UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN ÉLECTRONIQUE INDUSTRIELLE

PAR

MARC-ALAIN SANTERRE

L'ÉTUDE DE L'APPLICATION DU FILTRAGE DE KALMAN POUR LA RECONSTITUTION DE SIGNAUX DE MESURE; L'IMPLANTATION EN TECHNOLOGIE VLSI

MAI 1996

RÉSUMÉ

Le mémoire a pour objectif spécifique l'implantation du filtre de Kalman en technologie d'intégration à très grande échelle pour la reconstruction des signaux de mesure. La reconstitution d'un mesurande consiste à estimer un signal x(t) qui n'est pas mesurable directement, à partir des résultats de mesure d'un autre signal y(t) qui est lié avec le premier de façon causale. Les méthodes de reconstitution sont généralement basées sur certaines suppositions concernant le modèle mathématique de la relation entre les signaux et l'information *a priori* accessible sur le signal reconstruit et sur le bruit qui entache des résultats de conversion y(t). Le problème de reconstitution, posé d'une façon abstraite, est bien conforme aux nombreuses situations pratiques dans les différents domaines de la technique de mesure et de commande. Cela implique une demande particulière pour des processeurs spécialisés, dédiés à la vaste classe d'applications de reconstitution.

La contribution originale à cette étude est la réalisation d'un premier prototype de processeur spécialisé à la reconstitution de signaux utilisant le filtrage de Kalman. Cet algorithme offre l'avantage d'opérer en temps réel par un traitement en ligne en plus de permettre l'ajout facile d'une contrainte de positivité pour améliorer les résultats. Par contre, il requiert un temps de calcul suffisamment important pour demander un temps de réponse non négligeable de la part d'un processeur de traitement de signal commercial. Ainsi, les calculateurs conventionnels munis de microprocesseurs sont insuffisants pour atteindre une vitesse de

calcul satisfaisant les applications où la contrainte majeure est la reconstitution en temps réel.

Une étude de l'algorithme, pour les signaux stationnaires et invariants, révèle la possibilité d'effectuer les calculs sur deux processus indépendants et entrelacés pour augmenter les performances en temps réel. L'architecture des processeurs disponibles commercialement pour le traitement des signaux n'est pas adaptée à ce traitement parallèle. Une première architecture spécialisée, DSP_KAL, qui tire profit du parallélisme de l'algorithme est développée. Une deuxième version réduite de l'architecture, DSP_KAL+, est réalisée pour permettre son intégration sur silicium.

L'amélioration des performances du processeur par rapport au DSP56000 de Motorola a été évaluée en comparant le nombre de cycles machine requis par chacun des processeurs pour réaliser certaines fonctions. Pour effectuer une reconstitution, l'architecture proposée permet une réduction du nombre de cycles d'horloge qui tend vers 5 fois moins de cycles que le DSP56000 de Motorola pour une augmentation de fréquence de 2 fois supérieure.

Un prototype a été réalisé à la Société Cannadienne de Microélectronique (SCM). Ce prototype a été développé pour fonctionner comme coprocesseur d'un processeur maître. Il est composé de deux multiplieurs/accumulateurs (M/A) et d'un microséquenceur programmable controlant les unités opératives servant à exécuter l'algorithme stationnaire de Kalman sans et avec contrainte de positivité. La conception a été réalisée au moyen d'outils rendu disponibles par la SCM: CADENCE, SILOS, BNR DFT, SYNFUL et un langage de description matériel de haut niveau VERILOG. Pour une réduction des coûts de fabrication,

une architecture à chemin de données de 8-bits a été fabriquée dans une technologie CMOS de 1.2 um de la SCM. La complexité de la puce sans les mémoires est de 24 000 transistors pour une surface de silicium de 5.1 x 9.0 mm^2.

Cette version du processeur est un premier pas vers un processeur spécialisé pour la reconstitution des signaux basée sur le filtre de Kalman. D'autres améliorations sont envisagées afin de séparer les calculs vectoriels en sections et de les distribuer sur des processeurs séparés.

REMERCIEMENTS

Je me dois d'adresser des remerciements tous particuliers au Dr. Daniel Massicotte. Ses encouragements répétés de même que ses conseils et son amitié soutenue ont permis la rédaction de ce mémoire.

Je tiens également à remercier le Pr. Yvon Savaria mon codirecteur de maîtrise pour les judicieux conseils qu'il m'a donnés tout au long de ce travail. Le support financier de même que son assistance lors de la correction de la rédaction ont été grandement appréciés.

Je ne saurais passer sous silence la contribution du Pr. Andrzej Barwicz mon directeur de maîtrise. La confiance dont il a fait preuve à mon égard durant cette longue période, le support organisationnel et financier, ainsi que le soutien du laboratoire de systèmes de mesure méritent d'être mentionnés.

Je remercie également la Société canadienne de Micro-électronique (SCM) qui a permis la fabrication du circuit intégré qui est le résultat de ce travail de recherche.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SYMBOLES ET ABRÉVIATIONS

h	Réponse impulsionnelle;
I	Innovation;
k	Gain de Kalman;
η	Bruit de mesure;
t	Temps;
x	Entrée du système de mesure;
$\hat{x}$	Résultat de reconstitution;
y	Signal de sortie;
$\tilde{y}$	Signal de sortie bruité;
δ	Impulsion de Dirac;
ASIC	Application-specific integrated circuit;
ATPG	Automatic Test Pattern Generator;
BNR	Bell Northern Research;
CIF	Caltech Intermediate Form;
CLA	Carry Look-Ahead;
CLARA	Processeur spécialisé pour la reconstitution de mesurandes basée sur le filtrage de Kalman rapide;
DFT	Design For Testability;
DRC	Design Rules Check;

DSP	Digital Signal Processing;
DSP_KAL	Architecture originale pour la reconstitution de mesurandes;
DSP_KAL+	Architecture améliorée pour la reconstitution de mesurandes;
I/O	Input/Output;
M/A	Multiplicateur/Accumulateur;
MUX	Multiplexeur;
PC	Program Counter;
PGA	Pin Grid Array;
RAM	Random Access Memory;
SCM	Société Canadienne de Microélectronique;
SM	Scan Mode;
TM	Test Mode;
VLSI	Very Large Scale Integration;

CHAPITRE 1

INTRODUCTION

1.1 Objectifs

Le traitement des signaux dans un système de mesure consiste en deux phases: la conversion et la reconstitution. La figure 1-1 montre un système de mesure comportant des modules dédiés à ces phases.

La conversion dans un système de mesure numérique consiste en une série de transformations de signaux analogiques, reçus d'un objet de mesure, en des signaux numériques. Ces derniers sont ensuite traités afin d'obtenir des estimés des grandeurs à mesurer; cette opération est nommée reconstitution de mesurande.

La reconstitution de mesurande est un problème fondamental en métrologie. Suivant la figure 1-1, le signal de sortie du bloc de conversion d'un instrument de mesure est obtenu par la convolution du signal mesuré à son entrée, x(t), avec la réponse impulsionnelle, h(t), de l'appareil de mesure. De plus, un bruit de mesure, $\eta(t)$, qui entache le système, s'additionne à y(t) pour obtenir le signal de sortie bruité $\tilde{y}(t)$.

L'équation suivante caractérise la conversion des signaux dans un système de mesure:

$$\tilde{y}(t) = \int h(t-t')\,x(t')\,dt' + \eta(t) \qquad (1.1)$$

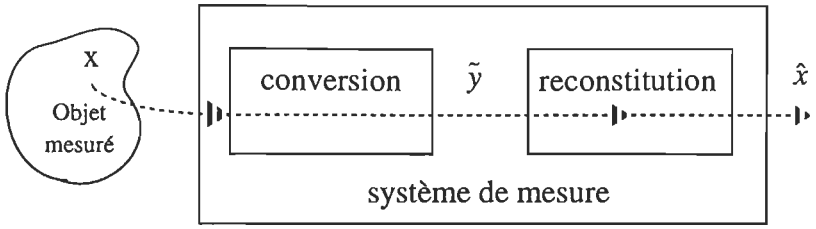

Figure 1-1 Schéma bloc d'un système de mesure.

La résolution de l'équation (1.1) devient un problème de reconstitution dans le cas où l'on cherche à reconstruire le signal d'entrée. Plusieurs algorithmes ont été développés pour permettre la reconstitution des signaux. Citons à titre d'exemple les algorithmes de: Van-Cittert, Jansson, Kalman, etc. Il n'existe pas de méthode universelle pour effectuer la reconstitution d'un signal. On choisit l'algorithme le plus approprié en fonction des connaissances *a priori* du signal. Cependant, l'algorithme de Kalman permet la reconstitution en temps réel aussi bien avec des signaux à variation rapide qu'avec des signaux à variation lente [MAS94]. Nous avons, en partie, cherché à améliorer le temps de reconstitution pour accroître la fréquence d'échantillonnage afin de mieux reconstituer les signaux à variation rapide.

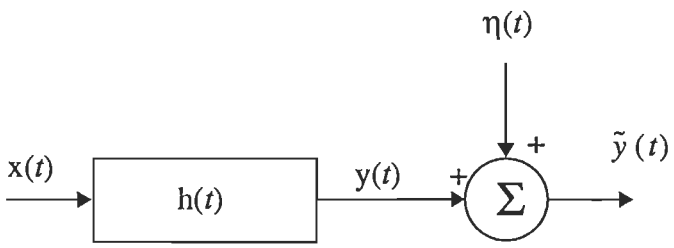

Figure 1-2 Bloc de conversion linéaire monodimensionnel avec bruit.

L'algorithme de Kalman demande une importante capacité de calcul de la part du processeur sur lequel il est mis en oeuvre. L'architecture des processeurs commerciaux pour le traitement des signaux n'est pas optimisée pour un algorithme en particulier. Par exemple, le DSP56000 requiert 2 cycles d'horloge pour une multiplication/addition et le résultat d'une addition doit être placé temporairement dans un registre avant d'être écrit en mémoire. L'exécution est répartie sur plusieurs cycles, ce qui entraîne une diminution des performances dans les applications en temps réel. Le seul processeur spécialisé ciblé vers le filtrage de Kalman connu au début des travaux en 1991 était CLARA [REY86]. Son architecture interne est semblable aux processeurs commerciaux, il n'utilise qu'un seul M/A.

Une analyse de l'algorithme [MAS92b] a révèlé la possibilité d'effectuer les calculs sur deux processus indépendants mais entrelacée. Ce mémoire porte sur des travaux visant la mise en oeuvre d'une première architecture spécialisée, adaptée à l'algorithme, conçue pour tirer profit de cette distribution des calculs afin d'améliorer le temps de reconstitution. L'architecture utilise 2 M/A externes de 16x16 bits sur lesquels sont répartis les calculs (DSP_KAL).

Ce mémoire traite aussi d'une deuxième version de l'architecture qui intègre les M/A pour minimiser le nombre de broches externes. Cela améliore aussi les performances en réduisant le temps requis pour une propagation des données du M/A. Une version compacte conçue pour supporter des opérandes de 8 bits a été réalisée sous la forme d'un prototype intégré sur silicium (DSP_KAL+).

1.2 Méthodologie

La méthodologie utilisée pour la conception et la validation de l'architecture intégrée est illustrée à la figure 1-3. Elle se décompose en six étapes distinctes visant: l'élaboration de l'algorithme, l'élaboration de l'architecture originale (DSP_KAL), la conception de l'architecture intégrée (DSP_KAL+), l'inclusion de structure permettant d'assurer la testabilité de l'ensemble, le placement et routage du circuit et enfin la simulation fonctionnelle. On retrouve dans cette figure l'ensemble des outils utilisés, à savoir: Verilog, Synful, la simulation fonctionnelle, Scancheck, ATPG, le placement et routage, le DRC et le PDcompare. De plus, la figure 1-3 décrit clairement l'ordre dans lequel ces outils permettent d'arriver à un circuit complet.

La première étape est discutée au chapitre 2. Ce chapitre inclut une étude de l'algorithme de Kalman utilisé pour la reconstitution. Les deuxième, troisième, quatrième et cinquième étapes sont décrites au chapitre 3. Elles démontrent en détail la première architecture DSP_KAL, les problèmes qui ont conduit au développement de l'architecture DSP_KAL+, l'analyse de la testabilité de l'architecture DSP_KAL+, effectuée avec l'aide d'une chaîne de balayage interne, et les principales étapes de l'intégration de l'architecture DSP_KAL+ sur silicium. Les simulations fonctionnelles réalisées pour la validation de l'architecture intégrée sont décrites au chapitre 4. Les chapitres 3 et 4 sont une contribution originale provenant de mon travail de recherche.

CHAPITRE 2

PROBLÈME DE LA RECONSTITUTION DE SIGNAUX

2.1 Description du problème

Dans le domaine des systèmes de mesure, nous considérons généralement qu'un instrument est précis lorsque sa réponse impulsionnelle peut être approchée par une impulsion de Dirac:

$$h(t) = a \cdot \delta(t) \tag{2.1}$$

où $\delta(t) = \begin{cases} 1 & si\ t = 0 \\ 0 & ailleurs \end{cases}$

On obtient alors la grandeur à mesurer par une relation de proportionnalité :

$$x(t) = \frac{y(t)}{a} \tag{2.2}$$

Cette approximation n'est acceptable que si le contenu fréquentiel du signal x(t) est limité à l'intérieur de la bande passante de l'instrument. Si nous utilisons l'appareil au-delà de ces limites, la réponse impulsionnelle ne peut plus être considérée comme une fonction de Dirac. Le signal à mesurer est alors convolué avec la réponse impulsionnelle réelle du système de mesure. Ce problème se pose à chaque fois que nous voulons mesurer une grandeur x(t) non directement accessible, par l'intermédiaire d'un instrument qui fournit une valeur expérimentale [DEM86].

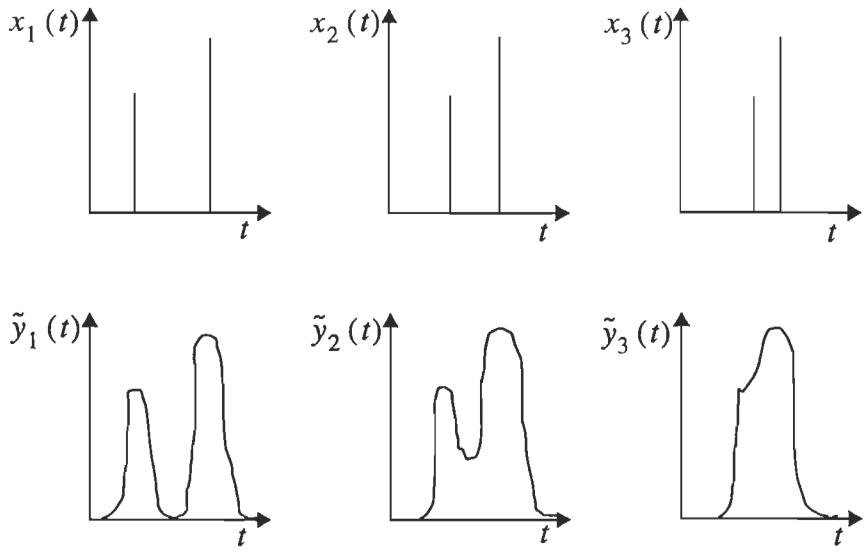

Figure 2-1 Exemple de signaux de sortie d'un système de mesure.

Il est important de comprendre l'effet de la convolution de la réponse impulsionnelle h(t) sur le signal x(t). La figure 2-1 illustre trois cas possibles de signaux spectrométriques où la distorsion qui est induite cause des erreurs de mesure[JAN84]. Nous remarquons que l'information contenue dans les raies de $x_3(t)$ qui est connue a priori est perdue lors de la convolution avec h(t) pour obtenir $\tilde{y}_3(t)$. Cet exemple illustre un cas ou l'usager aurait dépassé les limites de résolution de son appareil de mesure.

Dans un système linéaire, l'entrée x(t) d'un système et sa sortie y(t) sont reliées par une intégrale de première espèce [MOR93] :

$$y(t) = \int_{-\infty}^{\infty} h(t,t')\,x(t')\,dt' \tag{2.3}$$

Si le système est invariant, la relation devient une convolution :

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} h(t-t')\,x(t')\,dt' \tag{2.4}$$

Cette équation ne représente pas toujours la réalité. Il nous faut en effet tenir compte de la présence inévitable du bruit de mesure $\eta(t)$ qui, même à de très faibles valeurs, rend parfois difficile la résolution du problème. Nous obtenons alors dans le cas d'un système causal.

$$\tilde{y}(t) = \int h(t-t')\,x(t')\,d(t') + \eta(t) \tag{2.5}$$

La solution consiste à résoudre l'équation intégrale de la convolution pour reconstruire ou restaurer l'entrée du système $x(t)$ à partir de sa sortie $\tilde{y}(t)$. En général, on doit résoudre cette intégrale numériquement. Nous pouvons donc numériser l'équation (2.5) comme suit:.

$$\tilde{y}_k = \sum_{i=0}^{k} h_{k-i} x_i + \eta \tag{2.6}$$

Le terme η de l'équation (2.6) comporte au moins trois sources potentielles d'erreurs : des erreurs provenant de bruits dans la chaîne de conversion des signaux analogiques $\tilde{y}(t)$, des erreurs de quantification lors de l'échantillonnage des signaux $\tilde{y}(t)$ pour obtenir des signaux numériques $\tilde{y}_k$, des erreurs de repliement dûes à un sous-échantillonnage dans la quadrature du signal à restaurer.

2.2 Algorithme de Kalman pour signaux stationnaires

L'algorithme de reconstitution de Kalman d'un signal $\tilde{y}(t)$ comportant N échantillons est basé sur le modèle d'état discret du système de mesure représenté par les équations sui-

vantes [MAS92b].

$$z_{k+1} = Fz_k + bu_k \tag{2.7}$$

$$\tilde{y}_k = h^T z_k + \eta_k \tag{2.8}$$

où $z_0^k = 0$

pour k = 1, 2, 3, 4, . . , N. La matrice **F** et le vecteur **b** utilisés dans les équations (2.7) et (2.8) sont invariants et de forme canonique contrôlable :

$$F = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}, dim(F) = M \times M$$

$$b = [0|0|0|\dots|0|1]^T, dim(b) = M$$

$$h = [h_1|h_2|h_3|h_4|\dots|h_{M-1}|h_M]^T, dim(h) = M$$

À partir d'observations $\tilde{y}_1, \tilde{y}_2, \dots \tilde{y}_k, \dots, \tilde{y}_{N-1}, \tilde{y}_N$, nous désirons effectuer une estimation optimale (au sens de l'erreur quadratique moyenne minimale) $\hat{z}_{k/i}$ de l'entrée z_k. Nous noterons une telle estimation par $\hat{z}_{k/n}$, c'est-à-dire l'estimation à l'instant k compte tenu des informations disponibles à l'instant n.

Dans le cas d'un système linéaire et invariant dans le temps, le vecteur de gain peut être calculé indépendamment des mesures à effectuer [MAS91]. Le filtrage de Kalman se

réduit aux équations suivantes:

$$\hat{z}_{k+1/k} = \boldsymbol{F} \cdot \hat{z}_{k/k} \tag{2.9}$$

$$\hat{y}_{k+1} = \boldsymbol{h}^T \cdot \hat{z}_{k+1/k} \tag{2.10}$$

$$I_{k+1} = \tilde{y}_{k+1} - \hat{y}_{k+1} \tag{2.11}$$

$$\hat{z}_{k+1/k+1} = \hat{z}_{k+1/k} + \boldsymbol{k}_{\infty} \cdot I_{k+1} \tag{2.12}$$

Le résultat de reconstitution $\hat{x}_k$ est obtenu par l'extraction d'un élément du vecteur $\hat{z}_{k/k}$ à chaque point d'échantillonnage k [MAS95].

$$\hat{x}_k = \hat{z}_{k/k, \frac{M}{2}} \tag{2.13}$$

2.3 Séquencement de l'algorithme pour le calcul parallèle

Une analyse de l'algorithme révèle qu'il est possible de distribuer l'exécution des calculs sur deux processus indépendants, mais entrelacés [MAS92b]. Cela est possible en calculant l'estimation du signal de sortie du système de mesure $\hat{y}_{k+1}$ au même moment que l'estimation du vecteur d'état $\hat{z}_{k/k}$. Cette possibilité découle du fait que lors de la première mesure, le vecteur $\hat{z}_{0/0}$ est égal à zéro. Ceci implique que la première innovation, I_1, est égale à la première mesure bruité.

La figure 2-1 montre la séquence des calculs en parallèle. La colonne de droite présente les calculs séquentiels pour obtenir $\hat{z}_{k/k}$ et la colonne de gauche montre les calculs séquentiels pour obtenir $\hat{y}_{k+1}$.

L'architecture de la plupart des processeurs commerciaux pour le traitement des signaux (DSP5600, TMS320, ...) ne comprend qu'un seul M/A. Il est donc impossible de répartir le calcul sur deux processus indépendants. Une solution possible est de développer une architecture spécialisée capable de mettre à profit le parallélisme au niveau du traitement.

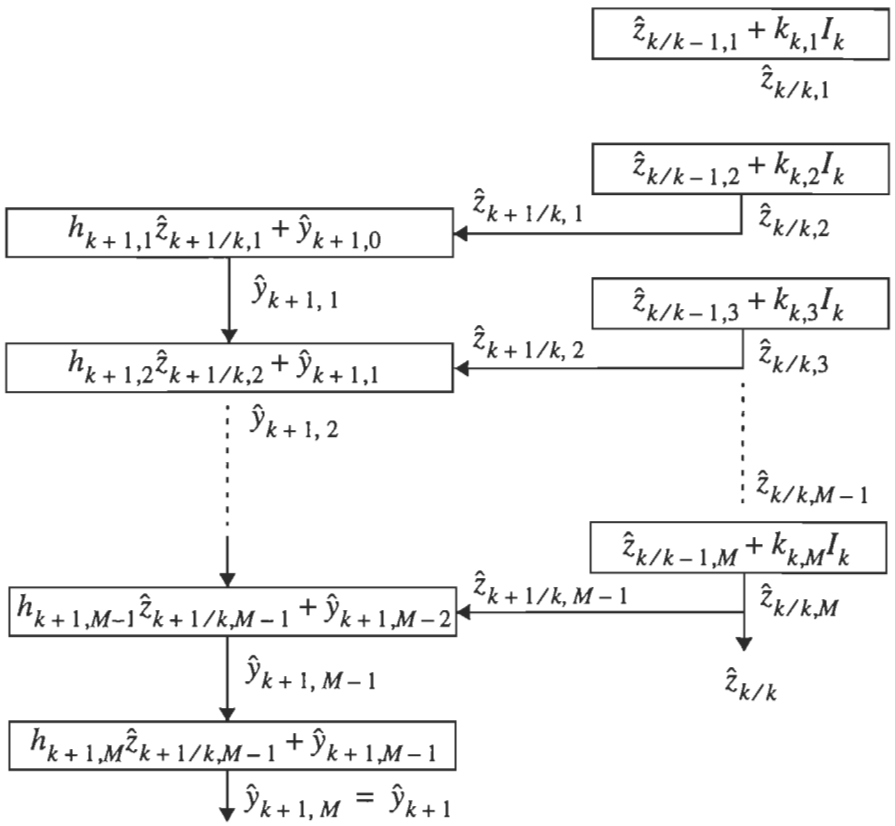

Figure 2-2 Séquencement des calculs en parallèles.

CHAPITRE 3

PROPOSITION D'UNE ARCHITECTURE SPÉCIALISÉE ET INTÉGRÉE

3.1 Système de reconstitution

Pour permettre d'utiliser efficacement le parallélisme dans l'algorithme, un processeur spécialisé avec deux M/A externes est proposé à la figure 3-2. Le processeur spécialisé est conçu pour fonctionner comme un co-processeur relié à l'ordinateur maître d'un système de mesure. L'ordinateur maître programme le processeur spécialisé avec les vecteurs de gain, la réponse impulsionnelle requise pour la reconstitution et la fréquence d'échantillonnage. Il reçoit ensuite les données reconstituées. Le M/A choisi est le TMC2210 de TRW. Il est disponible en quatre versions offrant des temps de calcul de 65, 80, 100 et 160 ns respectivement. Ceci permet de sélectionner le délai de propagation approprié en fonction de la technologie sur laquelle le processeur sera implanté. Afin que les M/A fonctionnent en parallèle, le temps de chargement additionné au délai de propagation interne par M/A doit être inférieur à la période de deux cycles d'horloge du processeur. Les caractéristiques sur lesquelles sont basées le choix du M/A sont la multiplication parallèle de 16 bits avec accumulateur de 35 bits, le fonctionnement en notation complément de deux et la capacité d'arrondissement du résultat.

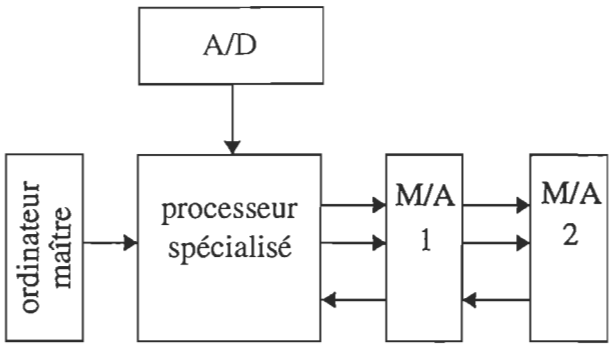

Figure 3-1 Schéma général du système de reconstitution.

3.2 Architecture avec M/A externes au processeur (DSP_KAL)

Le diagramme bloc de la figure 3-2 montre les blocs principaux de l'architecture du co-processeur, on y retrouve: les mémoires qui contiennent les vecteurs **z**, **h** et **k** utilisés par l'algorithme; les unités de génération d'adresses associées à chaque mémoire pour faciliter la manipulation des données; les bus internes et les multiplexeurs pour effectuer le transfert parallèle vers les ports d'entrée et de sortie de plusieurs données; le microséquenceur en mémoire RAM avec programmation horizontale qui synchronise les transferts parallèles en un seul cycle; l'unité opérative qui contient un comparateur pour la reconstitution avec contraintes et un bloc de compteurs qui permet de gérer les boucles nécessaires au traitement de vecteurs.

3.2.1 Unité de mémoire et de génération d'adresses

L'architecture est conçue pour maximiser le nombre d'opérations exécutables en parallèle. Chaque vecteur utilisé par l'algorithme est emmagasiné dans sa propre mémoire

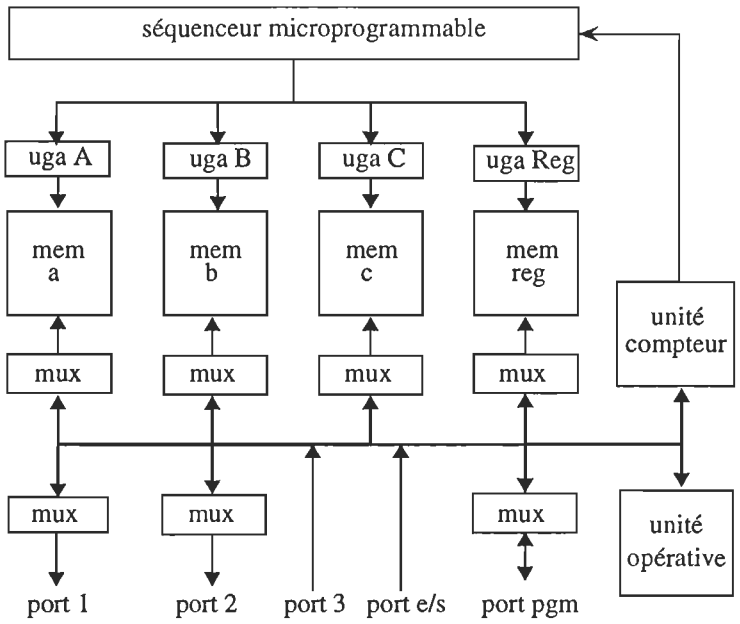

Figure 3-2 Diagramme bloc de l'architecture.

pour permettre un accès simultané aux différentes données. La décision d'utiliser des mémoires internes est partiellement justifiée par l'amélioration des temps d'accès. De plus, comme l'implantation tend à être limitée par le nombre d'entrées/sorties, ceci permet d'utiliser plus efficacement la surface de silicium. Précisons que dans sa forme actuelle, notre architecture requiert 144 broches. Les vecteurs **z** et **h** sont donc emmagasinés dans deux mémoires de 128 mots de 16 bits. La longueur de ces mémoires a été limitée à 128 mots puisque les applications envisagées ne requièrent pas un très long vecteur de réponse impulsionnelle pour caractériser le système de mesure. Le vecteur **k** est le gain stationnaire qui est calculé pour un niveau de bruit donné. Quatre exemplaires du vecteur **k** pour différents rapports signal à bruit sont stockés dans une mémoire de 512 mots de 16 bits pour permettre une meilleure reconstitution des signaux non-stationnaires. Le vecteur gain le mieux adapté

à la mesure peut donc être sélectionné. La quatrième mémoire est utilisée pour emmagasiner des informations telles que l'innovation et les données utiles à la reconstitution avec contraintes. Le choix de mots de 16 bits est fondé sur une analyse quantitative de l'effet de la quantification sur la qualité de reconstitution avec différents types de signaux spectrométriques [MAS95].

Pour éviter les délais en adressant les données, chaque mémoire possède sa propre unité de génération d'adresses contenant deux pointeurs indépendants. On retrouve à la figure 3-2, le diagramme bloc de l'unité de génération d'adresses avec les registres de base et les registres de déplacement propres à chaque pointeur [FEL90]. L'emploi de deux pointeurs permet de diviser la mémoire en sections que l'on adresse en sélectionnant le pointeur approprié. L'emploi d'un registre de déplacement avec l'additionneur facilite le traitement vectoriel et ne restreint l'accès aux pointeurs que dans les cas de chargement du registre.

3.2.2 Bus de données et multiplexeurs

Pour maximiser les transferts de données en parallèle sur un cycle, chaque mémoire possède son propre bus de données. L'interconnexion est assurée par un multiplexeur à l'entrée de chaque mémoire. Ceci permet une flexibilité au niveau de la manipulation des données advenant l'utilisation d'un algorithme différent. L'accès aux M/A à partir des différents bus est réalisé par un autre multiplexeur.

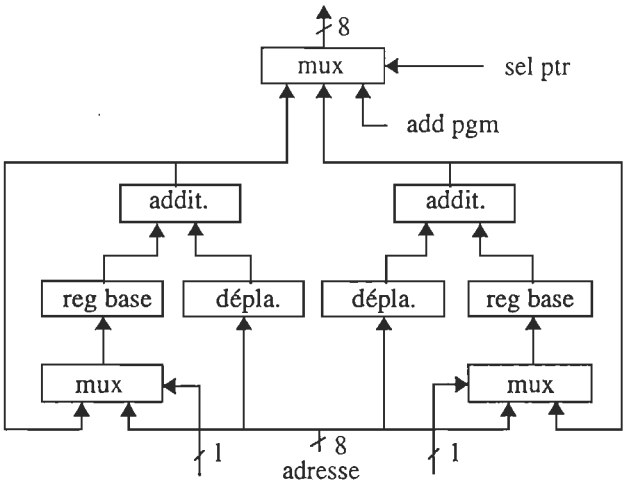

Figure 3-3 Unité de génération d'adresse.

3.2.3 Ports d'entrées et de sorties

La communication du processeur avec l'extérieur est assurée par 5 ports indépendants. Un port est dédié au convertisseur analogique/numérique. Ce dernier possède une largeur de 12 bits pour permettre la lecture d'un convertisseur en un cycle. La programmation des mémoires internes est accomplie par le processeur maître en utilisant le port PGM . Pour ce faire, il faut arrêter le processeur en activant le signal d'entrée HOLD. Avant d'autoriser la demande, le processeur se place en mode de programmation externe pour permettre l'accès à ses mémoires internes. Les ports 1, 2 et 3, d'une largeur de 16 bits, servent à communiquer avec les M/A. Les ports 1 et 2 servent de sorties pour charger les M/A, tandis que le port 3 est une entrée pour effectuer les lectures. Ainsi, l'opération de chargement et de lecture d'un M/A peut être effectuée en un cycle. Une erreur de quantification du résultat est introduite lors de la lecture parce que le résultat de la multiplication est sur 35 bits. Cette

erreur peut être minimisée à une résolution de 2^{-17} par une fonction du M/A qui arrondit à la valeur la plus proche. Il est important de noter que le chargement et la lecture simultanées du M/A impliquent que ce dernier se comporte comme un registre pipeline, dont il faut tenir compte dans la programmation.

3.2.4 L'unité arithmétique et logique

L'implantation d'une unité arithmétique et logique complète n'est pas rentable au niveau de la surface de silicium occupée par rapport à son utilisation. Deux unités indépendantes sont implantées pour réaliser les fonctions de base nécessaires. Par conséquent, l'unité opérative comprend un comparateur de 16 bits pour la reconstitution avec contraintes [MAS92a] et un additionneur de 16 bits pour permettre la résolution de l'équation (2.12) en un seul cycle. De plus, le bloc des compteurs est utilisé pour supporter l'exécution des boucles. Ce bloc comprend trois compteurs indépendants qui ont leurs fonctions propres. Un premier compteur de 16 bits sert de base de temps pour l'échantillonnage et les deux autres d'une largueur de 8 bits servent pour le contrôle de boucles. Un avantage de ce type de conception est de permettre un fonctionnement parallèle au transfert des mémoires.

3.2.5 Le microséquenceur programmable

Le microséquenceur programmable comporte une mémoire RAM de 128 mots x 80 bits. Il supporte une programmation horizontale afin d'exécuter le maximum d'opérations en parallèle en un cycle tout en minimisant la logique de commande. La figure 3-2 illustre les blocs principaux du séquenceur. On y retrouve la mémoire qui contrôle l'exécution des opé-

rations. Elle est séparée en différents champs et chacun contrôle une partie différente du processeur. Le multiplexeur de condition, le bloc de polarité et le bloc de sélection de la prochaine adresse réalisent les fonctions de branchement dans le microséquenceur [WHI81]. Le principe de fonctionnement général est le suivant: le multiplexeur de condition sélectionne lequel des états du système va influencer le branchement, le bloc de polarité détermine si le branchement va s'effectuer sur un état qui est présent ou non et le bloc d'affectation de la prochaine adresse va sélectionner d'où va provenir l'adresse de la prochaine instruction. L'adresse de branchement peut provenir du compteur de microprogramme de la mémoire pour un adressage immédiat ou du registre de sous-programme. La fonction sous-programme ne comporte qu'un seul niveau et ne peut donc pas permettre des sous-programmes imbriqués.

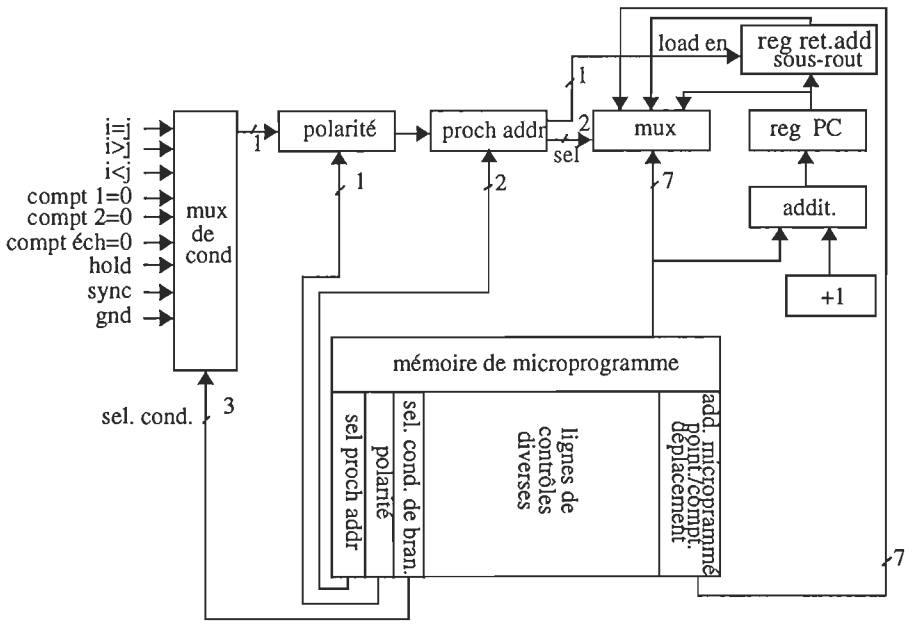

Figure 3-4 Microséquenceur programmable.

3.2.6 Intégration de l'architecture

L'intégration de l'architecture présentée dans ce chapitre pose plusieurs difficultés. Le nombre de broches requises évalué à 144 cause un problème au niveau du silicium. La surface du circuit résultant n'est plus déterminée par le nombre de portes logiques ou la quantité de mémoire intégré mais par la surface requise pour implanter les plots d'entrée/sortie. Le circuit est alors considéré comme "I/O bound". Ceci cause un gaspillage de la surface de silicium qui justifie difficilement sa fabrication. Le test du processeur après son intégration deviendrait difficile. Le testeur IMS disponible au laboratoire de VLSI de l'École Polytechnique de Montréal ne possède que 68 broches d'entrée/sortie. Un autre facteur qui affecte cette architecture provient de la nécessité de faire un transfert de données à chaque cycles d'horloge vers les multiplicateurs externes. Le délai de propagation diminue les performances du système de reconstitution. Il devient nécessaire pour permettre la fabrication du processeur d'optimiser son architecture sans affecter les performances.

3.3 Architecture avec M/A internes au processeur (DSP_KAL+)

Le gaspillage de la surface de silicium avec l'architecture DSP_KAL augmente les coûts de fabrication et diminue les possibilités d'utilisation à grande échelle. Une deuxième architecture (DSP_KAL+) est développée en fonction des compromis: surface de silicium occupée, performances et flexibilité de programmation d'une classe d'algorithmes. Il existe cinq différences notables entre l'architecture DSP_KAL et DSP_KAL+: l'intégration des M/A, l'élimination d'un banc de mémoire, l'optimisation du microséquenceur, la testabilité et le séquencement de l'algorithme.

3.3.1 Intégration des multiplicateurs/accumulateurs

L'intégration des multiplicateurs/accumulateurs externes à l'intérieur du processeur a pour effet de diminuer le nombre de broches externes requises et d'augmenter le nombre de portes logiques sur la surface de silicium. Le problème du circuit "I/O bound" de l'architecture DSP_KAL est éliminé et un gain en performance est obtenu en diminuant les délais d'interconnections externes. La figure 3-2 représente le diagramme bloc du multiplicateur/ accumulateur. On remarque les registres pipelines à l'entrée et à la sortie du M/A. La sortie du M/A est protégée par un tampon haute impédance car les deux M/A se partagent le même bus de sortie. Cette solution économise un bus interne car un des M/A est utilisé exclusivement pour la multiplication de deux vecteurs. Un multiplexeur à l'entrée de l'additionneur permet de sélectionner le M/A en mode multiplicateur seulement.

Le choix du type de M/A est important car il influence directement les performances du processeur. Un deuxième facteur qui influence les performances et la latence du circuit est l'insertion de registres pipeline dans le M/A. Cette approche n'a pas été retenue dans l'architecture DSP_KAL+ pour deux raisons. La première est l'augmentation de la surface de silicium. La possibilité de fabrication est directement reliée à la dimension du circuit. Il est donc important de la réduire au minimum. La deuxième raison est l'augmentation de la complexité du microséquenceur. Pour tenir compte de l'augmentation de la latence du M/A, le microséquenceur doit être modifié pour tenir compte du pipeline dans le cas ou un branchement conditionnel est effectué. Cette augmentation de la complexité n'est pas justifiée dans le cadre du projet. À noter qu'il serait intéressant pour l'exécution de l'algorithme Kal-

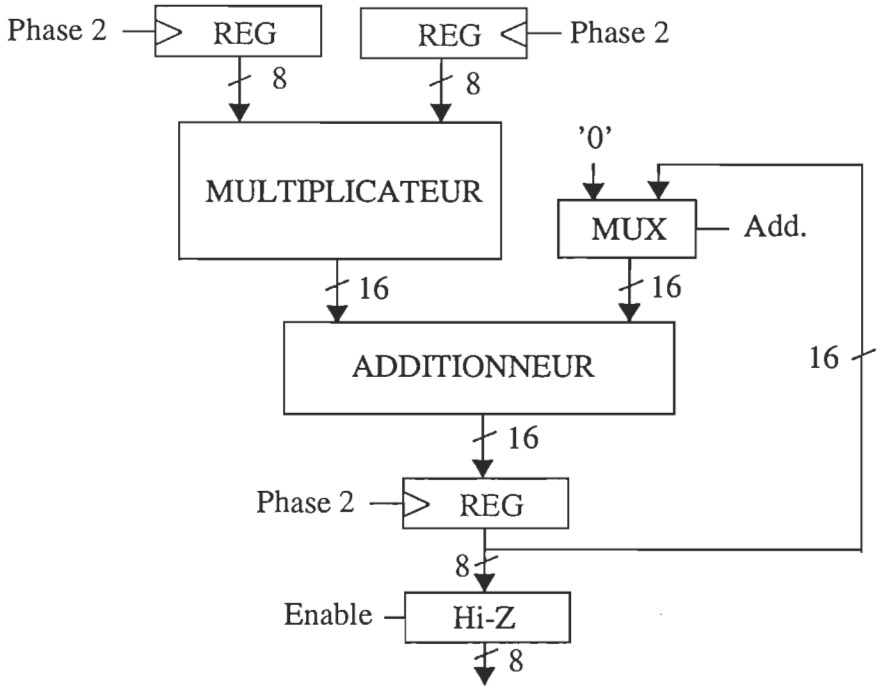

Figure 3-5 Diagramme bloc du multiplicateur/accumulateur.

man d'utiliser une technique de pipeline par vague ("wave pipeline") pour les M/A. Cette technique repose sur l'utilisation du délai de propagation de la logique combinatoire pour servir de registre pipeline [GHO95]. Pour le moment, aucun essai n'a été effectué pour validé l'utilisation de cette méthode sur l'architecture DSP_KAL+.

3.3.1.1 Architecture des multiplicateurs

Le choix de l'architecture des multiplicateurs est basé sur quatre critères précis. Le premier et le plus important est le nombre de portes logiques utilisées. La surface de silicium doit être la plus compacte possible pour augmenter les chances de se qualifier pour une implémentation par la Société Canadienne de Microélectronique (SCM). La proportion du

nombre de portes pour un M/A est d'environ 18% du nombre total de portes du processeur. Il est donc important de le minimiser. Le deuxième critère est la vitesse. Les performances globales du processeur sont directement reliées à celles des multiplicateurs. Le troisième critère est la régularité de la structure. Ce critère englobe un peu les deux premiers, une structure régulière de multiplicateur va diminuer la surface due aux interconnexions et augmenter la vitesse. Le quatrième et dernier critère est le format de numérisation d'entrée des nombres. Le multiplicateur doit être capable d'opérer avec des nombres en complément de deux.

Il existe une multitude d'architectures de multiplicateurs et de publications sur le sujet. Il est très difficile de déterminer avec rigueur la meilleure solution à notre problème car il n'existe pas de solution unique. La sélection est effectuée de façon intuitive en utilisant l'architecture la plus populaire commercialement soit l'algorithme de Booth modifié. Contrairement au multiplicateur matriciel où la multiplication de deux nombres de N bits est équivalente à additionner des nombres de N bits décalés correctement N fois, l'algorithme de Booth examine plusieurs bits à la fois et permet des opérations plus complexes sur le nombre multiplié pour diminuer le nombre de produits partiels [ANN86]. À titre d'exemple, le multiplicateur de 8x8 bits employé dans le prototype du processeur comprend 631 portes logiques et a un délai de propagation de 28 ns dans une technologie de 1.2 µm.

3.3.1.2 Architecture des accumulateurs

La recherche de performance maximale évaluée en terme de vitesse est le critère de sélection principal pour l'additionneur. La table 3-1 compare la surface et les delais de pro-

pagation des trois types d'additionneurs les plus répandus: "ripple", "carry look-ahead", "carry select". L'architecture choisie est du type "carry select". Elle double la vitesse par rapport à un additionneur classique mais cette performance est au détriment de la surface qui double elle aussi. Le coût requis d'environ 134 portes logiques est acceptable pour le projet. Ce type d'additionneur utilise quatre sections d'additionneurs de quatre bits du type "carry look-ahead" (CLA) pour les produits intermédiaires [ANN86].

	"Ripple"	"CLA"	"Carry select"
Nombre de portes	62	48	134
Délai de propagation (ns)	25	20	13
Nombre de portes*délai	1550	960	1742
Nombre de portes*délai^2	38750	19200	22646

Table 3-1 Comparaison des architectures d'additionneur

3.3.2 Banc de mémoire

L'architecture de DSP_KAL comporte quatre bancs de mémoire interne. Trois des quatre bancs de mémoire sont attribués au vecteur d'état, au gain de Kalman et à la réponse impulsionnelle. Les trois vecteurs ont une dimension de 128 éléments. Le quatrième banc de mémoire est utilisé pour sauvegarder des données temporaires ou des constantes utilisées dans la reconstitution avec contraintes [MAS92a]. Ce banc de mémoire a une dimension de 16 éléments. Pour sauvegarder le plus de surface de silicium possible, ce banc de mémoire est éliminé dans DSP_KAL+. Il est remplacé par quelques registres qui contiennent l'infor-

mation nécessaire. On élimine ainsi la surface requise pour la mémoire de 16x16 bits et l'unité de génération d'adresse. Le nombre d'entrées des multiplexeurs de bus est réduit de 5 à 4 puisque le nombre de bus internes diminue de un.

3.3.3 Optimisation du micro-séquenceur

Le microséquenceur de l'architecture DSP_KAL+ minimise la largeur de mot de la mémoire programme. La version DSP_KAL du microséquenceur utilise un mot de 80 bits par rapport à celle de DSP_KAL+ qui n'utilise que 64 bits. La table 3-4 présente le contenu du champ de microprogrammation du microséquenceur de DSP_KAL+. Une première réduction est obtenue avec l'abandon d'un des bancs de mémoire interne. On économise ainsi 11 bits pour le contrôle de l'unité de génération d'adresse. Les autres réductions proviennent de la diminution du nombre de bus internes suite à l'abandon du banc de mémoire. Le nombre de bits requis pour le contrôle des multiplexeurs passe de trois à deux, pour une économie totale de six bits.

La mémoire contenant le programme du microséquenceur est conçue en 8 sections de mots de 8 bits pour améliorer le temps d'accès à la mémoire comparativement à une mémoire utilisant des mots de 64 bits. Ce fractionnement de la mémoire facilite aussi le chargement à partir de l'interface externe. La sélection des sections de la mémoire du microséquenceur est faite à partir des contrôles CS4, CS3, CS2 et CS1. La table 3-2 indique la relation. La mémoire D contient les bits les moins significatifs (LSB) tandis que la mémoire K contient les bits les plus significatifs (MSB). Nous verrons au chapitre 4 une description d'une simulation du chargement de ces mémoires.

CS4	CS3	CS2	CS1	Mémoire sélectionnée
0	0	0	0	mémoire A (données)
0	0	0	1	mémoire B (données)
0	0	1	0	mémoire C (données)
0	0	1	1	mémoire D (µSéq. LSB)
0	1	0	0	mémoire E (µSéq.)
0	1	0	1	mémoire F (µSéq.)
0	1	1	0	mémoire G (µSéq.)
0	1	1	1	mémoire H (µSéq.)
1	0	0	0	mémoire I (µSéq.)
1	0	0	1	mémoire J (µSéq.)
1	0	1	0	mémoire K (µSéq. MSB)
1	0	1	1	sans usage
1	1	X	X	sans usage

Table 3-2 Sélection des mémoires internes

La logique de branchement du microséquenceur est presque identique à celle de l'architecture DSP_KAL à l'exception du registre d'adresse pour le retour d'une sous-routine. La logique pour les sous-routines a été abandonnée pour simplifier la conception logique. La figure 3-2 démontre l'architecture de la logique de branchement pour DSP_KAL+.

La description de la table de vérité du branchement se trouve à la table 3-3. Le signal "état" indique le résultat d'une opération sélectionnée par le multiplexeur de condition. Les opérations disponibles sont données à la table 3-3. Pour permettre un meilleur choix de branchement conditionnel la version DSP_KAL+ utilise un bit de plus que la version DSP_KAL. La sélection de l'entrée est faite avec les quatre bits du microséquenceur "sel. cond.". Le signal "polarité" indique si le résultat de l'opération sélectionnée est actif bas ou haut. Un "1" logique repésente un état actif bas et un "0" logique représente un état actif haut. Le signal "$\overline{\text{sel. bran.}}$" indique si un branchement conditionnel est permis ou non.

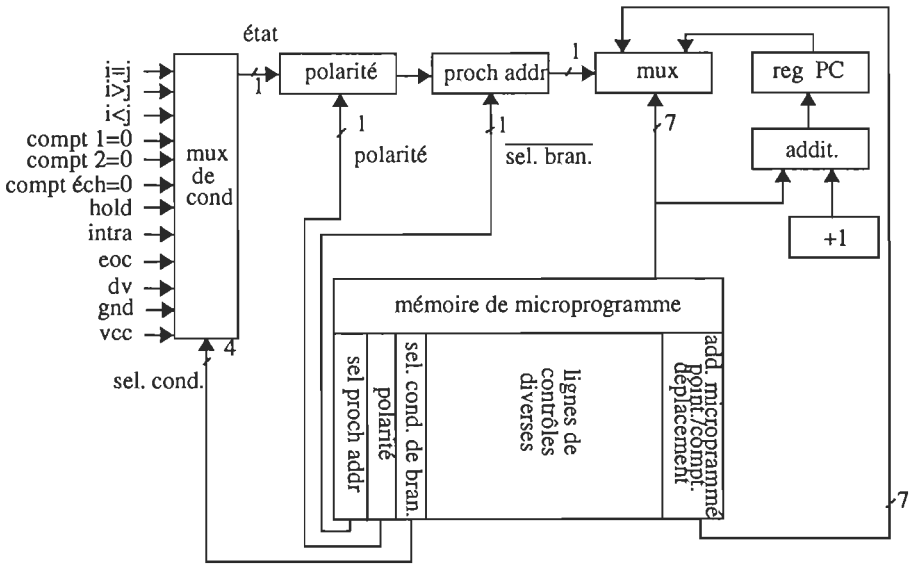

Figure 3-6 Microséquenceur programmable de DSP_KAL+.

état	polarité	sel. bran.	résultat
0	0	0	PC=PC+1
0	0	1	PC=PC+1
0	1	0	PC=Donnée (bran. cond.)
0	1	1	PC=PC+1
1	0	0	PC=Donnée (bran. cond.)
1	0	1	PC=PC+1
1	1	0	PC=PC+1
1	1	1	PC=PC+1

Table 3-3 Table de vérité du microséquenceur

bit	nom	description
0	sel_ptr_memA	Sélectionne le pointeur d'adresse de la mémoire A 0 -> sélectionne le pointeur 1 1 -> sélectionne le pointeur 2
1	wr_reg1_memA	Chargement du registre de base 1 note: Le signal reg_base1_memA doit être actif au même moment.
2	wr_reg2_memA	Chargement du registre de base 2 note: Le signal reg_base2_memA doit être actif au même moment.

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
3	reg_base1_memA	Incrémente le registre de base du pointeur d'adresse 1 de la mémoire A
4	reg_dep1_memA	Chargement du registre de déplacement du pointeur d'adresse 1 de la mémoire A
5	reg_base2_memA	Incrémente le registre de base du pointeur d'adresse 2 de la mémoire A
6	reg_dep2_memA	Chargement du registre de déplacement du pointeur d'adresse 2 de la mémoire A
7	r/$\overline{w}$_memA	Sélectionne l'écriture ou la lecture de la mémoire A 0 -> pour l'écriture 1 -> pour la lecture
[9:8]	mux_memA	Sélectionne une des 4 entrées du multiplexeur de bus de la RAM A 00 -> bus RAM B 01 -> bus RAM C 10 -> bus M/A 11 -> bus es_pgm
10	sel_ptr_memB	Sélectionne le pointeur d'adresse de la mémoire B 0 -> sélectionne le pointeur 1 1 -> sélectionne le pointeur 2
11	wr_reg1_memB	Chargement du registre de base 1 note: Le signal $\overline{\text{reg\_base1\_memB}}$ doit être actif au même moment.
12	wr_reg2_memB	Chargement du registre de base 2 note: Le signal $\overline{\text{reg\_base2\_memB}}$ doit être actif au même moment.

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
13	reg_base1_memB	Incrémente le registre de base du pointeur d'adresse 1 de la mémoire B
14	reg_dep1_memB	Chargement du registre de déplacement du pointeur d'adresse 1 de la mémoire B
15	reg_base2_memB	Incrémente le registre de base du pointeur d'adresse 2 de la mémoire B
16	reg_dep2_memB	Chargement du registre de déplacement du pointeur d'adresse 2 de la mémoire B
17	r/$\overline{w}$_memB	Sélectionne l'écriture ou la lecture de la mémoire B 0 -> pour l'écriture 1 -> pour la lecture
[19:18]	mux_memB	Sélectionne une des 4 entrées du multiplexeur de bus de la RAM B 00 -> bus RAM B 01 -> bus RAM C 10 -> bus M/A 11 -> bus es_pgm
20	sel_ptr_memC	Sélectionne le pointeur d'adresse de la mémoire C 0 -> sélectionne le pointeur 1 1 -> sélectionne le pointeur 2
21	wr_reg1_memC	Chargement du registre de base 1 note: Le signal $\overline{\text{reg\_base1\_memC}}$ doit être actif au même moment.
22	wr_reg2_memC	Chargement du registre de base 2 note: Le signal $\overline{\text{reg\_base2\_memC}}$ doit être actif au même moment.

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
23	reg_base1_memC	Incrémente le registre de base du pointeur d'adresse 1 de la mémoire C
24	reg_dep1_memC	Chargement du registre de déplacement du pointeur d'adresse 1 de la mémoire C
25	reg_base2_memC	Incrémente le registre de base du pointeur d'adresse 2 de la mémoire C
26	reg_dep2_memC	Chargement du registre de déplacement du pointeur d'adresse 2 de la mémoire C
27	r/$\overline{w}$_memC	Sélectionne l'écriture ou la lecture de la mémoire C 0 -> pour l'écriture 1 -> pour la lecture
[29:28]	mux_memC	Sélectionne une des 4 entrées du multiplexeur de bus de la RAM C 00 -> bus RAM B 01 -> bus RAM C 10 -> bus M/A 11 -> bus es_pgm
[31:30]	mux_ina_ma	Sélectionne une des 4 entrées du multiplexeur de bus pour l'accès à l'entrée A des multiplicateurs/accumulateur 00 -> bus RAM A 01 -> bus RAM B 10 -> bus RAM C 11 -> bus M/A

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
[33:32]	mux_inb_ma	Sélectionne une des 4 entrées du multiplexeur de bus pour l'accès à l'entrée B des multiplicateurs/accumulateur 00 -> bus RAM A 01 -> bus RAM B 10 -> bus RAM C 11 -> bus M/A
[35:34]	mux_pgm_es	Sélectionne une des 4 entrées du multiplexeur de bus pour le bus es_pgm 00 -> bus RAM A 01 -> bus RAM B 10 -> bus RAM C 11 -> bus M/A
[36]	sel_char_clk	Chargement du compteur de boucle sélectionné au bit 47-48
[40:37]	mux_cond	Sélectionne une des entrées de conditions 0000 -> comparateur égale 0001 -> comparateur plus petit 0010 -> comparateur plus grand 0011 -> compteur échant. 1 égale zéro 0100 -> compteur 1 égale zéro 0101 -> compteur 2 égale zéro 0110 -> entrée hold 0111 -> entrée intra 1000 -> entrée eoc 1001 -> entrée dv 1010 -> '0' 1011 -> '1'
[41]	polarité	Sélection du niveau sur lequel une entrée est active 0 -> actif haut 1 -> actif bas

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
[42]	sel_bran	Sélectionne le branchement si la condition est remplie. 0 -> branchement sur condition 1 -> prochaine addresse PC+1
[43]	wr_ma1_A	Charge l'entrée A du multiplicateur accumulateur 1
[44]	rd_ma1	Lecture du multiplicateur/accumulateur 1
[45]	wr_ma2	Charge l'entrée A et B du multiplicateur/accumulateur 2
[46]	rd_ma2	Lecture du multiplicateur/ accumulateur 2
[48:47]	sel_compt	Sélectionne le compteur de boucle à décrémenter ou charger si sel_char_clk = 0 00 -> nil 01 -> compteur d'échantillonnage 10 -> compteur 1 11 -> compteur 2
[49]	holda	Écriture sur la broche de sortie holda
[50]	cs_ad	Écriture sur la broche de sortie cs_ad
[51]	oe	Écriture sur la broche de sortie oe
[52]	sc	Écriture sur la broche de sortie sc
[57:53]	donnée	Champs de donnée pour le chargement des pointeurs ou l'adresse de branchement
[58]	add_ma1	Sélectionne le mode d'accumulation sur le multiplicateur/ accumulateur 1

Table 3-4 Description du champ de microprogrammation du microséquenceur

bit	nom	description
[59]	add_ma2	Sélectionne le mode d'accumulation sur le multiplicateur/ accumulateur 2
[60]	intr	Écriture sur la broche de sortie intr
[61]	wr_ma1_B	Charge l'entrée B du multiplicateur/accumulateur 1
[62]	rd_résult	Lecture de l'additionneur
[63]	pgm_ext	Sélection du mode de programmation des mémoires internes par le processeur maître

Table 3-4 Description du champ de microprogrammation du microséquenceur

3.3.4 Modifications pour la testabilité

La version originale du processeur est conçue pour fonctionner avec une horloge à deux phases sans recouvrement. L'emploi d'une horloge à deux phases permet d'utiliser des bascules actives sur un niveau à la place de bascules D ce qui permet une économie sur le nombre total de transistors requis. Une bascule active sur un niveau utilise 13 transistors par rapport à une bascule D qui en utilise 19 selon la technologie de la SCM. Cependant, l'utilisation d'une chaîne de balayage interne pour la testabilité du processeur requiert l'emploi de bascules D testables au lieu de bascules actives sur un niveau. Le remplacement des bascules actives sur un niveau par des bascules D s'est fait de façon directe dans tout le circuit, mis à part les boucles de rétroaction des pointeurs d'adresses de mémoire où on utilise la caractéristique des bascules D, actives sur une transition, pour simplifier la conception. Le remplacement des bascules actives sur un niveau par des bascules D implique de substituer

les bascules actives sur la phase 1 par des bascules D actives sur un front descendant et les bascules actives sur la phase 2 par des bascules D actives sur un front montant. Le section 3.4 décrit en détail la testabilité du processeur.

3.3.5 Séquencement de l'architecture

Le remplacement des bascules actives sur un niveau par des bascules D modifie le comportement de l'architecture mais le séquencement des opérations reste le même. Le diagramme de PERT, présenté à la figure 3-2, donne une représentation graphique du séquencement des diverses étapes dans l'opération du processeur. On remarque sur la figure que le graphique représentant le séquencement des opérations est replié sur lui-même. Ceci indique qu'au moment où une transition est effectuée entre la phase 1 et la phase 2 ou vice-versa, deux événements sont déclenchés et les opérations sont effectuées en parallèle.

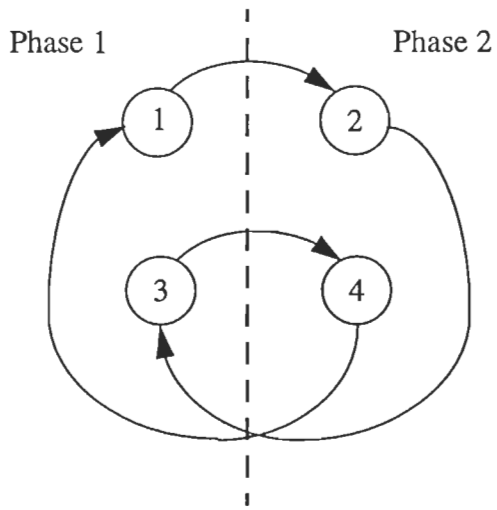

Figure 3-7 Diagramme de Pert de l'architecture DSP_KAL+.

C'est ce qu'on appelle le pipeline du circuit [ANC86]. L'architecture de DSP_KAL+ utilise un pipeline à deux niveaux qui augmente la complexité du circuit mais il permet d'en doubler la vitesse. Les quatre étapes du séquencement sont décrites dans la table 3-5. Les étapes 1 et 2 sont utilisées par le microséquenceur pour le contrôle. Les étapes 3 et 4 sont utilisées par le chemin de données, c'est à dire l'accès aux mémoires et aux M/A. Selon la figure 3-2, les étapes 1 et 3 sont effectuées en parallèle durant la phase 1 et les étapes 2 et 4 sont effectuées en parallèle durant la phase 2. Les quatre étapes sont exécutées en un seul cycle en utilisant une horloge à deux phases sans recouvrement.

Étape	Opération
1	Opération sur le compteur de programme
2	Accès à la mémoire du microséquenceur
	Adressage des bancs de mémoire interne
3	Applique les signaux de contrôle du microséquenceur
	Lecture des mémoires internes
4	Lecture des M/A
	Écriture aux mémoires internes
	Écriture aux M/A

Table 3-5 Étapes de séquencement de l'architecture

3.4 Testabilité de l'architecture DSP_KAL+

Cette section traite de la testabilité du circuit conçu. La conception d'un circuit intégré testable permet une diminution des coûts en permettant la détection d'une pièce défectueuse le plus tôt possible. La nécessité de se préoccuper de la testabilité découle du fait qu'en général une proportion substantielle des circuits intégrés manufacturés sont défectueux. La planification de la méthode de test fait donc partie intégrante du cycle de conception. Le choix de la méthode de test s'est porté sur l'insertion d'une chaîne de balayage interne. Ce choix est dicté par le coût modéré en logique additionnelle, le nombre minimum de broches supplémentaires requises et par la disponibilité au laboratoire VLSI d'un logiciel de génération automatique de vecteurs de test. Ce logiciel nommé DFT provient de BNR. L'emploi d'un outil de génération automatique de vecteurs de test permet de générer une couverture élevée des pannes potentielles, tout en diminuant le temps requis pour déterminer les vecteurs de test.

3.4.1 Insertion de la chaîne de balayage

L'utilisation d'une chaîne de balayage permet d'améliorer l'observabilité et la contrôlabilité des noeuds internes du circuit. Cette méthode est facilement mise en oeuvre dans les circuits de types synchrones. La technique requiert que toutes les bascules du circuit soient remplacées par leur équivalent testable. Les bascules sont ensuite reliées entre elles pour former une chaîne de balayage. Une bascule testable opère comme une bascule durant le mode de fonctionnement normal, mais elle fait partie d'un registre à décalage en mode balayage.

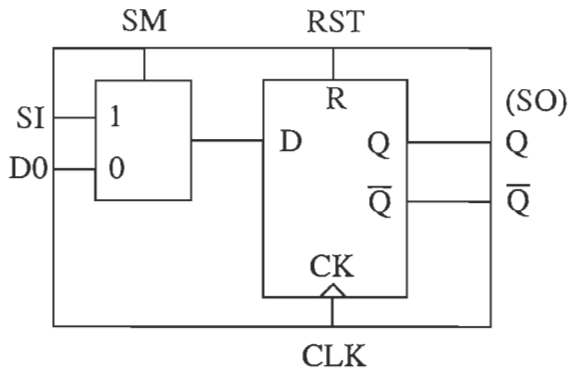

Figure 3-8 Représentation testable d'une bascule.

L'emploi d'une chaîne de balayage requiert quatre broches externes: SM (mode balayage), TM (mode test), SI (entrée de la chaîne de balayage), SO (sortie de la chaîne de balayage).

Dans le mode balayage, contrôlé par la broche SM, les éléments de la chaîne de balayage décalent les données le long de la chaîne, initialisant ainsi tous les éléments. Le circuit retourne en mode normal pour un cycle d'horloge et le vecteur qui est emmagasiné dans les bascules sert alors de stimulus pour la section combinatoire du circuit. Les résultats sont emmagasinés dans la chaîne de balayage au prochain coup d'horloge. Le circuit est remis en mode balayage et les données emmagasinées sont décalées vers la sortie pour être comparées avec la réponse prévue. La broche TM est gardée à un niveau constant durant le mode balayage. Elle sert à reconfigurer la logique du circuit durant le mode balayage, pour éviter les boucles de rétroaction qui ne sont pas testables, ainsi que les problèmes de contention sur les tampons à haute impédance.

La génération des vecteurs de test et de la réponse prévue est réalisée automatiquement avec DFT (Design For Testability) [DFT91]. Le logiciel DFT est séparé en trois sections indépendantes:

- La première section SCANCHECK vérifie la description du circuit pour détecter les violations aux règles de base de conception requise par le logiciel. L'annexe A décrit en détail les règles d'utilisation de DFT. La logique du circuit qui n'est pas connectée et les entrées qui sont connectées à un niveau logique constant sont enlevées du circuit. SCANCHECK remplace les portes logiques par leurs primitives, c'est-à-dire des portes de types OU, ET, inverseur et bascule. Le modèle de panne de chaque primitive de base est assigné aux portes logiques.

- La deuxième section, ATPG (Automatic Test Pattern Generator), est un générateur de vecteurs de test qui utilise le résultat de SCANCHECK. Ce générateur traite le circuit comme un circuit combinatoire accessible par la chaîne de balayage. Cet outil comporte aussi un simulateur de pannes. Ce dernier estime la couverture de pannes, qui sert de critère d'arrêt au processus de génération.

- La troisième section, SEQUENCE, formate les vecteurs de tests en fonction de la structure de la chaîne de balayage et des informations sur les broches d'entrées. Il faut préciser que la troisième section de DFT n'est pas disponible présentement au laboratoire. Les résultats obtenus s'arrêteront donc à la détermination de la couverture de pannes obtenue sur le circuit à partir du logiciel.

Une des premières étapes pour utiliser DFT a été de créer une représentation Verilog pour toutes les portes logiques utilisées dans le circuit. Ceci a permis la génération d'une description Verilog de notre circuit à partir de CADENCE. Le modèle de panne pour la plupart des portes logiques de la bibliothèque CMOS4S était déjà fourni par la CMC. Il ne restait que le développement de la représentation Verilog et du modèle fonctionnel avec les primitives des cellules peu utilisées tel les plots d'entrée/sortie et la mémoire interne.

3.4.2 Résultats de SCANCHECK

La première section SCANCHECK utilise comme entrée la description Verilog du circuit et le fichier chip_DSP.scancheck. Le fichier chip_DSP.scancheck définit la broche de commande du mode balayage SM et les entrées qui sont gardées à un niveau constant durant le test de balayage: TM et RESET. Le circuit est vérifié et les violations aux règles de base de conception sont rapportées dans le fichier chip_DSP.lis. On retrouve dans le même fichier les renseignements suivants sur le circuit: le nombre total de primitives (portes) testables utilisées par ATPG; le nombre de portes non-utilisées enlevées du circuit parce que leur sortie est constante ou non observable, le nombre de portes non testable parce que leur sortie est constante ou non observable durant la durée du test et le nombre total de portes.

Le processeur contient 4503 primitives (portes) testables par ATPG. Le nombre de portes inutilisées parce que leur sortie est constante ou non observable s'élève à 1386. Le fichier chip_DSP.unused contient la liste de toutes les portes non utilisées. Ce nombre ne représente pas la réalité. Il comprend toutes les alimentations et masses insérées automatiquement dans la description Verilog à chaque sous module. Le nombre de portes non-utili-

sées descend à 576 en enlevant ces cas particuliers. Du nombre restant, on retrouve en majorité des portes dont la sortie est non observable. Les mémoires internes en sont la cause principale, elles sont traitées par l'outil comme des boîtes noires inaccessibles. Par exemple, les entrées de la mémoire sont considérées non observables et la logique combinatoire provenant de points contrôlables jusqu'à la mémoire est enlevée et considérée comme non testable. Une autre conséquence est que les sorties de la mémoire sont considérées comme non contrôlable et la logique combinatoire qui en découle et qui dépend de ces sorties est enlevées et considérées comme non testable. L'annexe B mentionne quelques méthodes pour améliorer la couverture de pannes. Aucune de ces méthodes n'a été retenue dans le processeur pour limiter le nombre de portes logiques totales à un strict minimum.

Le nombre de portes éliminées parce que leur sortie est constante ou non observable durant la durée du test s'élève à 629. Ces portes ne sont pas testées par ATPG. D'autres vecteurs de test doivent donc être développés pour couvrir les pannes associées à ces portes. Le fichier chip_DSP.pruned contient la liste de toutes les portes éliminées. La sortie d'une porte est constante si l'une ou plusieurs de ses entrées TM est d'une valeur constante. D'autres portes sont éliminées parce que leurs sorties ne sont pas observables après le retrait des portes dont la sortie est constante. La presque totalité des portes éliminées fait partie des multiplexeurs avec un tampon haute impédance à la sortie. Pour éviter les problèmes de contention en mode test, les tampons sont forcés en haute impédance. Ceci empêche la sortie des multiplexeurs d'être observable.

Le nombre total de portes équivaut au total du nombre de portes testables, du nombre

de portes inutilisées et du nombre total de portes éliminées. Le nombre total est égal à 5708 portes.

SCANCHECK crée aussi deux fichiers de sortie qui seront utilisés dans les étapes suivantes. Le premier fichier chip_DSP.tgfile est utilisé par ATPG et il contient une description de la logique combinatoire reliée à la chaîne de balayage. Le deuxième fichier chip_DSP.scaninfo est utilisé par SEQUENCE et il décrit les connexions de la chaîne de balayage et le type des broches.

3.4.3 Résultats produits par ATPG

La deuxième section ATPG utilise le fichier chip_DSP.tgfile pour générer les vecteurs de test de la logique combinatoire. Elle réalise aussi les simulations de pannes du circuit combinatoire et elle écrit les résultats sur la couverture de pannes et le nombre de vecteurs générés à la suite du fichier chip_DSP.listing. La couverture de pannes garantie de la partie testable du processeur est 95.61% avec 118 vecteurs de test. Le fichier chip_DSP.pattern contient les 118 vecteurs de tests. La couverture de pannes garantie (solid fault) est le rapport du nombre de pannes dont la détection est garantie sur le nombre de pannes total.

Le pourcentage de pannes redondantes est de 4.39%. Une panne redondante est une panne pour lequel il est impossible de trouver un vecteur de test [DFT91]. Le fichier chip_DSP.rfl contient la liste de toutes les pannes redondantes. Le fichier rapporte beaucoup de pannes redondantes de manière erronée. La figure 3-2 est un schéma des mémoires du microséquenceur. En mode test, la sortie des tampons haute impédance est non-contrôlable,

et la mémoire est considérée comme une boîte noire. Le bus bus_pgm_int est laissé flottant, cependant, le modèle DFT du tampon haute impédance propage la valeur logique 0 à sa sortie. L'entrée D du registre pourrait être forcée à zéro et la fonction logique vue par ATPG serait identique. Une erreur de panne collée à zéro est donc rapportée pour l'entrée D du registre.

Une couverture nominale de 95% sur la logique testable a été atteinte sur le processeur. Ce facteur diminue à 80% si on inclut la logique non utilisée. Il faut mentionner que la couverture de 80% exclut le test des mémoires internes. Ce pourcentage n'est pas très élevé, mais il s'agit d'un compromis entre la couverture de pannes et le nombre total de portes logiques. Ces résultats ne seraient pas acceptables en production mais il s'agit ici d'un prototype de recherche. Pour atteindre 100% il faut poursuivre mais c'est long et coûteux.

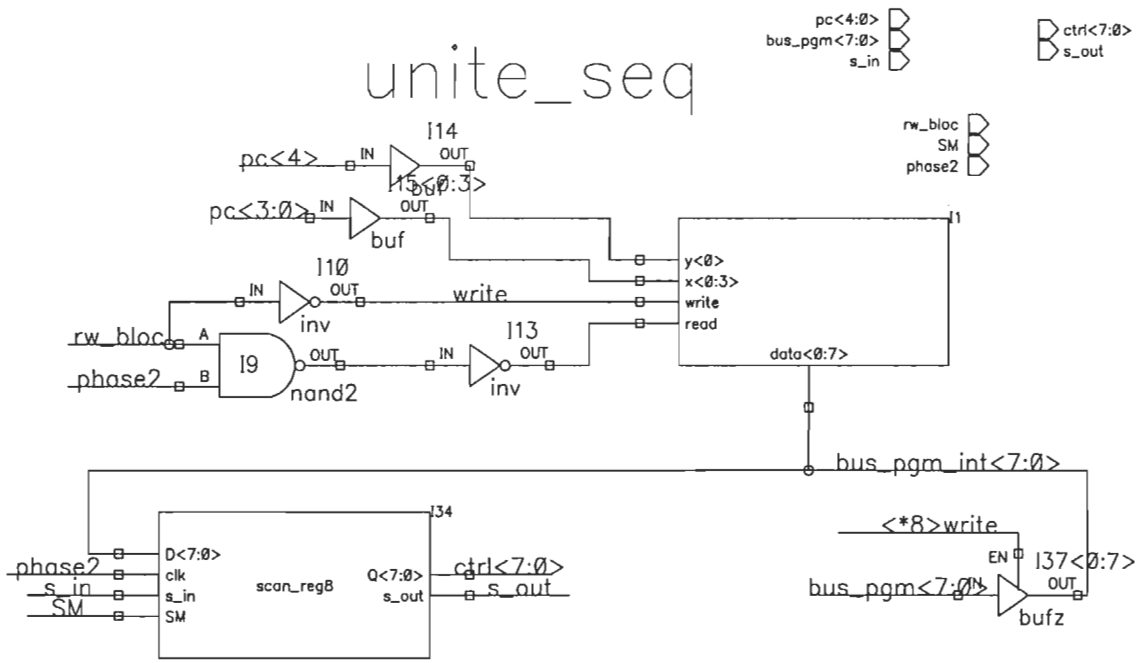

Figure 3-9 Mémoire du microséquenceur.

3.5 Intégration de l'architecture DSP_KAL+

3.5.1 Modèle Verilog et synthèse

La première étape de conception du processeur intégré est le développement du modèle de l'architecture DSP_KAL en langage Verilog. Ce modèle permet de valider par simulation les décisions prises sur l'architecture. L'objectif original était d'utiliser le logiciel de synthèse SYNFUL de BNR [SYN92] pour convertir le code Verilog en un circuit logique. Cependant, l'interface Verilog du logiciel de synthèse est très limitée. Il est impossible de faire la synthèse d'une boucle de rétroaction. Le logiciel est inutilisable pour les pointeurs d'adresses de mémoires. Pour la synthèse de logiques combinatoires le logiciel est utile mais la synthèse des multiplicateurs n'est pas supportée et les circuits de contrôles des mémoires internes introduisent des transitions parasites. Après plusieurs tentatives infructueuses pour obtenir un résultat adéquat l'approche est abandonnée au profit de la méthode d'entrée schématique pour l'architecture DSP_KAL+. Le modèle Verilog de DSP_KAL est à l'annexe C.

3.5.2 Prototype de l'architecture DSP_KAL+

Le nombre total de portes requises pour une mise en oeuvre avec une largeur de mots de 16 bits de l'architecture DSP_KAL+ diminue de beaucoup la possibilité de fabrication du processeur par la Société Canadienne de Microélectronique. La SCM attribue la surface de silicium en fonction du nombre de demandes, de la surface requise et du mérite du projet. Une version réduite de l'architecture DSP_KAL+ est proposée en tant que prototype. Cette

version prototype avec une largeur de mots de 8 bits permet une réduction substantielle de la surface de silicium requise. Les multiplicateurs de 8x8 bits requièrent environ 500 portes logiques au lieu de 2000 pour un multiplicateur de 16x16 bits. Le nombre de portes logiques requises pour le reste de la logique combinatoire et des bascules à transition est diminuées de moitié. Une économie d'environ 2500 portes logiques est réalisée.

Une autre économie de surface est réalisée en limitant à 32 mots de profondeur les bancs de mémoires A,B et C et la mémoire du microséquenceur. Normalement la profondeur des mémoires requise pour une reconstitution adéquate est de 64 mots [MAS92b]. La limitation de la mémoire du microséquenceur empêche d'exécuter le programme de reconstruction en une seule étape mais la quantité de surface de silicium économisée est non négligeable puisque la mémoire possède une largeur de mots de 64 bits. Le test du processeur doit cependant être fait à l'aide des opérations vectorielles de base utilisées dans l'algorithme de Kalman puisqu'il devient impossible de charger le programme de reconstitution au complet . Nous verrons au chapitre 4 une description détaillée des programmes de test.

3.5.3 Intégration sur silicium

L'intégration de l'architecture DSP_KAL+ sur silicium est réalisée avec le logiciel Edge 2.1 de Cadence disponible au laboratoire de VLSI de l'École Polytechnique de Montréal. Les différentes étapes requises pour le transfert du processeur conçu avec une entrée schématique vers le silicium sont: le placement, le routage, la vérification des règles de dessins ("DRC"), la vérification de la fonctionnalité et la génération du fichier de transfert pour la fonderie ("tapeout").

Le placement consiste à sélectionner l'emplacement des macros modules du processeur. Un bon choix sur le placement des macros permet d'améliorer les performances en limitant les délais d'interconnexions. Il faut cependant éviter la congestion pour le routage en plaçant une trop grande quantité d'interconnexion sur une petite surface. La figure 3-2 démontre l'emplacement des macros du processeur. Le processeur comprend trois types de macro: les mémoires, les M/A et la logique diverse. Après le placement des macros, le logiciel effectue le placement des cellules standards pour chaque macro. Les cellules standards sont placées une à la suite de l'autre en rangée. L'espacement entre les rangées peut être automatique ou contrôlé par le logiciel. Cet espace est utilisé comme canal par le routage.

L'étape suivante est le routage du circuit. Le logiciel utilise les canaux d'interconnexions laissés à sa disposition pour réaliser les connexions entre les différentes cellules standards.

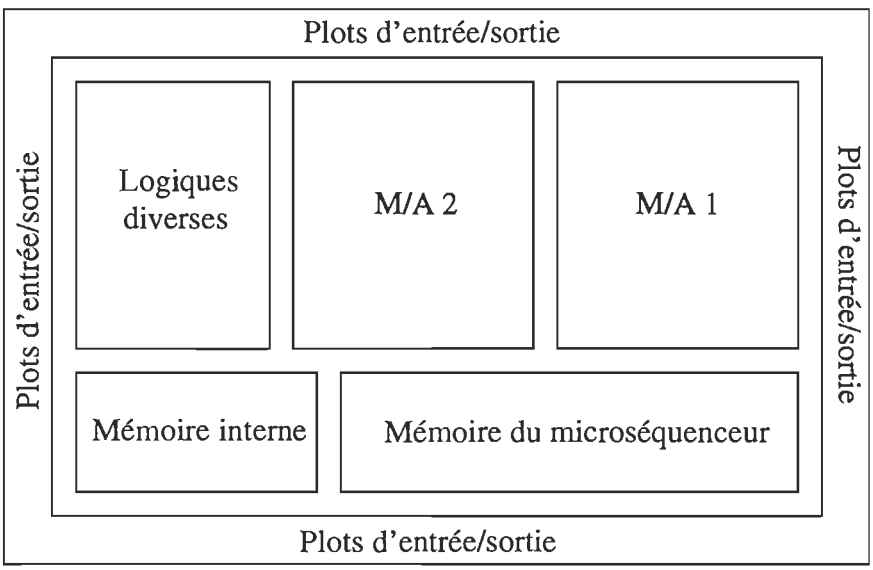

Figure 3-10 Placement des macros.

Si le logiciel ne réussit pas le routage du premier coup, on peut modifier les surfaces de placements pour ajuster les canaux d'interconnexion.

Une fois le routage réussi, le circuit est vérifié pour détecter les violations des règles de dessin ("DRC"). Le DRC utilise les règles de dessin de la technologie CMOS4S. Les violations détectées doivent être corrigées pour assurer la fiabilité du procédé de fabrication.

La vérification du fonctionnement du circuit après routage est réalisée avec le logiciel PDcompare. Le logiciel analyse les masques du circuit pour en extraire les transistors. Une représentation du circuit au niveau des transistors est ainsi créée. Les schématiques du processeur sont ensuite analysés jusqu'au niveau du transistor pour extraire la représentation. Une comparaison entre les deux circuits extraits permet de valider l'intégrité des étapes de placement et routage.

Le format de fichier de transfert pour la fonderie ("tapeout") utilisé par la SCM est le format "Caltech Intermediate Form" (CIF). La dernière étape est de générer ce fichier qui est ensuite transmis par courrier électronique à la SCM.

Le circuit intégré est fabriqué avec le procédé CMOS4S de 1.2 micron par la Société Canadienne de Microélectronique (ICAQT93A). La surface de silicium requise pour la fabrication est de 42.8 mm^2.

3.5.4 Description du boîtier

Après la fabrication sur silicium, le circuit intégré est encapsulé dans un boîtier de 68

broches du type PGA (Pin Grid Array). La figure 3-2 représente une vue du dessous du boîtier avec une numérotation matricielle des broches. Pour faciliter la tâche du personnel de la SCM l'emplacement des entrées/sorties du boîtier du processeur (le "bonding") est laissé à leur discrétion. La table 3-6 montre le lien entre les broches du boîtier et les plots du circuit intégré.

Les plots de sortie du processeur ont une capacité en courant de 8 mA. Cette capacité permet d'augmenter les fréquences d'opérations en permettant de charger et de décharger plus rapidement la charge capacitive à la sortie de la broche. La diminution des temps de transition sur les sorties cause cependant des oscillations sur les masses et les alimentations. Les oscillations peuvent causer un mauvais fonctionnement du circuit dans des conditions critiques.

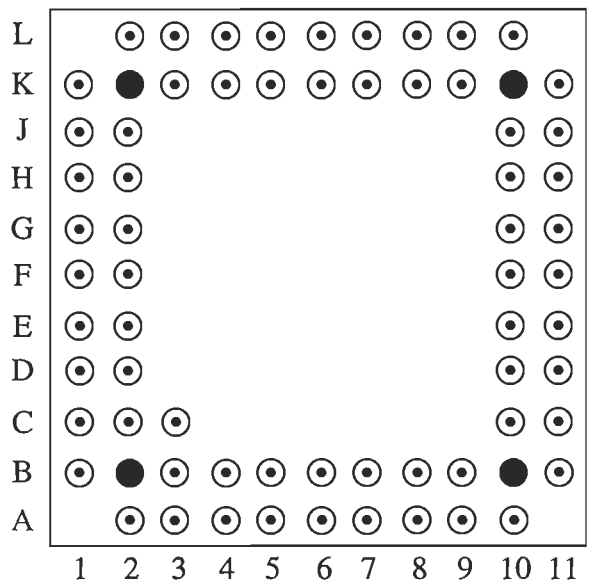

Figure 3-11 Vue du dessous du boîtier de 68 broches.

Nom	Broche	Nom	Broche	Nom	Broche
phase 2	B2	intr	K5	gnd core	E10
phase 1	B1	intra	L5	s_in	E11
reset	C2	gnd core	K6	tm	D10
vdd ring	C1	vdd core	L6	s_out	D11
gnd ring	D2	sc	K7	sm	C10
bus_ext[7]	D1	cs_ad	L7	vdd ring	C11
bus_ext[6]	E2	oe	K8	gnd ring	B11
bus_ext[5]	E1	eoc	K10	gcs	A9
bus_ext[4]	F2	dv	K11	cs[4]	B8
bus_ext[3]	F1	vdd ring	J10	cs[3]	A8
bus_ext[2]	G2	gnd ring	J11	cs[2]	B7
bus_ext[1]	G1	pgm_ext[4]	H10	cs[1]	A7
bus_ext[0]	H2	pgm_ext[3]	H11	gnd core	B6
gnd ring	H1	pgm_ext[2]	G10	vdd core	A6
vdd ring	J2	pgm_ext[1]	G11	r/$\overline{w}$ ext	B5
holda	J1	pgm_ext[0]	F10		

Table 3-6 Emplacement des I/O sur le boîtier.

Nom	Broche	Nom	Broche	Nom	Broche
hold	K1	vdd core	F11		

Table 3-6 Emplacement des I/O sur le boîtier.

Une des solutions utilisée dans le processeur consiste à séparer les masses et les alimentations du circuit de celles utilisées par les plots. Les oscillations causées par la commutation des sorties sont ainsi limitées au I/O du circuit. Le routage interne du circuit insère quatre cercles indépendants sur lesquels sont distribuées les masses et les alimentations de la logique et des I/O. La table 3-6 indique le type des masses et des alimentations.

L'autre solution consiste à utiliser un nombre suffisant de plots de masses et d'alimentations. Le guide d'utilisation de la technologie CMOS4S employé pour le processeur recommande d'utiliser pour les I/O un ensemble de masses et d'alimentations de chaque côté d'un bus de sortie de 8 bits utilisant des plots de 8 mA. Pour la logique interne, on recommande un ensemble de masses et d'alimentations pour chaque tranche de 1500 portes logiques.

3.5.5 Description des entrées/sorties du circuit intégré

On dénombre quatre catégories de signaux d'entrées/sorties sur le processeur: l'interface externe, l'interface de contrôle, l'interface analogue/digitale et l'interface de testabilité. L'interface externe comprend les signaux utiles pour la communication avec le processeur maître: $\overline{\text{intr}}$, $\overline{\text{intra}}$, $\overline{\text{hold}}$, $\overline{\text{holda}}$, bus_ext, $\overline{\text{gcs}}$, cs, pgm_ext et r/$\overline{\text{w}}$ ext. L'interface de contrôle comprend les signaux utiles pour la synchronisation du circuit: $\overline{\text{reset}}$, phase 1 et phase 2.

L'interface analogue/digitale comprend les signaux utiles pour le contrôle du convertisseur analogue/digital: sc, cs_ad, oe, eoc et dv. L'interface de testabilité comprend les signaux utiles pour la chaîne de balayage: s_in, s_out, sm et tm. La description des signaux d'entrées/sorties de chaque interfaces se trouve à la table 3-7.

Nom	Type	Description
phase 1	I	phase 1 de l'horloge à deux phases sans recouvrement
phase 2	I	phase 2 de l'horloge à deux phases sans recouvrement
reset	I	reset du processeur
intr	O	demande d'interruption
intra	I	interruption accordé
hold	I	demande d'accès
holda	O	accès accordé
bus_ext[7:0]	I/O	bus d'entrée/sortie
gcs	I	sélection du processeur
cs[4:1]	I	sélection du banc de mémoire interne. Voir Table XXX
pgm_ext[4:0]	I	addresse de la mémoire interne
r/$\overline{w}$ ext	I	lecture/écriture
sc	I	début de conversion du convertisseur A/D

Table 3-7 Description des broches externes du processeur

Nom	Type	Description
cs_ad	O	sélection du convertisseur A/D
oe	O	lecture externe du convertisseur A/D
eoc	I	fin de conversion du convertisseur A/D
dv	O	écriture au convertisseur A/D
s_in	I	entrée de la chaîne de balayage
tm	I	mode test
s_out	O	sortie de la chaîne de balayage
sm	I	mode scan

Table 3-7 Description des broches externes du processeur

CHAPITRE 4

SIMULATION DE L'ARCHITECTURE DSP_KAL+

La simulation fonctionnelle de toutes les possibilités de programmation du microséquenceur du co-processeur spécialisé est trop complexe pour être envisageable. En effet, la largeur de mot du microséquenceur est de 64 bits, il en résulte que le test de toutes les configurations possibles pour ne compter que les possibilités combinatoires, requiert 2^{64} instructions. Une deuxième option pour le test fonctionnel est choisie. Elle consiste à programmer et exécuter un certain nombre d'algorithmes de base sur le co-processeur. Cependant, la mémoire de 32 mots du microséquenceur est trop petite pour contenir l'algorithme de Kalman au complet. Ceci nous incite à décomposer l'algorithme de Kalman défini aux équations (2.9) à (2.12) en opérations de bases programmables sur le co-processeur. Cet algorithme se décompose en quatre opérations vectorielles de base: multiplication de deux vecteurs, multiplication d'un vecteur par un scalaire, addition de deux vecteurs et décalage d'un vecteur. On retrouve les mêmes opérateurs vectoriels dans le programme de reconstitution pour le DSP56000 à l'annexe A. L'opération vectorielle de décalage d'un vecteur est redondante avec les trois autres, elle n'est pas considérée comme un test essentiel. Par conséquent, il est donc possible de valider la fonctionnalité du co-processeur dans l'exécution de l'algorithme avec seulement les trois programmes de test énoncés.

Ce chapitre présente les résultats de simulations du co-processeur. Ces simulations consistent à réaliser les étapes suivantes:

- Charger la mémoire du microséquenceur,

- Charger les bancs de mémoires A, B et C,

- Exécuter le programme du microséquenceur.

4.1 Chargement des instructions du microséquenceur.

La première étape pour la mise en fonction du co-processeur consiste à charger la mémoire du microséquenceur. Elle contient les microcodes qui contrôlent et exécutent le séquencement des opérations du processeur. La programmation de cette mémoire peut être effectuée à n'importe quel moment mais elle perturbe le fonctionnement du co-processeur. La broche externe $\overline{gcs}$ permet de remplacer l'adresse générée par le compteur de programme par l'adresse qui provient directement des broches d'adresse externe. Les broches externes cs1 à cs4 sélectionnent la mémoire interne sur laquelle est appliquée le signal d'écriture/lecture externe $\overline{w}$/r. Une donnée de 8 bits arrive du port externe bidirectionnel bus_ext au bus de données de la mémoire par un tampon à haute impédance propre à chaque banc de mémoire interne. Ce tampon à haute impédance est nécessaire puisque le bus de données de la mémoire est bidirectionnel. La figure 4-1 représente la programmation de la mémoire D du microséquenceur. Le bus adr_buf<4:0> représente les broches d'adresse externe. Le bus bus_pgm<7:0> représente le port de données bidirectionnels externes utilisé pour le chargement des données. Le signal rw_bloc_d contrôle l'écriture de la mémoire, il est actif bas et

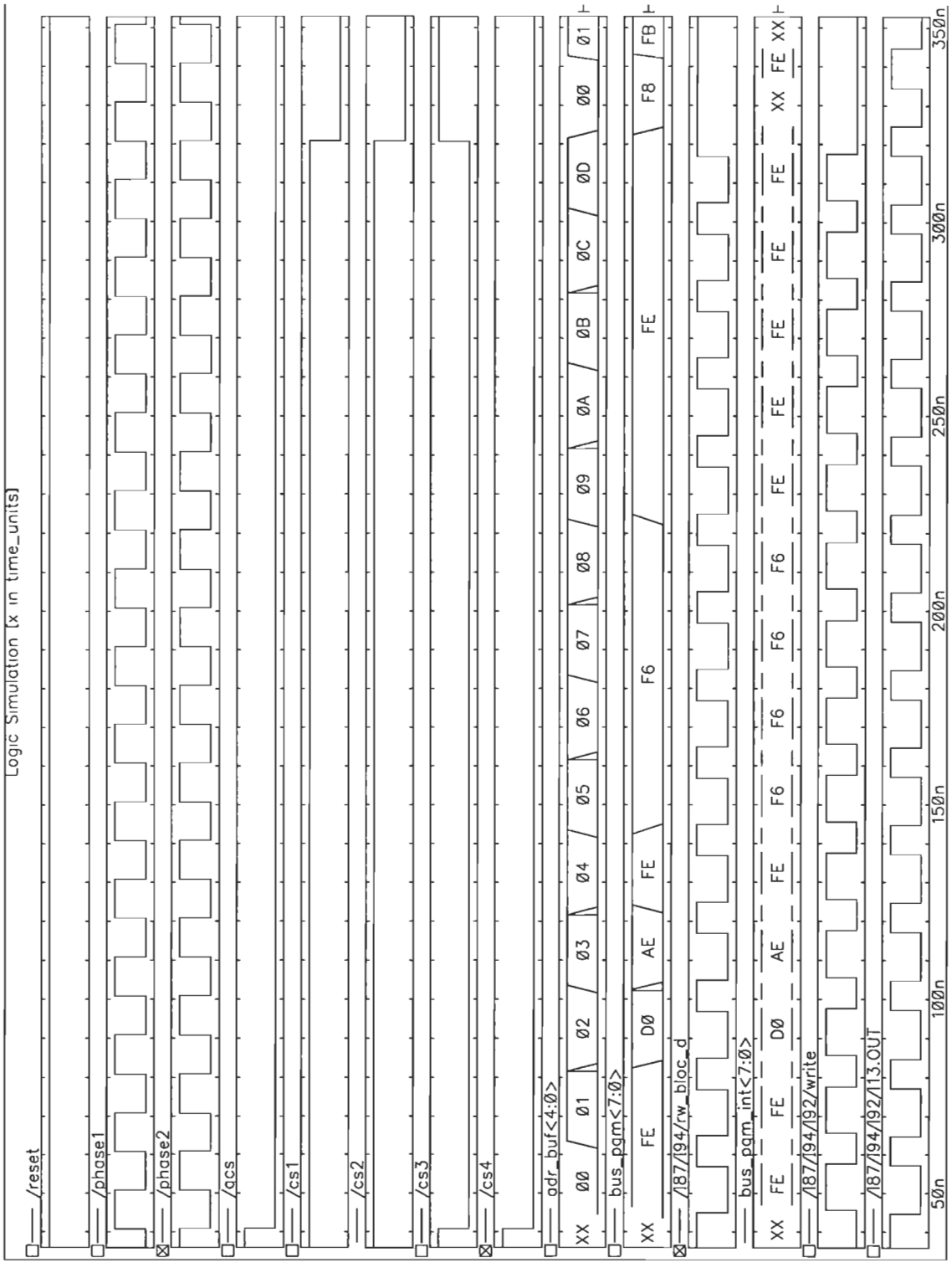

Figure 4-1 Chargement des instructions du microséquenceur.

est le résultat de la fonction logique ET entre le signal d'écriture/lecture externe $\overline{w}/r$ et la sélection du banc de mémoires.

4.2 Chargement des données vectorielles.

Il est important de se rappeler que l'accès aux trois bancs de mémoires internes se fait avec la participation du microséquenceur pour éviter des problèmes de contentions sur les mémoires. Les programmes de test vectoriels font intervenir les deux instructions suivantes pour régir l'accès aux mémoires:

#0 IF hold ≠ 0 THEN pc <= 2;

#1 holda <= 0; IF hold = 0 THEN pc <= 1;

La première instruction vérifie si le processeur hôte demande un arrêt pour la programmation des mémoires internes ($\overline{hold} = 0$). S'il n'y a pas de requête d'interruption le processeur continue l'exécution du programme à l'adresse 3 autrement la deuxième instruction est exécutée. La sortie $\overline{holda}$ est mise à zéro pour indiquer au processeur hôte que la demande est accordée et le processeur boucle sur la même instruction jusqu'à la fin de la requête ($\overline{hold} = 1$). Durant cette période le processeur se met en mode d'accès externe. La sélection de la mémoire interne sur laquelle le chargement de données est exécuté est faite à l'aide des broches externes cs1 à cs4. La table 3-2 indique le banc de mémoire sélectionné en fonction des broches externes cs1 à cs4. La sortie de l'unité de génération d'adresse du banc de mémoire sélectionné est remplacée par la donnée provenant directement des broches d'adresses externes. La sortie du multiplexeur de bus de données propre à la mémoire

interne sélectionnée provient du port de données externes bidirectionnels bus_ext.

La figure 4-1 représente la programmation de la mémoire interne A (cs1, cs2, cs3 et cs4 sont tous à 0). Le bus ptr_a représente la sortie du générateur d'adresses de la mémoire A. Le bus bus_1a représente la sortie du multiplexeur de bus de la mémoire A. Les multiplexeurs de bus ont une sortie avec haute impédance, ils ne sont actifs que durant une écriture à la mémoire. Le signal rw_bloca contrôle l'écriture de la mémoire, il est actif bas et est le résultat de la fonction logique ET entre le signal d'écriture/lecture externe $\overline{w}$/r et la sélection du banc de mémoire. La figure 3-9 représente le schéma de l'interface mémoire.

4.3 Simulation des opérations vectorielles.

4.3.1 Addition de deux vecteurs

Ce programme additionne le vecteur contenu dans la mémoire A avec celui de la mémoire B et place le résultat dans la mémoire C. La table 4-1 contient le microcode du programme d'addition de deux vecteurs. Les instructions aux adresses programme 0 et 1 permettent l'accès au trois bancs de mémoires internes. Les instructions aux adresses 2 et 3 initialisent les unités de génération d'adresses. L'instruction à l'adresse 4 programme le compteur pour fixer la longueur des vecteurs. L'instruction de l'adresse 5 constitue le corps du programme. Elle exécute 6 opérations en parallèle. Cette instruction fonctionne comme suit: le contenu de la mémoire A est additionné au contenu de la mémoire B, le résultat est placé dans la mémoire C. Le contenu des mémoires A, B et C est adressé par leur pointeur 1 respectif; les pointeurs 1 des unités de générations d'adresses A, B et C sont incrémentés des

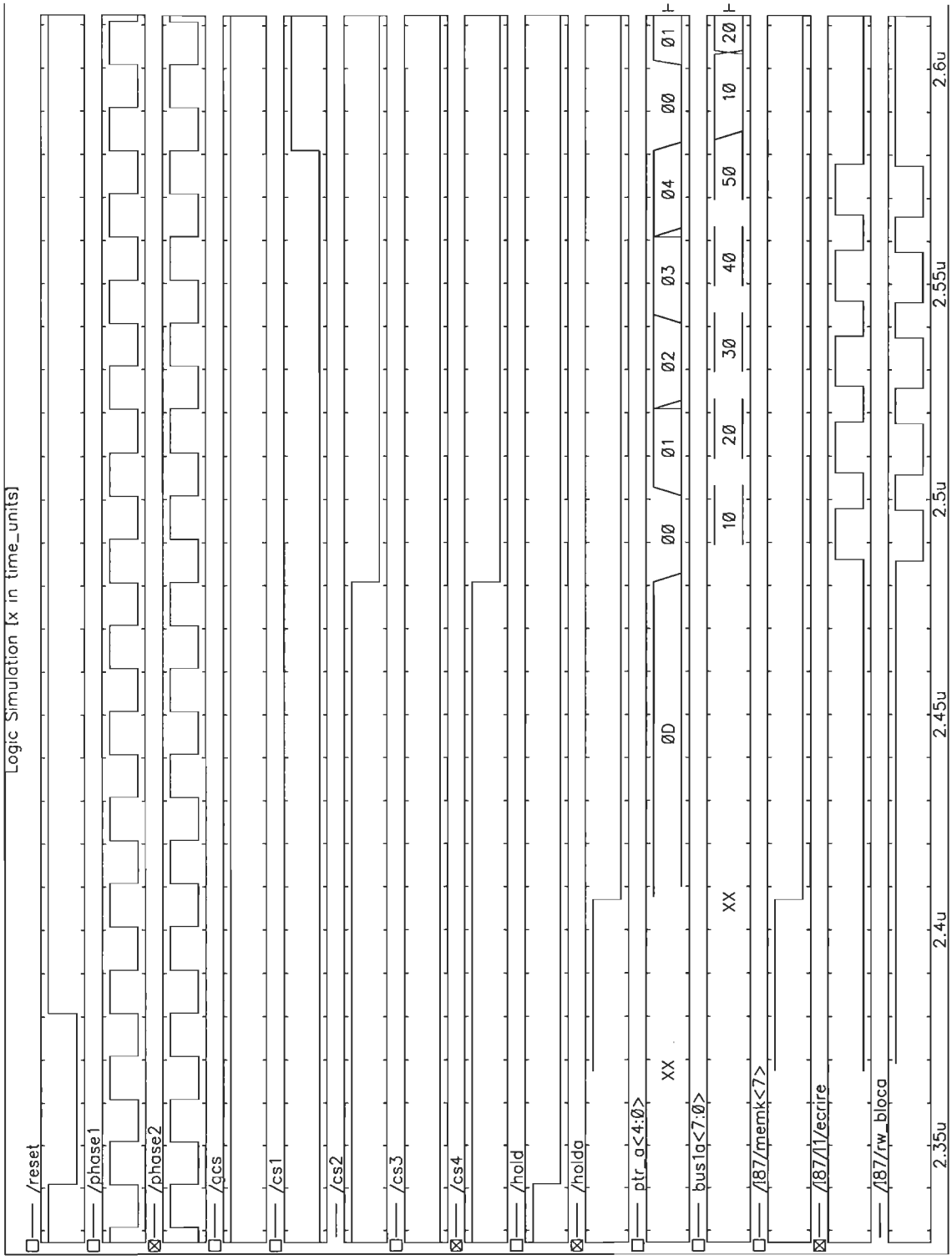

Figure 4-2 Chargement des données vectorielles.

valeurs contenues dans les registres de déplacement respectifs; le compteur de programme est décrémenté de 1 et le résultat est comparé avec 0; si le résultat est différent de zéro, le compteur de programme est remplacé par l'adresse 5. Le programme boucle sur place jusqu'à la fin de l'addition vectorielle. La table 4-2 contient le microcode binaire utilisé pour

Compteur programme	Description
0	IF hold ≠ 0 THEN pc <= 2;
1	holda <= 0; IF hold = 0 THEN pc <= 1;

2	reg_base_A_1 <= 1F;
	reg_base_B_1 <= 1F;
	reg_base_B_1 <= 1F;

3	reg_dep_A_1 <= 01;
	reg_dep_B_1 <= 01;
	reg_dep_C_1 <= 01;

4	compt_ech <= 04;

5	mem_C(reg_base_C_1 + reg_dep_C_1) <=
	mem_A(reg_base_A_1 + reg_dep_A_1) +
	mem_B(reg_base_B_1 + reg_dep_B_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	reg_base_C_1 <= reg_base_C_1 + reg_dep_C_1;
	IF compt_ech ≠ 0 THEN pc <= 5;

6	pc <= 0;

Table 4-1 Microcode pour l'addition de deux vecteurs

programmer le microséquenceur. La table 4-3 présente le contenu des mémoires vectorielles avant l'exécution du programme.

Add.	mem D	mem E	mem F	mem G	mem H	mem I	mem J	mem K
0	fe	f8	e3	0f	d0	78	5e	fc
1	fe	fb	ef	3f	d0	7a	3c	7c
2	d0	40	03	0d	50	3f	fe	ff
3	ae	b8	e2	0a	50	6f	3e	fc
4	fe	f8	e3	0f	50	ff	9e	fc
5	f6	d8	63	37	64	f8	be	bc
6	fe	f8	e3	0f	50	7b	1e	fc

Table 4-2 Microcode binaire pour l'addition de deux vecteurs

Add.	mem A	mem B	mem C
0	1	1	1
1	2	2	2
2	3	3	3
3	4	4	4
4	5	5	5

Table 4-3 Mémoire interne pour l'addition de deux vecteurs

La figure 4-1 présente une simulation du programme sur le simulateur SILOS. Les signaux ptr_a, ptr_b et ptr_c représentent la sortie des unités de génération d'adresse A, B et C respectivement. Les signaux bus1, bus2 et bus3 représentent les bus de données des mémoires A, B et C respectivement. Le signal "ecrire" représente le signal d'écriture de la mémoire C, il est actif haut. Le signal adr_buf représente la prochaine adresse du compteur de programme. Les vecteurs contenus dans les mémoires A et B sont d'une longueur de 4 mots. Pour analyser la simulation, on se réfère au front descendant de "ecrire". Au premier résultat, la sortie de la mémoire A et B est 01H. Le résultat de l'addition est 02H et on le retrouve sur le signal bus3. Ainsi de suite pour les trois autres additions.

4.3.2 Multiplication de deux vecteurs

Le programme de multiplication de deux vecteurs utilise les deux M/A internes pour accélérer les calculs. Le M/A A reçoit les éléments impairs des vecteurs contenus dans les mémoires A et B. Le M/A B reçoit les éléments pairs de ces mêmes vecteurs. Les résultats des deux M/A sont ensuite additionnés pour obtenir le résultat final. Ce résultat est placé dans la mémoire C pour ensuite être transmi à l'extérieur du processeur par le bus de données bidirectionnels bus_ext.

La table 4-4 contient le microcode du programme de multiplication de deux vecteurs. Les instructions 0 à 4 sont identiques à celles que l'on retrouve pour l'addition de vecteurs. Les instructions 5, 6, 7 et 8 représentent le corps du programme. L'instruction 5 charge le M/A 1 avec le contenu impair des mémoires A et B adressées par leur pointeur respectif. Les pointeurs de mémoires sont incrémentés et le compteur du nombre d'éléments des vecteurs

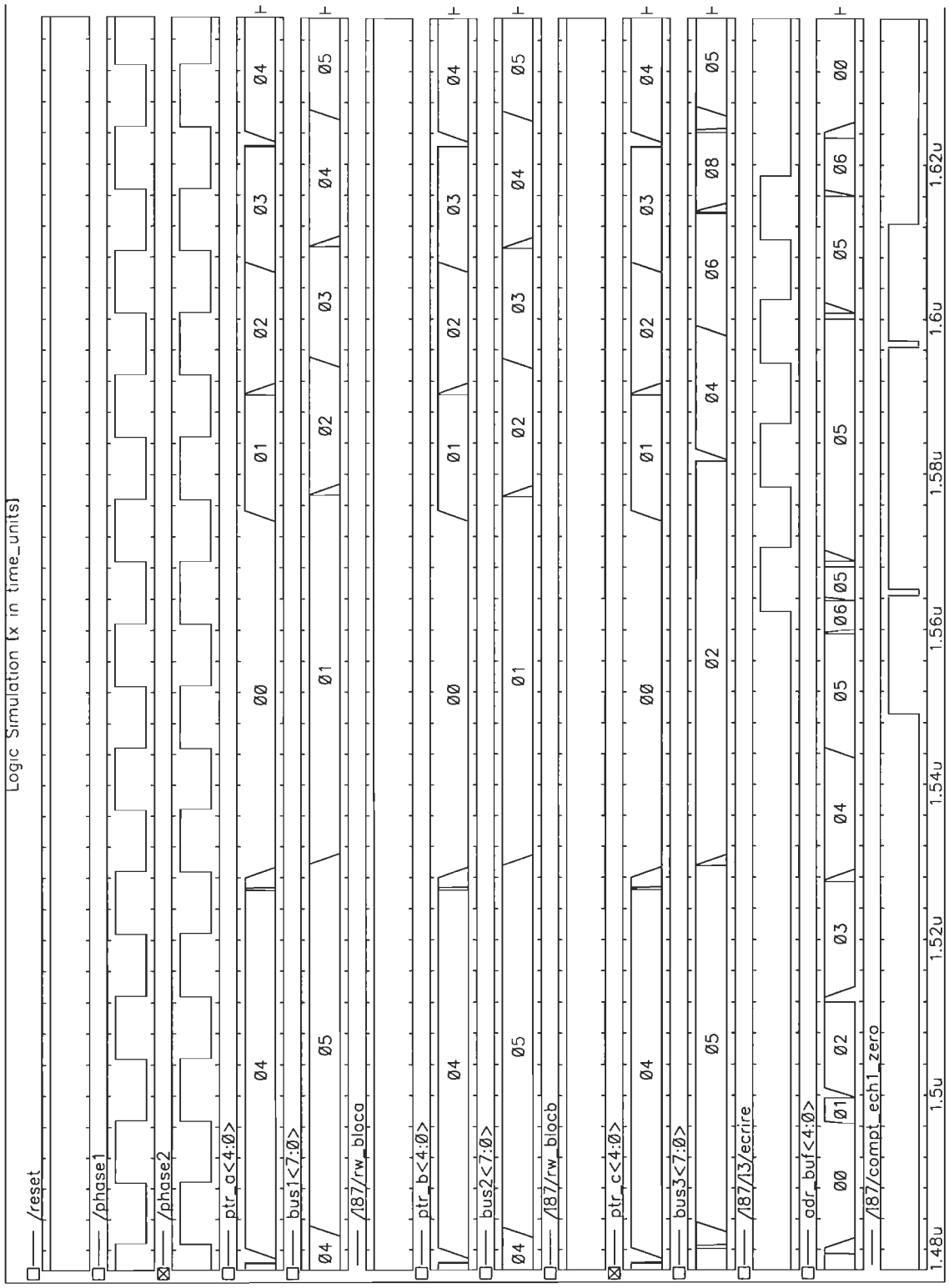

Figure 4-3 Addition de deux vecteurs.

est décrémenté. L'instruction 6 effectue les mêmes opérations que l'instruction 5, la seule différence est que le contenu vectoriel pair est dirigé vers le M/A 2. Les instructions 7 et 8 répètent les instructions 5 et 6 auxquelles s'ajoutent une instruction pour l'accumulation des multiplications et une autre instruction pour la comparaison du compteur avec zéro pour déterminer à quel moment un branchement doit être effectué. En mode accumulation, le contenu précédent est additionné avec le résultat de la multiplication. Le contenu de l'accumulateur doit être valide avant de programmer le mode accumulation. L'instruction des adresses 9 et A additionne le contenu des deux M/A pour obtenir le résultat. L'instruction de l'adresse B effectue une demande d'interruption externe. Elle boucle sur place en attente d'une réponse du processeur hôte via le signal $\overline{\text{intra}}$. Les instructions aux adresses C et D sont identiques. Le délai de propagation externe est trop élevé pour un seul cycle. Au lieu d'allonger la période d'horloge, la même instruction est utilisée deux fois.

Compteur programme	Description
0	IF hold ≠ 0 THEN pc <= 2;
1	holda <= 0; IF hold = 0 THEN pc <= 1;

2	reg_base_A_1 <= 1F;
	reg_base_B_1 <= 1F;
	reg_base_B_1 <= 1F;

3	reg_dep_A_1 <= 01;
	reg_dep_B_1 <= 01;
	reg_dep_C_1 <= 01;

Table 4-4 Microcode pour la multiplication de deux vecteurs

Compteur programme	Description
4	compt_ech <= 04;

5	MA1_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA1_B <= mem_B(reg_base_B_1 + reg_dep_B_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;

6	MA2_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA2_B <= mem_B(reg_base_B_1 + reg_dep_B_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;

7	MA1_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA1_B <= mem_B(reg_base_B_1 + reg_dep_B_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	som_ma1 <= som_ma1 + ma1;
	IF compt_ech = 0 THEN pc <= 9;

8	MA2_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA2_B <= mem_B(reg_base_B_1 + reg_dep_B_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	som_ma2 <= som_ma2 + ma2;
	IF compt_ech ≠ 0 THEN pc <= 7;

9	mem_A(reg_base_A_1 + reg_dep_A_1) <= som_ma1;
A	mem_C(reg_base_C_1 + reg_dep_C_1) <= som_ma2 + mem_A(reg_base_A_1 + reg_dep_A_1) ;

Table 4-4 Microcode pour la multiplication de deux vecteurs

Compteur programme	Description
B	intr <= 0; IF intra ≠ 0 THEN pc <= B;
C	bus_ext <= mem_C(reg_base_C_1 + reg_dep_C_1);
D	bus_ext <= mem_C(reg_base_C_1 + reg_dep_C_1);
E	pc <= 0;

Table 4-4 Microcode pour la multiplication de deux vecteurs

La table 4-5 contient le microcode binaire utilisé pour programmer le microséquenceur. La table 4-6 présente le contenu des mémoires vectorielles avant l'exécution du programme.

Add.	mem D	mem E	mem F	mem G	mem H	mem I	mem J	mem K
0	fe	f8	e3	0f	d0	78	5e	fc
1	fe	fb	ef	3f	d0	7a	3c	3c
2	d0	40	03	0d	50	3f	fe	ff
3	ae	b8	e2	0a	50	6f	3e	fc
4	fe	f8	e3	0f	50	ff	9e	fc
5	f6	d8	e3	0f	61	f3	3e	dc
6	f6	d8	e3	0f	61	d8	fe	fc

Table 4-5 Microcode binaire pour la multiplication de deux vecteurs

Add.	mem D	mem E	mem F	mem G	mem H	mem I	mem J	mem K
7	f6	d8	e3	0f	61	f3	fe	d9
8	f6	d8	e3	0f	61	d8	be	f4
9	7e	fa	e3	0f	50	6f	1e	fc
a	fe	f8	e3	37	5c	3f	1e	bc
b	fe	f8	e3	0f	f0	78	7e	ed
c	fe	f8	e3	0f	58	7f	1e	ec
d	fe	f8	e3	0f	58	7f	1e	ec
e	fe	f8	e3	0f	50	7b	1e	fc

Table 4-5 Microcode binaire pour la multiplication de deux vecteurs

Add.	mem A	mem B	mem C
0	10	10	10
1	20	20	20
2	30	30	30
3	40	40	40
4	50	50	50

Table 4-6 Mémoire interne pour la multiplication de deux vecteurs

La figure 4-1 présente une simulation du programme sur le simulateur SILOS. Les signaux sont presque tous identiques à ceux décrits dans l'addition de vecteur. On peut remarquer les symboles XX à gauche de la figure indiquant une indétermination causée par l'initialisation des unités de générations d'adresses à leur première utilisation. Les deux vecteurs à multiplier sont identiques. Ils contiennent les valeurs 10H, 20H, 30H et 40H. Les données impaires des vecteurs sont traitées par le M/A 1 et le résultat est écrit dans la mémoire A sur le front montant du signal rw_bloca. La donnée présente à l'entrée de la mémoire A est 0AH ce qui est identique au résultat de 10H*10H + 30H*30H. Le résultat du M/A 2 est additionné avec celui qui vient d'être placé dans la mémoire A. Le résultat de l'addition est écrit dans la mémoire C sur le front montant du signal rw_blocc. La donnée présente à l'entrée de la mémoire est 1EH ce qui est identique au résultat de 10H*10H + 20H*20H + 30H*30H + 40H*40H.

4.3.3 Multiplication d'un vecteur par des scalaires.

Ce programme multiplie un vecteur situé dans la mémoire A avec des scalaires situés dans la mémoire C. Le programme utilise les deux multiplicateurs internes pour accélérer les calculs. Le multiplicateur A reçoit les éléments impairs du vecteur contenus dans la mémoire A et le multiplicateur B reçoit les éléments pairs du même vecteur.

La table 4-7 contient le microcode du programme. Les instructions des adresses 0 à 4 sont identiques à celles que l'on retrouve pour l'addition de deux vecteurs. Les instructions aux adresses 5 et 6 ne sont exécutées qu'une seule fois pour remplir le registre pipeline de sortie des multiplicateurs 1 et 2. Les multiplicateurs 1 et 2 reçoivent les données impairs et

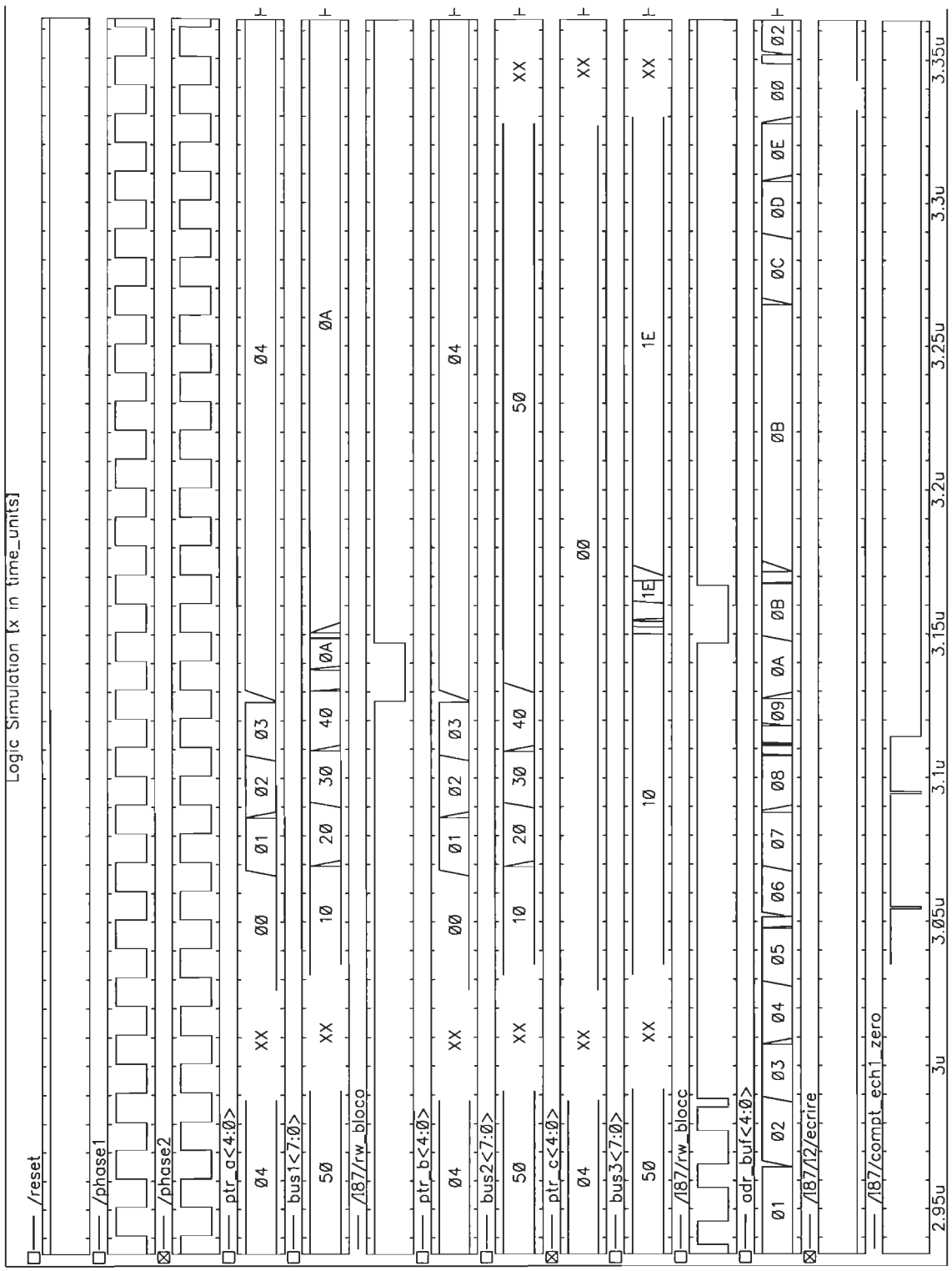

Figure 4-4 Multiplication de deux vecteurs.

pair respectivement. Les instructions aux adresses 7 et 8 sont le corps du programme. Elles effectuent les mêmes opérations de base que les instructions aux adresses 5 et 6. De plus, comme le pipeline des multiplicateurs est remplie au moment du chargement de nouvelles données, le résultat de l'opération précédente est transféré dans la mémoire B. Une comparaison du compteur avec zéro permet de déterminer quand un branchement doit être effectué. Les instructions des adresses 9 et A, ou B et C servent à vider les registres pipelines d'entrées des multiplicateurs.

Compteur programme	Description
0	IF hold ≠ 0 THEN pc <= 2;
1	holda <= 0; IF hold = 0 THEN pc <= 1;

2	reg_base_A_1 <= 1F;
	reg_base_B_1 <= 1F;
	reg_base_B_1 <= 1F;

3	reg_dep_A_1 <= 01;
	reg_dep_B_1 <= 01;
	reg_dep_C_1 <= 01;

4	compt_ech <= 04;

5	MA1_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA1_C <= mem_C(reg_base_C_1 + reg_dep_C_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_C_1 <= reg_base_C_1 + reg_dep_C_1;

Table 4-7 Microcode pour la multiplication d'un vecteur par des scalaires

Compteur programme	Description

6	MA2_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA2_C <= mem_C(reg_base_C_1 + reg_dep_C_1);
	compt_ech <= compt_ech - 1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_C_1 <= reg_base_C_1 + reg_dep_C_1;

7	MA1_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA1_C <= mem_C(reg_base_C_1 + reg_dep_C_1);
	compt_ech <= compt_ech - 1;
	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA1;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	reg_base_C_1 <= reg_base_C_1 + reg_dep_C_1;
	IF compt_ech = 0 THEN pc <= B;

8	MA2_A <= mem_A(reg_base_A_1 + reg_dep_A_1);
	MA2_C <= mem_C(reg_base_C_1 + reg_dep_C_1);
	compt_ech <= compt_ech - 1;
	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA2;
	reg_base_A_1 <= reg_base_A_1 + reg_dep_A_1;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	reg_base_C_1 <= reg_base_C_1 + reg_dep_C_1;
	IF compt_ech ≠ 0 THEN pc <= 7;

9	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA1; reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;

A	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA2;
	reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
	pc <= D;

B	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA2; reg_base_B_1 <= reg_base_B_1 + reg_dep_B_1;
C	mem_B(reg_base_B_1 + reg_dep_B_1) <= MA1;

Table 4-7 Microcode pour la multiplication d'un vecteur par des scalaires

Compteur programme	Description
D	pc <= 0;

Table 4-7 Microcode pour la multiplication d'un vecteur par des scalaires

La figure 4-1 présente une simulation du programme sur simulateur SILOS. Les signaux sont presque tous identiques à ceux décrits dans l'addition de deux vecteurs. Le vecteur à multiplier et les scalaires sont identiques. Ils contiennent les valeurs 10H, 20H, 30H et 40H. Le résultat de la multiplication est écrit dans la mémoire B sur le front descendant du signal "ecrire". La donnée présente à l'entrée de la mémoire B est en retard de deux cycles sur les entrées provenant des mémoires A et C. On trouve comme résultat sur le bus de données bus2 les valeurs 01H, 04H, 09H et 10H. Ceci correspond aux 8 bits les plus significatifs des multiplications 10H*10H, 20H*20H, 30H*30H et 40H*40H. Il ne faut pas oublier que le résultat des M/A est tronqué aux 8 bits les plus significatifs à la sortie.

La table 4-8 contient le microcode binaire utilisé pour programmer le microséquenceur. La table 4-9 présente le contenu des mémoires vectorielles avant l'exécution du programme.

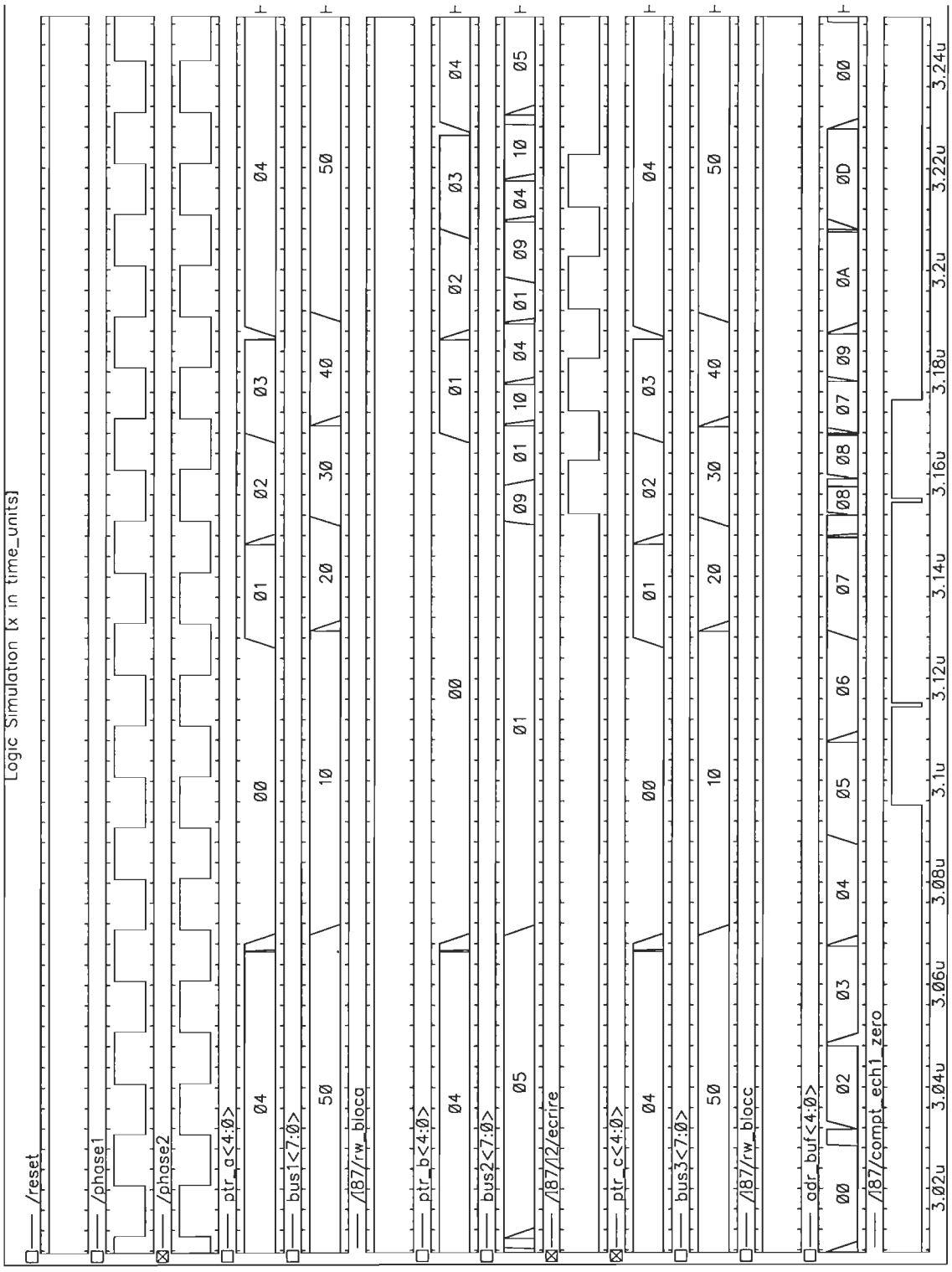

Figure 4-5 Multiplication d'un vecteur par un scalaire.

Add.	mem D	mem E	mem F	mem G	mem H	mem I	mem J	mem K
0	fe	f8	e3	0f	d0	78	5e	fc
1	fe	fb	ef	3f	d0	7a	3c	3c
2	d0	40	03	0d	50	3f	fe	ff
3	ae	b8	e2	0a	50	6f	3e	fc
4	fe	f8	e3	0f	50	ff	9e	fc
5	f6	f8	63	8f	40	f7	1e	dc
6	f6	f8	63	8f	40	df	1e	fc
7	f6	d8	69	8f	60	e2	7e	dc
8	f6	d8	69	8f	60	98	fe	fc
9	fe	d8	e9	8f	50	6f	fe	ff
a	fe	d8	e9	8f	50	3b	be	fd
b	fe	d8	e9	8f	50	3f	fe	ff
c	fe	f8	e9	8f	50	6f	fe	ff
d	fe	f8	e3	0f	50	7b	1e	fc

Table 4-8 Microcode binaire pour la multiplication d'un vecteur par un scalaire

Add.	mem A	mem B	mem C
0	10	1	10
1	20	2	20
2	30	3	30
3	40	4	40
4	50	5	50

Table 4-9 Mémoire interne pour la multiplication d'un vecteur par un scalaire

4.4 Synthèse des résultats

Pour évaluer les améliorations de l'architecture proposées par rapport aux processeurs de traitement de signaux commerciaux, nous avons programmé quelques fonctions de calcul vectoriel, ainsi que l'algorithme du filtre de Kalman selon les équations (2.9) à (2.12) pour effectuer la reconstitution sur le DSP56000. Ces mêmes fonctions ont été simulées sur le processeur spécialisé original DSP_KAL et sur le processeur intégré DSP_KAL+. Nous avons ensuite comparé les performances en fonction du nombre de cycles requis pour exécuter les programmes. La comparaison est basée sur les hypothèses que la durée d'un cycle d'horloge est égale pour les deux processeurs, que le pipeline du DSP56000 est plein, que les données du DSP56000 sont dans les mémoires internes, que le temps d'accès et le temps de calcul des M/A sont inférieurs ou égaux à la durée de deux cycles d'horloge.

Les résultats théoriques du taux minimum de réduction du nombre de cycles inclus à la table 4-10 démontrent l'amélioration découlant d'une architecture adaptée à l'algorithme. Pour une reconstitution sans contraintes, le taux minimum de réduction du nombre de cycles est de 88% alors que pour la reconstitution avec contraintes, ce même taux minimum est de 80%. L'écart est causé par l'opération de comparaison qui ne peut être exécutée en parallèle. Dans le cas d'opérations vectorielles, le taux de réduction atteint 84% pour l'addition de deux vecteurs. L'écart provient du fait que le DSP56000 ne possède que deux mémoires internes et que le résultat de l'addition doit être placé temporairement dans un registre avant d'être écrit en mémoire. L'opération est alors réalisée en plusieurs cycles. Le taux de réduction pour la multiplication de deux vecteurs est significatif puisqu'il démontre que l'emploi de deux M/A permet de réduire de 50% le nombre de cycles requis. Les résultats démontrent qu'il n'existe pas d'écart entre les performances de l'architecture DSP_KAL et l'architecture DSP_KAL+ sur les opérations vectorielles.

Opération vectorielles	DSP56000	Processeur original (DSP_KAL)		Processeur intégré (DSP_KAL+)	
	Nombre de cycles	Nombre de cycles	% min. réduction	Nombre de cycles	% min. réduction
Multiplication par un scalaire	26+M*4	2+M	75%	5+M	75%
Multiplication de deux vecteurs	20+M*2	4+M	50%	5+M	50%

Table 4-10 Nombre de cycles requis pour certains calculs vectoriels de dimension M.

Opération vectorielles	DSP56000	Processeur original (DSP_KAL)		Processeur intégré (DSP_KAL+)	
	Nombre de cycles	Nombre de cycles	% min. réduction	Nombre de cycles	% min. réduction
Addition de deux vecteurs	20+M*6	2+M	84%	3+M	84%
Décalage d'un vecteur	14+M*4	2+M*2	50%	7+M*2	50%
reconstitution sans contraintes	106+M*16	10+(M-3)*2	88%	N/D	N/D
reconstitution avec contraintes	106+M*20	14+(M-3)*4	80%	N/D	N/D

Table 4-10 Nombre de cycles requis pour certains calculs vectoriels de dimension M.

CHAPITRE 5

CONCLUSION

Dans ce mémoire, nous avons présenté une architecture de processeur spécialisé pour la reconstitution de signaux basée sur le filtre de Kalman. La reconstitution de signaux est présente dans plusieurs domaines. Dès qu'il est nécessaire d'effectuer une prise de données et qu'une déformation du mesurande est obtenue durant la conversion, nous pouvons appliquer une méthode de reconstitution. Plusieurs méthodes sont disponibles pour reconstituer un signal échantillonné à la sortie d'une chaîne de conversion. La plupart des méthodes sont basées sur des algorithmes fort complexes qui exigent un volume de calcul suffisamment important pour demander un temps de réponse non négligeable de la part d'un processeur de traitement de signaux commercial tel que le DSP56000 de Motorola.

Suite à des travaux algorithmiques effectués au Laboratoire de systèmes de mesure pour répondre à des besoins en spectrométrie et chromatographie, nous avons proposé d'intégrer en technologie VLSI des algorithmes basés sur le filtre stationnaire de Kalman. Le filtre stationnaire de Kalman a pour grand avantage: de posséder un traitement en ligne qui permet d'éliminer tout stockage de données échantillonnée, de permettre l'ajout d'une contrainte de positivité dans son traitement afin d'améliorer la reconstitution, de répondre à un vaste champ d'application, d'être basé sur une équation récursive permettant un traitement parallèle [MAS94], [MAS95]. La découverte de ce parallélisme latent nous a permis

de proposer une architecture spécialisée pour la reconstitution de signaux offrant des performances significatives en comparaison au processeur de traitement de signaux DSP56000. Dans une première approche d'intégration de l'algorithme, nous avons considéré que l'état des signaux est stationnaire, ce qui nous a permis d'utiliser le filtrage de Kalman rapide. Effectivement, nous avons démontré que l'utilisation de ce processeur augmente les performances de reconstitution en temps réel, en utilisant un algorithme de calcul parallèle, qui répartit les calculs sur deux processus indépendants mais entrelacés. L'architecture du processeur est conçue pour permettre un traitement en parallèle des données. Nous avons conservé une grande flexibilité à la mise en oeuvre d'algorithmes dans l'architecture proposée grâce au microséquenceur programmable qui utilise la programmation horizontale. Cette programmation horizontale permet d'obtenir moins d'un cycle par instruction ce qui autrement dit permet l'exécution de plusieurs instructions dans un seul cycle d'horloge. Toutes les opérations arithmétiques (multiplication, addition, soustraction, comparaison) sont exécutées en un seul cycle [SAN92].

L'amélioration des performances du processeur en comparaison avec le DSP56000 a été évaluée en comparant le nombre de cycles machines requis par chacun des processeurs pour réaliser certaines fonctions. Dans le cas d'une reconstitution sans contrainte de positivité, la réduction du nombre de cycles tend vers une limite théorique inférieure de 88%, pour le cas d'une reconstitution avec contrainte de positivité, le taux de réduction tend vers une limite inférieure théorique de 80%. La diminution des performances est causée par la comparaison qui ne peut être jumelée à d'autres opérations en parallèles. Autrement dit, pour effectuer une reconstitution, l'architecture proposée permet une réduction du nombre de

cycles d'horloge qui tend à demander 5 fois moins de cycle que le DSP56000 de Motorola. De plus, nous projetons une augmentation de la fréquence d'un facteur de deux par rapport à ce même processeur.

Un prototype a été réalisé à la Société Canadienne de Microélectronique (SCM). Ce prototype a été développé pour fonctionner comme co-processeur à un processeur maître. Il est composé de deux multiplicateurs/accumulateurs (M/A) et d'un microséquenceur programmable contrôlant les unités opératives servant à exécuter l'algorithme stationnaire de Kalman avec et sans contrainte de positivité. Deux unités M/A sont utilisées afin de répondre au compromis vitesse et surface de silicium. Cependant, l'architecture permet d'utiliser plus de deux unités M/A quand une augmentation de la vitesse est justifiée pour un coût de surface silicium acceptable. La conception a été réalisée au moyen d'outils disponibles de la SCM: CADENCE, SILOS, BNR DFT, SYNFUL et un langage de description matériel de haut niveau VERILOG. Ainsi la conception peut se réaliser en utilisant différentes longueur de mots numériques pour la quantification des nombres et pour répondre à différentes technologies. Pour une réduction des coûts de fabrication, une architecture avec un chemin de données de 8-bits a été fabriqué dans une technologie CMOS de 1.2 um de la SCM.

Dans ce prototype les mémoires A, B et C sont de 32x8-bits. Les unités M/A sont basées sur l'algorithme de Booth en arithmétique entière de 8x8-bits signée. Les nombreux multiplexeurs sont conçus à l'aide d'amplificateur trois états. Une horloge à deux phases sans recouvrement a été utilisée. La complexité du chip sans les mémoires est de 24 000 transistors pour une surface de silicium de 5.1 x 9.0 mm2. Un boîtier de 68-broches de type " pin

grid array" a été choisi et 59 broches sont utilisées.

Cette version du processeur est un premier pas vers un processeur spécialisé pour la reconstitution des signaux basée sur le filtre de Kalman. D'autres améliorations sont envisagées afin de séparer les calculs vectoriels en sections et de distribuer les calculs sur des processeurs séparés.

BIBLIOGRAPHIE

[ABD92] Abderrahman, A., Achour, C., Massicotte, D., Santerre, M.-A., Processeurs spécialisés pour la reconstitution de signaux à mesurer, basé sur le filtrage de Kalman, rapport de cours en conception de circuit électronique intégré II, École Polytechnique de Montréal, rapport interne, groupe de recherche en électronique industrielle, laboratoire de systèmes de mesure, Université du Québec à Trois-Rivières, mai 1992.

[ANC86] Anceau, F., The architecture of micro-processors, Addison-Wesley, 1986.

[ANN86] Annaratone, M., Digital CMOS circuit design, Kluwer Academic Publishers, 1986.

[BAR93] Barwicz, A, Massicotte, D., Savaria, Y., Santerre, M.A., Morawski, R.Z., "An Integrated Structure for Kalman-Filter-based Measurand Reconstruction in Spectrometric Applications", IEEE Instrument and Measurement Technology Conference, Hamanatsu, Japon, 1993.

[BAR94] Barwicz, A, Massicotte, D., Savaria, Y., Santerre, M.A., Morawski, R.Z., "An Integrated Structure for Kalman-Filter-based Measurand

Reconstruction", IEEE Transaction on Instrument and Measurement, juin 94.

[DEM86] Demoment, G., Déconvolution des signaux, Rapport interne No L2S 20/84 du Laboratoires des Signaux et Systemes (CNRS/ESE), no 3086/ 85, 1986.

[DFT91] DFT, Design For Test, Reference Guide, Northern Telecom, 1991.

[FEL90] Fellman, R.D., Kaneshiro, R.T., Konstantinos, K., "Design and Evaluation of an Architecture for a Digital Signal Processor for Instrumentation Applications", IEEE Transactions on acoustics, speech, and signal processing, Vol. 30, No 3, March, 1990, pp.537-546.

[GHO95] Ghosh, D., Nandy, S. K., "Design and Realization of a 8X8 Wavepipelined Multiplier in CMOS Technology", IEEE Transactions on VLSI Systems, March 1995.

[HAY75] Hayes, John P., " Detection of Pattern-Sensitive Faults in Random-Access Memories", IEEE Transactions on Computers, Vol. C-24, No 2, Fev. 1975, pp. 150-157.

[JAN84] Jansson, P.A., Deconvolution with application in Spectroscopy, Academic Press Inc., 1984.

[MAS91] Massicotte, D., Reconstitution des signaux basée sur le filtrage de Kal-

man: implantation sur la carte DSP56000, Rapport interne, Laboratoire de systèmes de mesure, Université du Québec à Trois-Rivières, 1991, 33 pages.

[MAS92a] Massicotte, D., Morawski, R.Z., Barwicz, A., "Incorporation of a positivity constraint into a Kalman-filter-based algorithm for correction of spectrometric data", Instrumentation/Measurement Technology Conference (IMTC/92), New-York, may 1992 pp.590-593.

[MAS92b] Massicotte, D., Santerre, M.A., Savaria, Y., Barwicz, A., "Structure de calculs parallèles pour le filtrage de Kalman dans la reconstitution de signaux", Congrès canadien en génie électrique et informatique, Toronto, Ontario, septembre 1992

[MAS94] Massicotte, D., Morawski, R.Z., Barwicz, A, "Kalman-Filter-Based Algorithms of Spectrophotometric Data Correction, Part 1: Incorporation of the Positivity Constraint Into an Iterative Algorithm of Correction", soumis à IEEE Transaction on Instrument and Measurement, juin 94.

[MAS95] Massicotte, D., Une approche à l'implantation en technologie VLSI d'une classe d'algorithme de reconstitusion de signaux, Thèse de Ph.D. École Polytechnique de Montréal, Département de Génie Électrique et Génie Informatique 1995

[MOR93] Morawski, R.Z., "Unified Approach to Measurement Signal Reconstruction", invited paper, IEEE Conference record IMTC/93

[OPP75] Oppenhein, A.V., Schafer, R.W., Digital signal processing, Prentice-Hall, 1975

[REY86] Reynaud, R., Filtrage de Kalman à la déconvolution de signaux monodimentionnels: étude d'un nouvel algorithme de type Kalman rapide, expérimentations numériques, implantation dans un processeur spécialisé opérant en arithmétique entière, Thèse de doctorat d'état, Paris XI, 1986.

[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 électrique et informatique, Toronto, Ontario, septembre 1992

[SAV88] Savaria, Y., Conception et vérification des circuits VLSI, Éditions de l'École Polytechnique de Montréal, 1988.

[SYN92] SYNFUL, Reference Guide, Bell Northern Research, 1992

[WHI81] White, D., Bit-Slice Design: Controllers and ALU, Garlands Publishing Inc., 1981

ANNEXE A

Programme de reconstitution sur DSP56000

```
;*****************************************************
;       Algorithme de Kalman sur DSP56000
;
;       Auteur: Marc-Alain Santerre
;
;       Le programme a pour fonction de mesurer les performances
;       de reconstruction du DSP5600 par rapport au processeur
;       spécialisé
;
;*****************************************************
opt         mex

; Definition des macros pour les opérations vectorielles
;

mul_vect   macro       nmul,veca,vecb,result
;cette macro multiplie deux vecteurs
;result = veca(i) * vecb(i)  i=1..nmul
;nmul = longeur des vecteurs
;veca = adresse de base du vecteur a
;vecb = adresse de base du vecteur b
;result = adresse de base du vecteur result

move        #0,a
move        #veca,r0
move        #vecb,r4
move        #result,r1
move                x:(r0)+,x0  y:(r4)+,y0
rep         #nmul
mac          x0,y0,a    x:(r0)+,x0  y:(r4)+,y0
move                a1,x:(r1)
endm

dec_vect   macro       vect,long
;cette macro decalle un vecteur
;  x1       x2
;  x2       x3
;
;          xn
;  xn       xn
;vect = l'adresse de base du vecteur
;long = longueur du vecteur

move        #vect+1,r0
```

```
move      #vect,r1
do        #long-1,_end
move                x:(r0)+,a
move                a,x:(r1)+
_end
endm

mul_sca   macro     vect,long,sca,result
;cette macro multiplie un vecteur par un scalaire
;result(i)=vect(i)*sca
;vect = adresse de base du vecteur
;long = longueur du vecteur
;sca = valeur du scalaire
;result = adresse de base du resultat

move      #result,r1
move      #vect,r4
move      #sca,r2
nop
move  x:(r2),a0
move      a0,x1
move                y:(r4)+,x0
do        #long,_end
mpy       x1,x0,a   x:(r4)+,x0
move      a1,x:(r1)+
_end
endm

add_vect  macro     vecta,vectb,nadd,result
;cette macro additionne deux vecteurs
;result(i)= vecta(i) + vectb(i) i=1..nadd
;nadd = longueur du vecteur
;vecta = adresse de base du vecteur a
;vectb = adresse de base du vecteur b
;result = adresse de base du vecteur result

move      #vecta,r0
move      #vectb,r2
move      #result,r1
move                x:(r0)+,x0
do        #nadd,_end
move                x:(r2)+,a1
add       x0,a      x:(r0)+,x0
move      a1,x:(r1)+
```

```
_end
endm

;*********************************************************
;       programme principale
;
;       Les équations 2.9 à 2.12 sont implémenté pour
;       résoudre la déconvolution de signaux stationnaires
;       et invariants.
;
;*********************************************************

nb_pts    equ       128
nb_ptsh equ      80

org       x:$0
don_so    dsm       nb_pts

org       x:$100
xk        dsm       nb_ptsh

org       x:$300
est_so    dsm       2
erreur    dsm       2

org       y:$0
rep_h     dsm       nb_ptsh

org       y:$100
coef_g    dsm       nb_ptsh

org       x:$200
temp      dsm       nb_ptsh

datin     equ       $ffef
sortie equ    xk; sortie lissé
sortie equ   xk+nb_ptsh  ; sortie filtré

org       p:$0
;
; chargement de la table de réponse impulsionnelle à partir
; du fichier
;
 move   #$400,r0
```

```
move      #$0,r4
do        #nb_ptsh,end1
move      x:(r0),a
move      a1,y:(r4)+
end1
;
; chargement de la table des coefficients de gains à partir
; du fichier
;
 move  #$401,r0
 move      #$0,r4
 do        #nb_ptsh,end2
 move      x:(r0),a
 move      a1,y:(r4)+
end2

move      #don_so,r3

do        #nb_pts,_end
dec_vect   xk,nb_ptsh
mul_vect   nb_ptsh,xk,rep_h,est_so
movep      y:datin,a
move       x:est_so,x0
sub        x0,a
move       a1,x:erreur
mul_sca    coef_g,nb_ptsh,erreur,temp
add_vect   xk,temp,nb_ptsh,xk
;
;  Sortie du résultat
;
move       #sortie,r0
move       #sortie2,r1
nop
move       x:(r0),x0
move       x:(r1),y0
move       #don_so+1,r2
nop
move       y0,x:(r2)
move       x0,x:(r3)

_end

end
```

ANNEXE B

Utilisation du logiciel DFT

B.1 Règles d'utilisation de DFT

Les règles d'utilisation décrites dans la section suivante permettent au logiciel de génération automatique de vecteurs de test de générer une bonne couverture de pannes pour le circuit. Les règles qui nous concernent sont divisées en trois catégories: les règles de conception pour un circuit synchrone, les règles de connexions de la chaîne de balayage et les règles pour les tampons à haute impédance. Les règles de conception sont tirées du manuel d'utilisation de DFT par BNR [DFT91].

B.1.1 Règles de conception synchrone

Le circuit utilisé doit être complètement synchrone. Les règles de conception synchrone s'appliquent sur les parties du circuit qui sont composées de logique combinatoire et de bascules D actives sur une transition. Les autres éléments de mémoire tels que les bascules actives sur un niveau, bascule de type SR et autres doivent être utilisés de manière différente et ils sont décrits plus en détail à la fin du chapitre.

Règle 1: L'entrée horloge d'une bascule D doit être contrôlable d'une entrée primaire. Le terme contrôlable permet l'insertion de tampons, d'inverseurs et autres logiques entre l'entrée primaire et l'entrée d'horloge tant et aussi longtemps que la logique résultante équivaut à une seule entrée primaire qui contrôle l'entrée d'horloge.

Règle 2: Toutes les bascules dont l'horloge provient de la même entrée primaire doivent opérer sur le même front. Cette règle permet à toutes les bascules d'être échantillonnées simultanément à partir d'une source externe en mode test.

Règle 3: Il ne doit pas y avoir de chemin entre une entrée d'horloge de bascule et une entrée de donnée de bascule. Cette règle prévient les problèmes de courses sur les entrées de données causées par les transitions d'horloge.

Règle 4: Les entrées remise-à-zéro/remise-à-un asynchrone d'une bascule doivent contrôler une entrée primaire ou être bloquées. Cette règle assure que les entrées remise-à-zéro/ remise-à-un ne changeront pas l'état d'une bascule durant le test.

Règle 5: Les entrées remise-à-zéro/remise-à-un asynchrone des bascules contrôlées à partir d'une entrée primaire particulière doivent opérer sur le même niveau logique. Il serait impossible autrement d'opérer le circuit de façon synchrone.

Règle 6: Ne pas utiliser une entrée primaire pour contrôler la remise-à-zéro/remise-à-un asynchrone d'une bascule et l'entrée d'horloge. L'entrée remise-à-zéro/remise-à-un asynchrone serait inévitablement activée durant le test puisque toutes les entrées d'horloges sont continuellement exercées. Il deviendrait donc impossible de transmettre des données par la chaîne de balayage.

Règle 7: Toutes les portes logiques autre que les bascules sont reliées ensemble pour former un circuit combinatoire. La sortie du circuit correspondant à une entrée particulière est toujours la même, peu importe le vecteur d'entrée précédent qui a été appliqué.

B.1.2 Règles pour la chaîne de balayage

Les règles suivantes s'appliquent lorsque le circuit est en mode balayage (TM=1 et

SM=1).

Règle 1: Une entrée (D) de bascule doit être contrôlable à partir de la sortie (Q) d'une autre bascule ou d'une entrée primaire.

Règle 2: La sortie (Q) d'une bascule doit contrôler l'entrée (D) d'une autre bascule ou une sortie primaire.

Règle 3: La chaîne de balayage doit commencer à une entrée primaire et finir à une sortie primaire.

Règle 4: La chaîne de balayage doit être non-inverseuse. Les inversions à l'intérieur de la chaîne de balayage sont permises, en autant que les données transmises à l'entrée de la chaîne de balayage soit transmise à la sortie de la chaîne de balayage avec la même polarité.

Règle 5: Une seule chaîne de balayage est permise à un moment donné.

B.1.3 Règles pour les tampons haute-impédance

Il existe des restrictions pour l'utilisation de tampons haute-impédance à l'intérieur du circuit intégré. Le tampon doit apparaître comme un circuit combinatoire. Ceci est nécessaire pour éliminer les problèmes de contention et assurer que le bus est dans un état connu durant tout le test.

B.2 Test des bascules actives sur un niveau logique (latch)

Les bascules actives sur un niveau logique ne sont pas testables à cause de la boucle

de rétroaction dans leur conception interne. L'entrée de la bascule est non observable et la logique combinatoire provenant de points contrôlables jusqu'à la bascule est enlevée et considérée comme non testable. La sortie de la bascule est non contrôlable et la logique combinatoire vers un point d'observation est enlevée et est aussi considérée comme non testable. Pour éviter de diminuer la couverture de pannes du circuit, les bascules sont mises en mode transparent durant la période de test. La sortie non contrôlable est ainsi remplacée par son entrée.

B.3 Test des mémoires internes

Les mémoires internes ne sont pas testables à cause de la boucle de rétroaction dans leur conception interne. L'entrée de la RAM est non observable et la logique combinatoire provenant de points contrôlables jusqu'à la RAM est enlevée et considérée comme non testable. La sortie de la RAM est non contrôlable et la logique combinatoire vers un point d'observation est enlevée et aussi considérée comme non testable. Une solution consiste à contourner la RAM par l'ajout d'un multiplexeur [DFT91]. En mode test, l'entrée de la RAM est réinjectée à sa sortie. La sortie non contrôlable est ainsi remplacée par une autre sortie contrôlable. Un modèle de panne de type boîte noire est utilisé dans les deux cas pour éviter les erreurs rapportées par SCANCHECK. La logique combinatoire provenant de points contrôlable jusqu'au bus d'adresse et la ligne de contrôle d'écriture/lecture peuvent être observées par l'ajout de registre fantôme (shadow). L'entrée du registre est reliée au bus d'adresse et a la ligne de contrôle d'écriture/lecture. La fonctionnalité du circuit reste inchangée et la logique est maintenant testable. Il s'agit de trouver un compromis entre la couverture de

pannes et le nombre total de portes logiques.

Le test des mémoires internes est possible par l'emploi d'autres techniques. Il existe deux options: La première consiste à entourer la RAM d'un circuit de type BIST (Built-In Selft-Test). Le circuit comprend alors un générateur de vecteur pseudo-aléatoire conçu à partir d'un registre à décalage à rétroaction linéaire et d'un analyseur de signature basé sur la division polynômial [SAV88]. La solution permet le test interne des mémoires cependant, elle est limitée quant à l'algorithme de test des mémoires et elle requiert environ 300 à 400 portes logiques.

La deuxième solution profite de l'accès externe des mémoires. Un test fonctionnel des mémoires internes est effectué par un processeur extérieur. Une série de lecture et d'écriture permet de valider les pannes de cellules et les erreurs d'adresses. Le test fonctionnel idéal qui détecte toutes les défectuosités possibles devient rapidement trop coûteux pour être pratique. En effet une mémoire de N cellules requiert la complexité d'un algorithme de l'ordre de 2^N pour vérifier la sensibilité de chaque cellule face à toutes les combinaisons possibles des autres cellules [HAY75]. Un compromis est fait entre le pourcentage de détection des pannes et le temps de test requis.

Le test choisi pour le processeur est une version modifiée de la procédure de test utilisé par LSI Logic. Un patron de damier est programmé dans la RAM en écrivant la valeur 55H sur les adresses paires suivi de la valeur AAH sur les adresses impaires. Les adresses sont relues et comparées avec la valeur originale. Le patron de damier inverse est programmé dans la RAM en écrivant la valeur AAH sur les adresses paires suivies de la valeur

55H sur les adresses impaires. Les adresses de la RAM sont relues et comparées avec la valeur originale. La procédure permet de tester plusieurs types de pannes: cellule ouverte, cellule court-circuitée, cellule sensible au voisinage , etc.

Le test pour l'unicité des adresses est effectué en écrivant une diagonale de un logique sur un fond de zéro. Les adresses de la RAM sont relues et comparées avec la valeur originale. Les tests effectués permettent aussi de vérifier la logique située sur le chemin d'accès des RAM internes.

ANNEXE C

Modèle Verilog de l'architecture DSP_KAL

```
/*
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++
+
+       PROCESSEUR SPECIALISE EN RECONSTRUCTION DE SIGNAUX
+       BASEE SUR L'ALGORITHME DE KALMAN
+
+       realisee par : Marc-Alain Santerre
+       le 1 fevrier 1993 (Ecole Polytechnique)
+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/

//      Architecture du processeur specialise


module chip (bus_es_pg, cs_ad, oe, eoc, sc, dv, adresse, gcs, cs1, cs2, cs3, cs4,
rw_ext, phase1, phase2, reset, intra, hold, holda, intr);

parameter size=7;

inout [size:0] bus_es_pg;
input [4:0] adresse;
input eoc, dv, gcs, cs1, cs2, cs3, cs4, rw_ext, phase1, phase2, reset, intra, hold;
output cs_ad, oe, sc, holda, intr;

wire [1:0] mux_mema, mux_memb, mux_memc, mux_ina_ma, mux_inb_ma,
mux_pgm_es, sel_compt;
wire [size:0] bus1, bus2, bus3, bus4, ina_ma, inb_ma, bus_es_pgm;
wire [4:0] adr_mema, adr_memb, adr_memc, donnee;

micro_seq microseq(reset,  phase1, phase2, egale, pp, pg, compt_ech1_zero, compt1_zero,
compt2_zero, hold, intra, eoc, dv, sel_ptr_mema, wr_reg1_mema, wr_reg2_mema,
reg_base1_mema, reg_dep1_mema, reg_base2_mema, reg_dep2_mema, rw_mema,
mux_mema, sel_ptr_memb, wr_reg1_memb, wr_reg2_memb, reg_base1_memb,
reg_dep1_memb, reg_base2_memb, reg_dep2_memb, rw_memb, mux_memb,
sel_ptr_memc, wr_reg1_memc, wr_reg2_memc, reg_base1_memc, reg_dep1_memc,
reg_base2_memc, reg_dep2_memc, rw_memc, mux_memc, mux_ina_ma, mux_inb_ma,
mux_pgm_es, sel_char_clk, wr_ma1, rd_ma1, wr_ma2, rd_ma2, sel_compt, donnee, holda,
cs_ad, oe, sc, overflow, add_ma1, add_ma2, intr, wr_data, rd_result, pgm_ext, bus_es_pgm,
gcs, cs1, cs2, cs3, cs4, rw_ext, rw_bloca, rw_blocb, rw_blocc);

memoire mema(bus1, adr_mema, rw_bloca, pgm_ext, phase2);
memoire #(7, "memb.txt") memb(bus2, adr_memb, rw_blocb, pgm_ext, phase2);
```

```
memoire #(7, "memc.txt") memc(bus3, adr_memc, rw_blocc, pgm_ext, phase2);

muxhiz muxa(bus1, bus2, bus3, bus4, bus_es_pgm, rw_bloca, mux_mema);
muxhiz muxb(bus2, bus1, bus3, bus4, bus_es_pgm, rw_blocb, mux_memb);
muxhiz muxc(bus3, bus1, bus2, bus4, bus_es_pgm, rw_blocc, mux_memc);
muxhiz mux_ina(ina_ma, bus1, bus2, bus3, bus4, 1'b0, mux_ina_ma);
muxhiz mux_inb(inb_ma, bus1, bus2, bus3, bus4, 1'b0, mux_inb_ma);
muxhiz mux_es_pgm(bus_es_pg, bus1, bus2, bus3, bus4, intra || !gcs, mux_pgm_es);

bufhiz bufh(bus_es_pgm, bus_es_pg, gcs && (hold || holda));

multi_addit ma1(ina_ma, inb_ma, bus4, wr_ma1, rd_ma1||(!rd_ma2), phase1, phase2,
add_ma1, reset);
multi_addit ma2(ina_ma, inb_ma, bus4, wr_ma2, rd_ma2||(!rd_ma1), phase1, phase2,
add_ma2, reset);

gen_adr gen_mema(phase1, phase2, sel_ptr_mema, donnee, wr_reg1_mema,
reg_base1_mema, reg_dep1_mema, wr_reg2_mema, reg_base2_mema, reg_dep2_mema,
adresse, adr_mema, reset, pgm_ext);
gen_adr gen_memb(phase1, phase2, sel_ptr_memb, donnee, wr_reg1_memb,
reg_base1_memb, reg_dep1_memb, wr_reg2_memb, reg_base2_memb, reg_dep2_memb,
adresse, adr_memb, reset, pgm_ext);
gen_adr gen_memc(phase1, phase2, sel_ptr_memc, donnee, wr_reg1_memc,
reg_base1_memc, reg_dep1_memc, wr_reg2_memc, reg_base2_memc, reg_dep2_memc,
adresse, adr_memc, reset, pgm_ext);

compteur compt(phase1, phase2, donnee, sel_char_clk, sel_compt, compt_ech1_zero,
compt1_zero, compt2_zero, reset);

comparateur egal(ina_ma, inb_ma, pg, pp, egale, phase1);

adder addit(bus4, bus1, wr_data, bus_es_pgm, (rd_result || !holda || !gcs), overflow, phase1,
phase2);
endmodule
//------------------------------------------------------------------------
/* Le module du microsequenceur est charge de generer les signaux
   de controle du processeur. Il comprend les memoires et une logique
   de controle pour permettre l'acces a partir du processeur maitre.
   ref. cadence : micro_seq
*/

module micro_seq(reset, phase1, phase2, egale, pp, pg, compt_ech1_zero, compt1_zero,
compt2_zero, hold, intra, eoc, dv, sel_ptr_mema, wr_reg1_mema, wr_reg2_mema,
reg_base1_mema, reg_dep1_mema, reg_base2_mema, reg_dep2_mema, rw_mema,
```

```
mux_mema, sel_ptr_memb, wr_reg1_memb, wr_reg2_memb, reg_base1_memb,
reg_dep1_memb, reg_base2_memb, reg_dep2_memb, rw_memb, mux_memb,
sel_ptr_memc, wr_reg1_memc, wr_reg2_memc, reg_base1_memc, reg_dep1_memc,
reg_base2_memc, reg_dep2_memc, rw_memc, mux_memc, mux_ina_ma, mux_inb_ma,
mux_pgm_es, sel_char_clk, wr_ma1, rd_ma1, wr_ma2, rd_ma2, sel_compt, donnee, holda,
cs_ad, oe, sc, overflow, add_ma1, add_ma2, intr, wr_data, rd_result, pgm_ext, bus_es_pgm,
gcs, cs1, cs2, cs3, cs4, rw_ext, rw_bloca, rw_blocb, rw_blocc);

parameter size=7;
input reset, phase1, phase2, egale, pp, pg, compt_ech1_zero, compt1_zero, compt2_zero,
hold, intra, eoc, dv, overflow, gcs, cs1, cs2, cs3, cs4, rw_ext;
input [size:0] bus_es_pgm;
output wr_reg1_mema, wr_reg2_mema, reg_base1_mema, reg_dep1_mema,
reg_base2_mema, reg_dep2_mema, rw_mema, wr_reg1_memb, wr_reg2_memb,
reg_base1_memb, reg_dep1_memb, reg_base2_memb, reg_dep2_memb, rw_memb,
wr_reg1_memc, wr_reg2_memc, reg_base1_memc, reg_dep1_memc, reg_base2_memc,
reg_dep2_memc, rw_memc, wr_ma1, rd_ma1, wr_ma2, rd_ma2, holda, cs_ad, oe, sc,
add_ma1, add_ma2, intr, sel_char_clk, wr_data, rd_result, pgm_ext, sel_ptr_mema,
sel_ptr_memb, sel_ptr_memc, rw_bloca, rw_blocb, rw_blocc;
output [1:0] mux_mema, mux_memb, mux_memc, mux_ina_ma, mux_inb_ma,
mux_pgm_es, sel_compt;
output [4:0] donnee;

reg [4:0] prog_compt, adresse;

wire [7:0] reg_instrd;
wire [7:0] reg_instre;
wire [7:0] reg_instrf;
wire [7:0] reg_instrg;
wire [7:0] reg_instrh;
wire [7:0] reg_instri;
wire [7:0] reg_instrj;
wire [7:0] reg_instrk;

reg etat;

memoire_seq memd(reg_instrd, bus_es_pgm, adresse, rw_blocd, phase1);
memoire_seq #(7, "meme.txt") meme(reg_instre, bus_es_pgm, adresse, rw_bloce, phase1);
memoire_seq #(7, "memf.txt") memf(reg_instrf, bus_es_pgm, adresse, rw_blocf, phase1);
memoire_seq #(7, "memg.txt") memg(reg_instrg, bus_es_pgm, adresse, rw_blocg, phase1);
memoire_seq #(7, "memh.txt") memh(reg_instrh, bus_es_pgm, adresse, rw_bloch, phase1);
memoire_seq #(7, "memi.txt") memi(reg_instri, bus_es_pgm, adresse, rw_bloci, phase1);
memoire_seq #(7, "memj.txt") memj(reg_instrj, bus_es_pgm, adresse, rw_blocj, phase1);
memoire_seq #(7, "memk.txt") memk(reg_instrk, bus_es_pgm, adresse, rw_block, phase1);
```

```
program programme(gcs, cs1, cs2, cs3, cs4, rw_mema, rw_memb, rw_memc, rw_ext,
pgm_ext, rw_bloca, rw_blocb, rw_blocc, rw_blocd, rw_bloce, rw_blocf, rw_blocg,
rw_bloch, rw_bloci, rw_blocj, rw_block);

assign sel_ptr_mema=reg_instrd[0],
        wr_reg1_mema=reg_instrd[1],
        wr_reg2_mema=reg_instrd[2],
        reg_base1_mema=reg_instrd[3],
        reg_dep1_mema=reg_instrd[4],
        reg_base2_mema=reg_instrd[5],
        reg_dep2_mema=reg_instrd[6],
        rw_mema=reg_instrd[7],
        mux_mema=reg_instre[1:0],
        sel_ptr_memb=reg_instre[2],
        wr_reg1_memb=reg_instre[3],
        wr_reg2_memb=reg_instre[4],
        reg_base1_memb=reg_instre[5],
        reg_dep1_memb=reg_instre[6],
        reg_base2_memb=reg_instre[7],
        reg_dep2_memb=reg_instrf[0],
        rw_memb=reg_instrf[1],
        mux_memb=reg_instrf[3:2],
        sel_ptr_memc=reg_instrf[4],
        wr_reg1_memc=reg_instrf[5],
        wr_reg2_memc=reg_instrf[6],
        reg_base1_memc=reg_instrf[7],
        reg_dep1_memc=reg_instrg[0],
        reg_base2_memc=reg_instrg[1],
        reg_dep2_memc=reg_instrg[2],
        rw_memc=reg_instrg[3],
        mux_memc=reg_instrg[5:4],
        mux_ina_ma=reg_instrg[7:6],
        mux_inb_ma=reg_instrh[1:0],
        mux_pgm_es=reg_instrh[3:2],
        sel_char_clk=reg_instrh[4],
        wr_ma1=reg_instri[3],
        rd_ma1=reg_instri[4],
        wr_ma2=reg_instri[5],
        rd_ma2=reg_instri[6],
        sel_compt={reg_instrj[0], reg_instri[7]},
        holda=reg_instrj[1],
        cs_ad=reg_instrj[2],
        oe=reg_instrj[3],
```

```
        sc=reg_instrj[4],
        donnee={reg_instrk[1:0], reg_instrj[7:5]},
        add_ma1=reg_instrk[2],
        add_ma2=reg_instrk[3],
        intr=reg_instrk[4],
        wr_data=reg_instrk[5],
        rd_result=reg_instrk[6],
        pgm_ext=reg_instrk[7];

// mux_cond=reg_instr[40:37]
// polarite=reg_instr[41]
// sel_bran=reg_instr[42]

always @(negedge reset)
        begin
        adresse=6'h00;
        end

always @(phase1 or egale or pp or pg or compt_ech1_zero or compt1_zero or
        compt2_zero or hold or intra or eoc or dv or overflow)
        if (phase1)
                #0 begin

                        case ({reg_instri[0], reg_instrh[7:5]})
                        4'b0000:etat=egale;
                        4'b0001:etat=pp;
                        4'b0010:etat=pg;
                        4'b0011:etat=compt_ech1_zero;
                        4'b0100:etat=compt1_zero;
                        4'b0101:etat=compt2_zero;
                        4'b0110:etat=hold;
                        4'b0111:etat=intra;
                        4'b1000:etat=eoc;
                        4'b1001:etat=dv;
                        4'b1010:etat=1'b0;
                        4'b1011:etat=1'b1;
                        4'b1100:etat=overflow;
                        default etat=1'b1;
                        endcase
                        if (reg_instri[1]) etat=etat;
                        else etat=~etat;
                end

always @(phase2)
```

```
        if (phase2)
                begin
                        prog_compt=adresse+1;
                        if (etat || reg_instri[2]) adresse=prog_compt;
                        else adresse={reg_instrk[1:0], reg_instrj[7:5]};
                end
endmodule


//----------------------------------------------------------------------
/* Le module genere les signaux d'ecritures et lectures provenant du
  microsequenceur ou du processeur maitre.
  ref. cadence : controle_rw
*/

module program(gcs, cs1, cs2, cs3, cs4, rw_mema, rw_memb, rw_memc, rw_ext, pgm,
rw_bloca, rw_blocb, rw_blocc, rw_blocd, rw_bloce, rw_blocf, rw_blocg, rw_bloch,
rw_bloci, rw_blocj, rw_block);
input gcs, cs1, cs2, cs3, cs4, rw_mema, rw_memb, rw_memc, rw_ext, pgm;
output rw_bloca, rw_blocb, rw_blocc, rw_blocd, rw_bloce, rw_blocf, rw_blocg, rw_bloch,
rw_bloci, rw_blocj, rw_block;
reg rw_bloca, rw_blocb, rw_blocc, rw_blocd, rw_bloce, rw_blocf, rw_blocg, rw_bloch,
rw_bloci, rw_blocj, rw_block;

always @(gcs or cs1 or cs2 or cs3 or cs4 or rw_mema or rw_memb or rw_memc
        or rw_ext or pgm)
        begin
        rw_bloca= 1'b1;
        rw_blocb= 1'b1;
        rw_blocc= 1'b1;
        rw_blocd= 1'b1;
        rw_bloce= 1'b1;
        rw_blocf= 1'b1;
        rw_blocg= 1'b1;
        rw_bloch= 1'b1;
        rw_bloci= 1'b1;
        rw_blocj= 1'b1;
        rw_block= 1'b1;

        if (!pgm||!gcs)
        case ({cs4, cs3, cs2, cs1})
                4'b0000:rw_bloca= rw_ext;
                4'b0001:rw_blocb= rw_ext;
                4'b0010:rw_blocc= rw_ext;
```

```
                4'b0011:rw_blocd= rw_ext;
                4'b0100:rw_bloce= rw_ext;
                4'b0101:rw_blocf= rw_ext;
                4'b0110:rw_blocg= rw_ext;
                4'b0111:rw_bloch= rw_ext;
                4'b1000:rw_bloci= rw_ext;
                4'b1001:rw_blocj= rw_ext;
                4'b1010:rw_block= rw_ext;
        endcase
        else
                begin
                        rw_bloca= rw_mema;
                        rw_blocb= rw_memb;
                        rw_blocc= rw_memc;
                        rw_blocd= 1'b1;
                        rw_bloce= 1'b1;
                        rw_blocf= 1'b1;
                        rw_blocg= 1'b1;
                        rw_bloch= 1'b1;
                        rw_bloci= 1'b1;
                        rw_blocj= 1'b1;
                        rw_block= 1'b1;
                end
        end
endmodule

//-------------------------------------------------------------------------
/* Unite de memoire ou sont garde les vecteurs utiles a la reconstruction
  du signal. La lecture est active sur phase1 et l'ecriture sur la phase2.
*/

module memoire (data, adresse, rw_, pgm_ext, phase2);
//memoire de 16 X 8 bits
parameter size=7, fichier_ram="mema.txt";

input rw_, pgm_ext, phase2;
input [4:0] adresse;
inout [size:0] data;

reg [size:0] data_reg;
reg [size:0] memoire [0:31];

assign data =data_reg;
always @(data or adresse or rw_ or phase2)
```

```
        if ( rw_) data_reg=memoire[adresse];
always @(data or adresse or rw_ or phase2 or pgm_ext)
        if ((phase2 || !pgm_ext) && !rw_)
        begin
                data_reg=8'bz;
                memoire[adresse]=data;
        end
initial
        $readmemh(fichier_ram, memoire);
endmodule

//-------------------------------------------------------------------------
/* Unite de memoire pour la memoire de microprogrammation du microsequenceur.
  ref. cadence:
*/

module memoire_seq (data, bus_es_pgm, adresse, rw_, phase1);
//memoire de 32 X 8 bits
parameter size=7, fichier_ram="memd.txt";

// faire attention pour mettre un buffer haute impedance pour empecher
// les donnees de fairent collision avec le mode lecture

input rw_, phase1;
input [4:0] adresse;
input [size:0] bus_es_pgm;
output [size:0] data;

reg [size:0] data_reg;
reg [size:0] memoire [0:31];

assign data =data_reg;
always @(data or adresse or rw_ or phase1 )
        if (phase1 && rw_) data_reg=memoire[adresse];
always @(data or adresse or rw_ )
        if (!rw_) memoire[adresse]=bus_es_pgm;
initial
        $readmemb(fichier_ram, memoire);
endmodule


//-------------------------------------------------------------------------

/* Module du multiplicateur/additionneur. Le delai simule est de
```

```
  25 pas du simulateur.
  ref. cadence: mult_add
*/

module multi_addit(opa, opb, sortie, wr_ma, rd_ma, phase1, phase2, addit, reset);

parameter size=7,  longsize=15;

input phase1, phase2, addit, wr_ma, rd_ma, reset;
input [size:0] opa, opb;
output [size:0] sortie;
reg addit_latch;
reg [size:0] sortie;
reg [longsize:0] shift_opa, shift_opb, result, sortie2, resultat;


always @(phase2 or opa or opb or wr_ma or addit)
        #5 if (phase2 && !wr_ma)
        begin :mult
        shift_opa=opa;
        shift_opb=opb;
        addit_latch=addit;

        #20 result=shift_opa*shift_opb;

        if (!addit_latch) sortie2=result+sortie2;
        if (!addit_latch) resultat=sortie2;
        else resultat=result;

        end

 always @(phase2 or rd_ma or resultat)
        begin
        #2 if (phase2 && !rd_ma)
                begin
                        sortie=resultat [7:0];  // attention !!!!!
                        sortie2=16'b0;
                end
        end
always @(negedge phase2) sortie=8'bz;

always @(negedge reset)
        begin
                resultat=16'b0;
```

```
                sortie2=16'b0;
        end

endmodule

//------------------------------------------------------------------------
/* Module du multiplexeur 4 dans 1 avec sortie a haute impedance
  ref. cadence: mux_mux_8_scan
*/

module muxhiz(busout, bus0, bus1, bus2, bus3, enable, s);
        parameter size=7;
        parameter Zee=8'bz;
        output [size:0] busout;
        input [size:0] bus0, bus1, bus2, bus3;
        input enable;
        input [1:0] s;

        reg [size:0] data;
        reg [size:0] busout;

always @(bus0 or bus1 or bus2 or bus3 or enable or s)
        begin
                case (s)
                        2'b00:data=bus0;
                        2'b01:data=bus1;
                        2'b10:data=bus2;
                        2'b11:data=bus3;
                endcase

                if (!enable) busout=data;
                else busout=Zee;
        end


endmodule

//------------------------------------------------------------------------
/* Module de buffer a haute impedance. Il sert pour le bus d'entre
  sortie bidirectionnelle.
  ref. cadence:bufhiz
*/

module bufhiz(busout, busin, enable);
```

```
        parameter size=7;
        parameter Zee=8'bz;
        output [size:0] busout;
        input [size:0] busin;
        input enable;

        reg [size:0] busout;

always @(busin or enable)

        begin
                if (!enable) busout=busin;
                else busout=Zee;
        end
endmodule

//----------------------------------------------------------------------
/* Module de comparateur pour la reconstitution avec contrainte
  ref.cadence: comparateur ( realise avec synful)
*/

module comparateur(port1, port2, pg, pp, egale, phase1);

parameter size=7;
input [size:0] port1, port2;
input phase1;
output pg, pp, egale;
reg pp, pg, egale;

always @(phase1 or port1 or port2)
        if (phase1)
                begin
                        pp=1'b1;
                        pg=1'b1;
                        egale=1'b1;
                        if (port1 < port2)  pp=1'b0;
                        if (port1 == port2) egale=1'b0;
                        if (port1 > port2) pg=1'b0;
                end
endmodule

//----------------------------------------------------------------------
/* Module de compteur pour implementer les boucles
  ref.cadence: ? (realiser par synful)
```

```
*/

module compteur(phase1, phase2, donnee, sel_char_clk, sel_compt, compt_ech1_zero,
compt1_zero, compt2_zero, reset);

input phase1, phase2, sel_char_clk, reset;
input [1:0] sel_compt;
input [4:0] donnee;
output compt1_zero, compt2_zero, compt_ech1_zero;


reg [4:0] compt_ech, compt1, compt2, compt_ech_fb, compt1_fb, compt2_fb;
reg compt1_zero, compt2_zero, compt_ech1_zero;

always @( phase1 or donnee or sel_char_clk or sel_compt)
        if (phase1)
                begin
                compt_ech1_zero=1'b1;
                compt1_zero=1'b1;
                compt2_zero=1'b1;
        if (sel_char_clk)
                case (sel_compt)
                2'b00:;
                2'b01:compt_ech_fb=donnee;
                2'b10:compt1=donnee;
                2'b11:compt2=donnee;
                endcase
        else
                case (sel_compt)
                2'b00:;
                2'b01:begin
                        compt_ech_fb=compt_ech-1;
                        if (compt_ech_fb==0) compt_ech1_zero=1'b0;
                        end
                2'b10:begin
                        compt1_fb=compt1-1;
                        if (compt1_fb==0) compt1_zero=1'b0;
                        end
                2'b11:begin
                        compt2_fb=compt2-1;
                        if (compt2_fb==0) compt2_zero=1'b0;
                        end
                endcase
                end
```

```
always @(phase2)
   begin
      compt_ech=compt_ech_fb;
      compt1=compt1_fb;
      compt2=compt2_fb;
   end

always @(negedge reset)
                  begin
                  compt_ech=5'b0;
                  compt1=5'b0;
                  compt2=5'b0;
                  end

endmodule

//-------------------------------------------------------------------------
/* Module d'additionneur
  ref.cadence:
*/

module adder (data_a, data_b, wr_data, result, rd_result, overflow, phase1, phase2);

parameter size=7;

input[size:0] data_a, data_b;
input wr_data, rd_result, phase1, phase2;
output[size:0] result;
output overflow;

reg [size:0] result, result1;
reg overflow, overflow1;

always @(data_a or data_b or wr_data or rd_result or phase2)

         if (phase2 && !wr_data) {overflow1, result1}=data_a+data_b;

always @(data_a or data_b or wr_data or rd_result or phase2)

         if (phase2 && !rd_result) begin
                                   result=result1;
                                   overflow=overflow1;
                                   end
```

```
always @(negedge phase2)
                        begin
                        result=8'bz;
                        overflow=1'bz;
                        end

endmodule

//-------------------------------------------------------------------------
/* Module de generation d'addresse pour les memoires principales. Il
  est compose de deux pointeurs independants.
  ref.cadence:

        Note: modifie pour implementer la testabilite avec synful mais
        realise avec l'entree schematique
*/

module gen_adr(phase1, phase2, sel_ptr, donnee, wr_reg1, reg_base1, reg_dep1, wr_reg2,
reg_base2, reg_dep2, pgm_adr, pointeur, reset, pgm_ext);

parameter ptr_size=4;

input phase1, phase2, wr_reg1, reg_base1, reg_dep1, wr_reg2, reg_base2, reg_dep2, reset,
sel_ptr, pgm_ext;
input [ptr_size:0] donnee;
input [ptr_size:0] pgm_adr;
output [ptr_size:0] pointeur;
reg [ptr_size:0] add1, add2, registre_base1, registre_dep1, registre_base2, registre_dep2,
reg_fb1, reg_fb2;
reg [ptr_size:0] pointeur;

always @(phase1 or phase2 or sel_ptr or donnee or wr_reg1 or reg_base1 or reg_dep1 or
wr_reg2 or reg_base2 or reg_dep2 or pgm_adr or reset or pgm_ext)


   begin
        if (!pgm_ext) pointeur=pgm_adr[ptr_size:0];
        else
        case (sel_ptr)
        1'b0:
                begin
                if (phase1 && !reg_dep1) registre_dep1=donnee;
```

```
                    else registre_dep1=registre_dep1;
                if (phase1 && !reg_base1)
                    if (!wr_reg1) registre_base1=donnee;
                        else registre_base1=reg_fb1;
                    else registre_base1=registre_base1;
                add1=registre_base1+registre_dep1;
                pointeur=add1[ptr_size:0];
                if (phase2) reg_fb1=add1;
                end
        1'b1:
                begin
                if (phase1 && !reg_dep2) registre_dep2=donnee;
                    else registre_dep2=registre_dep2;
                if (phase1 && !reg_base2)
                    if (!wr_reg2) registre_base2=donnee;
                        else registre_base2=reg_fb2;
                    else registre_base2=registre_base2;
                add2=registre_base2+registre_dep2;
                pointeur=add2[ptr_size:0];
                if (phase2) reg_fb2=add2;
                end

        endcase
    end
always @(negedge reset)
        begin
        registre_base1=5'h00;
        registre_dep1=5'h00;
        registre_base2=5'h00;
        registre_dep2=5'h00;
        reg_fb1=5'h00;
        reg_fb2=5'h00;
        add1=5'h00;
        add2=5'h00;
        end

endmodule
```

ANNEXE D

Schémas de DSP_KAL+

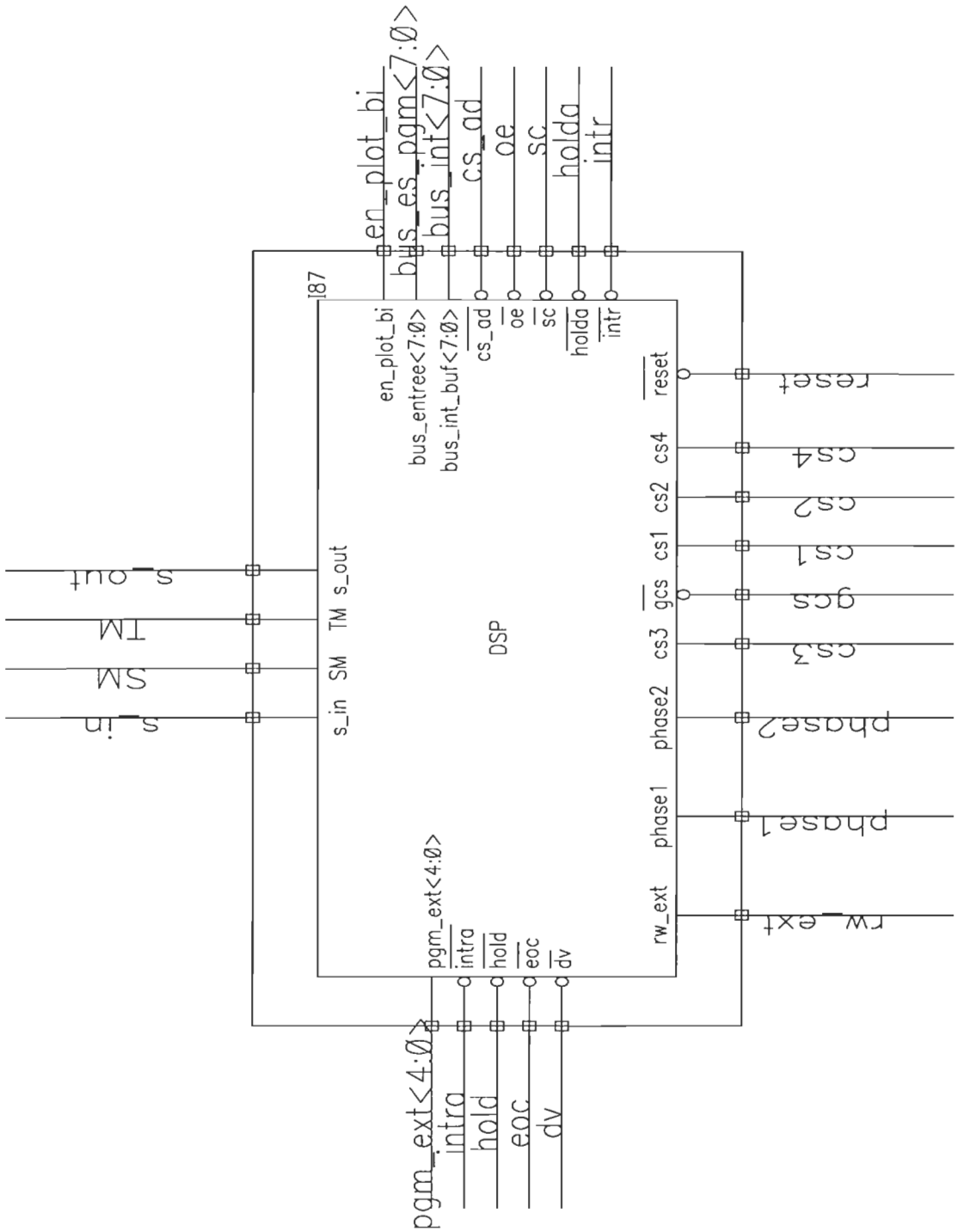

Figure D-1 Symbole du circuit intégré

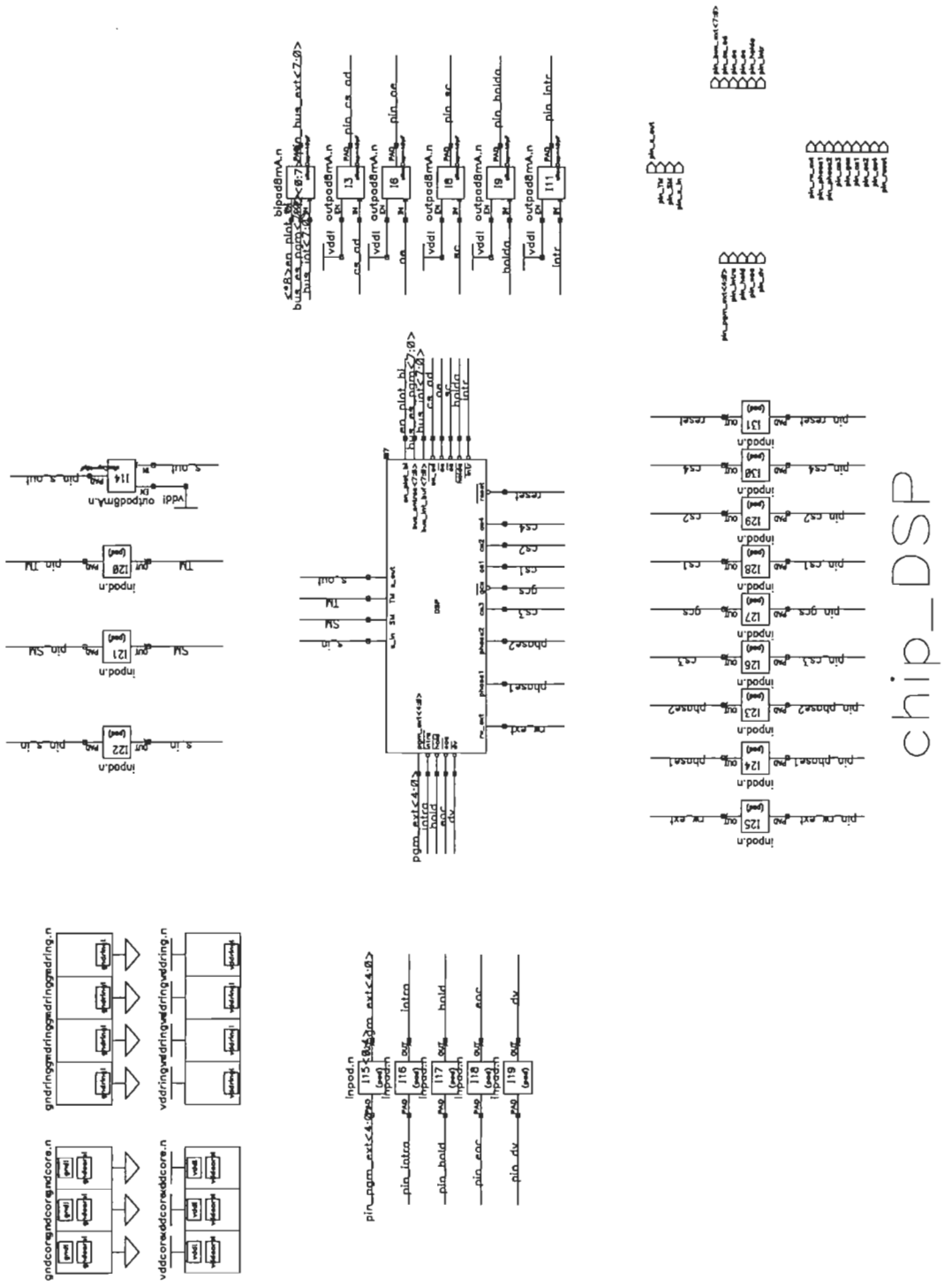

Figure D-2 Schéma principal.

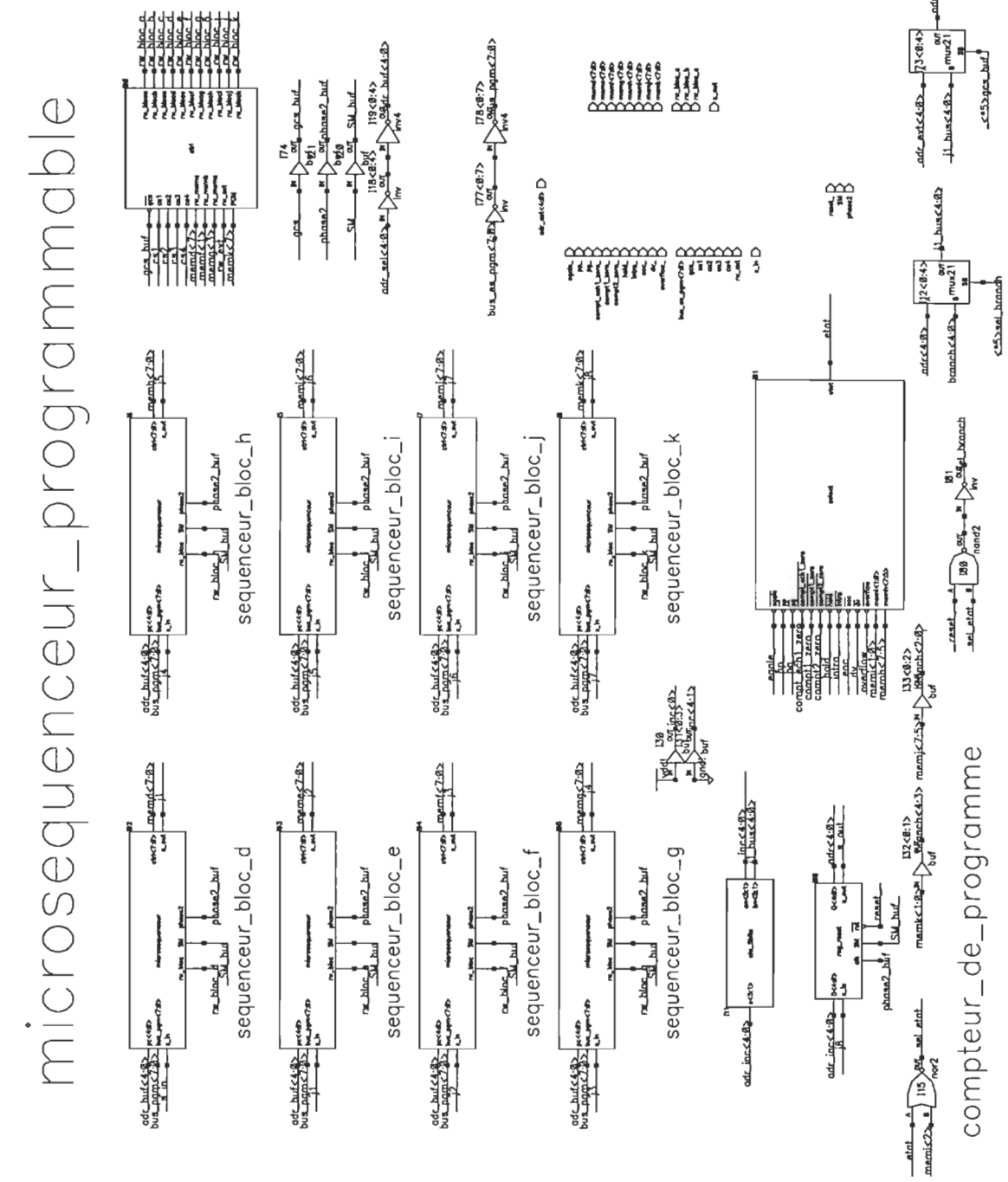

Figure D-3 Microséquenceur programmable.

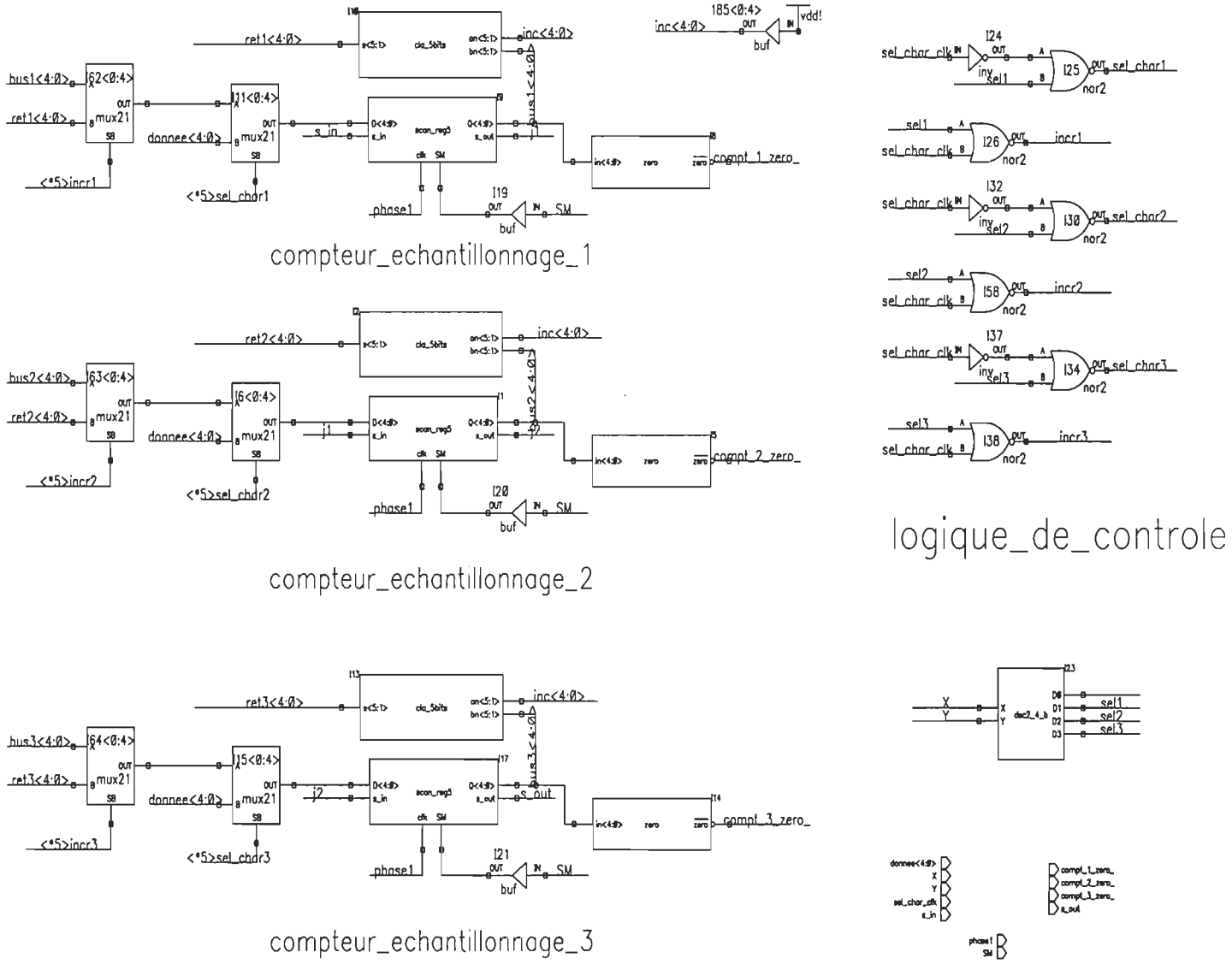

Figure D-4 Compteurs d'échantillonnages.

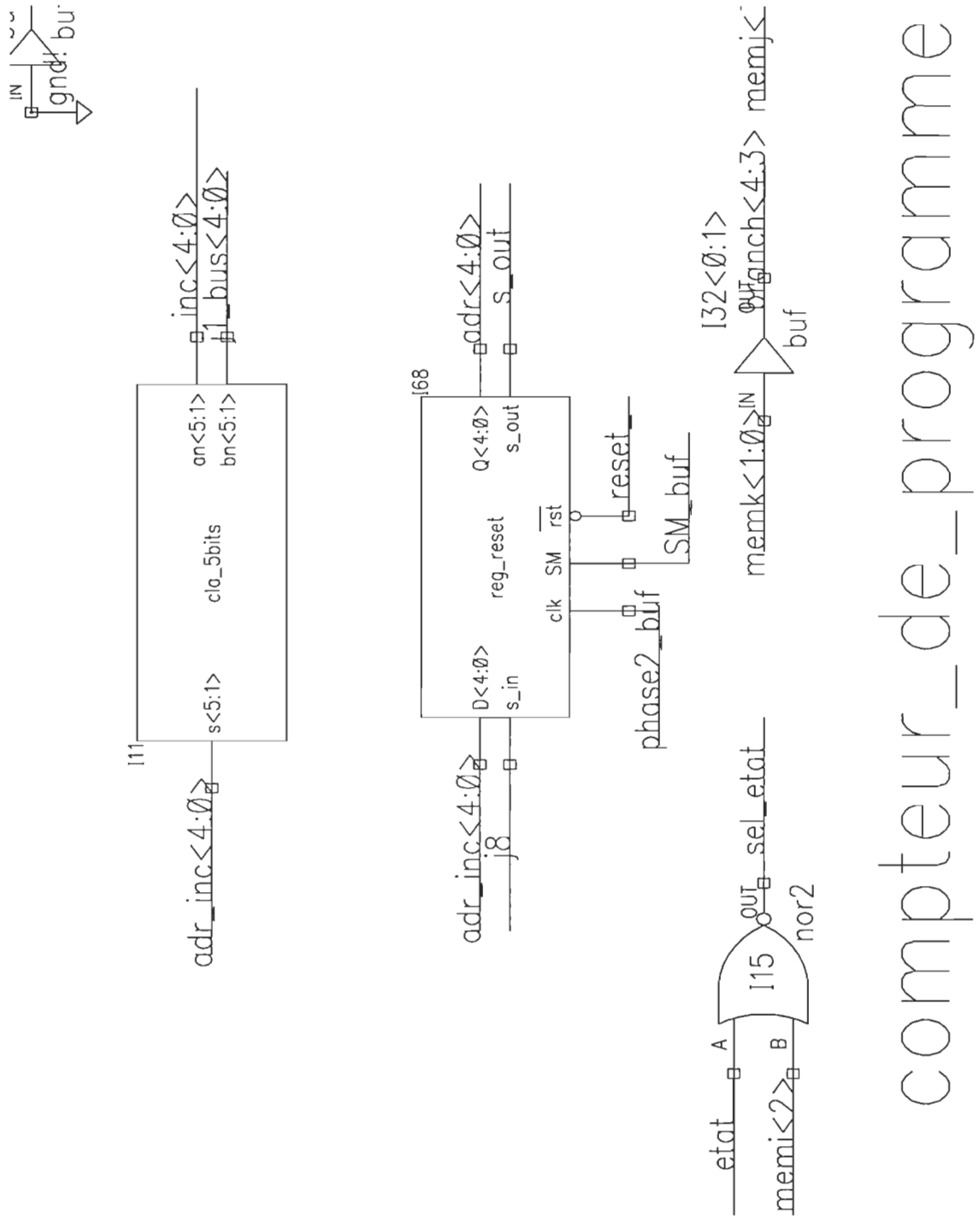

Figure D-5 Compteur de programme.

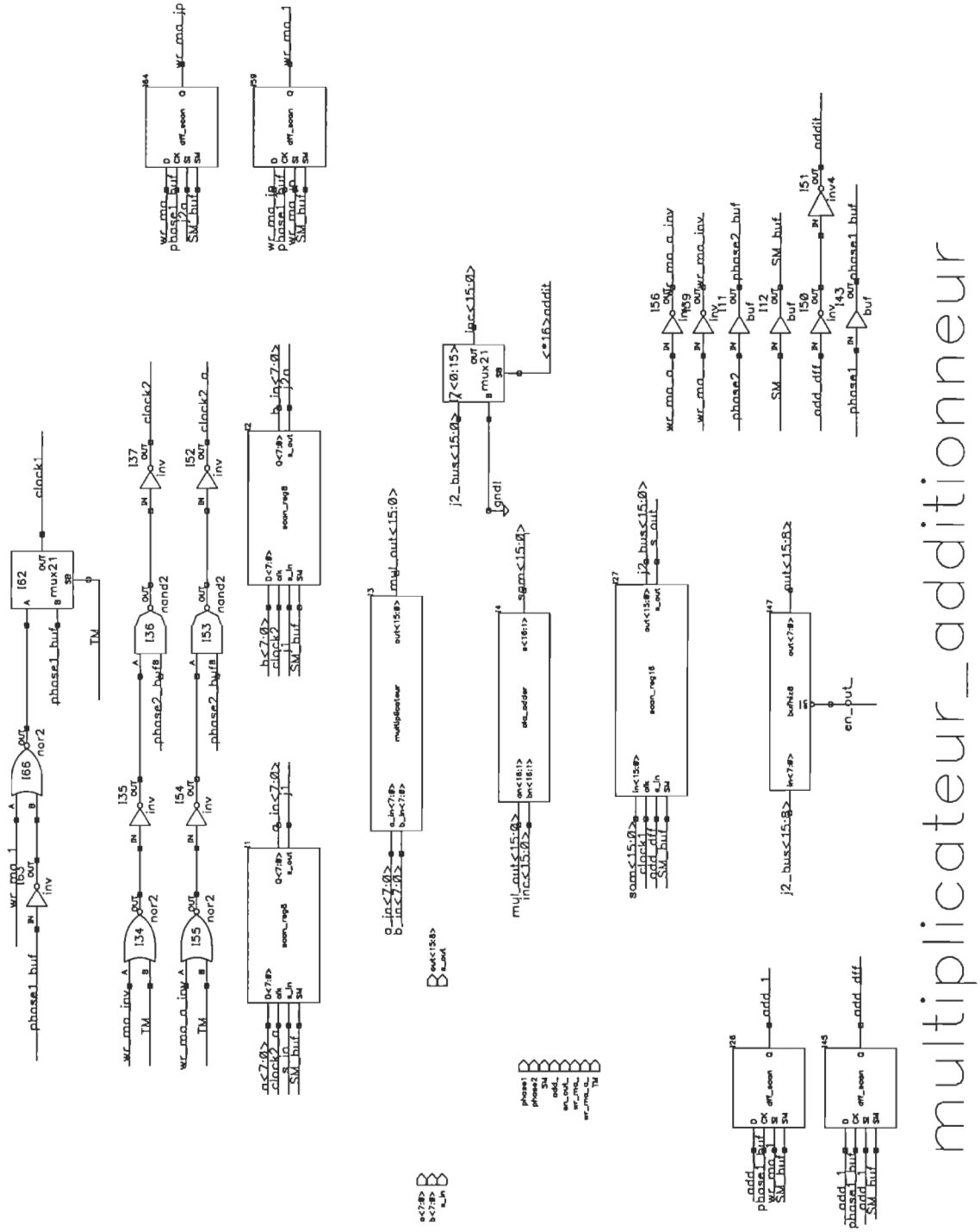

Figure D-6 Multiplicateur/accumulateur.

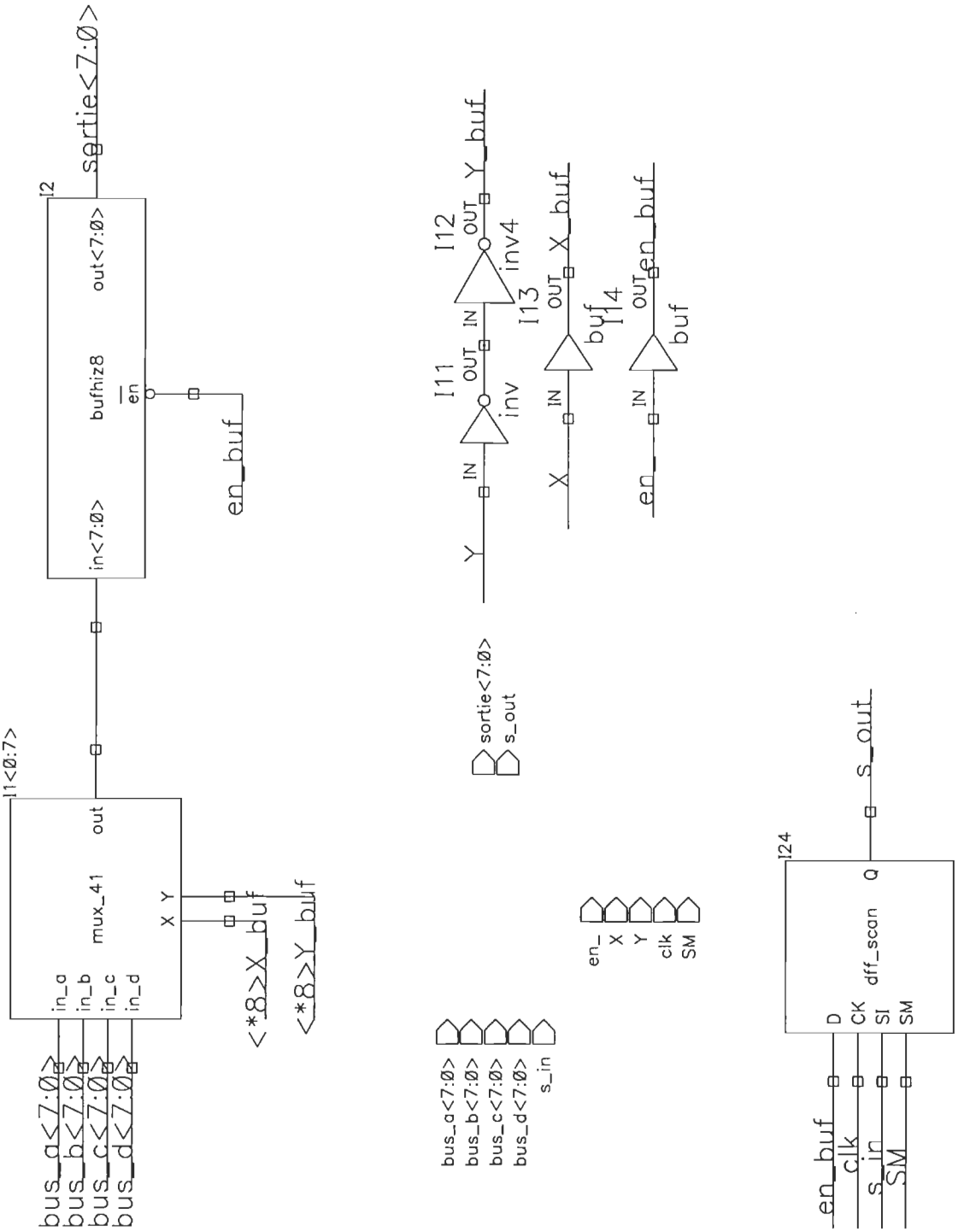

Figure D-7 Multiplexeur de bus.

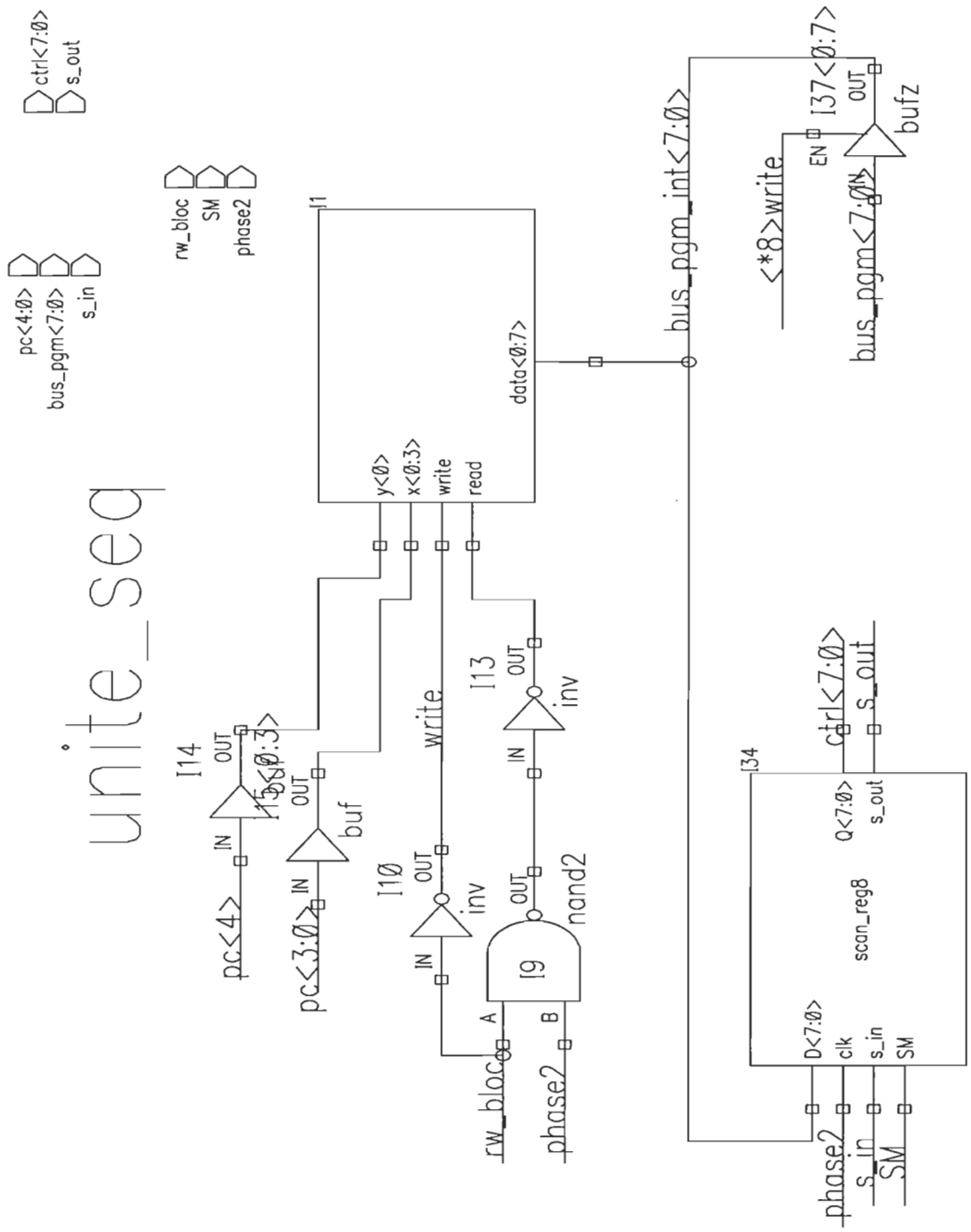

Figure D-8 Mémoire du microséquenceur.

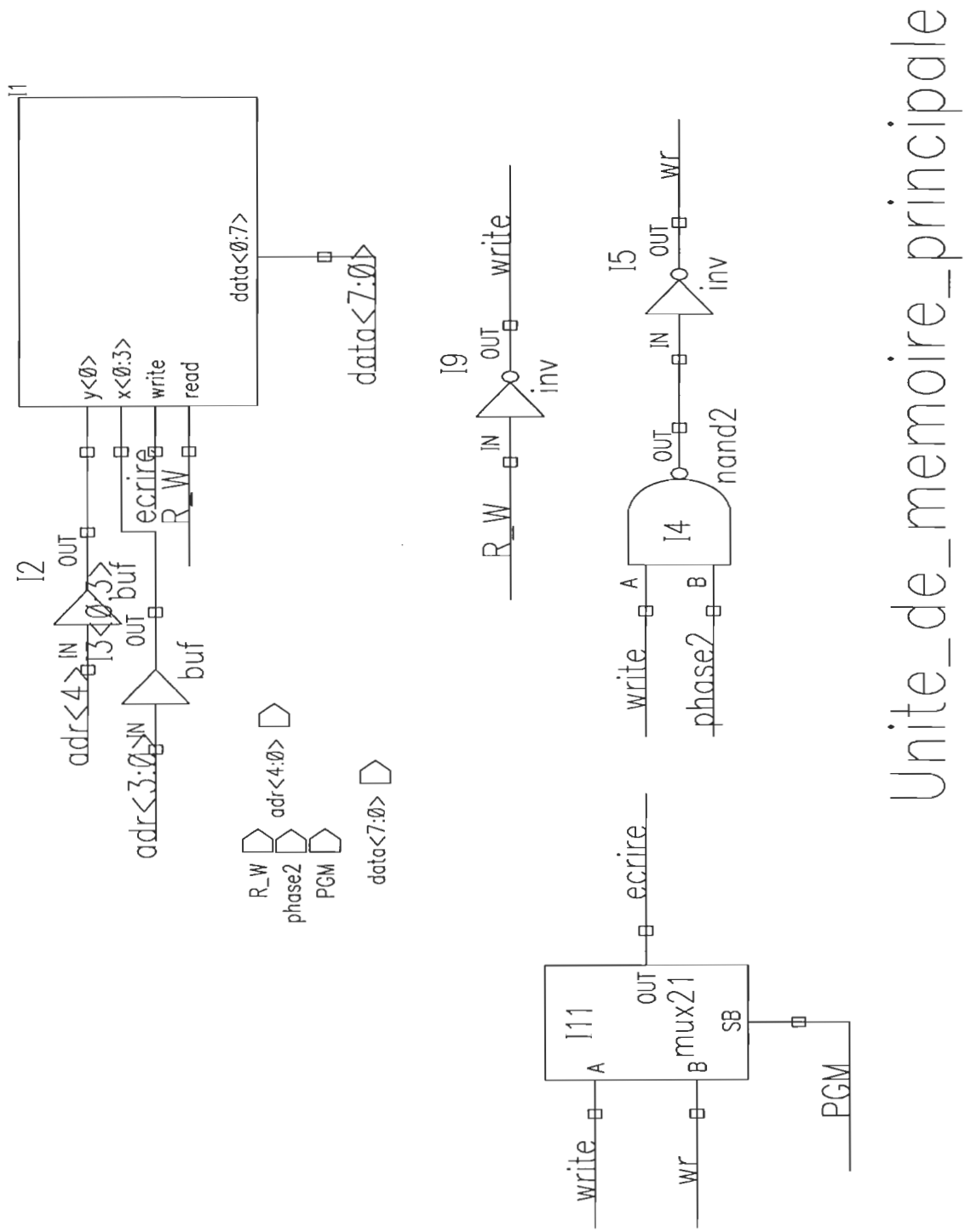

Figure D-9 Unité de mémoire principale.

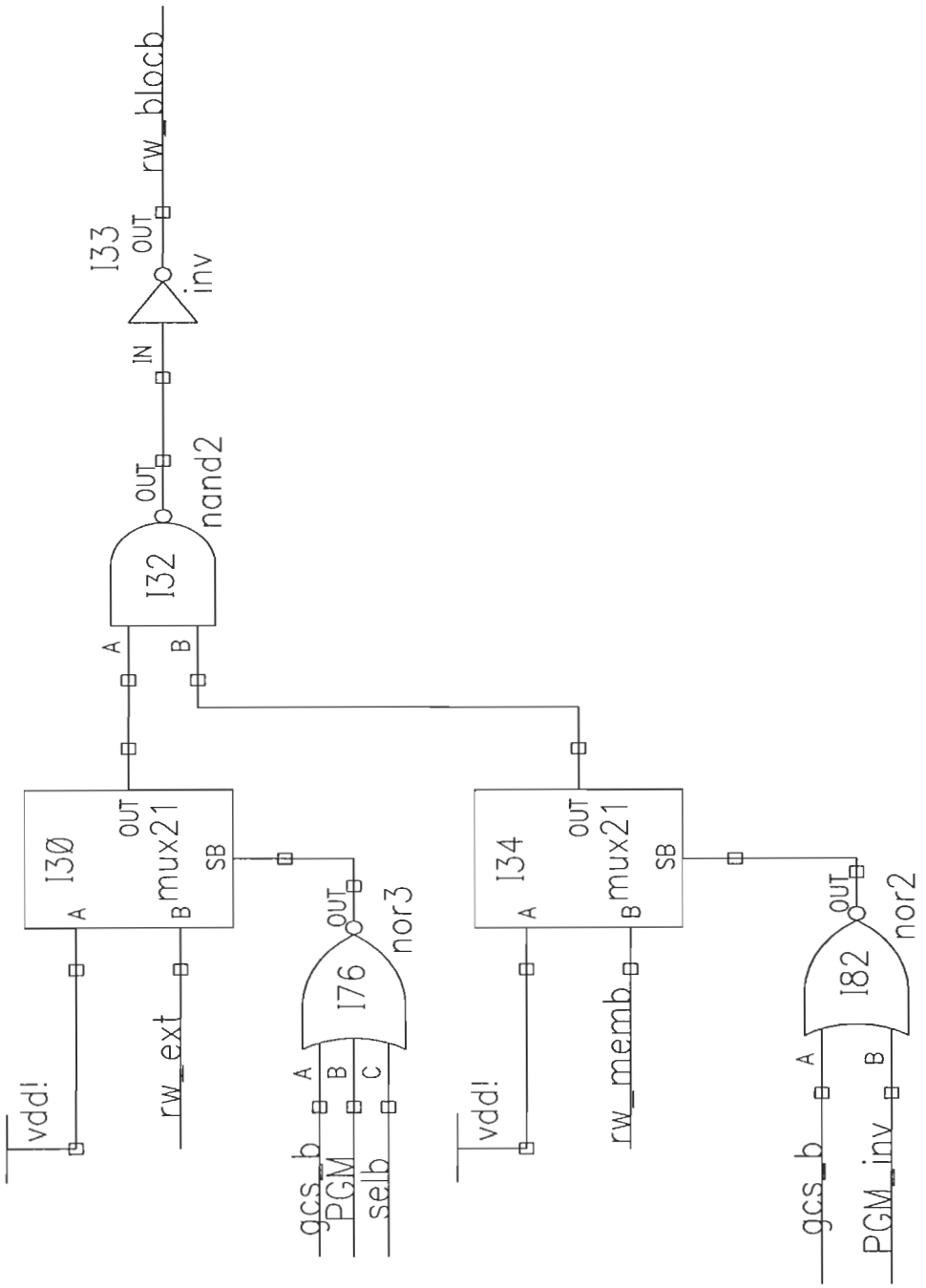

Figure D-10 Contrôle d'accès aux mémoires principales.