolution en échographie ", Communication présentée au 8ème congrès de la SFAUMB, RBM, vol. 6, N° 1, 1991, pp. 41-50.

[HOR83] E. Horowitz et A. Zorat, Divide-and-Conguer For parallel processing, IEEE Transactions Computer, Vol.32, N° 6, 1983, p 582-585.

[JAN84] P. A. JANSSON, Deconvolution with Application in Spectroscopy, Academic Press, 1984.

[KAL60] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", ASME, Journal of Basic Engineering, Vol. 82-D, pp.34-45, 1960.

[KRA94] T. P. Krauss, L. Shure, J. N. Littlr, "Signal processing toolbox for use with MATLAB", The Math Work Inc., February 1994

[KUN91] M. Kunt, "Techniques modernes de traitement numériques des signaux", Presses Politechniques et universitaires romandes, Paris, 1991.

[LEG91] S. LEGENDRE, Étude de la Méthode de Jansson de reconstitution de Signaux, Activité de Synthèse en Génie Électrique ,Université du Québec à Trois-Rivières, Décembre 1991.

[LZC] L. Szczecinski, R.Z. Morawski, A. Barwicz. " Original-domain Tikhonov regularization and non-negativity constraint improve resolution of spectrophotometric analyses ", journal of the International Measurement Confederation.

[MAS92] A. Barwicz, D. Massicotte, R. Z. Mowski, "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.

[MAS93] 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.

[MAS95a] Daniel MASSICOTTE, " Une Approche à l'Implantation en Technologie VLSI d'une Classe d'Algorithmes de Reconstitution de signaux ", thèse présentée en vue de l'obtention du diplôme de philosophiae doctor, Mai 1995.

[MAS95b] D. Massicotte, R.Z. Morawski, A. Barwicz, " Incorporation of a Positivity Constraint Into a Kalman-Filter-Based Algorithm for Correction of Spectrometric Data ", soumis à IEEE Transactions on Instrumenttation and Measurement, Vol.44, N0 1, fevrier 1995, pp. 2-7.

[MAS97] D. Massicotte, R.Z. Morawski, A. Barwicz, " Kalman-Filter-Based Algorithms of Spectrometric Data Correction Part 1: An Iterative Algorithm of Deconvolution", IEEE Trans.Instrumentation and Measurement, vol. 46, N°3, JUNE 1997.

[MAS98a] D. Massicotte and M.B Elouafay " A Systolic Architecture For Kalman Filter-Based Signal Reconstruction Usign The Wave Pipeline Methode ", ICECS'98

[MAS98b] D. Massicotte " A Systolic VLSI Implementation of Kalman Filter-Based Algorithm for Signal Recontruction", IEEE Int. Conference on Acoustic, Speech, and Signal Processing, Seattle, 12-15 May 98

[MEA83] C. Mead, L. conway, "Introduction aux Systèmes VLSI", InterEditions, 1983, p. 168-169.

[MEN86] J. M. MENDEL, " Some Modeling Problems in Reflection Seismology ", IEEE Acoustics, Speed, and Signal processing Magazine, Avril 1986, pp. 4-17.

[MIL91] M. S. MILEWKI, Comparative Study of Iterative Algorithms for Measurement Signal Reconstruction, Vol. 1, Master's Thesis of Warsaw University of Technology, 1991.

[MOR77] Morawski, R. Z., Szczecinski, L., and Barwicz, A. (1995). J. Chemon. 9, 3-20, 1997, p178.

[MOR94] R. Z. Morawski, "Unified Approach to measurement Signal Reconstruction", IEEE Transactions on Instrumentation and Measurement, vol. 43, No 2, avril 1994, pp. 226-231.

[PAN95] P. PANGO, Architecture d'un Processeur Spécialisé pour la Reconstitution de signaux Basée sur le Filtrage de Kalman, Mémoire de Maîtrise en électronique industrielle, UQTR, 1995.

[PES95] Frederic PESSON, Modélisation d'un Processeur de Reconstitution dans l'environnement de Mentor Graphic, rapport de stage de DESS, Sept 95.

[SAN92] Santerre, M.A., Massicotte, D., Barwicz, A., Savaria, Y., " Architecture d'un Processeur Spécialisé pour la Reconstitution de Signaux Basée sur le Filtrage de Kalman", Congrès canadien en génie electrique et informatique, Toronto, Ontario, septembre 1992.

[SLI91] M. Ben SLIMA, Étude d'applications d'un processeur numérique de signaux (DSP) , Mémoire de Maîtrise en électronique industrielle UQTR 1991.

[SLI92] M Ben SLIMA, R. Z. MORAWSKI et A. BARWICZ, "Splined-based Variational Method with Constraints for Spectrophotometric Data Correction", IEEE Trans.Instrumentation and Measurement, vol. 41, N°6, Décembre 1992, pp.786-790.

[SLI97] Note de cours de TRAITEMENT NUMÉRIQUE DU SIGNAL, par Mouhamed Ben Slima, Hiver 1997, Université du québec à trois-Rivières.

[TIK77] A.N. TIKHONOV ET V.Y. ARSENINE, Solutions of Ill-Posed Problems, Wiley/Winston, London, 1977.

[RAO91] P. Rao and M. Bayoumi, "An Algorithm Specific VLSI Parallel Architecture for kalman Filter", IEEE Press: VLSI Signal Processor, IV, 1991, pp. 264-273.

ANNEXE A

CODES EN MATLAB

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CODE  DE L'ALGORITHME ITÉRATIF DE KALMAN AVEC AJOUT DU
% CRITÈRE J_X , FAIT PAR DANIEL MASSICOTTE, MODIFIÉ PAR DIOMANDÉ
%  ABOU
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [w,xr,Y,Y1,Y2,Kopt,c,e] = abouw3(yb,h,K_kal,dt,C,top,x,y,g);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% yb= mesure de y avec du bruit,
% h= réponse impulsionnelle reduite,
% K_kal= gain de Kalman,
% dt= pas d'échantillonnage,
% C= constante de positivité,
% top= nombre d'itération maximale,
% x = objet a mesure,
% y=  objet mesur,
% g=réponse impulsionnelle du systeme
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ordre = 1;
Nb_pt_y =length(yb);
Nb_pt_x = Nb_pt_y;
Nb_pt_h = max(size(h));


z = zeros(Nb_pt_h,1);

Y=[];
```

```
V=[];
to=0.72;
c=0;
cri=6;

b=h;
b1=[0];
for it=1: top
for k=1:Nb_pt_y

  F_z = [ z(1) ; z(1:Nb_pt_h-1) ];
%   ye(k) = h'*F*z;
   ye(k) = h'*F_z;

  i(k) = yb(k) - ye(k);
%   z = F*z + K_kal*i(k);
   z = F_z + K_kal*i(k);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Application de la contrainte de positivité sur x_kal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  for n = 1:Nb_pt_h
    if z(n) < 0
      z(n) = z(n)*C;

    end
  end


  x_fkal(k) = z(Nb_pt_h);

 end

x_fkal = x_fkal'; % /dt;
z = z; % /dt;
i = i';
ye = ye';

for k = 1:retard
   x_fkal(Nb_pt_x+k-1) = z(retard-k+1);

end


xr = x_fkal(retard : Nb_pt_x + retard-1);
```

```
xr=xr(:);

b=conv(h,b);

w(it)=norm(x/max(x)-xr/max(xr))/norm(x/max(x));% w bon

Kopt(it)=to*norm(xr/max(xr))+(1-to)*norm(y/max(y)-yb/max(yb));

%%%%%%%%%%%%%%%%%%%%%%%%
%  APPLICATION DU CRITERE L-CUVE
%%%%%%%%%%%%%%%%%%%%%%%%

g_len=length(g)/2;
b_len=length(b)/2;
b1_len=length(b1)/2;

if fix(g_len)~=g_len,
        h1=[zeros(g_len+1,1);b;zeros(g_len,1)];
        h3=[zeros(g_len+1,1);b1;zeros(g_len,1)];
else
        h1=[zeros(g_len,1);b;zeros(g_len,1)];
        h3=[zeros(g_len,1);b1;zeros(g_len,1)];
end

if fix(b_len)~=b_len,
        h2=[zeros(b_len+1,1);g;zeros(b_len,1)];
else
        h2=[zeros(b_len,1);g;zeros(b_len,1)];
end

if fix(b1_len)~=b1_len,
        h4=[zeros(b1_len+1,1);g;zeros(b1_len,1)];
else
        h4=[zeros(b1_len,1);g;zeros(b1_len,1)];
end


v1=(norm(yb/max(yb)))^2/(norm(xr/max(xr)))^2; % CRITERE L-CUVE


v2=((norm(y/max(y)-yb/max(yb)))^2/(norm(y/max(y)-xr/max(xr))))^2;

v=v1-v2; %CRITERE L-CUVE
```

```
%%%%%%%%%%%%%%%%%%%
% SOUSTRACTION de G et h
%%%%%%%%%%%%%%%%%%%



figure(1)
plot(w)
xlabel('Nombre iteration')

label('Erreur')
yb=xr;
b1=b;

fprintf('iter= %d\n',it);
fprintf('cri= %f\t Kopt= %f\n',cri,Kopt(it));

if cri-Kopt(it)<0
        break;
else
cri=Kopt(it);

end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CREATION DES SIGNAUX SYNTETIQUE POUR LA SIMULATION DE
%
% L'ALGORITHME
%
% ITERATIF DE KALMAN
%
% FAIT PAR DANIEL MASSICOTTE MODIFIE PAR DIOMANDE ABOU
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [x,g,hh,y,yb,tx,SNR] = clepat1(LB,sigma_b);
dt=1; % j'ai fixe dt=1



tx = 0:dt:127;
tg = 0:dt:63;
tx = tx';
tg = tg';

%%%%%%%%%%%%%%%%%%%%%%%%
% Definition des signaux synthétiques
%%%%%%%%%%%%%%%%%%%%%%%%

%x=6*gaussoid(tx-40,3)+2*gaussoid(tx-70,3);%x0 essai2
%x =  2*gaussoid(tx-10,2.25) + 6*gaussoid(tx-30,2.25);% X0 essai

%x =  4*gaussoid(tx-50,2.25) + gaussoid(tx-70,2.25);%x1

x = gaussoid(tx-50,2.25)+4*gaussoid(tx-70,2.25) + 3*gaussoid(tx-90,2.25);%x2


x   =   8*gaussoid(tx-25,4)+4*gaussoid(tx-30,4)+6.5*gaussoid(tx-80,4)+2*gaussoid(tx-
100,4)+3*gaussoid(tx-150,4)+gaussoid(tx-160,4)+4*gaussoid(tx-170,4)+gaussoid(tx-
230,4);%tx = 0:dt:256;%x4


g = 0.0525*gaussoid(tg-32,8); % LA pour faire varier g
```

```
hh = gaussoid(tg-32,LB);


Nb_pt_x = length(x);
Nb_pt_y = Nb_pt_x;
Nb_pt_g = length(g);

y = conv(x,g)*dt;
y = y(Nb_pt_g/2+1:Nb_pt_g/2+Nb_pt_x);

%
%  Bruit sur la mesure
%

randn('seed',1) % j'ai five pois a 1
bruit = sigma_b * randn(Nb_pt_y,1);

if sigma_b == 0
  bruit = 0;
  SNR = inf;
  yb = y;
else
  bruit = sigma_b * randn(Nb_pt_y,1);
  bruit = sigma_b/std(bruit) * bruit;
yb = y + bruit;
  SNR = 20*log10( norm(y) / norm(yb-y) );
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% PROGRAMME PRINCIPAL PERMETANT LA DETERMINATION
% AUTOMATIQUE DE L' ITERATION Kopt avec JLCh
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [x,xr,y,w,Y,Y1,Y2,yb,Kopt]= EREKA4(LB,sigma_b,Q_R,top);

[x,g,hh,y,yb,tx,SNR] = clepat1(LB,sigma_b);
dt=1;
 Nb_pt_g = length(g);
 h=hh(Nb_pt_g/2+1-Nb_pt_g/8:Nb_pt_g/2+1+Nb_pt_g/8-1);

 Nb_pt = 3*max(size(h));
 [K,Erel_K] = gain_k(h,Nb_pt,Q_R,dt);
 retard=length(h)/2;
 K_kal=K;

 C=0;%modification



          [w,xr,Y,Y1,Y2,Kopt]=abouw33(yb,h,K_kal,dt,retard,C,top,x,y,g);

          end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% PROGRAMME PERMETTANT LA DETERMINATION
% AUTOMATIQUE DE L' 'ITERATION Kopt,     % par  Jx
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [x,xr,y,w,Y,Y1,Y2,yb,Kopt,c,e,S1]= EREKA12(LB,sigma_b,Q_R,top);

[x,g,hh,y,yb,tx,SNR] = clepat2(LB,sigma_b);
dt=1;
 Nb_pt_g = length(g);
 h=hh(Nb_pt_g/2+1-Nb_pt_g/8:Nb_pt_g/2+1+Nb_pt_g/8-1);

 Nb_pt = 3*max(size(h));
 [K,Erel_K] = gain_k(h,Nb_pt,Q_R,dt);
 retard=length(h)/2;
 K_kal=K;

 C=0;%modification


            [w,xr,Y,Y1,Y2,Kopt,c,e,S1]=abouw3(yb,h,K_kal,dt,retard,C,top,x,y);

            end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  SOUS PROGRAMME EREKA4
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x,xr,y,w,Y,Y1,Y2,yb,Kopt,c,e,S1]= EREKA4(LB,sigma_b,Q_R,top);

[x,g,hh,y,yb,tx,SNR] = clepat2(LB,sigma_b);
dt=1;
 Nb_pt_g = length(g);
 h=hh(Nb_pt_g/2+1-Nb_pt_g/8:Nb_pt_g/2+1+Nb_pt_g/8-1);

 Nb_pt = 3*max(size(h));
 [K,Erel_K] = gain_k(h,Nb_pt,Q_R,dt);
 retard=length(h)/2;
 K_kal=K;

 C=0;%modification


                [w,xr,Y,Y1,Y2,Kopt,c,e,S1]=abouw33(yb,h,K_kal,dt,retard,C,top,x,y);

                end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  SOUS PROGRAMME EREKA12
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x,xr,y,w,Y,Y1,Y2,yb,Kopt,c,e,S1]= EREKA12(LB,sigma_b,Q_R,top);

[x,g,hh,y,yb,tx,SNR] = clepat2(LB,sigma_b);
dt=1;
 Nb_pt_g = length(g);
 h=hh(Nb_pt_g/2+1-Nb_pt_g/8:Nb_pt_g/2+1+Nb_pt_g/8-1);

 Nb_pt = 3*max(size(h));
 [K,Erel_K] = gain_k(h,Nb_pt,Q_R,dt);
 retard=length(h)/2;
 K_kal=K;

 C=0;%modification


                [w,xr,Y,Y1,Y2,Kopt,c,e,S1]=abouw3(yb,h,K_kal,dt,retard,C,top,x,y);

                end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                     %
% PROGRAMME DE CONVERSION DECIMALE/BINAIRE VIRGULE FIXE  %
% COMPLEMENT A 2                                                      %
%                                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function c=bin(x,Nb_bit);
seuil=.5;
N_MAX=2^(Nb_bit-1)-1;


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

if x<0

x=N_Max-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
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FICHIER FORCE FILE POUR LA SIMULATION DE SYSKALY1 ET SYSKALY2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


fid=fopen('/u/hping/abou/mentor/DESIGN/stimul/SYSKALI_1_et_2.stim','w');

load k1.mat
load k2.mat
load h1.mat
load h2.mat
load y1.mat
load y2.mat


fprintf(fid,'forc  clk       0\n',1);
fprintf(fid,'Forc  pil_iny    0000000000000000\n',1);
fprintf(fid,'Forc  distanc    100001\n',1);
fprintf(fid,'Forc  chain      0\n',1);
fprintf(fid,'Forc  chain1     0\n',1);
fprintf(fid,'Forc  clr_z      1\n',1);
fprintf(fid,'Forc  clr_x      1\n',1);
fprintf(fid,'Forc  clr_cal    1\n',1);
fprintf(fid,'Forc  clr_acc    1\n',1);
fprintf(fid,'Forc  clr_s      1\n',1);
fprintf(fid,'Forc   s        1\n',1);
fprintf(fid,'Forc  en_Z       1\n',1);
fprintf(fid,'Forc  en_K       1\n',1);
fprintf(fid,'Forc  en_h       1\n',1);
fprintf(fid,'Forc  en_y       0\n',1);
fprintf(fid,'Forc  en_s       0\n',1);
fprintf(fid,'Forc  en_acc     0\n',1);
fprintf(fid,'Forc  d1  01000000\n',1);
fprintf(fid,'Forc  en_0       0\n',1);
fprintf(fid,'run             5\n',1);
fprintf(fid,'                \n',1);
fprintf(fid,'                \n',1);
```

```
% charge des piles K et H ********************************

for i=1:8
        fprintf(fid,' Forc clk      1\n',1);

                if length(num2str(k1(i)))==1
                        x1=strcat('0000000',num2str(k1(i)));
           elseif length(num2str(k1(i)))==2
                x1=strcat('000000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==3
                x1=strcat('00000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==4
                x1=strcat('0000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==5
                x1=strcat('000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==6
                x1=strcat('00',num2str(k1(i)));
                elseif length(num2str(k1(i)))==7
                x1=strcat('0',num2str(k1(i)));
                else
                x1=num2str(k1(i));
        end


                if length(num2str(k2(i)))==1
                        x2=strcat('0000000',num2str(k2(i)));
           elseif length(num2str(k2(i)))==2
                x2=strcat('000000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==3
                x2=strcat('00000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==4
                x2=strcat('0000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==5
                x2=strcat('000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==6
                x2=strcat('00',num2str(k2(i)));
                elseif length(num2str(k2(i)))==7
                x2=strcat('0',num2str(k2(i)));
                else
                x2=num2str(k2(i));
        end


                if length(num2str(h1(i)))==1
                        x3=strcat('0000000',num2str(h1(i)));
```

```
    elseif length(num2str(h1(i)))==2
        x3=strcat('000000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==3
        x3=strcat('00000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==4
        x3=strcat('0000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==5
        x3=strcat('000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==6
        x3=strcat('00',num2str(h1(i)));
        elseif length(num2str(h1(i)))==7
        x3=strcat('0',num2str(h1(i)));
        else
        x3=num2str(h1(i));
end


        if length(num2str(h2(i)))==1
                x4=strcat('0000000',num2str(h2(i)));
    elseif length(num2str(h2(i)))==2
        x4=strcat('000000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==3
        x4=strcat('00000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==4
        x4=strcat('0000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==5
        x4=strcat('000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==6
        x4=strcat('00',num2str(h2(i)));
        elseif length(num2str(h2(i)))==7
        x4=strcat('0',num2str(h2(i)));
        else
        x4=num2str(h2(i));
end

        fprintf(fid,' Forc chain        1\n',1);
        fprintf(fid,' Forc pil_ink1     %8s\n',x1);
        fprintf(fid,' Forc pil_ink2     %8s\n',x2);
        fprintf(fid,' Forc pil_inh1     %8s\n',x3);
        fprintf(fid,' Forc pil_inh2     %8s\n',x4);
        fprintf(fid,' run             5\n',1);
        fprintf(fid,' Forc clk          0\n',1);
        fprintf(fid,' Forc en_s         1\n',1);
        fprintf(fid,' run             5\n',1);

end;
```

```
%*************************************************************************

fprintf(fid,'              Forc clk          1\n',1);
fprintf(fid,'              Forc pil_ink1     0000000000000000\n',1);
fprintf(fid,'              Forc pil_ink2     0000000000000000\n',1);
fprintf(fid,'              Forc pil_inh1     0000000000000000\n',1);
fprintf(fid,'              Forc pil_inh2     0000000000000000\n',1);
fprintf(fid,'              run               5\n',1);
fprintf(fid,'                    Forc chain        0\n',1);
fprintf(fid,'              run               5\n',1);

% charge de la pile Y ***********************************************

for j=1:length(y1)


    if length(num2str(y1(j)))==1
                x5=strcat('0000000',num2str(y1(j)));
    elseif length(num2str(y1(j)))==2
        x5=strcat('000000',num2str(y1(j)));
        elseif length(num2str(y1(j)))==3
        x5=strcat('00000',num2str(y1(j)));
        elseif length(num2str(y1(j)))==4
        x5=strcat('0000',num2str(y1(j)));
        elseif length(num2str(y1(j)))==5
        x5=strcat('000',num2str(y1(j)));
        elseif length(num2str(y1(j)))==6
        x5=strcat('00',num2str(y1(j)));
        elseif length(num2str(y1(j)))==7
        x5=strcat('0',num2str(y1(j)));
        else
        x5=num2str(y1(j));
end;


        if length(num2str(y2(j)))==1
                x6=strcat('0000000',num2str(y2(j)));
    elseif length(num2str(y2(j)))==2
        x6=strcat('000000',num2str(y2(j)));
        elseif length(num2str(y2(j)))==3
        x6=strcat('00000',num2str(y2(j)));
        elseif length(num2str(y2(j)))==4
        x6=strcat('0000',num2str(y2(j)));
        elseif length(num2str(y2(j)))==5
        x6=strcat('000',num2str(y2(j)));
```

```
            elseif length(num2str(y2(j)))==6
            x6=strcat('00',num2str(y2(j)));
            elseif length(num2str(y2(j)))==7
            x6=strcat('0',num2str(y2(j)));
            else
            x6=num2str(y2(j));
        end;
            x7=strcat(x5,x6);

            fprintf(fid,' Forc clk          1\n',1);
            fprintf(fid,' Forc en_y         1\n',1);
            fprintf(fid,' Forc chain1       1\n',1);
            fprintf(fid,' Forc pil_iny      %s\n',x7);
            fprintf(fid,' run             5\n',1);
            fprintf(fid,' Forc clk          0\n',1);
            fprintf(fid,' Forc en_x         0\n',1);
            fprintf(fid,' Forc clr_x        1\n',1);
            fprintf(fid,' run             5\n',1);

end;


            fprintf(fid,'Forc   clk         1\n',1);
            fprintf(fid,'Forc   chain       0\n',1);
            fprintf(fid,'Forc   clr_z       0\n',1);
            fprintf(fid,'Forc   clr_x       0\n',1);
            fprintf(fid,'Forc   clr_cal     0\n',1);
            fprintf(fid,'Forc   s           1\n',1);
            fprintf(fid,'Forc   clr_acc     1\n',1);
            fprintf(fid,'Forc   en_cal      1\n',1);
            fprintf(fid,'Forc   pil_iny     %s\n',x7);
            fprintf(fid,'run              5\n',1);
            fprintf(fid,'Forc   clk         0\n',1);
            fprintf(fid,'                   Forc chain      0\n',1);
            fprintf(fid,'Forc   en_cal      0\n',1);
            fprintf(fid,'Forc   chain1      0\n',1);
            fprintf(fid,'Forc   en_K        1\n',1);
            fprintf(fid,'Forc   en_h        1\n',1);
            fprintf(fid,'Forc   en_acc      1\n',1);
            fprintf(fid,'forc   clr_acc     0\n',1);
            fprintf(fid,'run              5\n',1);
            fprintf(fid,'Forc   clk         1\n',1);


                fprintf(fid,'Forc   en_x        0\n',1);
                fprintf(fid,'Forc   en_0        0\n',1);
```

```
        fprintf(fid,'forc   clr_s       0\n',1);
        fprintf(fid,'Forc   en_cal      0\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk         0\n',1);
        fprintf(fid,'forc   en_x        0\n',1);
        fprintf(fid,'run              5\n',1);


        for i=1:5
                fprintf(fid,'Forc   clk   1\n',1);
                fprintf(fid,'run        5\n',1);
                fprintf(fid,'Forc   clk   0\n',1);
                fprintf(fid,'run        5\n',1);
        end;

        fprintf(fid,'Forc   clk         1\n',1);
        fprintf(fid,'Forc   s           0\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk         0\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk         1\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk         0\n',1);
        fprintf(fid,'Forc   en_x        0\n',1);
        fprintf(fid,'run              5\n',1);


a=0;
for j=1:length(y1)
        a=a+1;
        if a==3
                a=1;
        end;
        fprintf(fid,'Forc   clk         1\n',1);

                fprintf(fid,'Forc   s           1\n',1);
                fprintf(fid,'Forc   clr_acc     1\n',1);
                fprintf(fid,'Forc   en_cal      1\n',1);
                fprintf(fid,'Forc   en_y        1\n',1);
                fprintf(fid,'run              5\n',1);
                fprintf(fid,'Forc   clk         0\n',1);
                fprintf(fid,'Forc   en_y        0\n',1);
                fprintf(fid,'Forc   en_cal      0\n',1);
                fprintf(fid,'Forc   en_K        1\n',1);
                fprintf(fid,'Forc   en_h        1\n',1);
```

```
            fprintf(fid,'Forc   en_acc       1\n',1);
            fprintf(fid,'forc   clr_acc      0\n',1);
            fprintf(fid,'run                5\n',1);
            fprintf(fid,'Forc   clk          1\n',1);
            if a==2
                fprintf(fid,'Forc   en_x         1\n',1);
                fprintf(fid,'Forc   en_0         1\n',1);
            else
                fprintf(fid,'Forc   en_x         0\n',1);
                fprintf(fid,'Forc   en_0         0\n',1);
            end;
            fprintf(fid,'forc   clr_s        0\n',1);
            fprintf(fid,'Forc   en_cal       0\n',1);
            fprintf(fid,'run                5\n',1);
            fprintf(fid,'Forc   clk          0\n',1);
            fprintf(fid,'forc   en_x         0\n',1);
            fprintf(fid,'run                5\n',1);


            for i=1:5
                fprintf(fid,'Forc   clk   1\n',1);
                fprintf(fid,'run         5\n',1);
                fprintf(fid,'Forc   clk   0\n',1);
                fprintf(fid,'run         5\n',1);
            end;

            fprintf(fid,'Forc   clk          1\n',1);
            fprintf(fid,'Forc   s           0\n',1);
            fprintf(fid,'run                5\n',1);
            fprintf(fid,'Forc   clk          0\n',1);
            fprintf(fid,'run                5\n',1);
            fprintf(fid,'Forc   clk          1\n',1);
            fprintf(fid,'run                5\n',1);
            fprintf(fid,'Forc   clk          0\n',1);
            fprintf(fid,'Forc   en_x         0\n',1);
            fprintf(fid,'run                5\n',1);
end;


fclose(fid);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FICHIER FORCE FILE POUR LA SIMULATION DE SYSKALY3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid=fopen('/u/hping/abou/mentor/DESIGN/SYSKALI_3.stim','w');

load mizata.mat % Ficher contenant les donnes h,k,et y.

fprintf(fid,'forc   clk        0\n',1);
fprintf(fid,'Forc   y_mes      0000000000000000\n',1);
fprintf(fid,'Forc   distanc    100001\n',1);
fprintf(fid,'Forc   chain      0\n',1);
fprintf(fid,'Forc   clr_z      1\n',1);
fprintf(fid,'Forc   clr_x      1\n',1);
fprintf(fid,'Forc   clr_cal    1\n',1);
fprintf(fid,'Forc   clr_acc    1\n',1);
fprintf(fid,'Forc   clr_s      1\n',1);
fprintf(fid,'Forc   s          1\n',1);
fprintf(fid,'Forc   en_Z       1\n',1);
fprintf(fid,'Forc   en_K       1\n',1);
fprintf(fid,'Forc   en_h       1\n',1);
fprintf(fid,'Forc   en_s       0\n',1);
fprintf(fid,'Forc   en_p       1\n',1);
fprintf(fid,'Forc   en_acc     0\n',1);
fprintf(fid,'Forc   Z_out19    0000000000000000\n',1);
fprintf(fid,'Forc   Z_out20    0000000000000000\n',1);
fprintf(fid,'Forc   Z_out21    0000000000000000\n',1);
fprintf(fid,'Forc   Z_out22    0000000000000000\n',1);
fprintf(fid,'run               5\n',1);
fprintf(fid,'                  \n',1);
fprintf(fid,'                  \n',1);
```

```
% charge des piles K et H ********************************


for i=1:8
        fprintf(fid,' Forc clk      1\n',1);

                if length(num2str(k1(i)))==1
                        x1=strcat('0000000',num2str(k1(i)));
           elseif length(num2str(k1(i)))==2
                x1=strcat('000000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==3
                x1=strcat('00000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==4
                x1=strcat('0000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==5
                x1=strcat('000',num2str(k1(i)));
                elseif length(num2str(k1(i)))==6
                x1=strcat('00',num2str(k1(i)));
                elseif length(num2str(k1(i)))==7
                x1=strcat('0',num2str(k1(i)));
                else
                x1=num2str(k1(i));

        end;


                if length(num2str(k2(i)))==1
                        x2=strcat('0000000',num2str(k2(i)));
           elseif length(num2str(k2(i)))==2
                x2=strcat('000000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==3
                x2=strcat('00000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==4
                x2=strcat('0000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==5
                x2=strcat('000',num2str(k2(i)));
                elseif length(num2str(k2(i)))==6
                x2=strcat('00',num2str(k2(i)));
                elseif length(num2str(k2(i)))==7
                x2=strcat('0',num2str(k2(i)));
                else
                x2=num2str(k2(i));

        end;
        x22=strcat(x1,x2);
```

```
        if length(num2str(h1(i)))==1
                x3=strcat('0000000',num2str(h1(i)));
   elseif length(num2str(h1(i)))==2
        x3=strcat('000000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==3
        x3=strcat('00000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==4
        x3=strcat('0000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==5
        x3=strcat('000',num2str(h1(i)));
        elseif length(num2str(h1(i)))==6
        x3=strcat('00',num2str(h1(i)));
        elseif length(num2str(h1(i)))==7
        x3=strcat('0',num2str(h1(i)));
        else
        x3=num2str(h1(i));
end;


        if length(num2str(h2(i)))==1
                x4=strcat('0000000',num2str(h2(i)));
   elseif length(num2str(h2(i)))==2
        x4=strcat('000000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==3
        x4=strcat('00000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==4
        x4=strcat('0000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==5
        x4=strcat('000',num2str(h2(i)));
        elseif length(num2str(h2(i)))==6
        x4=strcat('00',num2str(h2(i)));
        elseif length(num2str(h2(i)))==7
        x4=strcat('0',num2str(h2(i)));
        else
        x4=num2str(h2(i));

end;
 x44=strcat(x3,x4);

        fprintf(fid,' Forc chain          1\n',1);
        fprintf(fid,' Forc pil_ink1       %8s\n',x22);
        fprintf(fid,' Forc pil_ink2       %8s\n',x22);
        fprintf(fid,' Forc pil_inh1       %8s\n',x44);
        fprintf(fid,' Forc pil_inh2       %8s\n',x44);
        fprintf(fid,' run               5\n',1);
```

```
            fprintf(fid,' Forc clk          0\n',1);
            fprintf(fid,' Forc en_s          1\n',1);
            fprintf(fid,' run              5\n',1);

        end;

fprintf(fid,'           Forc clk          1\n',1);
fprintf(fid,'           Forc pil_ink1     0000000000000000\n',1);
fprintf(fid,'           Forc pil_ink2     0000000000000000\n',1);
fprintf(fid,'           Forc pil_inh1     0000000000000000\n',1);
fprintf(fid,'           Forc pil_inh2     0000000000000000\n',1);
fprintf(fid,'           run              5\n',1);
fprintf(fid,'           Forc clk          0\n',1);
fprintf(fid,'           Forc chain        0\n',1);
fprintf(fid,'           Forc clr_Z        0\n',1);
fprintf(fid,'           Forc clr_cal      0\n',1);
fprintf(fid,'           Forc clr_x        0\n',1);
fprintf(fid,'           Forc en_h         0\n',1);
fprintf(fid,'           Forc en_K         0\n',1);
fprintf(fid,'           run              50\n',1);


% charge de la pile Y ************************************************

for j=1:length(y10)

        if length(num2str(y10(j)))==1
                x5=strcat('0000000',num2str(y10(j)));
        elseif length(num2str(y10(j)))==2
            x5=strcat('000000',num2str(y10(j)));
            elseif length(num2str(y10(j)))==3
            x5=strcat('00000',num2str(y10(j)));
            elseif length(num2str(y10(j)))==4
            x5=strcat('0000',num2str(y10(j)));
            elseif length(num2str(y10(j)))==5
            x5=strcat('000',num2str(y10(j)));
            elseif length(num2str(y10(j)))==6
            x5=strcat('00',num2str(y10(j)));
            elseif length(num2str(y10(j)))==7
            x5=strcat('0',num2str(y10(j)));
            else
            x5=num2str(y10(j));
        end;


            if length(num2str(y20(j)))==1
```

```
                x6=strcat('0000000',num2str(y20(j)));
  elseif length(num2str(y20(j)))==2
       x6=strcat('000000',num2str(y20(j)));
       elseif length(num2str(y20(j)))==3
       x6=strcat('00000',num2str(y20(j)));
       elseif length(num2str(y20(j)))==4
       x6=strcat('0000',num2str(y20(j)));
       elseif length(num2str(y20(j)))==5
       x6=strcat('000',num2str(y20(j)));
       elseif length(num2str(y20(j)))==6
       x6=strcat('00',num2str(y20(j)));
       elseif length(num2str(y20(j)))==7
       x6=strcat('0',num2str(y20(j)));
       else
       x6=num2str(y20(j));
end;

  x7=strcat(x5,x6);



 if length(num2str(y100(j)))==1
                x15=strcat('0000000',num2str(y100(j)));
  elseif length(num2str(y100(j)))==2
       x15=strcat('000000',num2str(y100(j)));
       elseif length(num2str(y100(j)))==3
       x15=strcat('00000',num2str(y100(j)));
       elseif length(num2str(y100(j)))==4
       x15=strcat('0000',num2str(y100(j)));
       elseif length(num2str(y100(j)))==5
       x15=strcat('000',num2str(y100(j)));
       elseif length(num2str(y100(j)))==6
       x15=strcat('00',num2str(y100(j)));
       elseif length(num2str(y100(j-60)))==7
       x15=strcat('0',num2str(y100(j)));
       else
       x15=num2str(y100(j));
end;


       if length(num2str(y200(j)))==1
                x16=strcat('0000000',num2str(y200(j)));
  elseif length(num2str(y200(j)))==2
       x16=strcat('000000',num2str(y200(j)));
       elseif length(num2str(y200(j)))==3
```

```
        x16=strcat('00000',num2str(y200(j)));
        elseif length(num2str(y200(j)))==4
        x16=strcat('0000',num2str(y200(j)));
        elseif length(num2str(y200(j)))==5
        x16=strcat('000',num2str(y200(j)));
        elseif length(num2str(y200(j)))==6
        x16=strcat('00',num2str(y200(j)));
        elseif length(num2str(y200(j)))==7
        x16=strcat('0',num2str(y200(j)));
        else
        x16=num2str(y200(j));
end;

   x8=strcat(x15,x16);



        fprintf(fid,'Forc   clk          1\n',1);
        fprintf(fid,'Forc   s           1\n',1);
        fprintf(fid,'Forc   clr_acc      1\n',1);
        fprintf(fid,'Forc   en_cal       1\n',1);
        fprintf(fid,'Forc   en_x         1\n',1);
        fprintf(fid,'Forc   y_mes        %s\n',x7);% x8 en x7
        fprintf(fid,'Forc   Z_out19      %s\n',x8);
        fprintf(fid,'Forc   Z_out20      %s\n',x8);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk          0\n',1);
        fprintf(fid,'Forc   en_K         1\n',1);
        fprintf(fid,'Forc   en_h         1\n',1);
        fprintf(fid,'forc   en_x         0\n',1);
        fprintf(fid,'Forc   en_acc       1\n',1);
        fprintf(fid,'forc   clr_acc      0\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk          1\n',1);
        fprintf(fid,'forc   clr_s        0\n',1);
        fprintf(fid,'Forc   en_cal       0\n',1);
        fprintf(fid,'run              5\n',1);
        fprintf(fid,'Forc   clk          0\n',1);
        fprintf(fid,'run              5\n',1);



        for i=1:29
                fprintf(fid,'Forc   clk   1\n',1);
                fprintf(fid,'run         5\n',1);
```

```
                fprintf(fid,'Forc   clk   0\n',1);
                fprintf(fid,'run        5\n',1);

            end;
            fprintf(fid,'Forc   clk       1\n',1);
            fprintf(fid,'Forc   s         0\n',1);
            fprintf(fid,'run            5\n',1);
            fprintf(fid,'Forc   clk       0\n',1);
            fprintf(fid,'run            5\n',1);
            fprintf(fid,'Forc   clk       1\n',1);
            fprintf(fid,'run            5\n',1);
            fprintf(fid,'Forc   clk       0\n',1);
            fprintf(fid,'Forc   en_x      0\n',1);
            fprintf(fid,'run            5\n',1);


 end;


fclose(fid);
```

ANNEXE B

CODES EN VHDL

```
------------------------------------------------------------------------------------------------------
--
-- PROGRAMME VHDL DU PROCESSEUR SYSKALI1
--
------------------------------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.pack_processeur_syskali.all;

ENTITY pro1_entier IS

        PORT(pil_ink1,pil_ink2    :IN    std_logic_vector(15  downto 0);
         pil_inh1,pil_inh2,d1  :IN    std_logic_vector(15  downto 0);
           clk,clr_Z ,chain, s   :IN    std_logic;
         en_Z,en_K,en_s,en_h   :IN    std_logic;
         pil_iny           :IN    std_logic_vector(15 downto 0);
         clr_acc,en_acc,clr_s  :IN    std_logic;
         distanc           :IN    std_logic_vector(5 downto 0);
         en_x,clr_x,en_0,chain1:IN    std_logic;
         clr_cal,en_cal,en_y   :IN    std_logic;
           xout              :out   std_logic_vector(15 downto 0));


  END pro1_entier  ;

ARCHITECTURE comportement OF pro1_entier IS

signal  xout1,add_sortie,add_entree,must:std_logic_vector(15 downto 0);
signal  mult1,mult2,add_out1,add_out2   :std_logic_vector(15 downto 0);
signal  mux_out, Z_out1,Z_out2        :std_logic_vector(15 downto 0);
signal  pil_outk1 ,pil_outk2          :std_logic_vector(15 downto 0);
signal  pil_outh1 ,pil_outh2          :std_logic_vector(15 downto 0);
signal  sortie1 ,sortie2            :std_logic_vector(15 downto 0);
signal  yout1,yout2, yout             :std_logic_vector(15 downto 0);
signal  yacc_in, yacc_out             :std_logic_vector(15 downto 0);
```

```
signal  y_cal_in, y_cal_out,xout2      :std_logic_vector(15 downto 0);
signal  inovation ,y_mes               :std_logic_vector(15 downto 0);
begin

soustracteur : soustract1          port map (y_mes , y_cal_out , inovation);

MULT_1     : multiplieur_16_8      port map ( inovation,pil_outk1,mult1 );
MULT_2     : multiplieur_16_8      port map ( inovation,pil_outk2,mult2 );
ADD_1      : adder1                port map (mult1,mux_out, add_out1);
ADD_2      : adder1                port map (mult2,Z_out1 , add_out2);
Reg1       : reg16                 port map (add_out1,clk,en_s,clr_s,sortie1);
Reg2       : reg16                 port map (add_out2,clk,en_s,clr_s,sortie2);
BLOC_Z1    : bloc_stock_donnees_32 port map (clk,en_Z,clr_Z,sortie1,Z_out1);
BLOC_Z2    : bloc_stock_donnees_32 port map (clk,en_Z,clr_Z,sortie2,Z_out2);
Multiplex  : mux2a1                port map (Z_out1,Z_out2,s,mux_out);
BLOC_K1    : PILE_34_16            port map (clk, en_K, chain, pil_ink1, pil_outk1);
BLOC_K2    : PILE_34_16            port map (clk, en_K, chain, pil_ink2, pil_outk2);
MULT_3     : multiplieur_16_16     port map ( sortie1,pil_outh1,yout1 );
MULT_4     : multiplieur_16_16     port map ( sortie2,pil_outh2,yout2 );
BLOC_h1    : PILE_34_16            port map (clk, en_h, chain, pil_inh1, pil_outh1);
BLOC_h2    : PILE_34_16            port map (clk, en_h, chain, pil_inh2, pil_outh2);
ADD_3      : adder1                port map (yout1,yout2 , yout);
ADD_4      : adder1                port map (yout,yacc_in , yacc_out);
Reg3       : reg16                 port map (yacc_out,clk,en_acc,clr_acc,yacc_in);
decaleur   : decaleur_gauch_droit  port map (distanc,yacc_in,y_cal_in);
Reg4       : reg16                 port map (y_cal_in,clk,en_cal,clr_cal,y_cal_out);
Reg6       : reg16                 port map (z_out2 ,clk,en_x,clr_x,xout1);
Reg7       : reg16                 port map (xout1 ,en_x,en_x,clr_x,add_entree);
ADD1_sortie : adder1               port map (add_entree,xout1,add_sortie);
MULT_sortie : multiplieur_16_16    port map ( add_sortie,d1,must );
Multiplex_S : mux2a1               port map (xout1,must,en_0,xout2);
BLOC_X         : PILE_SORTIE_X     port map
(clk,en_y,chain1,xout2,pil_iny,y_mes);
Reg_sortie  : reg16                port map (xout2 ,clk,en_cal,clr_cal,xout);

END comportement ;
```

```
--------------------------------------------------------------------------------------------------
--
-- PROGRAMME VHDL DU PROCESSEUR SYSKALI2
--
--------------------------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.pack_processeur_syskal.all;

ENTITY pro2_entier IS

        PORT(pil_ink1,pil_ink2    :IN    std_logic_vector(7  downto 0);
         pil_inh1,pil_inh2,d1     :IN    std_logic_vector(7  downto 0);
           clk,clr_Z ,chain, s    :IN    std_logic;
         en_Z,en_K,en_s,en_h      :IN    std_logic;
         y_mes                    :IN    std_logic_vector(15 downto 0);
         clr_acc,en_acc,clr_s     :IN    std_logic;
         distanc                  :IN    std_logic_vector(5 downto 0);
         en_x , clr_x,en_0        :IN    std_logic;
         clr_cal, en_cal          :IN    std_logic;
           xout                   :out   std_logic_vector(15 downto 0));


  END pro2_entier  ;

ARCHITECTURE comportement OF pro2_entier IS

signal  xout1,add_sortie,add_entree,must:std_logic_vector(15 downto 0);
signal  mult1,mult2,add_out1,add_out2   :std_logic_vector(15 downto 0);
signal  mux_out, Z_out1,Z_out2          :std_logic_vector(15 downto 0);
signal  Z_out3,Z_out4                   :std_logic_vector(15 downto 0);
signal  Z_out5,Z_out6                   :std_logic_vector(15 downto 0);
signal  pil_outk1 ,pil_outk2            :std_logic_vector(15 downto 0);
signal  pil_outh1 ,pil_outh2            :std_logic_vector(15 downto 0);
signal  sortie1 ,sortie2                :std_logic_vector(15 downto 0);
signal  yout1,yout2, yout,xout2         :std_logic_vector(15 downto 0);
signal  yacc_in, yacc_out               :std_logic_vector(15 downto 0);
signal  y_cal_in, y_cal_out,y_cal_out1  :std_logic_vector(15 downto 0);
signal  inovation,inovation1            :std_logic_vector(15 downto 0);
begin
```

```
soustracteur : soustract1          port map (y_mes , y_cal_out , inovation1);

MULT_1    : multiplieur_16_16      port map ( inovation,pil_outk1,mult1 );
MULT_2    : multiplieur_16_16      port map ( inovation,pil_outk2,mult2 );
ADD_1     : adder1                 port map (mult1,mux_out, add_out1);
ADD_2     : adder1                 port map (mult2,Z_out1 , add_out2);
Reg1      : reg16                  port map (add_out1,clk,en_s,clr_s,sortie1);
Reg2      : reg16                  port map (add_out2,clk,en_s,clr_s,sortie2);
BLOC_Z1   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,sortie1,Z_out1);
BLOC_Z3   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,Z_out1,Z_out3);
BLOC_Z5   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,Z_out3,Z_out5);
BLOC_Z2   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,sortie2,Z_out2);
BLOC_Z4   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,Z_out2,Z_out4);
BLOC_Z6   : bloc_stock_donnees_8   port map (clk,en_Z,clr_Z,Z_out4,Z_out6);
Multiplex : mux2a1                 port map (Z_out5,Z_out6,s,mux_out);
BLOC_K1   : PILE_8_16              port map (clk, en_K, chain, pil_ink1, pil_outk1);
BLOC_K2   : PILE_8_16              port map (clk, en_K, chain, pil_ink2, pil_outk2);
MULT_3    : multiplieur_16_16      port map ( sortie1,pil_outh1,yout1 );
MULT_4    : multiplieur_16_16      port map ( sortie2,pil_outh2,yout2 );
BLOC_h1   : PILE_8_16              port map (clk, en_h, chain, pil_inh1, pil_outh1);
BLOC_h2   : PILE_8_16              port map (clk, en_h, chain, pil_inh2, pil_outh2);
ADD_3     : adder1                 port map (yout1,yout2 , yout);
ADD_4     : adder1                 port map (yout,yacc_in , yacc_out);
Reg3      : reg16                  port map (yacc_out,clk,en_acc,clr_acc,yacc_in);
decaleur  : decaleur_gauch_droit   port map (distanc,yacc_in,y_cal_in);
Reg4      : reg16                  port map (y_cal_in,clk,en_cal,clr_cal,y_cal_out1);
Reg8      : reg16                  port map (y_cal_out1,clk,en_cal,clr_cal,y_cal_out);
Reg6      : reg16                  port map (z_out6 ,clk,en_x,clr_x,xout1);
Reg_sortie : reg16                 port map (xout1 ,en_x,en_x,clr_x,add_entree);
ADD1_sortie : adder1               port map (add_entree,xout1,add_sortie);
MULT_sortie : multiplieur_16_16    port map ( add_sortie,d1,must );
Multiplex_S : mux2a1               port map (must,xout1,en_0,xout2);
Reg2_sortie : reg16                port map (xout2 ,clk,en_cal,clr_cal,xout);
Multiplex_y : mux2a1               port map (inovation1,xout2,en_cal,inovation);

END comportement ;
```

```
--------------------------------------------------------------------------------------------------
--
-- PROGRAMME VHDL DU PROCESSEUR SYSKALI3
--
--------------------------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.pack_processeur_syskali.all;

ENTITY Pro3_entier IS

        PORT(pil_ink1,pil_ink2    :IN    std_logic_vector(15  downto 0);
         pil_inh1,pil_inh2    :IN    std_logic_vector(15  downto 0);
           clk,clr_Z ,chain, s   :IN    std_logic;
         en_Z,en_K,en_s,en_h   :IN    std_logic;
         y_mes,Z_out19,Z_out20 :IN    std_logic_vector(15 downto 0);
         Z_out21,Z_out22       :IN    std_logic_vector(15 downto 0);
         clr_acc,en_acc,clr_s  :IN    std_logic;
         distanc               :IN    std_logic_vector(5 downto 0);
         en_x , clr_x,en_p     :IN    std_logic;
         clr_cal, en_cal       :IN    std_logic;
           xout                :out   std_logic_vector(15 downto 0));


  END Pro3_entier  ;

ARCHITECTURE comportement OF Pro3_entier IS

signal  mult1,mult2,add_out1,add_out2   :std_logic_vector(15 downto 0);
signal  mux_out                         :std_logic_vector(15 downto 0);
signal  pil_outk1 ,pil_outk2            :std_logic_vector(15 downto 0);
signal  pil_outh1 ,pil_outh2            :std_logic_vector(15 downto 0);
signal  sortie1 ,sortie2                :std_logic_vector(15 downto 0);
signal  yout1,yout2, yout               :std_logic_vector(15 downto 0);
signal  yacc_in, yacc_out               :std_logic_vector(15 downto 0);
signal  y_cal_in, y_cal_out             :std_logic_vector(15 downto 0);
signal  inovation                       :std_logic_vector(15 downto 0);
signal  Z_out1,Z_out2,Z_out3,Z_out4     :std_logic_vector(15 downto 0);
```

```
signal Z_out5,Z_out6,Z_out7,Z_out8    :std_logic_vector(15 downto 0);
signal Z_out9,Z_out10,Z_out11,Z_out12 :std_logic_vector(15 downto 0);
signal Z_out13,Z_out14,Z_out15,Z_out16 :std_logic_vector(15 downto 0);
begin

soustracteur : soustract1          port map (y_mes , y_cal_out , inovation);

MULT_1    : multiplieur_16_16      port map ( inovation,pil_outk1,mult1 );
MULT_2    : multiplieur_16_16      port map ( inovation,pil_outk2,mult2 );
ADD_1     : adder1                 port map (mult1,mux_out, add_out1);
ADD_2     : adder1                 port map (mult2,Z_out1 , add_out2);
Reg1      : reg16                  port map (add_out1,clk,en_s,clr_s,sortie1);
Reg2      : reg16                  port map (add_out2,clk,en_s,clr_s,sortie2);
BLOC_Z1   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,sortie1,Z_out1);
BLOC_Z3   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out1,Z_out3);
BLOC_Z5   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out3,Z_out5);
BLOC_Z7   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out5,Z_out7);
BLOC_Z9   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out7,Z_out9);
BLOC_Z11  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out9,Z_out11);
BLOC_Z13  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out11,Z_out13);
BLOC_Z15  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out13,Z_out15);
BLOC_Z2   : bloc_stock_donnees_32  port map (clk,en_Z,clr_Z,sortie2,Z_out2);
BLOC_Z4   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out2,Z_out4);
BLOC_Z6   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out4,Z_out6);
BLOC_Z8   : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out6,Z_out8);
BLOC_Z10  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out8,Z_out10);
BLOC_Z12  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out10,Z_out12);
BLOC_Z14  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out12,Z_out14);
BLOC_Z16  : bloc_stock_donnees_33  port map (clk,en_Z,clr_Z,Z_out14,Z_out16);

Multiplex : mux2a1                 port map (Z_out15,Z_out16,s,mux_out);
BLOC_K1   : PILE_8_16              port map (clk, en_K, chain, pil_ink1, pil_outk1);
BLOC_K2   : PILE_8_16              port map (clk, en_K, chain, pil_ink2, pil_outk2);
MULT_3    : multiplieur_16_16      port map ( sortie1,pil_outh1,yout1 );
MULT_4    : multiplieur_16_16      port map ( sortie2,pil_outh2,yout2 );
BLOC_h1   : PILE_8_16              port map (clk, en_h, chain, pil_inh1, pil_outh1);
BLOC_h2   : PILE_8_16              port map (clk, en_h, chain, pil_inh2, pil_outh2);
ADD_3     : adder1                 port map (yout1,yout2 , yout);
ADD_4     : adder1                 port map (yout,yacc_in , yacc_out);
Reg3      : reg16                  port map (yacc_out,clk,en_acc,clr_acc,yacc_in);
decaleur  : decaleur_gauch_droit   port map (distanc,yacc_in,y_cal_in);
Reg4      : reg16                  port map (y_cal_in,clk,en_cal,clr_cal,y_cal_out);
Reg10     : mux2a1                 port map (Z_out16 ,Z_out2,en_p,xout);

END comportement ;
```

```
--------------------------------------------------------------------------------------------------
--
-- PACKAGE DES PROCESSEURS SYSKALI
--
--------------------------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;


PACKAGE pack_processeur_syskali IS


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;


PACKAGE pack_processeur_syskali3 IS



COMPONENT  multiplieur_16_8

        port ( A      : in  std_logic_vector (15  DOWNTO 0)  ;
          B      : in  std_logic_vector (7   DOWNTO 0)  ;
          Sortie  : out std_logic_vector (15  DOWNTO 0)  );


END component ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

COMPONENT  multiplieur_16_16

        port ( A      : in  std_logic_vector (15  DOWNTO 0)  ;
          B      : in  std_logic_vector (15  DOWNTO 0)  ;
          Sortie  : out std_logic_vector (15  DOWNTO 0)  );


END component ;
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
COMPONENT  adder1
        PORT( input1: IN std_logic_vector (15 DOWNTO 0);
            input2: IN std_logic_vector (15 DOWNTO 0);
            output: OUT std_logic_vector(15 DOWNTO 0));
END component ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
COMPONENT  soustract1
        PORT( input1: IN std_logic_vector (15 DOWNTO 0);
            input2: IN std_logic_vector (15 DOWNTO 0);
            output: OUT std_logic_vector(15 DOWNTO 0));
        END component ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
COMPONENT PILE_34_8
GENERIC (larg     : INTEGER :=7);
         PORT   (clk      : IN std_logic;
                        en      : IN std_logic;
                        chain   : IN std_logic;
            pile_in  : IN std_logic_vector(larg DOWNTO 0);
            pile_out : BUFFER std_logic_vector(larg DOWNTO 0));
END COMPONENT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
COMPONENT PILE_8_16
GENERIC (larg     : INTEGER :=15);
         PORT   (clk      : IN std_logic;
                        en      : IN std_logic;
                        chain   : IN std_logic;
            pile_in  : IN std_logic_vector(larg DOWNTO 0);
            pile_out : BUFFER std_logic_vector(larg DOWNTO 0));
END COMPONENT;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
COMPONENT mux2a1
        PORT(IN0      : in std_logic_vector(15 downto 0);
         IN1      : in std_logic_vector(15 downto 0);
         S        : in std_logic;
         OUT_MUX  : out std_logic_vector(15 downto 0));
END COMPONENT;
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
COMPONENT reg16
   PORT( d          : IN std_logic_vector (15 DOWNTO 0);
         clk, en, clr : IN std_logic;
         q          : OUT std_logic_vector (15 DOWNTO 0));
END COMPONENT;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
COMPONENT bloc_stock_donnees_33
   PORT( clk     : IN  std_logic;
         en      : IN  std_logic;
         clr     : IN  std_logic;
         pile_in : IN  std_logic_vector(15 DOWNTO 0);
         pile_out: OUT std_logic_vector(15 DOWNTO 0));

END COMPONENT;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
COMPONENT bloc_stock_donnees_32
   PORT( clk     : IN  std_logic;
         en      : IN  std_logic;
         clr     : IN  std_logic;
         pile_in : IN  std_logic_vector(15 DOWNTO 0);
         pile_out: OUT std_logic_vector(15 DOWNTO 0));

END COMPONENT;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
COMPONENT bloc_stock_donnees_8
   PORT( clk     : IN  std_logic;
         en      : IN  std_logic;
         clr     : IN  std_logic;
         pile_in : IN  std_logic_vector(15 DOWNTO 0);
         pile_out: OUT std_logic_vector(15 DOWNTO 0));

END COMPONENT;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
COMPONENT decaleur_gauch_droit



PORT(distance:IN std_logic_vector (5 DOWNTO 0);
```

```
                input: IN std_logic_vector (15 DOWNTO 0);
                output: OUT std_logic_vector (15 DOWNTO 0));

END COMPONENT;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

END pack_processeur_syskali3 ;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------


LIBRARY IEEE ;

USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_ARITH.all ;



ENTITY multiplieur_16_8 IS
  port ( A      : in  std_logic_vector (15  DOWNTO 0)  ;
       B      : in  std_logic_vector (7   DOWNTO 0)  ;
       Sortie  : out std_logic_vector (15  DOWNTO 0)  );

END multiplieur_16_8;

ARCHITECTURE behav OF multiplieur_16_8 IS
 BEGIN
 PROCESS(A,B)
        variable aa   : std_logic_vector (14  DOWNTO 0):="000000000000000" ;
  variable bb   : std_logic_vector (6   DOWNTO 0):= "0000000" ;
        variable sig  : std_logic;
        variable S    : std_logic_vector(21 downto 0);


                begin
                aa  := A(14 downto 0);
                bb  := B( 6 downto 0);
                S   := unsigned(aa)*unsigned(bb);

                sig := A(15) xor B(7);

                Sortie <= sig & S(21 downto 7);
```

```
 end PROCESS;
END behav;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


LIBRARY IEEE ;

USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_ARITH.all ;



ENTITY multiplieur_16_16 IS
  port ( A      : in  std_logic_vector (15  DOWNTO 0)  ;
       B      : in  std_logic_vector (15  DOWNTO 0)  ;
       Sortie  : out std_logic_vector (15  DOWNTO 0)  );

END multiplieur_16_16;

ARCHITECTURE behav OF multiplieur_16_16 IS
 BEGIN
 PROCESS(A,B)
        variable aa   : std_logic_vector (14  DOWNTO 0):="000000000000000" ;
  variable bb   : std_logic_vector (14  DOWNTO 0):="000000000000000" ;
        variable sig  : std_logic;
        variable S    : std_logic_vector(29 downto 0);


                begin
                aa  := A(14 downto 0);
                bb  := B(14 downto 0);
                S   := unsigned(aa)*unsigned(bb);

                sig := A(15) xor B(15);

                Sortie <= sig & S(29 downto 15);
 end PROCESS;
END behav;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

```
USE IEEE.std_logic_arith.all;


ENTITY adder1 IS
        PORT( input1: IN std_logic_vector (15 DOWNTO 0);
              input2: IN std_logic_vector (15 DOWNTO 0);
              output: OUT std_logic_vector(15 DOWNTO 0));
        END adder1;

ARCHITECTURE behave OF adder1 IS
BEGIN
PROCESS(input1,input2)
   VARIABLE e1,e2 ,s : std_logic_vector (15 DOWNTO 0);
   BEGIN
        IF (input1(15)= '0')then
           e1:= input1;
        elsif (input1="1000000000000000")then
           e1:= NOT(input1(15))&input1(14 downto 0);
        else
           e1:= input1(15)&NOT(input1(14 downto 0));
        END IF;

        IF (input2(15)= '0')then
           e2:= input2;
        elsif (input2="1000000000000000")then
           e2:= NOT(input2(15))&input2(14 downto 0);
        else
           e2:= input2(15)&NOT(input2(14 downto 0));
        END IF;

        s:= unsigned(e1) + unsigned(e2) ;

        IF (s(15)= '0')then
            output<= s;
        elsif (s ="1000000000000000")then
           output<= NOT(s(15))&s(14 downto 0);
         else
           output<= s(15)&NOT(s(14 downto 0));
        END IF;
END PROCESS;

END behave;
```

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

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;



ENTITY  soustract1  IS
        PORT( input1: IN std_logic_vector (15 DOWNTO 0);
            input2: IN std_logic_vector (15 DOWNTO 0);
            output: OUT std_logic_vector(15 DOWNTO 0));
        END soustract1  ;

ARCHITECTURE behave OF soustract1  IS

BEGIN
PROCESS(input1,input2)
   VARIABLE e1,e2 ,s : std_logic_vector (15 DOWNTO 0);
   BEGIN


       if (input1(15)='0') then
          e1 := input1;
                          else e1 := input1(15)& not(input1(14 downto 0));
       end if;

       if (input2(15)='0')then
          e2 := Not(input2);
       else
          e2 := not(input2(15))&input2(14 downto 0);

        END IF;
        s:= unsigned(e1) + unsigned(e2);

        IF (s(15)= '0')then
           output<= s;


                                  else
          output<= s(15) & NOT(s(14 downto 0));
        END IF;
END PROCESS;
```

```
END behave;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;



ENTITY PILE_34_8 IS
     GENERIC (larg     : INTEGER :=7);
          PORT   (clk      : IN std_logic;
                       en      : IN std_logic;
                       chain   : IN std_logic;
           pile_in  : IN std_logic_vector(larg DOWNTO 0);
           pile_out : BUFFER std_logic_vector(larg DOWNTO 0));
END PILE_34_8;

ARCHITECTURE behav OF PILE_34_8 IS
          CONSTANT stack_depth: INTEGER := 31;
     TYPE pile_vect IS ARRAY (stack_depth DOWNTO 0) OF std_logic_vector(larg
DOWNTO 0);
          SIGNAL a: std_logic_vector(larg DOWNTO 0);
          SIGNAL gated_clk : std_logic;

BEGIN
          gated_clk <= clk AND en;
          mux: PROCESS (chain,pile_in,pile_out)
                       VARIABLE aa : std_logic_vector(larg DOWNTO 0);
          BEGIN
                       CASE chain IS
                            WHEN   '1' => aa := pile_in;
                       WHEN   '0' => aa := pile_out;
                            WHEN OTHERS => aa := (OTHERS => 'X');
                       END CASE;
                  a <= aa;
          END PROCESS mux;

     reg_ar: PROCESS (gated_clk)
                  VARIABLE temp_pile: pile_vect;
     BEGIN
                IF ((gated_clk = '1')  AND (gated_clk'EVENT)) THEN
                                  pile_out <= temp_pile(0);
                                  FOR i IN 0 TO ((stack_depth)-1) LOOP
```

```
                                        temp_pile(i) := temp_pile(i+1);
                                    END LOOP;
                                    temp_pile(stack_depth) := a;
                END IF;
            END PROCESS reg_ar;
END behav;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;



ENTITY PILE_8_16 IS
    GENERIC (larg     : INTEGER :=15);
            PORT    (clk      : IN std_logic;
                     en       : IN std_logic;
                     chain    : IN std_logic;
             pile_in  : IN std_logic_vector(larg DOWNTO 0);
             pile_out : BUFFER std_logic_vector(larg DOWNTO 0));
END PILE_8_16;

ARCHITECTURE behav OF PILE_8_16 IS
            CONSTANT stack_depth: INTEGER := 7;
    TYPE pile_vect IS ARRAY (stack_depth DOWNTO 0) OF std_logic_vector(larg
DOWNTO 0);
            SIGNAL a: std_logic_vector(larg DOWNTO 0);
            SIGNAL gated_clk : std_logic;

BEGIN
            gated_clk <= clk AND en;
            mux: PROCESS (chain,pile_in,pile_out)
                        VARIABLE aa : std_logic_vector(larg DOWNTO 0);
            BEGIN
                        CASE chain IS
                            WHEN   '1' => aa := pile_in;
                        WHEN   '0' => aa := pile_out;
                            WHEN OTHERS => aa := (OTHERS => 'X');
                        END CASE;
                    a <= aa;
            END PROCESS mux;

    reg_ar: PROCESS (gated_clk)
                    VARIABLE temp_pile: pile_vect;
```

```
    BEGIN
            IF ((gated_clk = '1')  AND (gated_clk'EVENT)) THEN
                        pile_out <= temp_pile(0);
                        FOR i IN 0 TO ((stack_depth)-1) LOOP
                              temp_pile(i) := temp_pile(i+1);
                        END LOOP;
                        temp_pile(stack_depth) := a;
            END IF;
        END PROCESS reg_ar;
END behav;


--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;

ENTITY mux2a1 IS
      PORT(IN0      : in std_logic_vector(15 downto 0);
       IN1      : in std_logic_vector(15 downto 0);
       S        : in std_logic;
       OUT_MUX   : out std_logic_vector(15 downto 0));
END mux2a1;

ARCHITECTURE behv OF mux2a1 IS
BEGIN
  mux2a1_process: PROCESS(S,IN0,IN1)
  BEGIN
      CASE S IS
          WHEN '0' => OUT_MUX <= IN0;
          WHEN '1' => OUT_MUX <= IN1;
          WHEN OTHERS => NULL;
      END CASE;
  END PROCESS mux2a1_process;
END behv;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;

-- D flip-flop avec clear et enable
```

```
--      type = D
--      W = 12 bits
--      clock_edge = RISING
--      load_enable = MUX_GATE
--      clear = SYNC
--      set = NONE
--      test = NONE

ENTITY reg16 IS
        PORT(    d          : IN std_logic_vector (15 DOWNTO 0);
                 clk, en, clr : IN std_logic;
                 q          : OUT std_logic_vector (15 DOWNTO 0));
END reg16;

ARCHITECTURE behv OF reg16 IS
BEGIN
        PROCESS(clk)
        BEGIN
                IF ((clk='1') and (clk'event) ) THEN
                        IF clr= '1' THEN
                                q <= "0000000000000000";
                        ELSIF en = '1' THEN
                                q <= d;

                        END IF;
                END IF;
        END PROCESS;
END behv;


--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;



ENTITY  bloc_stock_donnees_33 IS
  PORT(clk : IN  std_logic;
                  en : IN  std_logic;
                  clr : IN  std_logic;
   pile_in : IN  std_logic_vector(15 DOWNTO 0);
  pile_out : OUT std_logic_vector(15 DOWNTO 0));
```

```
END bloc_stock_donnees_33;

ARCHITECTURE behv OF  bloc_stock_donnees_33 IS
        CONSTANT stack_depth: INTEGER := 31;
      TYPE pile_vect IS ARRAY ((stack_depth) DOWNTO 0) OF std_logic_vector(15
DOWNTO 0);
        SIGNAL gated_clk :std_logic;
BEGIN
        gated_clk <= clk AND en;
        shift_stack15_process: PROCESS (gated_clk, clr)
        VARIABLE temp_pile: pile_vect;
  BEGIN
                IF      ( clr = '1' ) THEN
                          pile_out <= "0000000000000000";
                          temp_pile := ( OTHERS => "0000000000000000");
                ELSIF   ((gated_clk = '1') AND (gated_clk'EVENT)) THEN
                                        pile_out <= temp_pile(0);
                                        FOR i IN 0 TO ((stack_depth)-1) LOOP
                                                temp_pile(i) := temp_pile(i+1);
                                        END LOOP;
                                        temp_pile(stack_depth) := pile_in;
                END IF;
        END PROCESS shift_stack15_process;
END behv;



--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;



ENTITY  bloc_stock_donnees_32 IS
  PORT(clk : IN  std_logic;
                    en : IN  std_logic;
                   clr : IN  std_logic;
   pile_in : IN  std_logic_vector(15 DOWNTO 0);
  pile_out : OUT std_logic_vector(15 DOWNTO 0));
END bloc_stock_donnees_32;

ARCHITECTURE behv OF  bloc_stock_donnees_32 IS
```

```
        CONSTANT stack_depth: INTEGER := 30;
      TYPE pile_vect IS ARRAY ((stack_depth) DOWNTO 0) OF std_logic_vector(15
DOWNTO 0);
        SIGNAL gated_clk :std_logic;
BEGIN
        gated_clk <= clk AND en;
        shift_stack15_process: PROCESS (gated_clk, clr)
        VARIABLE temp_pile: pile_vect;
  BEGIN
                IF      ( clr = '1' ) THEN
                        pile_out <= "0000000000000000";
                        temp_pile := ( OTHERS => "0000000000000000");
                ELSIF   ((gated_clk = '1')  AND (gated_clk'EVENT)) THEN
                                        pile_out <= temp_pile(0);
                                        FOR i IN 0 TO ((stack_depth)-1) LOOP
                                                temp_pile(i) := temp_pile(i+1);
                                        END LOOP;
                                        temp_pile(stack_depth) := pile_in;
                END IF;
        END PROCESS shift_stack15_process;
END behv;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;




ENTITY  bloc_stock_donnees_8 IS
  PORT(clk : IN  std_logic;
                  en : IN  std_logic;
                 clr : IN  std_logic;
   pile_in : IN  std_logic_vector(15 DOWNTO 0);
  pile_out : OUT std_logic_vector(15 DOWNTO 0));
END bloc_stock_donnees_8;

ARCHITECTURE behv OF  bloc_stock_donnees_8 IS
        CONSTANT stack_depth: INTEGER := 7;
      TYPE pile_vect IS ARRAY ((stack_depth) DOWNTO 0) OF std_logic_vector(15
DOWNTO 0);
```

```
        SIGNAL gated_clk :std_logic;
BEGIN
        gated_clk <= clk AND en;
        shift_stack15_process: PROCESS (gated_clk, clr)
        VARIABLE temp_pile: pile_vect;
  BEGIN
                IF      ( clr = '1' ) THEN
                        pile_out <= "0000000000000000";
                        temp_pile := ( OTHERS => "0000000000000000");
                ELSIF   ((gated_clk = '1')  AND (gated_clk'EVENT)) THEN
                                        pile_out <= temp_pile(0);
                                        FOR i IN 0 TO ((stack_depth)-1) LOOP
                                                temp_pile(i) := temp_pile(i+1);
                                        END LOOP;
                                        temp_pile(stack_depth) := pile_in;
                END IF;
        END PROCESS shift_stack15_process;
END behv;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;

-- version 0 14/02/95 16:45
-- decaleur a barillet de 16 bits
-- decale de 1 a 16 vers la droite ou vers la gauche en une etape
-- insere des 0 a gauche ou a droite

ENTITY  decaleur_gauch_droit IS
        PORT(distance : IN  std_logic_vector (5 DOWNTO 0);
                        input   : IN  std_logic_vector (15 DOWNTO 0);
                        output  : OUT std_logic_vector (15 DOWNTO 0));
END decaleur_gauch_droit ;

ARCHITECTURE behv OF decaleur_gauch_droit IS
BEGIN
        decaleur_gauch_droit_process:
        PROCESS (distance,input)
```

```
BEGIN
        IF ( input(15) = '0' ) THEN
            CASE distance IS

                -- decalage vers la droite MSB(distance) = '0' (division)


                WHEN "000000" => output <= input;
                WHEN "000001" => output <= '0' & input(15 DOWNTO 1);
                WHEN "000010" => output <= "00" & input(15 DOWNTO
2);
                WHEN "000011" => output <= "000" & input(15 DOWNTO
3);
                WHEN "000100" => output <= "0000" & input(15
DOWNTO 4);
                WHEN "000101" => output <= "00000" & input(15
DOWNTO 5);
                WHEN "000110" => output <= "000000" & input(15
DOWNTO 6);
                WHEN "000111" => output <= "0000000" & input(15
DOWNTO 7);
                WHEN "001000" => output <= "00000000" & input(15
DOWNTO 8);
                WHEN "001001" => output <= "000000000" & input(15
DOWNTO 9);
                WHEN "001010" => output <= "0000000000" & input(15
DOWNTO 10);
                WHEN "001011" => output <= "00000000000" & input(15
DOWNTO 11);
                WHEN "001100" => output <= "000000000000" & input(15
DOWNTO 12);
                WHEN "001101" => output <= "0000000000000" & input(15
DOWNTO 13);
                WHEN "001110" => output <= "00000000000000" &
input(15 DOWNTO 14);
                WHEN "001111" => output <= "000000000000000" &
input(15);
                WHEN "010000" => output <= "0000000000000000";


                -- decalage vers la gauche MSB(distance) = '1'
(multiplication)


                WHEN "100000" => output <= input;
                WHEN "100001" => output <= input(15) & input(15-2
DOWNTO 0) & '0';
```

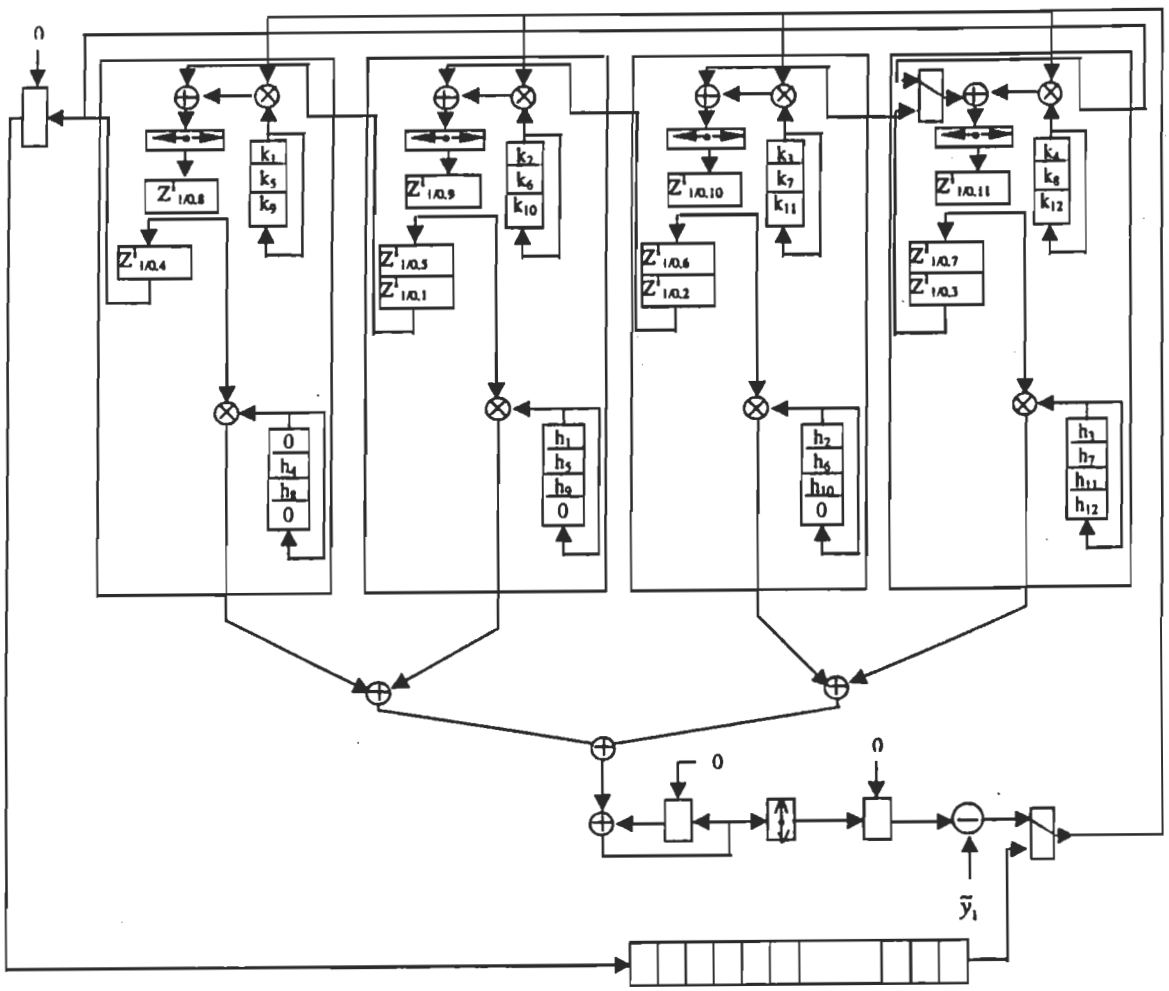
```
                        WHEN "100010" => output <= input(15) & input(15-3
DOWNTO 0) & "00";
                        WHEN "100011" => output <= input(15) & input(15-4
DOWNTO 0) & "000";
                        WHEN "100100" => output <= input(15) & input(15-5
DOWNTO 0) & "0000";
                        WHEN "100101" => output <= input(15) & input(15-6
DOWNTO 0) & "00000";
                        WHEN "100110" => output <= input(15) & input(15-7
DOWNTO 0) & "000000";
                        WHEN "100111" => output <= input(15) & input(15-8
DOWNTO 0) & "0000000";
                        WHEN "101000" => output <= input(15) & input(15-9
DOWNTO 0) & "00000000";
                        WHEN "101001" => output <= input(15) & input(15-10
DOWNTO 0) & "000000000";
                        WHEN "101010" => output <= input(15) & input(15-11
DOWNTO 0) & "0000000000";
                        WHEN "101011" => output <= input(15) & input(15-12
DOWNTO 0) & "00000000000";
                        WHEN "101100" => output <= input(15) & input(15-13
DOWNTO 0) & "000000000000";
                        WHEN "101101" => output <= input(15) & input(15-14
DOWNTO 0) & "0000000000000";
                        WHEN "101110" => output <= input(15) & input(15-15) &
"00000000000000";
                        WHEN "101111" => output <= input(15) &
"000000000000000";
                        WHEN "110000" => output <= "0000000000000000";

                        WHEN OTHERS => NULL;

                END CASE;

        ELSIF ( input(15) = '1' ) THEN

                CASE distance IS

                        -- decalage vers la droite MSB(distance) = '0'        (division)

                        WHEN "000000" => output <= input;
                        WHEN "000001" => output <= '1' & input(15 DOWNTO 1);
                        WHEN "000010" => output <= "11" & input(15 DOWNTO
2);
```

```
                    WHEN "000011" => output <= "100" & input(14 DOWNTO
2);
                    WHEN "000100" => output <= "1000" & input(14
DOWNTO 3);
                    WHEN "000101" => output <= "11111" & input(15
DOWNTO 5);
                    WHEN "000110" => output <= "111111" & input(15
DOWNTO 6);
                    WHEN "000111" => output <= "1111111" & input(15
DOWNTO 7);
                    WHEN "001000" => output <= "11111111" & input(15
DOWNTO 8);
                    WHEN "001001" => output <= "111111111" & input(15
DOWNTO 9);
                    WHEN "001010" => output <= "1111111111" & input(15
DOWNTO 10);
                    WHEN "001011" => output <= "11111111111" & input(15
DOWNTO 11);
                    WHEN "001100" => output <= "111111111111" & input(15
DOWNTO 12);
                    WHEN "001101" => output <= "1111111111111" & input(15
DOWNTO 13);
                    WHEN "001110" => output <= "11111111111111" &
input(15 DOWNTO 14);
                    WHEN "001111" => output <= "111111111111111" &
input(15);
                    WHEN "010000" => output <= "111111111111111";

                    -- decalage vers la gauche MSB(distance) = '1'
(multiplication)

                    WHEN "100000" => output <= input;
                    WHEN "100001" => output <= input(15) & input(15-2
DOWNTO 0) & '0';
                    WHEN "100010" => output <= input(15) & input(15-3
DOWNTO 0) & "00";
                    WHEN "100011" => output <= input(15) & input(15-4
DOWNTO 0) & "000";
                    WHEN "100100" => output <= input(15) & input(15-5
DOWNTO 0) & "0000";
                    WHEN "100101" => output <= input(15) & input(15-6
DOWNTO 0) & "00000";
                    WHEN "100110" => output <= input(15) & input(15-7
DOWNTO 0) & "000000";
```

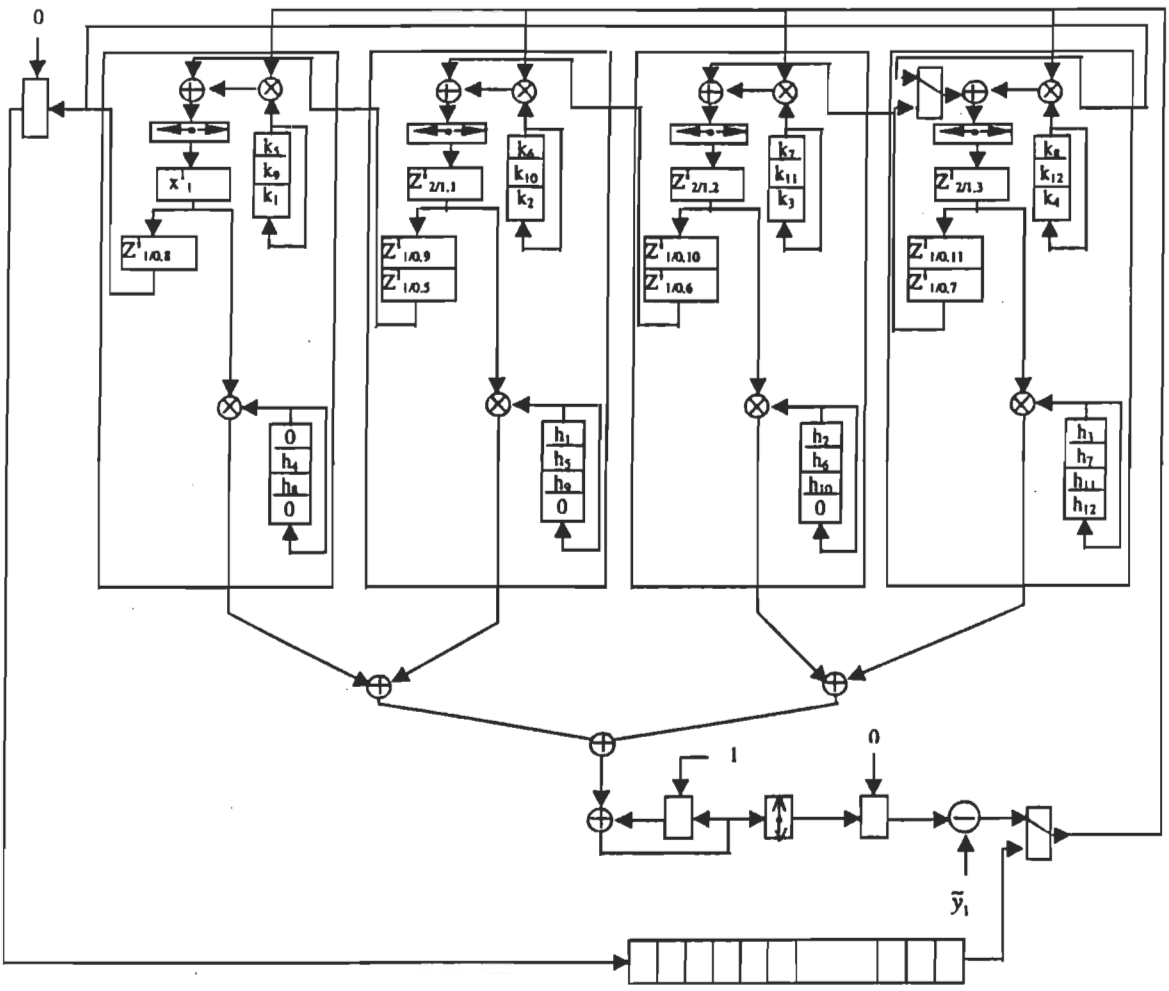
```
                    WHEN "100111" => output <= input(15) & input(15-8
DOWNTO 0) & "0000000";
                    WHEN "101000" => output <= input(15) & input(15-9
DOWNTO 0) & "00000000";
                    WHEN "101001" => output <= input(15) & input(15-10
DOWNTO 0) & "000000000";
                    WHEN "101010" => output <= input(15) & input(15-11
DOWNTO 0) & "0000000000";
                    WHEN "101011" => output <= input(15) & input(15-12
DOWNTO 0) & "00000000000";
                    WHEN "101100" => output <= input(15) & input(15-13
DOWNTO 0) & "000000000000";
                    WHEN "101101" => output <= input(15) & input(15-14
DOWNTO 0) & "0000000000000";
                    WHEN "101110" => output <= input(15) & input(15-15) &
"00000000000000";
                    WHEN "101111" => output <= input(15) &
"000000000000000";
                    WHEN "110000" => output <= "0000000000000000";

                    WHEN OTHERS => NULL;

                END CASE;
            END IF;
        END PROCESS decaleur_gauch_droit_process;
END behv;


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

ANNEXE C

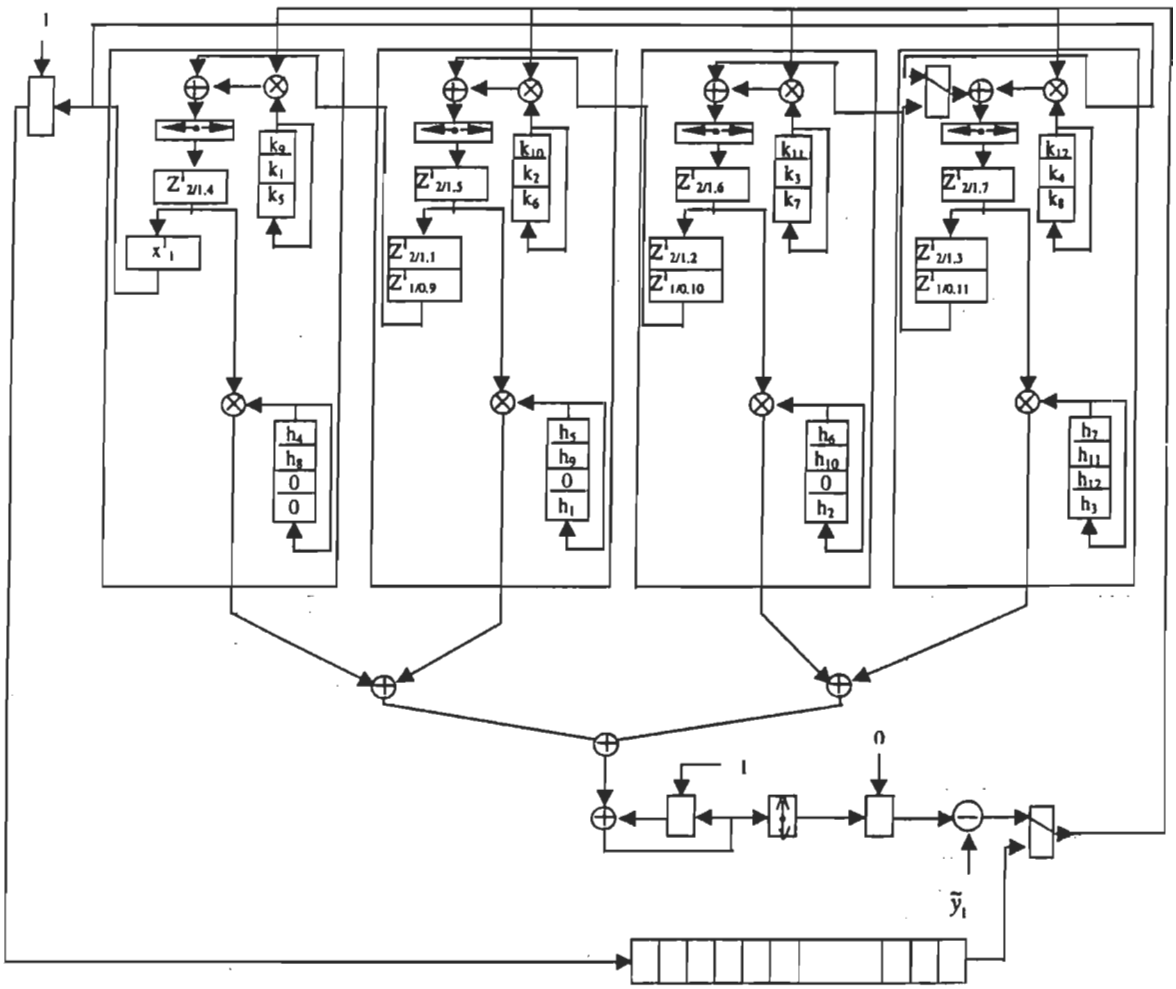
EXEMPLE DE DÉPLACEMENT DES FLOTS DE DONNÉES DANS LES ARCHITECTURES SYSKALI



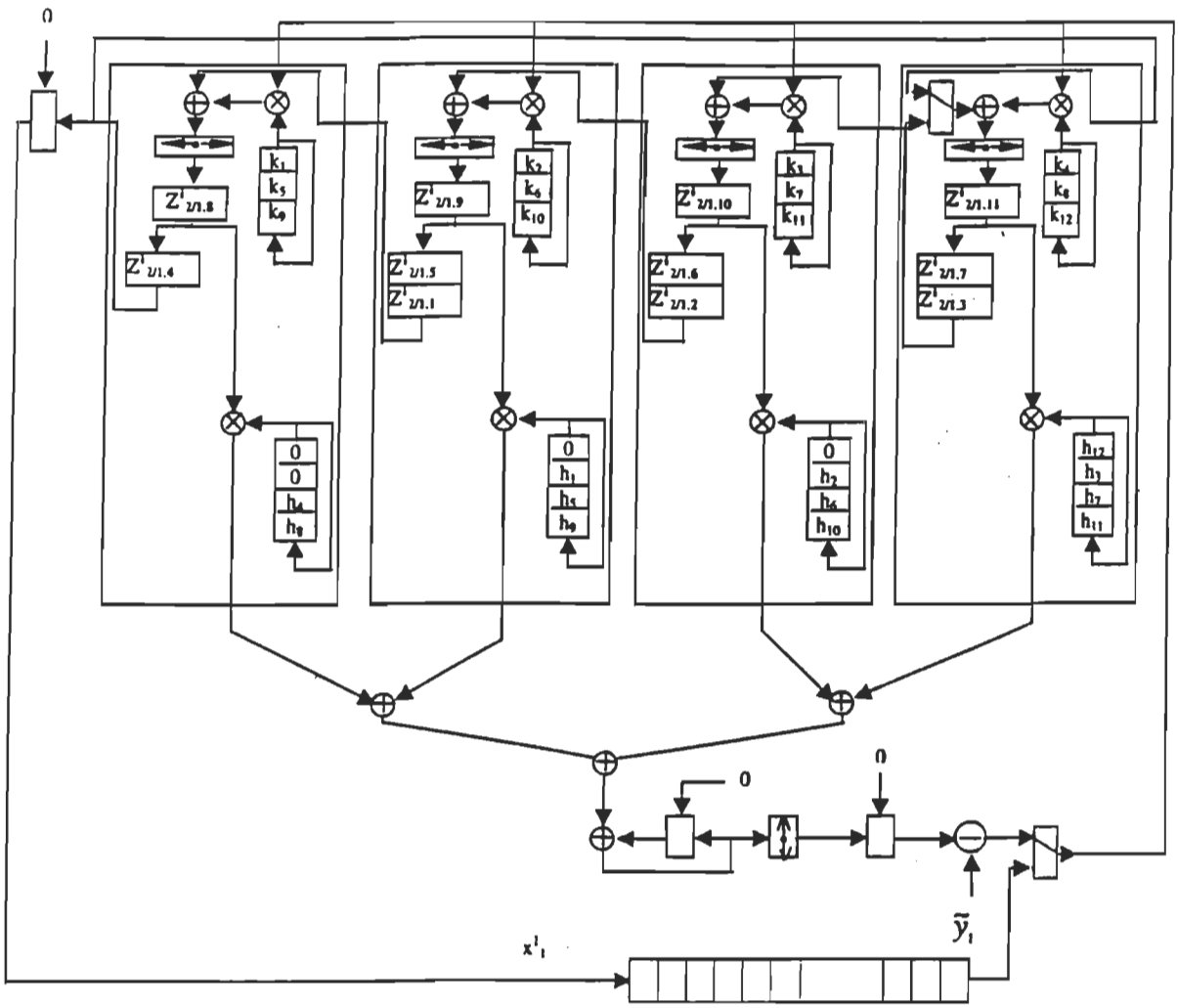
Exemple de fonctionnement de l'architecture $SYSKALI_1$ à t=0 et n=1



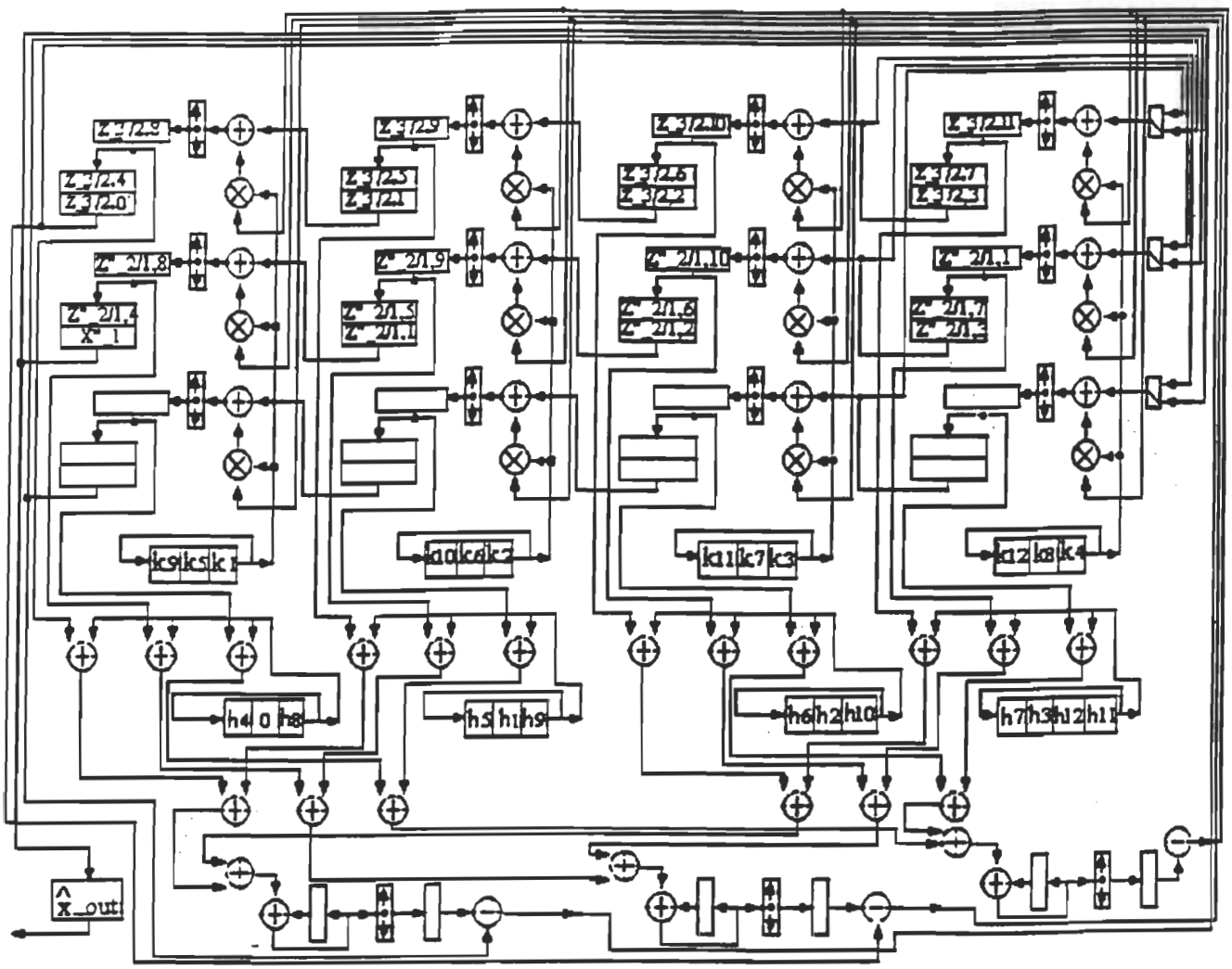
Exemple de fonctionnement de l'architecture $SYSKALI_1$ à t=1 et n=1



Exemple de fonctionnement de l'architecture $SYSKALI_1$ à t=2 et n=1

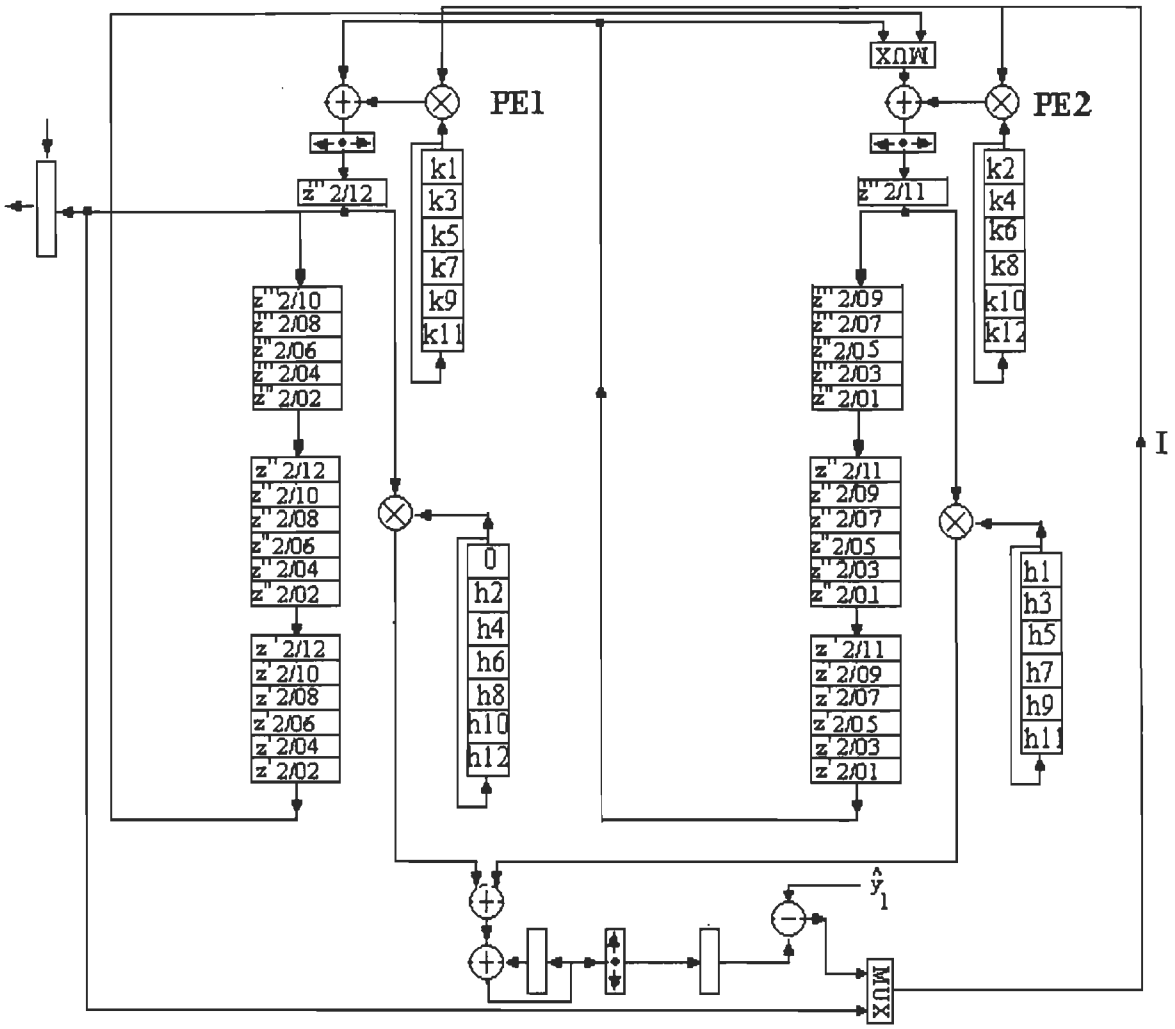


Exemple de fonctionnement de l'architecture $SYSKALI_1$ à t=3 et n=1



Exemple de fonctionnement de l'architecture $SYSKALI_2$ à t=8 et n=2

NB : Les données z', z'' correspondent respectivement aux valeurs calculées à la première, et deuxième itération. Notons également que les séquences avant et après sont basées sur la même logique que celle de l'architecture $SYSKALI_1$.



Exemple de fonctionnement de l'architecture $SYSKALI_3$ à t=4 et n=1

NB : Les données z', z'' et z''' correspondent respectivement aux valeurs calculées à la première, deuxième et troisième itération. Notons également que les séquences avant et après sont basées sur la même logique que celle de l'architecture $SYSKALI_1$.

ANNEXE D

GUIDE D'UTILISATION DES ARCHITECTURES SYSKALI

Au lancement de Mentor Graphics par la commande *dmg*, l'utilisateur voit apparaître la fenêtre principale du logiciel. C'est en effet à partir de cette fenêtre que seront lancés les différents outils utilisés.

Les outils de Mentor Graphics qui sont disponibles, sont visible dans la fenêtre "TOOLS". Si cette fenêtre n'est pas ouverte, on peut l'ouvrir de plusieurs manières : à partir de la barre d'outils placée sur la droite de l'écran en cliquant sur l'icône "TOOLS", ou à partir du bouton droite de la souris en choisissant "Open Tools Window".

Il est également utile d'avoir à l'écran le répertoire où l'on se trouve et celui de la liste des fichiers que contient ce répertoire. Ceci est disponible par le "navigator", En cliquant sur l'icône Tools.

En effet, le "navigator" permet de visualiser les fichiers et les répertoires sous forme de texte ou sous forme d'icône, et de se déplacer parmi ces derniers. Les répertoires et fichiers ont des icônes différents suivant leur type : fichier VHDL, symbole ou schématique. L'ouverture des outils se fait, entre autres, par un double clic sur l'icône de l'outil situé dans la boîte à outils. Il faut dire que toute la modélisation du processeur, que ce soit les programmes VHDL, la création des symboles, ou le dessin des schémas, est

réalisée grâce à l'outil Design Architect. Ici également, tous les éléments à ouvrir, peuvent l'être soit par le biais de la souris, ou de la barre d'outils. Par la suite, la méthode employée sera celle de la barre d'outils.

Il faut tout d'abord indiquer son répertoire de travail à partir du menu; soit:

MGC>Location Map>Set Working Directory .

Notons également que pour créer un ficher VHDL ou ouvrir un programme existant, il faut commencer par cliquer sur l'icône OPEN VHDL. Une fenêtre de dialogue apparaît alors, dans laquelle il faut indiquer le nom déjà existant. S'il s'agit d'un nouveau fichier, il faut charger le répertoire où il devra se situer.

Modélisation des blocs élémentaires des architectures SYSKALI

La modélisation des architectures SYSKALI a été réalisée au moyen du langage VHDL, à savoir tous les éléments de base le constituant.

Pour chaque élément, il fallait créer une entité, c'est à dire une boîte dont on ne connaît que les entrées et les sorties, puis une architecture qui définie le fonctionnement des entités en fonction de ses entrées et de ses sorties. Nous présenterons en annexe C, les codes VHDL de tous les blocs principaux présentés ci-dessous.

L'additionneur "Adder"

Le module additionneur a été réalisé sous le nom "Adder". Il a comme entrées input1 et input2, et comme sortie output. Ces trois ports sont des bus de 16 bits. Il faut noter que cet additionneur n'a ni retenue en entrée, ni retenue en sortie. L'addition est réalisée en VHDL par l'opérateur "+".

Le soustracteur "soustrct"

Le module soustracteur a été réalisé sous le nom "soustrct". Il réalise la soustraction des bus d'entrée sur 16 bits *input1* et *input2* et fournit le résultat sur un bus de sortie 16 bits nommé *output*. Tout comme l'additionneur, le soustracteur n'utilise ni retenue entrante ni retenue sortante. La soustraction est réalisée en VHDL par l'opérateur " – ".

Le multiplicateur "mult"

Le module multiplicateur a été réalisé sous le nom "mult", et réalise la multiplication de deux opérandes signés. Le premier, *mc0*, est sur un bus de 8 bits, le second, *mp0*, est sur 16 bits. Le résultat est disponible en sortie sur le bus de sortie *dout* 16 bits.

La pile "Shift stack"

Ce module est une pile de type fifo. Les entrées de cette pile sont des signaux de contrôle sur 1 bit :

clk le signal d'horloge actif sur front montant,

en enable, la validation de la pile,

clr la remise à 0 des registres de la pile.

La pile a aussi comme entrée le bus pile_in de 16 bits correspondant à la largeur des mots dans la pile, et en sortie le bus pile_out de 16 bits également. Les mots placés dans les registres de la pile se décalent vers la sortie à chaque front montant de l'horloge lorsque le signal enable est à " 1 ".

La pile "Shift rotate stack"

Le module *Shift rotate stack* est une pile ayant deux modes de fonctionnement distincts. Un mode correspond à une pile de type fifo, l'autre mode correspond à une pile de type circulaire (la sortie est bouclée sur l'entrée). Cette pile possède trois signaux de contrôle sur 1 bit, en entrée:

clk le signal d'horloge actif sur front montant,

en le signal de validation de la pile,

chain le signal de sélection du type de pile.

Suivant le signal *chain,* la pile se comporte de manière différente : si *chain=0* alors la pile est de type circulaire, si *chain=1* alors la pile est de type fifo. Le fonctionnement des décalages est le même que pour la pile *shift_stack.*

Le décaleur à barillet "Barrel shifter"

Le module *barrel-shifter* est un décaleur à barillet qui reçoit en entrée deux bus ;

l'un de 16 bits et l'autre de 4 bits. Le bus de 16 bits se nomme *input* et transporte la donnée à décaler. Le bus de 4 bits se nomme *distance* et transporte la valeur du décalage. Enfin la sortie *output* sur 16 bits transporte le résultat du décalage.

Le décalage de 1, 2, 3, 4 bits, suivant la valeur de *distance,* est uniquement vers la droite, et le décaleur insère des " 1 "à gauche car il n'agit que sur des nombres négatifs (complément

à 2).

Ainsi le décaleur n'est actif que si le bit de poids fort du *bus input est* à " 1 ". Dans le cas inverse la donnée en sortie est celle présentée en entrée. De plus, une cinquième valeur de *distance* met la sortie à zéro.

Le décaleur à barillet "Barrel shifter left right"

Le module *barrel_shifter* est également un décaleur à barillet mais qui, suivant l'une des données en entrée, la décale soit vers la droite, soit vers la gauche. Ce décaleur dispose aussi de deux bus en entrée : le bus de données *input* sur 16 bits, et le bus indiquant le nombre de décalage à effectuer, *distance,* sur 6 bits. La sortie du décaleur est le bus *output* de 16 bits. C'est la donnée *distance* qui indique dans quelle direction le décalage doit être effectué. Ainsi, si le bit de poids fort est un "0" (nombre positif) le décalage est réalisé vers la droite, ce qui correspond à une division de la donnée en entrée. Si le bit de poids fort est un "1" (nombre négatif) le décalage est réalisé vers la gauche, ce qui correspond à une multiplication de la donnée entrées. Ensuite, suivant la polarité de la donnée à décaler et le sens de décalage, il faut rajouter soit des " 0 ", soit des " 1 ".

Le compteur "Address- counter"

Le module *address-counter* est un compteur ayant trois signaux de contrôle sur 1bit :

clk le signal d'horloge actif sur front montant,

en le signal de validation du compteur,

remet le signal de remise à zéro.

Ce module a comme sortie un bus de 9 bits nommé *addess* qui se trouve incrémenté de 1 à chaque front montant de l'horloge quand le signal enable est à " 1 ". Comme son nom

l'indique, ce compteur fournit une adresse correspondant à une position en mémoire de certaines données spécifiques, à charger dans le processeur.

Registres

Plusieurs registres ont été définis. Ils possèdent tous, trois signaux de contrôle, un bus d'entrée et un bus de sortie. C'est seulement sur la taille du bus que les registres diffèrent. Ainsi, les registres existant sont : le registre 16 bits, 6 bits, 5 bits, 4 bits et 2 bits. Les signaux de contrôle communs à tous les registres sont :

clk le signal d'horloge actif sur front montant,

en le signal de validation,

clr la remise à zéro.

La donnée en sortie est affectée par la donnée en entrée sur un front montant de l'horloge, quand le signal enable est à " 1 ".

Le multiplexeur "mux2a1"

Le module *mux2al* est un multiplexeur deux entrées / une sortie. Il a donc en entrée deux bus de 16 bits *INO et IN1* et un signal de sélection S, et en sortie un bus de 16 bits *out - mux.* Lorsque le signal de sélection est à " 0 ", la sortie reçoit l'entrée *INO,* et inversement, quand S vaut " 1 " la sortie reçoit l'entrée *IN1.*

Le démultiplexeur''de*mux1a2*''

Le module ''de*mux1a2*'' est une démultiplexeur une entrée / deux sorties. il a donc en entrée un bus 16 bits *in-mux* et un signal de sélection S et, en sortie deux bus de 16 bits *OUT0 et OUT1*. Lorsque le signal de sélection est à "0', la sortie *OUT0* reçoit l'entrée, et inversement, quand S vaut '' 1 ''la sortie *OUT1* reçoit l'entrée. Il faut noter également que toutes les architectures des processeurs SYSKALI proposées contiennent des mémoires séparées pour les vecteurs du gain K, h, Z et d'autres paramètres nécessaires à la reconstitution (dimension des vecteurs, initialisation des compteurs, paramètres de reconstitution avec contraintes, etc).