UNIVERSITÉ DE MONTRÉAL

CONCEPTION ET IMPLANTATION EN MICROÉLECTRONIQUE DE PROCESSEURS PARALLELES POUR LE TRAITEMENT D'IMAGES

par

Gilles CHOUINARD

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU GRADE DE MAITRISE ES SCIENCES APPLIQUEES (M.Sc.A)

avril 1990

c Gilles Chouinard 1990

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-58112-3

Canada

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE

Ce mémoire intitulé:

CONCEPTION ET IMPLANTATION EN MICROELECTRONIQUE DE PROCESSEURS PARALLELES POUR LE TRAITEMENT D'IMAGES

présenté par : Gilles Chouinard

en vue de l'obtention du grade de: maîtrise ès sciences appliquées (M.Sc.A.)

a été dûment accepté par le jury d'examen constitué de:

M. Yvon Savaria	, Ph.D., président
M. Jean-Louis Houle	, Ph.D.
M. Laval Tremblay	, Ing.

SOMMAIRE

Pour plusieurs applications nécessitant du traitement en temps réel, telle que le traitement numérique d'images (TNI), on utilise des algorithmes et des architectures parallèles pour atteindre la performance désirée. Avec le développement des circuits ITGE (intégration à très grande échelle) de telles machines sont maintenant abordables et à un haut niveau de performance.

Considérant ces faits, un circuit intégré (c.i.) contenant huit processeurs identiques a été conçu, fabriqué et intégré comme coprocesseur à un poste de travail SUN. La machine IMAGE utilise une architecture parallèle de type IUDM (instruction unique, données multiples) implantée sous forme d'une matrice bidimensionelle de 256 processeurs. Ce travail résume les étapes importantes du développement de la machine IMAGE. D'abord le circuit intégré IMAGE-2 a été modélisé et simulé à l'aide d'outils de CAO (conception assistée par ordinateur) afin d'assurer un haut rendement lors de la fabrication. Une fois le c.i. fabriqué, des logiciels de test ont été développés afin de faciliter la vérification de l'implantation. En se basant sur des caractéristiques des algorithmes de TNI, la machine IMAGE a ensuite été conçue et construite.

Le système IMAGE montre que l'utilisation de circuits ITGE et de structures parallèles peuvent offrir des performances temps réel dans des applications de traitement numérique d'images en utilisant des technologies récentes.

ABSTRACT

For real-time image processing, architectures and algorithms use parallelism to attain required performances. The development of VLSI (Very Large Scale Integration) circuits can bring these machines to a good performance level with a reasonable complexity.

Based on these facts, a chip containing eight identical processors has been designed, fabricated and integrated as a coprocessor to a workstation. The IMAGE system uses a SIMD (Single Instruction, Multiple Data) architecture implemented as a bidimensional matrix containing 256 processors. The present work covers some important steps in the development of the IMAGE engine. First, the IMAGE-2 chip was modeled and simulated on sophisticated CAD (Computer Aided Design) tools in order to obtain a good yield when fabricated. After the fabrication, test softwares were developped to facilitate the verification of the design. Finally, after a study of typical image processing algorithms and their caracteristics, the IMAGE engine was designed and built. The system presently incorporates one IMAGE-2 chip.

The IMAGE system demonstrates that parallel structures and VLSI circuits can offer real-time performance in image processing applications using today's technology.

REMERCIEMENTS

Je tiens particulièrement à remercier mon directeur de recherche, le professeur Jean-Louis Houle pour son aide et son support tout au long de ce travail. J'ai aussi grandement apprécié les conseils et les suggestions du professeur Yvon Savaria qui a été activement impliqué dans le projet IMAGE.

Daniel Audet, l'un des deux concepteurs du c.i. IMAGE-2, reçoit aussi toute ma gratitude pour son aide lors des simulations et de la vérification d'IMAGE-2. Il a aussi contribué activement aux discussions lors de la conception de la machine IMAGE. Le stagiaire Franck Bertaud a aussi été d'une aide inestimable lors de l'implantation de la machine IMAGE. Plusieurs autres étudiants avec lesquelles j'ai eu le plaisir de travailler m'ont offert un support moral et je leurs en suis tous reconnaissant.

Je tiens aussi à remercier la compagnie MATROX pour le don du processeur d'images MVP, et les ingénieurs L. Tremblay, D. Connoly et M. Robinson qui ont participé aux discussions. Cette recherche n'aurait été possible sans les dons de logiciels et de matériels des compagnies Mentor Graphics, Apple Corporation ainsi que de la CMC (Canadian Microelectronic Corporation).

Finalement, le projet IMAGE a bénéficié des supports financiers du Conseil de Recherches en Sciences Naturelles et en Génie (CRSNG) et du Fonds pour la Formation de Chercheurs et d'Aide à la Recherche (FCAR) des professeurs Jean-Louis Houle et Yvon Savaria. L'auteur remercie également l'Ecole Polytechnique de Montréal et le FCAR pour leurs supports sous forme de bourses d'études.

TABLE DES MATIERES

SOMMAIRE iv

ABSTRACT v

REMERCIEMENTS vi

LISTE DES FIGURES xii

LISTE DES TABLEAUX xiv

TABLE DES DÉFINITIONS ET ACRONYMES xv

CHAPITRE 1 - INTRODUCTION 1

CHAPITRE 2 - REVUE DE L'ÉTAT DE L'ART 4

2.1 Le traitement numérique d'images 4

2.2 Architectures de machines de TNI 7

2.2.1 Parallélisme 7

2.2.2 Parallélisme de voisinage 7

2.2.3 Parallélisme d'opérations 8

2.2.3.1 Architecture pipeline 8

2.2.3.2 Architecture IMDM 10

2.2.4 Parallélisme d'image 12

2.2.4.1 Caractéristiques 12
2.2.4.2 L'ordinateur CLIP 4 14
2.2.4.3 L'ordinateur MPP 16

CHAPITRE 3 - ARCHITECTURE DU SYTEME IMAGE 18
3.1 Description globale du système IMAGE 18
3.1.1 Le poste de travail SUN 19
3.1.2 Le panier à cartes 22
3.1.2.1 Le numérisateur MVP 22
3.1.2.2 Les répéteurs 24
3.1.2.3 La micro-machine 26
3.1.2.4 Le générateur d'adresses 26
3.1.2.5 La matrice de processeurs 26
3.2 Description de la machine IMAGE 27
3.3 L'interface VME 28
3.4 L'interface MVP 30
3.4.1 Généralités 30
3.4.2 Fonctionnement 31
3.5 La micro-machine 33
3.5.1 Le micro-séquenceur Am2910 33
3.5.2 La mémoire de micro-code 35
3.5.3 La file de commandes (FIFO) 38
3.6 Les communications 38
3.6.1 Les communications de l'ordinateur hôte vers IMAGE 40

3.6.2 Les communications de IMAGE vers l'ordinateur hôte 40

3.7 Le générateur d'adresses 41

CHAPITRE 4 - LA MATRICE DE PROCESSEURS 47

4.1 Vue globale 47

4.2 Le circuit intégré IMAGE-2 47

4.3 La mémoire externe 52

4.3.1 Généralités 52

4.3.2 Mode fenêtre 54

4.3.3 Mode pyramidal 55

4.3.4 Mode ligne 57

4.4 Circuits périphériques 58

4.4.1 Le traitement des bordures 58

4.4.1.1 Configuration cylindrique 60

4.4.1.2 Configuration cylindrique décalée 60

4.4.1.3 Registre K 60

4.4.2 Les multiplexeurs d'entrée et le registre K 61

4.5 La table de données 62

CHAPITRE 5 - IMPLANTATION D'ALGORITHMES 65

5.1 Introduction 65

5.2 Opérations de pixels 66

5.3 Opérations de voisinage 71

5.4 Transformations globales 75

5.4 Calculs statistiques 75

CHAPITRE 6 - MODÉLISATION ET VÉRIFICATION 80

6.1 Aspects de la simulation 80

6.2 Outils de modélisation et de simulation 81

6.2.1 Description du matériel 81

6.2.2 Processus de développement 81

6.3 Modélisation du c.i. IMAGE-2 85

6.3.1 Buts 85

6.3.2 Types de modèles 85

6.3.3 Description du modèle utilisé 87

6.3.4 Qualité de la modélisation 90

6.4 Vérification 92

6.4.1 Méthode utilisée 92

6.4.2 Environnement matériel 93

6.4.2.1 La plaquette de test 93

6.4.2.2 Le générateur de vecteurs et l'analyseur de signaux. 93

6.4.3 Environnement logiciel 97

6.4.3.1 Le micro-assembleur DECO 97

6.4.3.2 Le programme SUN2HP 99

6.4.3.3 Le programme de transfert HP2GEN 99

6.4.4 Vérification du système IMAGE 100

6.5 Performance et qualité des tests 100

CONCLUSION 102

BIBLIOGRAPHIE 104

Annexe A - Schémas des circuits de la machine IMAGE 107

Annexe B.1 - Le micro-assembleur DECO 114

Annexe B.2 - La table de symboles mnémoniques 143

Annexe C - Le programme SUN2HP 148

Annexe D - Le programme HP2GEN 151

Annexe E - Le programme SUN2IMAGE 154

Annexe F - Le programme SUN2HP 158

LISTE DES FIGURES

Figure 2.1 Etapes d'une opération de TNI 5
Figure 2.2 Schéma-bloc d'une architecture à parallélisme de voisinage . . 9
Figure 2.3 Opération de voisinage 9
Figure 2.4 Schéma-bloc de l'ordinateur Cytocomputer 11
Figure 2.5 Architecture IMDM 11
Figure 2.6 Interconnexions entre les processeurs 13
Figure 2.7 Image de 4x4 pixels d'une résolution de 3 bits 13
Figure 2.8 Schéma-bloc de l'ordinateur CLIP 4 15
Figure 2.9 Schéma-bloc d'un processeur de l'ordinateur CLIP 4 15
Figure 2.10 Schéma-bloc d'un processeur de l'ordinateur MPP 17
Figure 3.1 Partage des étapes d'une opération de TNI 18
Figure 3.2 Le système IMAGE 20
Figure 3.3 Schéma-bloc du système IMAGE 23
Figure 3 .4 Schéma-bloc du numérisateur MVP 25
Figure 3.5 Conversion des données du format parallèle à sériel 32
Figure 3.6 Schéma-bloc de l'interface du MVP 34
Figure 3 .7 Transfert d'images par les doubles tampons 3 4
Figure 3.8 Schéma-bloc de la micro-machine 3 7
Figure 3.9 Circuits de chargement du micro-code 37
Figure 3.10 Liens de communication entre les sous-systèmes 39
Figure 3.11 Schéma-bloc du générateur d'adresses 43
Figure 3.12 Décalage des adresses pour une opération de voisinage 43
Figure 4.1 La matrice de processeurs IMAGE-2 48
Figure 4.2 Schéma-bloc d'un processeur élémentaire 50

Figure 4.3 Exécution en pipeline d'une addition 8 bits 51

Figure 4.4 La photomicrographie du c.i. IMAGE-2 53

Figure 4.5 Mode d'adressage en fenêtre 56

Figure 4.6 Mode d'adressage pyramidal 56

Figure 4.7 Mode d'adressage en ligne 56

Figure 4.8 Configurations des bordures 59

Figure 4.9 La table de données 64

Figure 5.1 Opération de seuillage 68

Figure 5.2 Opération de seuillage avec pipeline 68

Figure 5.3 Organisation de la mémoire des PE pour un seuillage 68

Figure 5.4 Exemple de masque de convolution 72

Figure 5.5 Opération de convolution sur un pixel 72

Figure 5.6 Décalage des adresses des pixels voisins 73

Figure 6.1 Processus de développement d'un circuit intégré 82

Figure 6.2 Processus de développement du c.i. IMAGE-2 84

Figure 6.3 Différents modèles de simulation 91

Figure 6.4 Symboles du modèle INTERRUPTEUR utilisés 91

Figure 6.5 La plaquette de test 94

Figure 6.6 Démultiplexage des signaux de commande du c.i. IMAGE-2 . 96

Figure 6.7 Processus de vérification des c.i. IMAGE-2 101

Figure A.1 Schéma de l'interface du bus VME 108

Figure A.2 Schéma de l'interface du MVP 109

Figure A.3 Schéma de la micro-machine 110

Figure A.4 Schéma du générateur d'adresses 111

Figure A.5 Schéma de la matrice de processeurs 112

Figure A.6 Schéma d'une cellule d'IMAGE-2 113

LISTE DES TABLEAUX

Tableau 5.1(a) Séquence des adresses pour un seuillage en utilisant les deux registres d'adresse. 70

Tableau 5.1(b) Séquence des adresses pour un seuillage en utilisant un seul registre d'adresse. 71

Tableau 5.2 Séquence des adresses pour une opération de convolution . . . 74

Tableau 6.1 Les 12 états des signaux du simulateur Quicksim 89

Tableau 6.2 Résolution des contentions des signaux 89

TABLE DES DÉFINITIONS ET ACRONYMES

c.i.	: **c**ircuit **i**ntégré.
CAO	: **c**onception **a**ssistée par **o**rdinateur.
IUDM	: **i**nstruction **u**nique, **d**onnées **m**ultiples.
	(SIMD, Single Instruction, Multiple Data)
PAL:	: Programmable Array Logic.
PE	: **p**rocesseur **é**lémentaire.
TNI	: **t**raitement **n**umérique d'**i**mages.
MVP	: numérisateur d'images commercial.
TRF	: **t**ransformée **r**apide de **F**ourier.
VLSI	: **V**ery **L**arge **S**cale **I**ntegration (intégration à très grande échelle).
ITGE	: **i**ntégration à **t**rès **g**rande **é**chelle.
pixel	: **pi**cture **el**ement (élément d'image).
MOS	: procédé de fabrication à trois couches **m**étal-**o**xide-**s**emiconducteur.
CMOS	: **C**omplementary **M**etal-**O**xide-**S**emiconductor.
KIC	: éditeur de masques pour les ordinateurs SUN.

CHAPITRE 1 - INTRODUCTION

Le traitement numérique d'images (TNI) est l'un des domaines qui a profité grandement de l'évolution de l'électronique et de la microélectronique. En effet, le TNI nécessite un grand nombre de données qui doivent être traitées à un débit élevé. Grâce à l'évolution des deux dernières décennies, avec l'avènement des circuits ITGE (intégration à très grande échelle), les machines de TNI peuvent maintenant devenir suffisamment puissante et à un coût abordable pour justifier des applications pratiques et commerciales. C'est pourquoi, à l'Ecole Polytechnique de Montréal, le projet de recherche IMAGE a démarré avec comme objectif l'étude et l'implantation d'un système de traitement numérique d'images basé sur des circuits ITGE développés à cette institution.

Le projet IMAGE a débuté en 1987 avec le développement d'un circuit intégré utilisant une architecture parallèle pour le traitement numérique d'images [1]. Sous la direction des professeurs Jean-Louis Houle et Yvon Savaria, les étudiants Daniel Audet et Claude Cyr ont conçu un premier prototype du circuit intégré appelé IMAGE, qui a été suivi par la deuxième et courante version IMAGE-2.

Le projet de maîtrise, qui fait l'objet du présent mémoire, consiste à concevoir et implanter un système de TNI en utilisant le circuit intégré (c.i.) IMAGE-2 mis au point au département de génie électrique. Le coeur du système sera composé d'une matrice bidimensionnelle de processeurs fonctionnant en parallèle dans un environnement IUDM (instruction unique, données multiples). Un

poste de travail de type SUN 3[1] servira de processeur hôte pour le développement d'algorithmes et comme une interface avec les utilisateurs.

Ce projet, jusqu'à ce jour, a connu deux étapes importantes, premièrement le développement du c.i. IMAGE-2, et deuxièmement, la conception et la construction de la machine IMAGE qui agit comme coprocesseur à un ordinateur SUN.

Les deux étapes importantes du développement d'IMAGE-2 auxquelles l'auteur a participé sont la simulation et la vérification. La conception d'IMAGE-2 a coïncidé avec l'acquisition des postes de travail APOLLO[2] et des logiciels de Mentor Graphics[3]. Ces logiciels permettent de modéliser et de simuler des circuits électroniques aussi bien au niveau analogique que logique. Ces outils nous ont permis de modéliser le c.i. IMAGE-2 et de simuler la plupart de ses fonctions au niveau fonctionnel ainsi que quelques sections critiquent au niveau analogique. Après la fabrication d'IMAGE-2, il a fallu vérifier sa fonctionnalité, c'est-à-dire s'il se comportait tel que prévu au niveau logique. Ceci a nécessité le développement de nouveaux outils logiciels, tels qu'un micro-assembleur et des programmes de transfert afin d'envoyer des vecteurs de test au circuit intégré. Cet environnement logiciel et matériel a démontré rapidement la fonctionnalité d'IMAGE-2, ce qui nous a permis d'amorcer la conception de la machine IMAGE.

---

[1]SUN est une marque déposée de SUN Microsystems, Inc.

[2]APOLLO est une marque déposée de Apollo Computer, Inc.

[3]Mentor Graphics est une marque déposée de Mentor Graphics Corp.

Déjà plusieurs machines existantes utilisent une architecture parallèle pour le TNI. Après l'étude de ces machines et des algorithmes types de TNI pour lesquels la machine IMAGE est conçue, une architecture a été développée qui assure une certaine performance tout en gardant suffisamment de flexibilité au niveau fonctionnel. La construction du prototype commence par l'implantation d'un seul c.i. qui sera suivi par une matrice de 8 x 8 processeurs. Cette architecture devra aussi pouvoir accommoder une matrice de 16 x 16 processeurs par l'ajout de c.i. IMAGE-2 supplémentaires.

Le présent document décrit d'abord, au chapitre 2, les différentes architectures parallèles qui ont été implantées dans des machines de TNI tels que les ordinateurs CLIP 4, Cytocomputer et MPP. Les deux chapitres suivants décrivent l'architecture de la machine IMAGE par la méthode descendante. Le chapitre 3 donne d'abord un aperçu global du système IMAGE. Celui-ci est composé de la machine IMAGE, d'un numérisateur d'images et d'un ordinateur hôte de type SUN 3. Ce chapitre décrit ensuite plus précisément la machine IMAGE et explique les architectures de ses différents sous-circuits. Le chapitre suivant est consacré à la matrice de processeurs qui forme le coeur du système IMAGE. Cette matrice inclue des circuits supplémentaires qui permettent d'augmenter d'une façon significative la performance de plusieurs algorithmes de TNI. Le chapitre 5 montre, par l'implantation d'algorithmes types, la performance de la machine IMAGE et l'utilisation de ses ressources. Finalement, le chapitre 6 décrit la manière utilisée pour tester IMAGE-2 grâce aux ordinateurs SUN et aux appareils de mesures HP du laboratoire de ITGE qui ont été mis en commun pour faciliter la vérification d'IMAGE-2.

CHAPITRE 2 - REVUE DE L'ÉTAT DE L'ART

2.1 Le traitement numérique d'images

Dans le présent document, le traitement numérique d'images se limite au sens large à deux grands aspects[1]:

- le graphisme sur ordinateur;
- l'analyse pour la reconnaissance de formes.

Le graphisme en général concerne tout ce qui a rapport à la création et à la transformation d'une image pour améliorer son aspect visuel dans un but précis. On utilise le graphisme par ordinateur dans les outils de CAO, les affichages de simulateurs de vol ou simplement comme moyen d'expression de l'art.

D'un autre côté, l'analyse d'images est surtout orientée vers l'extraction d'information ou de caractéristiques spécifiques d'une image en transformant un amas de pixels ("**pic**ture **el**ements") en un nombre limité de résultats. Par exemple sur une photographie aérienne, on voudra mettre en évidence les cours d'eau, ou bien sur des chèques bancaires, on voudra détecter les fausses signatures.

Le système IMAGE, qui fait l'objet du présent mémoire, est plutôt orienté vers l'analyse d'images, c'est pourquoi nous utilisons, comme dans les publications techniques, le terme 'traitement numérique d'images', (TNI) pour désigner en fait l'analyse d'images.

Une opération de TNI peut se diviser en plusieurs étapes telles que représentées à la figure 2.1.

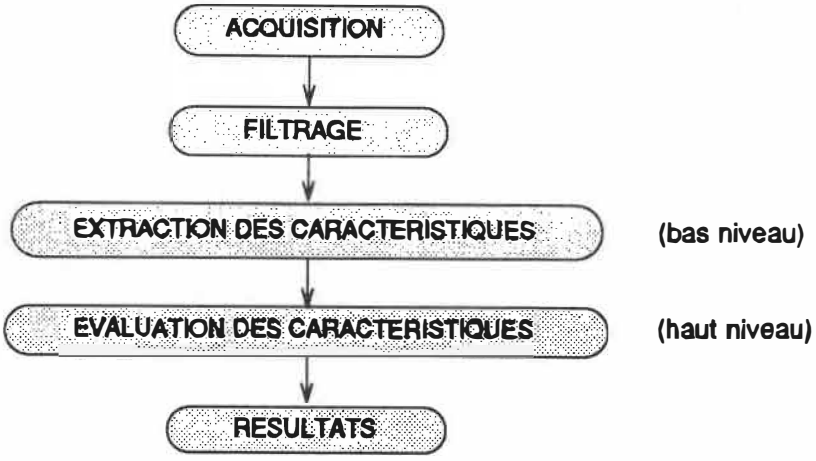


Figure 2.1 - Etapes d'une opération de TNI

L'image est d'abord numérisée à l'aide d'une caméra ou d'un satellite par exemple, et est ensuite filtrée, de façon analogique ou numérique, afin d'éliminer le bruit et le chevauchement (aliasing) dans l'image reçue. A la troisième étape, les caractéristiques de l'image telles que les intensités maximales et minimales, l'histogramme d'intensité et les arêtes sont extraits à l'aide d'opérations simples. L'évaluation des caractéristiques, par contre, nécessite des opérations plus complexes et englobe des analyses telles que la reconnaissance de formes, la restauration d'images et la vision artificielle. Finalement, on obtient le résultat de ces calculs qui peut être simplement binaire (oui ou non, 0 ou 1) ou prendre la forme d'une nouvelle image.

Le système Image réalise toutes ces opérations d'une façon qui sera expliquée dans les chapitres suivants. Brièvement, l'ordinateur hôte effectue les opérations complexes nécessaires à l'évaluation finale du problème, tandis que le

coeur du système, soit la matrice de processeurs IMAGE-2 [2], est conçu pour calculer efficacement les opérations de bas niveau.

Depuis le début du développement des machines de TNI, on s'est surtout concentré à l'optimisation des algorithmes de bas niveau parce qu'ils sont arithmétiquement simples. La plupart de ces algorithmes ont des caractéristiques communes qui devront se refléter dans la conception de la machine si on veut atteindre de grandes performances. Parmi ces caractéristiques, notons [3][4]:

- des opérations élémentaires généralement simples ;
- des opérations qui se répètent sur chaque pixel;
- un grand nombre de données,

  par exemple, une image de 512x512 pixels de 8 bits occupe 262 144 octets;
- un taux de transfert élevé,

  par exemple, pour respecter la norme NTSC (National Television Standard Code) de transfert en temps réel, on doit acheminer au processeur 262 144 pixels en 1/30 seconde, soit 64 méga-bits par seconde pour une profondeur de 8 bits par pixel;

Ces critères exigeants ont amené le développement de plusieurs architectures d'ordinateur dédiées presque exclusivement au TNI.

2.2 Architectures de machines de TNI

2.2.1 Parallélisme

Depuis les années soixante, plusieurs architectures ont été proposées pour résoudre des problèmes de traitements numériques d'images. A cause des caractéristiques de ces algorithmes énumérées précédemment, plusieurs concepteurs ont suivis la voie du parallélisme. Étant donné la nature simple des opérations, on utilise plusieurs processeurs simples fonctionnant en parallèle. Lougheed et Swenger [3] démontrent bien l'augmentation de performance que procure une matrice de processeurs simples comme le Cyto-HSS par rapport à un processeur unique mais relativement performant comme un VAX 11-780 pour des algorithmes typiques de bas niveaux de TNI. Ils démontrent, en utilisant des algorithmes simples pouvant tirer profit du parallélisme du Cyto-HSS, qu'on peut obtenir un temps de traitement beaucoup plus court avec une machine plus simple et moins coûteuse, mais conçue pour l'application spécifique qu'est le traitement numérique d'images.

Daniellson et Levialdi [1] ont proposé une façon intéressante de classer les différentes formes de parallélisme utilisées dans les machines de TNI existantes: le parallélisme de voisinage, le parallélisme d'opérations et le parallélisme d'images.

2.2.2 Parallélisme de voisinage

Tel que montré à la figure 2.2 [5], avec cette forme de parallélisme, l'image remplit de façon sérielle les registres à décalage et la matrice de registres,

de telle sorte qu'à chaque cycle, les huit pixels des registres N1 à N8 formant le voisinage immédiat du pixel central P alimentent simultanément un processeur logique effectuant une opération de transformation sur le pixel central. Cette opération de transformation, représentée à la figure 2.3, est très utilisée et très puissante pour le TNI.

L'une des première machine à utiliser le parallélisme de voisinage a été développée par Preston pour la firme Perkin-Elmer sous le nom de Diff3[6]. Cette machine a la particularité d'utiliser une fenêtre hexagonale [7] au lieu d'une fenêtre carrée de 3x3. Le Picap I [8] développé à l'Université de Linkoping par Kruse utilise aussi cette technique pour des images de 64 x 64 pixels de 4 bits de résolution.

2.2.3 Parallélisme d'opérations

Les architectures à parallélisme d'opérations sont plus complexes et peuvent être divisées principalement en deux catégories: les architectures pipelines et les architectures IMDM (instructions multiples, données multiples).

2.2.3.1 Architecture pipeline

Dans une architecture pipeline, plusieurs étages connectés entre eux dans un pipeline opèrent simultanément mais effectuent des opérations différentes. Cette technique est utilisée depuis plusieurs décennies dans différentes applications, mais le Cytocomputer [9,3] est probablement la machine qui implante le mieux cette architecture avec pas moins de 88 étages tel que montré à la figure 2.4. Les

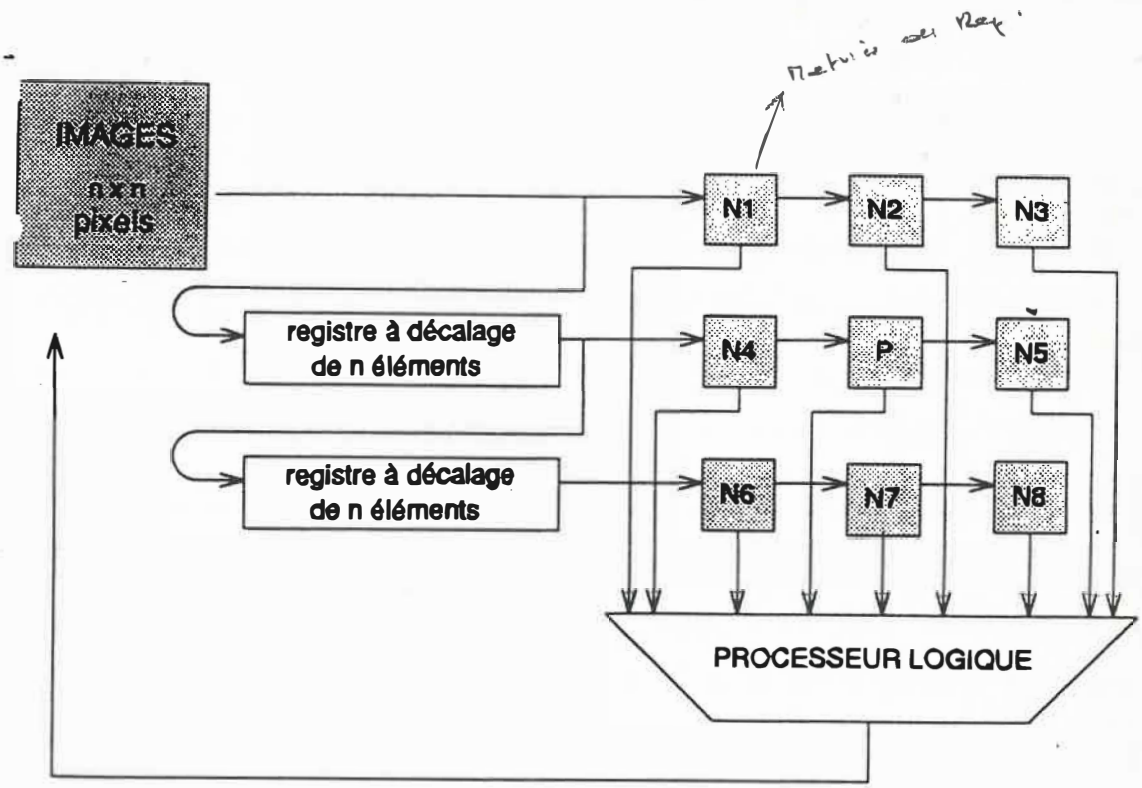
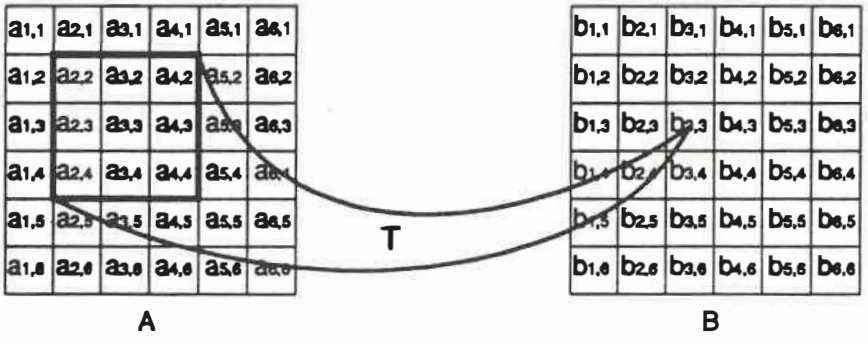


Figure 2.2 - Schéma-bloc d'une architecture à parallélisme de voisinage
(un pixel P et les huit registres des voisins N1 à N8)



$$b_{i,j} = T(a_{i-1,j-1}, a_{i,j-1}, a_{i,j+1}, a_{i-1,j}, \ldots, a_{i+1,j+1})$$

i.e. B = TA

Figure 2.3 - Opération de voisinage

images entrent dans le pipeline par une extrémité et, après avoir été traitées successivement par les différents étages, sortent à l'autre extrémité aussi rapidement.

Plus récemment, la machine Warp [10], mise au point à l'Université de Carnegie-Mellon, compte dix étages dans son pipeline. Chacun des dix processeurs formant le pipeline est muni d'une mémoire de 32k mots de 32 bits et d'un ALU à virgule flottante qui le rend très performant pour les algorithmes de TNI complexes comme les transformées rapides de Fourier (TRF).

2.2.3.2 Architecture IMDM

Dans les machines de type IMDM, tel que montré à la figure 2.5, chaque processeur a sa propre mémoire, mais peut aussi accéder à une mémoire partagée et communiquer avec les autres processeurs à l'aide d'un réseau d'interconnexions. Cette architecture est relativement complexe en terme de construction et au moins autant en terme de programmation puisque chaque module peut être programmé indépendamment. Cependant, c'est le prix d'une flexibilité et d'une performance plus grande particulièrement pour les algorithmes de haut niveau. Le PASM [11], développé à l'Université de Purdue, implante une architecture hybride de type IUDM, expliquée au paragraphe suivant, et IMDM dans laquelle 16 unités de contrôle indépendantes commandent chacune une machine IUDM composée de 64 processeurs.

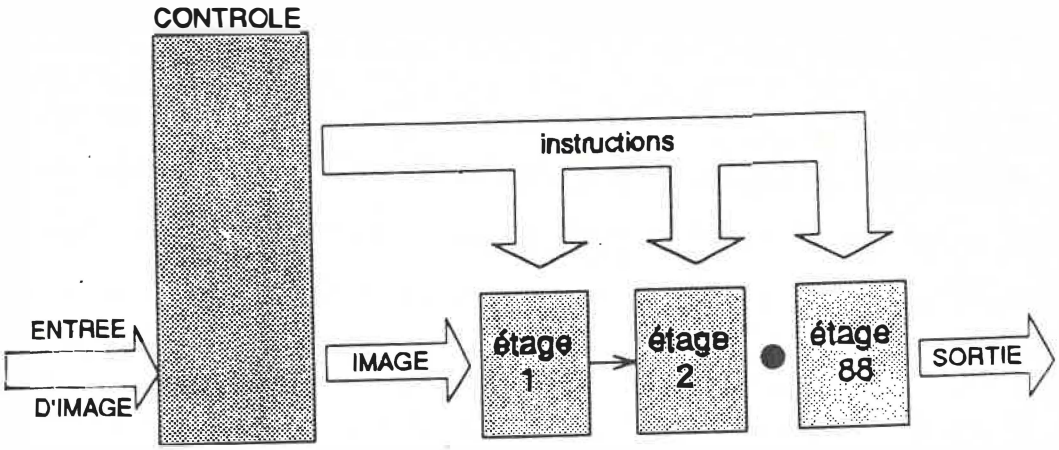


Figure 2.4 - Schéma-bloc du Cytocomputer

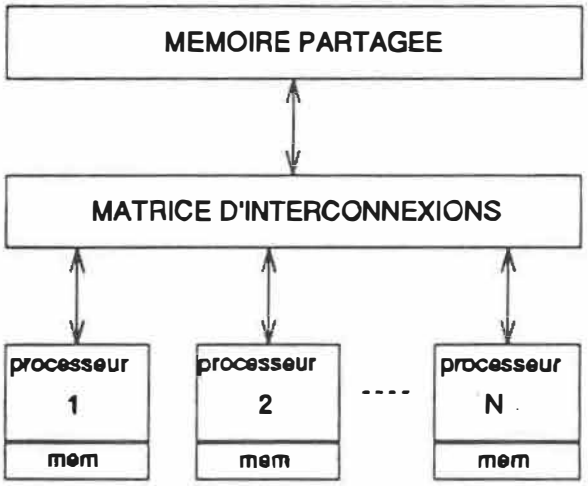


Figure 2.5 - Architecture IMDM

2.2.4 Parallélisme d'image

Les architectures à parallélisme d'image utilisent une matrice bidimensionnelle de processeurs, chacun opérant sur une section différente de l'image. Elles sont les mieux adaptées pour le TNI puisque les données du problème à résoudre, i.e. les images, ont la même structure bidimensionnelle. Ces architectures, aussi appelées IUDM (instruction unique, données multiples) [12] jouissent d'une grande popularité dans le domaine du TNI à cause de leur simplicité et de leur facilité d'implantation dans les circuits intégrés.

2.2.4.1 Caractéristiques

Unger a été le premier, en 1958 [13], à proposer une matrice de processeurs bidimensionnelle spécialisée pour le traitement numérique d'images. Parmi les caractéristiques principales, qui sont encore à la base des machines construites aujourd'hui, on retrouve des processeurs binaires simples interconnectés à leurs voisins pour pouvoir effectuer des opérations de transformations avec voisinages, tel que représenté à la figure 2.6. Chaque processeur est sériel par bit pour minimiser la complexité des interconnexions, et chacun possède aussi sa propre mémoire locale pour emmagasiner une partie de l'image et des résultats intermédiaires. L'architecture sérielle par bit permet à la machine de s'adapter à différentes résolutions de l'image et à différentes précisions de calculs. De cette façon, on peut aussi bien traiter une image binaire qu'une image de 8 bits avec 256 niveaux de gris. Tel que montré à la figure 2.7, une image de n bits de résolution est considérée comme n plan de pixels où tous les pixels d'un plan sont traités simultanément par tous les processeurs. Effectivement, une seule unité de

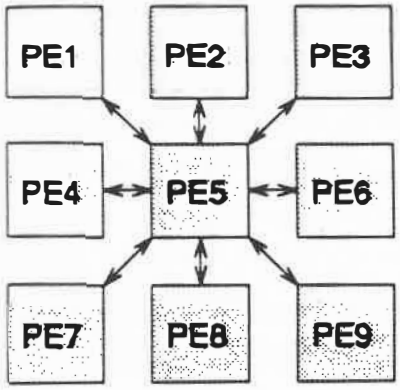


Figure 2.6 - Interconnexions entre les processeurs

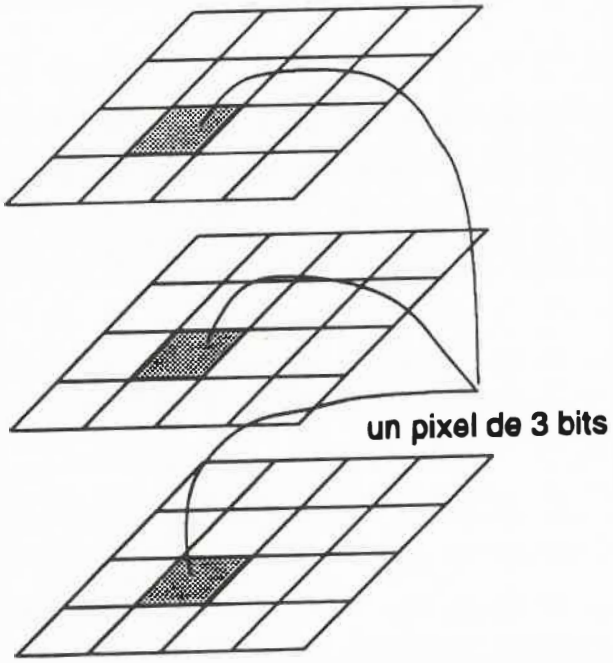


Figure 2.7 - Image de 4 x 4 pixels d'une résolution de 3 bits

contrôle envoie la même commande à tous les processeurs, mais chacun opère avec des données différentes, d'où le nom d'architecture à instruction unique et données multiples (IUDM).

Au début, les machines étaient dispendieuses et difficiles à construire, mais avec l'avènement de l'intégration à très grande échelle (ITGE,"VLSI"), plusieurs machines ont été construites suivant l'architecture IUDM. Effectivement, les circuits ITGE se prêtent bien à ce genre de problème étant donnée la nature répétitive et simple des processeurs. Nous en décrivons ici brièvement quelques-unes.

2.2.4.2 L'ordinateur CLIP 4

Duff a développé, au Collège Universitaire de Londres, une série de processeurs spécialisés pour le TNI [14], le dernier né étant le CLIP 4. Le schéma global de la figure 2.8 montre les 9216 processeurs arrangés en matrice carrée de 96x96. Pour obtenir des taux de transfert élevés aux entrées et sorties, deux banques de registres à décalage permettent la transformation des données sérielles venant d'une caméra en un format compatible à la matrice de processeurs. De façon semblable, des registres à décalage en sortie font la transformation inverse pour alimenter le moniteur.

Chaque processeur binaire représenté à la figure 2.9 compte 32 registres de un bit, un additionneur à un bit, une retenue, deux registres à un bit d'usage général et un circuit logique (ALU) qui réalise les 16 fonctions logiques de deux bits. De plus, un réseau d'interconnexions permet à chaque processeur de

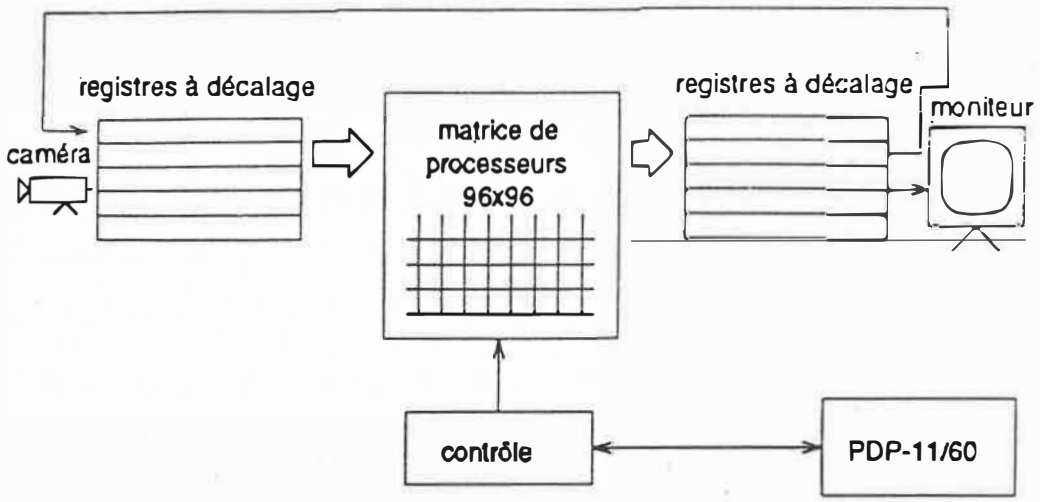


Figure 2.8 - L'ordinateur CLIP 4

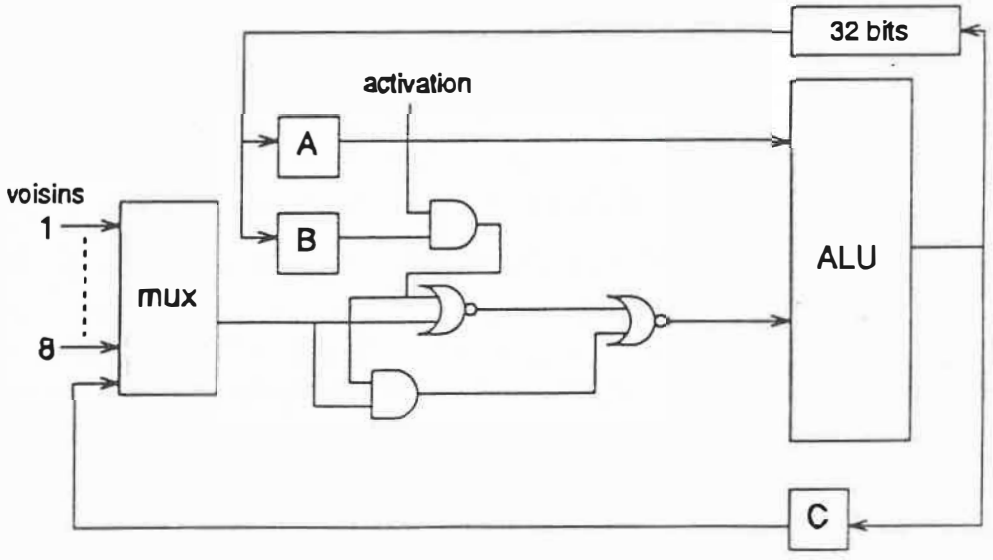


Figure 2.9 - Processeur de l'ordinateur CLIP 4

communiquer directement avec ses huit voisins immédiats.

2.2.4.3 L'ordinateur MPP

Le MPP (Massively Parallel Processor) a été conçu par la société Goodyear pour la NASA par Batcher [15]. Cette machine se distingue par sa grande quantité de processeurs arrangés en matrice de 128 x 128 PEs (processeurs élémentaires).

Pour pouvoir alimenter suffisamment rapidement ces processeurs, un circuit spécial est responsable de la transformation des données qui peuvent être transférées à une vitesse de 160 méga-octets par seconde[16].

Les processeurs, représentés à la figure 2.10, sont encore une fois très simples, chacun possédant un accumulateur, un ALU, quatre registres à un bit et un registre à décalage de longueur variable qui permet d'accélérer les multiplications. De plus, chaque processeur possède une mémoire locale d'un kilobit et est relié à ses quatre voisins pour les opérations de voisinage.

Le système IMAGE, qui utilise aussi une architecture IUDM, est décrit dans les deux prochains chapitres.

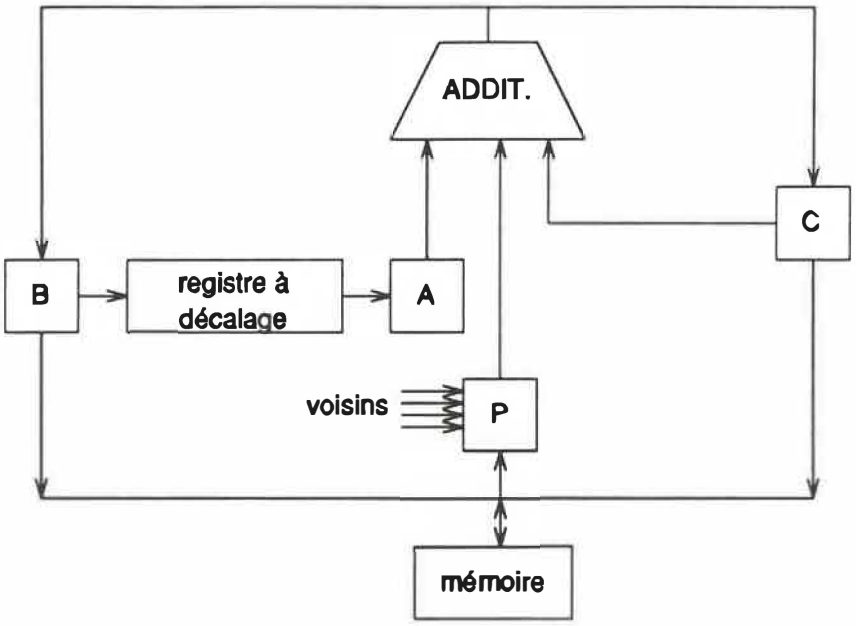


Figure 2.10 - Le processeur de l'ordinateur MPP

CHAPITRE 3 - ARCHITECTURE DU SYTEME IMAGE

3.1 Description globale du système IMAGE

Le système de TNI IMAGE a été conçu pour effectuer des traitements complets de TNI, par conséquent il contient les circuits et les interfaces nécessaires à l'acquisition d'images, à leur traitement, à leur stockage et à la représentation de résultats. La figure 3.1 montre les étapes que doit franchir le système IMAGE pour réaliser une opération type de TNI. A chacune de ces étapes correspond une ou plusieurs parties du système qui seront expliquées plus loin.

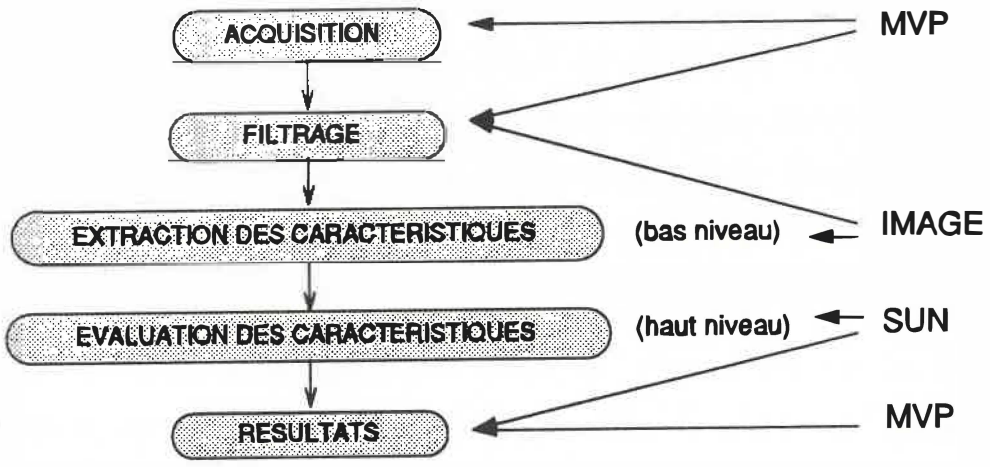


Figure 3.1 - Etapes d'une opération de TNI

Les étapes de traitement proprement dites peuvent être divisées en deux grandes catégories, les routines de bas niveau et les routines de haut niveau. Les routines de bas niveau impliquent des opérations simples, répétitives et facilement mises en parallèle, tandis que les routines de haut niveaux nécessitent des opérations beaucoup plus complexes, difficilement mises en parallèle, mais elles

utilisent par contre un plus petit nombre de données, réduites par les algorithmes de bas niveaux.

Le système IMAGE reflète cette dichotomie des algorithmes dans la structure des différents processeurs qui le composent. Les algorithmes de haut niveaux sont effectués par un ordinateur SUN 3, de type poste de travail, tandis que la matrice de processeurs parallèles IMAGE-2 s'occupe des algorithmes de bas niveaux.

La figure 3.2 donne un aperçu de l'organisation physique du système. Le poste de travail SUN est relié à un panier à cartes qui contient le reste du système, soit le répéteur, le MVP et la machine IMAGE composée de trois cartes: la matrice de processeurs est sur la première, la deuxième contient la micro-machine, tandis que le générateur d'adresses et les interfaces du MVP et du bus VME sont câblés sur la troisième.

3.1.1 Le poste de travail SUN

Le but du projet étant d'avoir un système complet de TNI, un ordinateur hôte devait commander la machine IMAGE. Ceci facilite grandement le développement et le déverminage du système ainsi que le développement de programmes de tests et d'applications. L'ordinateur SUN a été choisi étant donné la grande disponibilité de ces machines dans notre département de génie électrique et le laboratoire de ITGE (Intégration à Très Grande Echelle). Il commande, par l'intermédiaire du bus VME, toutes les autres parties du système. L'usager utilise le SUN, par le système d'exploitation UNIX, pour interagir avec le système IMAGE

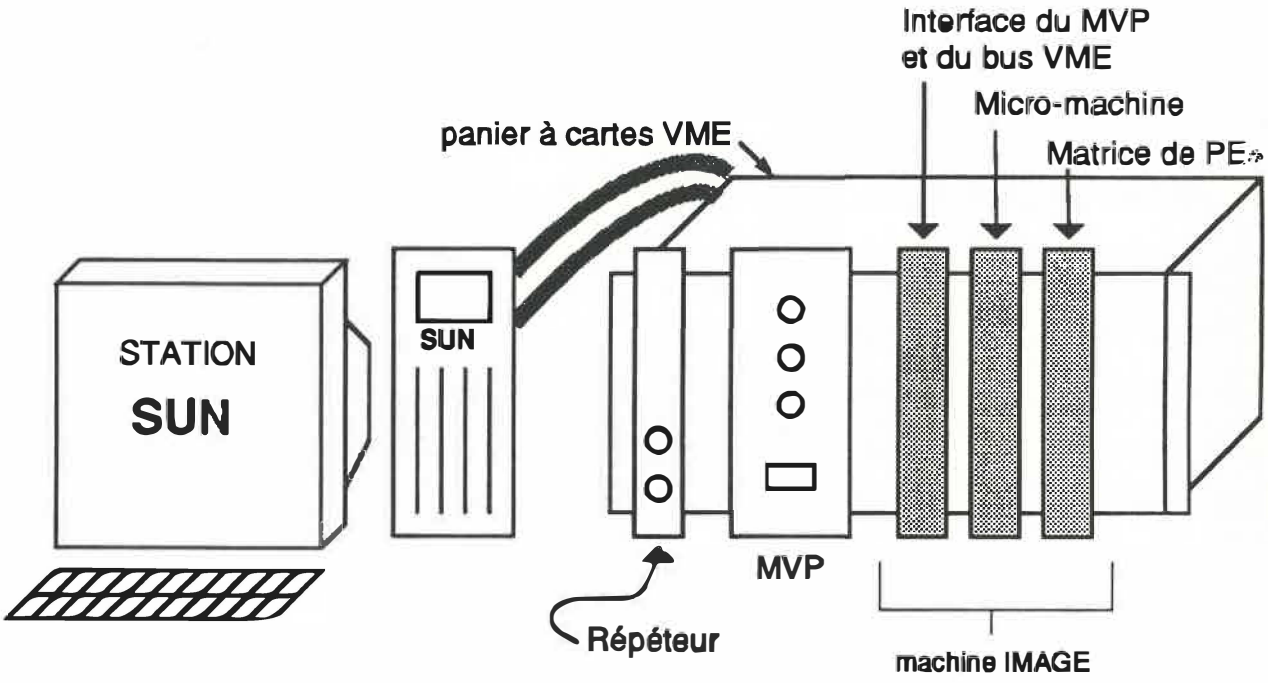


Figure 3.2 - Organisation physique du système IMAGE

et pour programmer les algorithmes et les programmes de contrôle des différents modules. Lors d'une opération de TNI, l'ordinateur SUN 3, d'une architecture à processeur simple de type MC68020, effectue aussi les algorithmes de haut niveau qui sont difficilement mis en parallèle pour la matrice de processeurs IMAGE-2. Le bus d'expansion du SUN, conforme à la norme VME, facilite le contrôle des circuits supplémentaires tels que le MVP et la machine IMAGE qui sont accessibles directement et rapidement.

3.1.2 Le panier à cartes

Le schéma-bloc de la figure 3.3 montre l'ensemble des circuits qui se trouvent dans le panier à cartes à l'extérieur du SUN. Ces circuits sont composés d'un répéteur de bus, du numérisateur MVP et de la machine IMAGE proprement dite formée de trois cartes: la micro-machine, le générateur d'adresses et la matrice de processeurs.

3.1.2.1 Le numérisateur MVP

Le numérisateur d'images MVP [17] est une carte qui nous a été gracieusement offerte par la firme MATROX Electronic Systems. Cette carte permet de faire de l'acquisition et du traitement numérique d'images ainsi que du graphisme. Elle se branche sur le bus VME et reçoit ses commandes de l'ordinateur SUN. Les images peuvent être transférées d'une part dans la mémoire secondaire à disque du SUN par le bus VME grâce à une interface à haute vitesse, et d'autre part avec la matrice de processeurs à l'aide d'un connecteur d'expansion. La figure 3.4 montre un schéma-bloc du MVP dont les caractéristiques les plus pertinentes sont:

- un méga-octets de mémoire permettant d'emmagasiner quatre images de 512x512x8 bits (FB0 à FB3);
- un cpu MC68000 responsable des communications avec le SUN;
- un processeur graphique rapide HD68384;
- des tables de conversion (LUT) et un ALU pour le traitement d'images;

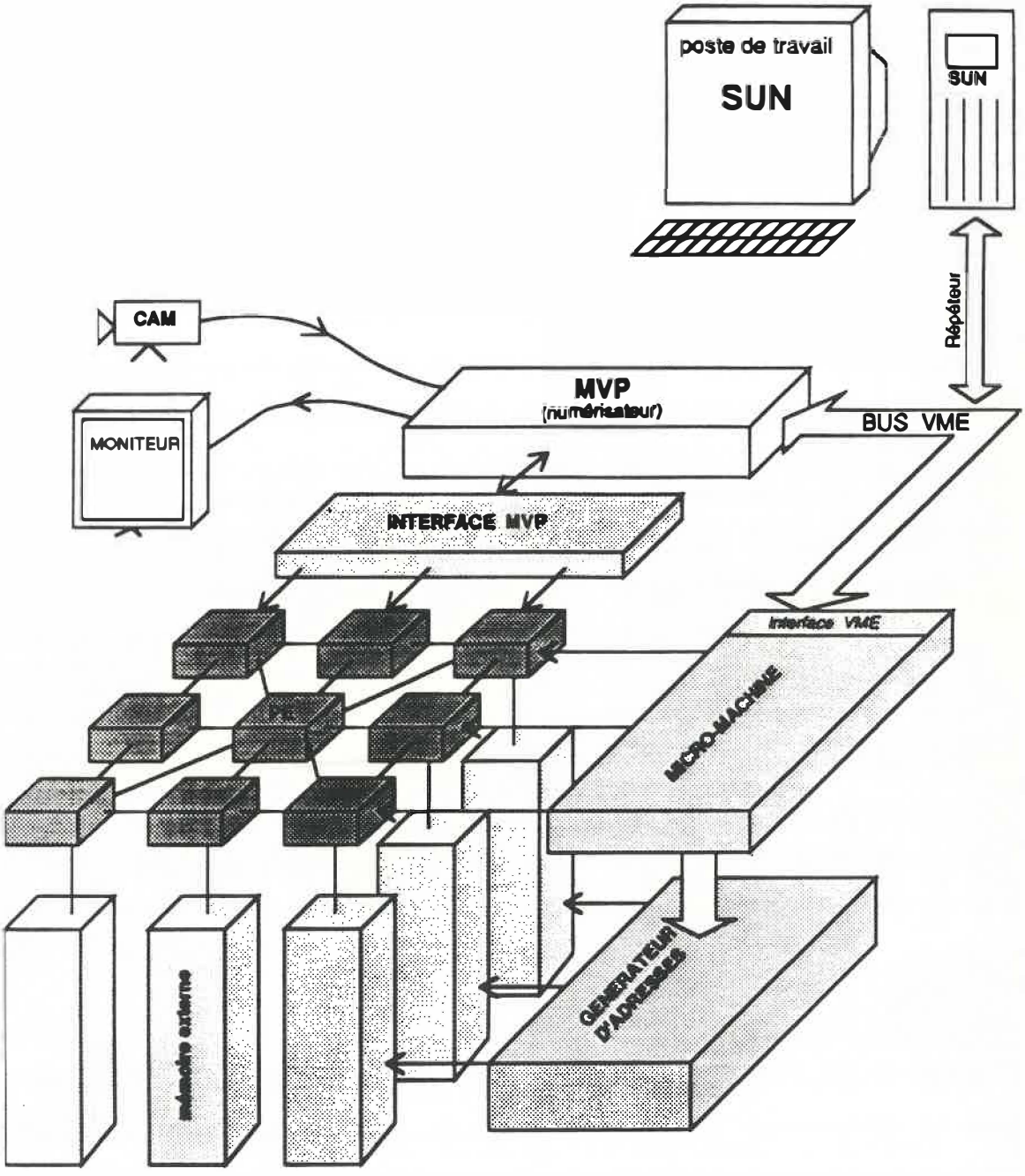


Figure 3.3 - Schéma-bloc du système IMAGE

- un numérisateur d'images à quatre entrées;
- un bus d'expansion permettant le transfert d'images à la matrice de processeurs IMAGE-2.

Dans le système IMAGE, le numérisateur MVP sert à l'acquisition d'images et à l'affichage des images résultantes. Les images peuvent être transférées à la machine IMAGE à la même vitesse qu'elles peuvent être numérisées, soit un trentième de seconde, i.e. près de 8 méga-octets par seconde.

3.1.2.2 Les répéteurs

Tel que montré aux figures 3.2 et 3.3, la machine IMAGE est construite à l'extérieur du SUN dans un panier à cartes pouvant contenir 21 cartes de format VME-6U. Ce panier est relié au SUN par quatre câbles plats à 50 conducteurs de deux mètres et un répéteur.

La première fonction du répéteur est d'amplifier tout les signaux venant du bus VME à l'intérieur du SUN vers le panier à cartes. Sa deuxième fonction est d'offrir une isolation électrique entre le bus VME du SUN et le bus VME du panier utilisé pour notre montage. De cette façon, une erreur de manipulation lors des essais ne risque pas d'endommager le SUN. De plus, ce montage permet de travailler aisément à l'extérieur du SUN pour le développement et le déverminage des cartes qui constitueront le système complet.

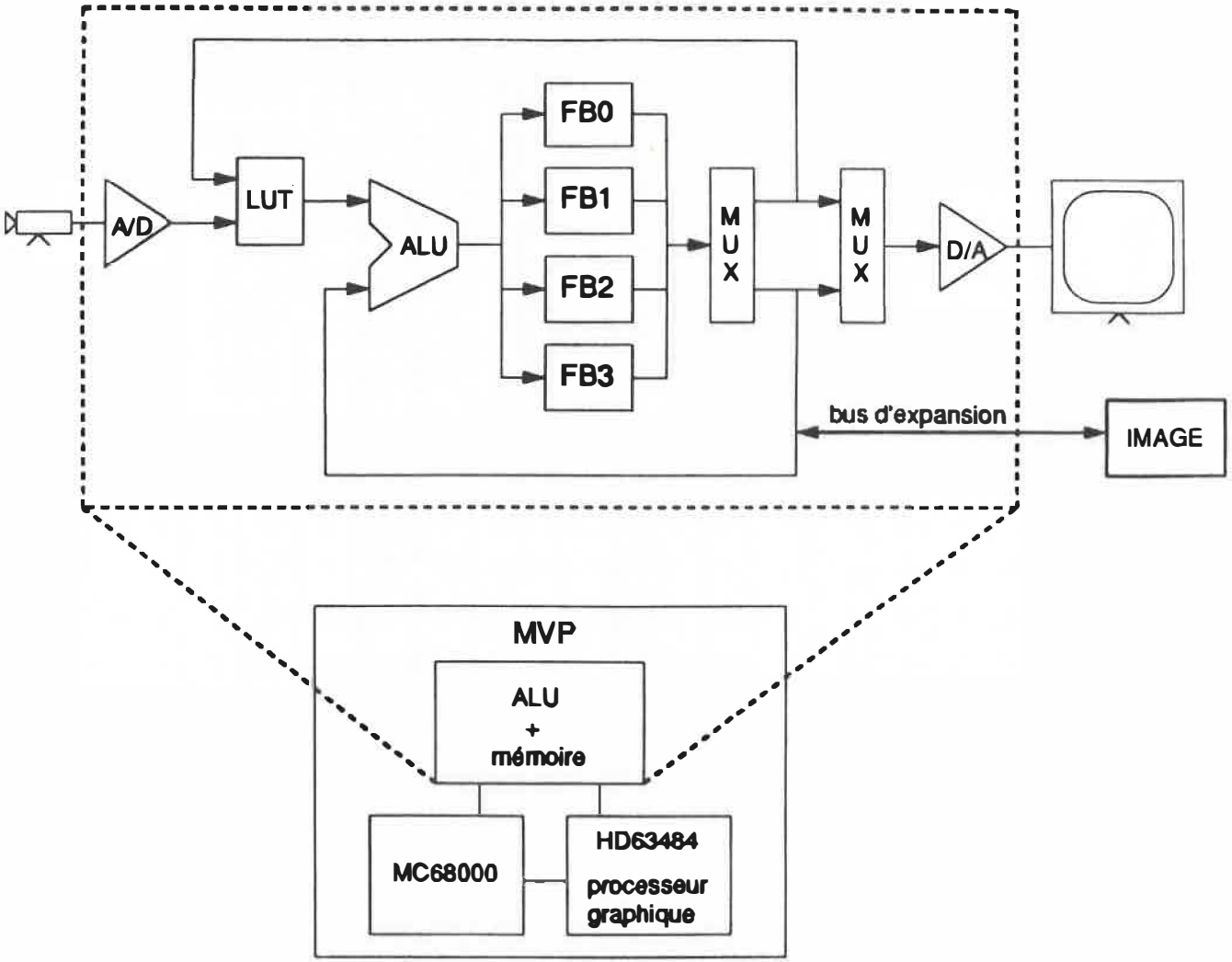


Figure 3.4 - Schéma-bloc du numérisateur MVP

3.1.2.3 La micro-machine

La micro-machine génère, à chaque phase d'horloge, le micro-mot nécessaire au contrôle des processeurs IMAGE-2 et au reste des circuits de la machine IMAGE. Le coeur de la micro-machine est composé d'un micro-séquenceur de type Am2910A opérant à une fréquence de 10 MHz et d'une mémoire de micro-code contenant 4096 mots de 96 bits. La mémoire de micro-mot contient, en fait, les routines qui seront exécutées par les processeurs IMAGE-2. Cette mémoire est chargée par l'ordinateur hôte par l'intermédiaire du bus VME. La micro-machine est expliquée plus en détail à la section 3.5.

3.1.2.4 Le générateur d'adresses

Le générateur d'adresses s'occupe de calculer une séquence d'adresses nécessaire pour l'accès aux données dans les algorithmes de TNI. Les adresses font référence aux mémoires externes des processeurs IMAGE-2. La synchronisation et le fonctionnement du générateur d'adresses sont gérés par le micro-séquenceur. La section 3.7 décrit d'une façon plus détaillée le fonctionnement du générateur d'adresses.

3.1.2.5 La matrice de processeurs

La matrice de processeurs forme le coeur de la machine IMAGE. Tel que montré à la figure 3.3, elle est formée d'une matrice bidimensionnelle de processeurs IMAGE-2 interconnectés entre eux, chacun ayant sa propre mémoire

externe. Le chapitre quatre est consacré à son architecture et à ses circuits périphériques.

3.2 Description de la machine IMAGE

Le choix de l'architecture de la machine IMAGE (dorénavant, le mot IMAGE signifiera la machine IMAGE) a été guidé principalement par l'architecture du c.i. IMAGE-2 et par les objectifs de simplicité qu'on s'était imposés.

Une architecture de type IUDM a été choisie pour la matrice de processeurs IMAGE-2 pour deux raisons. Premièrement, ce type d'architecture a un très bon rapport performance/complexité tel qu'il a été démontré dans plusieurs applications commerciales[14][15]. Deuxièmement, c'est pour ce type d'architecture que le c.i. IMAGE-2 a été développé. La machine IMAGE est composée de plusieurs sous-circuits dont les principaux ont été expliqués brièvement à la section précédente. Ces sous-circuits sont expliqués ici au niveau des blocs fonctionnels, mais le lecteur est invité à consulter les annexes du présent mémoire et le rapport technique du système IMAGE [18] s'il est intéressé aux schémas détaillés des circuits.

Tel que montré dans le diagramme-bloc de la figure 3.3, IMAGE est divisée en cinq sections intimement reliées: l'interface VME, l'interface MVP, la matrice de processeurs, la micro-machine et le générateur d'adresses.

3.3 L'interface VME

Toutes les informations que s'échangent l'ordinateur hôte et la machine IMAGE transitent par le bus VME [19]. Le schéma détaillé des circuits d'interface avec le bus VME est représenté à l'annexe A.1.

Les accès du bus VME à IMAGE sont synchronisés à l'aide d'un PAL (Programmable Array Logic), tandis que des circuits de comparaison d'adresses permettent de choisir l'adresse de l'espace mémoire occupée par IMAGE. IMAGE est adressable directement par l'ordinateur SUN et est perçue comme une région de mémoire de 8 kilo-mots de 32 bits dans l'espace de mémoire virtuelle de l'ordinateur SUN.

Cette espace de 8 kilo-mots est découpée en 8 sous-régions de la façon suivante:

1) table de données;
2) table de données;
3) table de données;
4) table de données;
5) registre K;
6) registre de l'adresse du générateur d'adresses;
7) registre de statut et de contrôle;
8) contrôle de la micro-machine.

Une table de données, qui peut être modifiée par l'ordinateur hôte et par IMAGE, occupe les quatre premiers kilo-mots. Cette table, large de huit bits, contient différents paramètres utilisés lors de l'exécution d'algorithmes. Son utilisation est expliquée plus en détails au chapitre cinq.

Le registre K, large de huit bits, peut être chargé d'un résultat global provenant de la matrice de processeurs, ou d'une valeur de la table de données. Le rôle et l'utilisation de ce registre est aussi expliqué en détail au chapitre cinq.

La sixième région est occupée par le registre de sortie du générateur d'adresses. Ce registre peut mémoriser une adresse calculée par le générateur d'adresses. Le section 3.7 montre l'implantation de ce registre.

Les registres de statut et de contrôle occupent la septième région. Ces registres maintiennent différentes valeurs du statut d'IMAGE et servent à contrôler certaines fonctions de la machine.

Finalement, la dernière région sert au chargement et à la vérification de la mémoire de micro-code. Ces fonctions sont expliquées à la section 3.5 dans la description de la micro-machine.

L'interface VME commande aussi les demandes d'interruption originant d'IMAGE pour l'ordinateur hôte. Le rôle des interruptions est expliqué à la section 3.6.

3.4 L'interface MVP

3.4.1 Généralités

Dans les machines de traitement numérique d'images, le problème d'acquisition et de transfert des images a toujours des particularités propres au système dans lequel la machine de TNI est implantée, par contre l'architecture globale de la matrice de processeurs IUDM est sensiblement la même. Etant donné la puissance de calcul de l'architecture IUDM, le temps de transfert des images peut devenir un paramètre important de la performance globale de la machine. Dans le cas de l'ordinateur MPP [15][16] et de la Connection Machine [20], les images sont emmagasinées dans l'ordinateur hôte et peuvent être transférées à très grande vitesse grâce à des circuits dédiées, respectivement 'the vault' et la 'staging memory'. D'autre machines, par contre, telles l'ordinateur CLIP 4 [1] et le GRID [21] reçoivent les données directement de l'appareil d'acquisition, généralement une caméra, et les traitent en temps réel.

Dans le cas du système IMAGE, l'acquisition se fait par l'intermédiaire du MVP qui peut numériser une image en un trentième de seconde. L'interface du MVP permet de transférer des images vers ou en provenance du MVP à la même vitesse, i.e. en temps réel. De plus, grâce au MVP, les images peuvent être emmagasinées dans la mémoire de masse de l'ordinateur hôte. Le schéma détaillé de cette interface est reproduit à l'annexe A.2. Cette annexe inclut aussi les circuits du registre K et de la table de données.

Cette interface doit aussi résoudre deux problèmes pour faire communiquer les deux machines MVP et IMAGE. Premièrement, les machines ne sont pas synchronisées, elles fonctionnent à des fréquences différentes et sans relation de phase. Deuxièmement, les données du MVP sont en format parallèle par bit, c'est-à-dire que tous les bits d'un pixel arrivent simultanément, alors qu'IMAGE fonctionne en format sériel par bit, c'est-à-dire que les bits d'un pixel sont traités séquentiellement, mais IMAGE traite par contre plusieurs pixels simultanément. La figure 3.5 montre cette conversion des données.

Le schéma-bloc de la figure 3.6 montre les deux blocs principaux, soient la mémoire à deux ports et le bloc des registres à décalage. La mémoire à deux-ports doit être vue comme deux mémoires tampons, chacune pouvant contenir 512 pixels, soit une ligne de pixels. Ces mémoires peuvent être alternativement connectées sur le bus d'expansion du MVP en synchronisme avec son horloge de pixel (DOTCLOCK) ou sur le bus d'IMAGE en synchronisme avec l'horloge phi 2 de cette machine.

3.4.2 Fonctionnement

Le diagramme de la figure 3.7 décrit brièvement un transfert d'images du MVP vers IMAGE. Premièrement, une ligne (512 pixels) est transférée dans le tampon #1. Dans un deuxième temps, une seconde ligne est emmagasinée dans le tampon #2 pendant qu'IMAGE lit la première ligne du tampon #1. La troisième ligne est ensuite emmagasinée dans le tampon #1 pendant qu'IMAGE lit la deuxième du tampon #2, et ainsi de suite jusqu'à la fin de l'image. Le phénomène inverse se produit lors du transfert d'images à partir d'IMAGE vers le MVP.

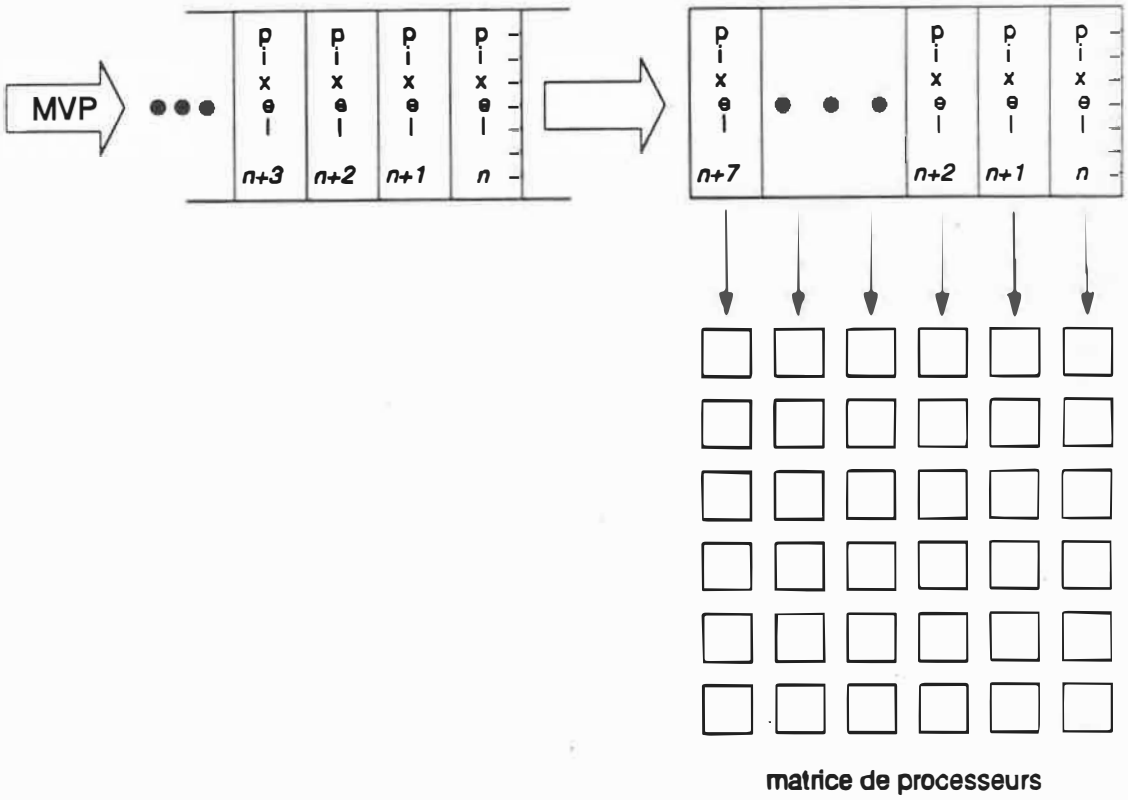


Figure 3.5 - Conversion des données du format parallèle à sériel

Lorsque les pixels sont envoyés à la matrice de processeurs, il sont sous un format parallèle par bit, alors que les processeurs IMAGE-2 ont une architecture sériel par bit. Le bloc de registres à décalage fait la conversion du format des pixels dans les deux sens. Cette interface transfère les images en temps réel au taux de 8 méga-octets par seconde ce qui est le taux de transfert maximal du numérisateur MVP.

3.5 La micro-machine

La micro-machine contrôle et synchronise toutes les parties de la machine IMAGE. Elle est responsable du séquencement des commandes, des données et des adresses lors de l'exécution des algorithmes. L'architecture de la micro-machine, dont le schéma-bloc est représenté à la figure 3.8, est divisée en trois blocs principaux: le micro-séquenceur, la mémoire de micro-code et la file de macro-commandes. Le schéma détaillé est représenté à l'annexe A.3. Cette architecture s'inspire de celle de la Connection Machine [20] dans laquelle les macro-commandes, originant de l'ordinateur hôte, appellent des routines micro-codées exécutées par le micro-séquenceur.

3.5.1 Le micro-séquenceur Am2910

Le coeur de la micro-machine est composé du micro-séquenceur Am2910. A chaque cycle d'horloge, deux micro-mots de 96 bits sont envoyés successivement pour contrôler toutes les fonctions d'IMAGE dont 35 forment la micro-instruction du CI IMAGE-2. Ce micro-séquenceur a été choisi en raison de sa grande disponibilité, mais surtout parce qu'il remplissait les critères nécessaires à

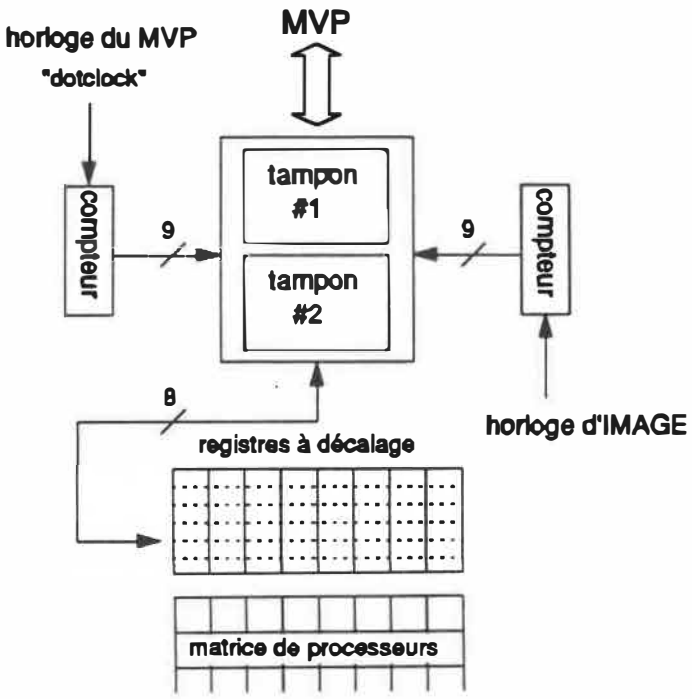
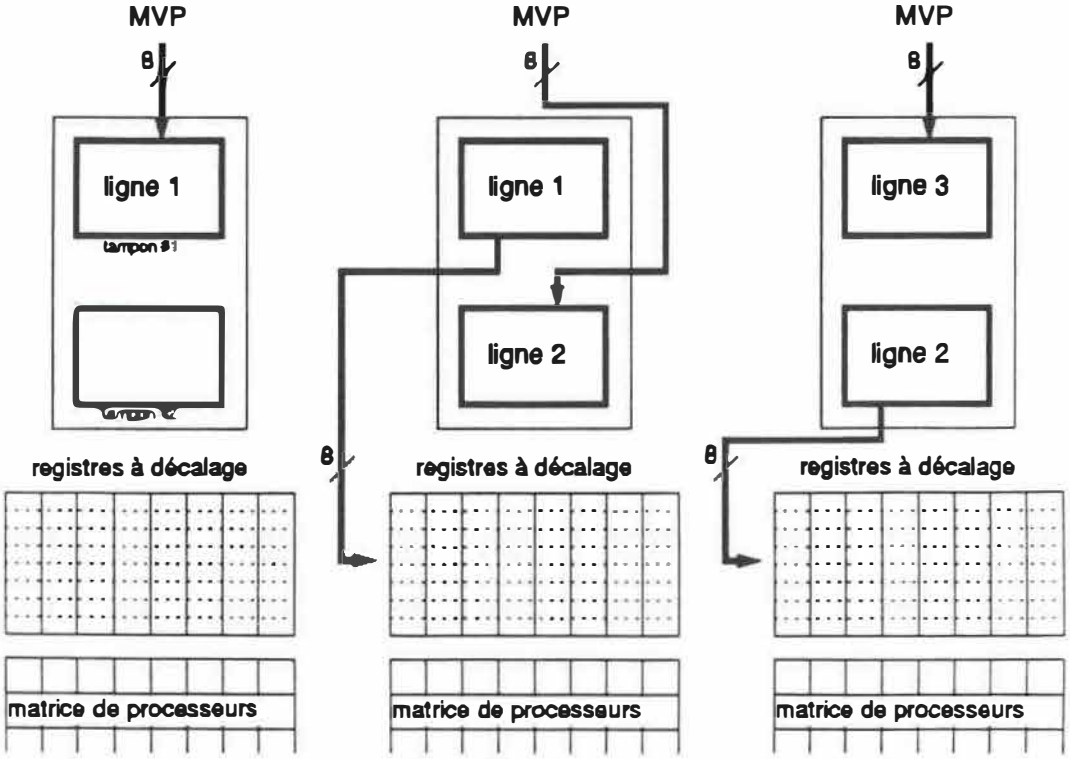


Figure 3.6 - Schéma-bloc de l'interface du MVP



(a) premier temps (b) deuxième temps (c) troisième temps

Figure 3.7 - Transfert d'images par les tampons

notre application. Ces critères sont les suivants.

Fréquence d'horloge de 10 Mhz :

Cette fréquence est la fréquence d'opération pour laquelle le CI IMAGE-2 a été conçu. De plus, c'est la fréquence d'opération du bus d'expansion du MVP qui doit être égalée pour transférer des lignes complètes en temps réel.

Compteur interne:

Etant donnée la nature répétitive des opérations de TNI, un compteur est essentiel pour une grande performance.

Branchements conditionnels:

Le micro-séquenceur est appelé à réagir selon l'état de certains signaux, il est donc muni d'instructions de branchement conditionnel.

Pile :

Le micro-séquenceur doit aussi contenir une pile pour l'exécution de sous-routines.

3.5.2 La mémoire de micro-code

La mémoire de micro-code contient les sous-routines exécutées par la machine IMAGE, elle est large de 96 bits et profonde de 4096 mots et contrôle l'ensemble d'IMAGE, y compris le micro-séquenceur lui-même. Etant donné l'architecture du CI IMAGE-2, cette mémoire doit fonctionner au double de la

fréquence d'horloge du reste du système, c'est-à-dire qu'à chaque cycle du Am2910, deux micro-mots de 96 bits sont lus.

Ces 96 bits sont divisés comme suit:

	# bits	
0-35	36:	instruction du c.i. IMAGE-2
36-43	8 :	contrôle de la matrice de processeurs
44-50	7 :	contrôle du générateur d'adresses
51-56	6 :	contrôle de l'interface MVP
57-59	3 :	bits de réserve
60-75	6 :	donnée de 16 bits à usages multiples (micro-data)
76-79	4 :	bits de réserve
80-91	12:	adresse de la prochaine micro-instruction
92-95	4 :	prochaine instruction du micro-séquenceur (Am2910)

Le micro-code est écrit avec l'ordinateur hôte à l'aide d'outils de développement tel que le micro-assembleur, décrit au chapitre six, et transféré à la mémoire de micro-code pour être exécuté. La largeur de la mémoire (96 bits) n'était pas sans causer de problèmes, puisqu'elle doit être accessible à la fois par l'ordinateur hôte en mode écriture et lecture, et par la micro-machine en mode lecture. En utilisant une architecture conventionnelle avec un chargement parallèle des données, le nombre de répéteurs aurait été prohibitif étant donné la grandeur de la carte utilisée tel que montré à la figure 3.9(a).

Afin de minimiser le nombre de répéteurs, nous avons utilisé des registres de diagnostique tel que montré à la figure 3.9(b). Ces registres permettent de charger et de relire d'une façon sérielle la mémoire de micro-code pour son chargement et sa vérification à partir de l'ordinateur hôte, tout en ayant la possibilité de décharger les 96 bits en parallèle pour l'exécution du micro-code.

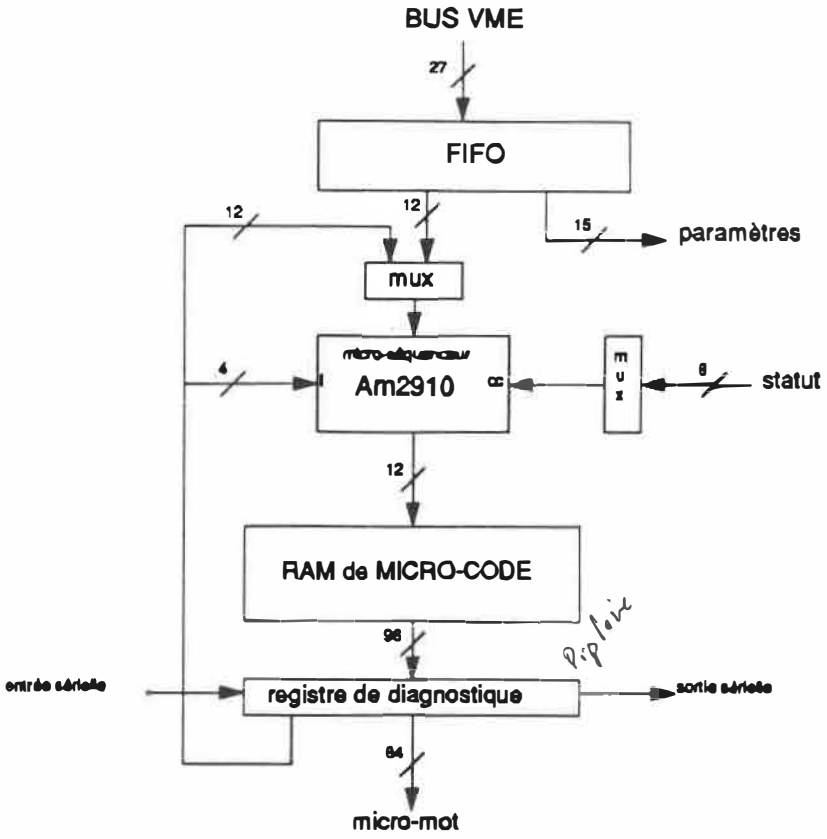
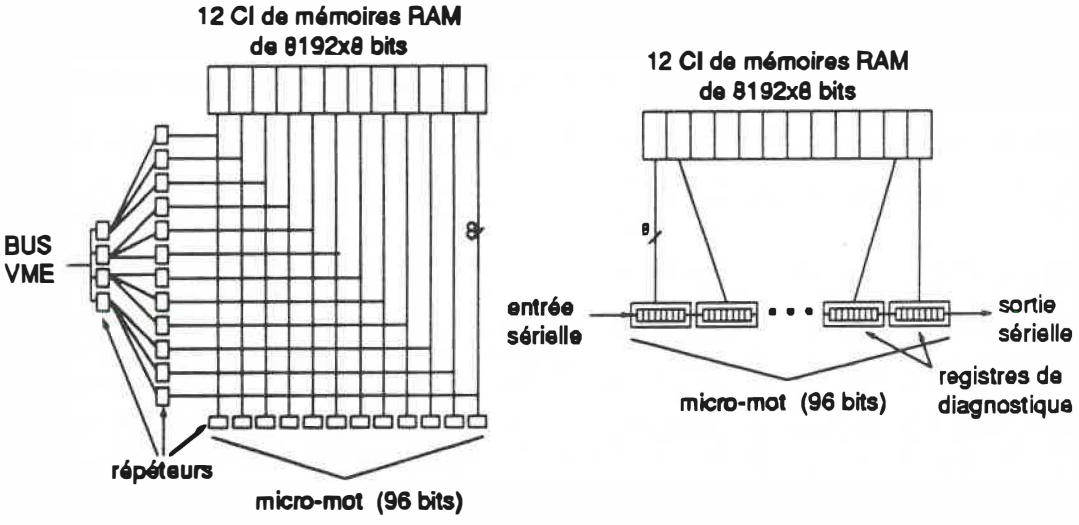


Figure 3.8 - Schéma-bloc de la micro-machine



(a) chargement parallèle (b) chargement sériel

Figure 3.9 - Circuits de chargement du micro-code

Cette architecture permet aussi à l'hôte d'aller modifier aléatoirement n'importe quel mot de 96 bits. Cette technique est évidemment beaucoup plus lente qu'un chargement parallèle, mais la performance n'est pas affectée puisque le micro-code n'est pas modifié pendant l'exécution d'un algorithme.

3.5.3 La file de commandes (FIFO)

Une fois la mémoire de micro-code chargée de micro-routines, l'ordinateur SUN envoie des séquences de micro-routines à exécuter par l'intermédiaire d'une file FIFO. Cette file contient des pointeurs à la mémoire de micro-code et des paramètres tels que des adresses pour le générateur d'adresses et la table de données.

Cette section du circuit n'est pas décrite davantage puisqu'elle est l'objet d'autres recherches présentement en cours.

3.6 Les communications

Le système IMAGE est composé de trois sous-systèmes principaux, l'ordinateur SUN, le numérisateur MVP et bien entendu, la machine IMAGE. Pour assurer une bonne performance, les communications entre ces trois sous-systèmes doivent être rapides et accommoder différentes sortes d'informations, telles que des programmes, des données, des images et des commandes. Le diagramme de la figure 3.10 montre quelles sortes d'informations sont échangées entre les différentes

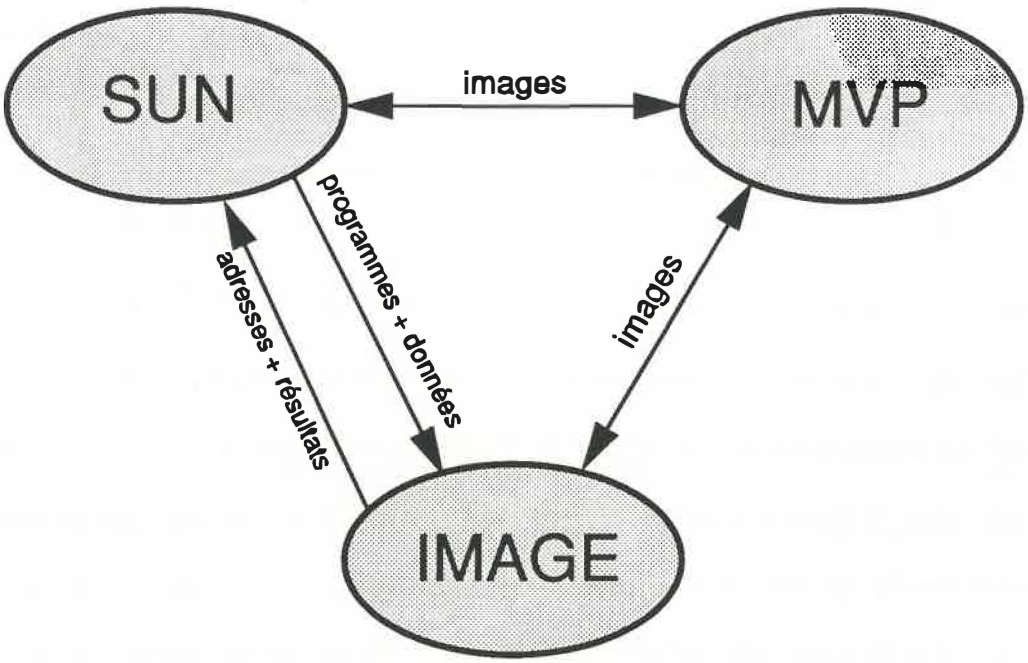


Figure 3.10 - Liens de communication dans le système IMAGE

sections.

Ces informations sont échangées de différentes façons: par accès directs à la mémoire, par interruptions ou par des circuits spéciaux tel que l'interface MVP.

3.6.1 Les communications de l'ordinateur hôte vers IMAGE

Le transfert d'informations de l'ordinateur hôte vers la machine IMAGE peut suivre deux chemins suivant le type de donnée. D'une part, les images sont transférées par l'intermédiaire du MVP qui bénéficie d'un interface très rapide avec l'ordinateur SUN. D'autre part, les programmes sont transférés directement à la machine IMAGE par la mémoire de micro-code, tandis que les données utilisent la table de données et le registre K qui seront expliqués plus loin (figure 4.1).

3.6.2 Les communications de IMAGE vers l'ordinateur hôte

De façon semblable à la section précédente, les images sont transférées d'IMAGE vers l'ordinateur SUN par l'intermédiaire du MVP. Par contre, puisque le SUN fonctionne dans un environnement multi-tâches et multi-usagers, les échanges d'autres informations de IMAGE vers l'ordinateur SUN se font à l'aide des interruptions. Lorsqu'un algorithme est terminé, et pour toute autre situation particulière, comme un débordement, ou une demande de nouveaux paramètres, le micro-séquenceur peut faire une demande d'interruption (IRQ - Interrupt ReQuest)

à l'ordinateur hôte. Celui-ci peut alors lire le registre K, le registre d'adresse ou le registre de statut qui contiendra l'information nécessaire concernant la source de l'interruption.

3.7 Le générateur d'adresses

Les algorithmes de TNI doivent souvent traiter un grand nombre de données. Par exemple, une image d'une dimension de 512 x 512 pixels de huit bits contient 262 184 octets. Etant donné le grand nombre de données, le générateur doit être suffisamment flexible pour pouvoir accommoder la plupart des algorithmes de façon efficace. Puisque l'architecture est de type IUDM, une seule adresse est générée pour les mémoires externes de tous les processeurs. Les générateurs d'adresses des machines IUDM sont généralement simples, celui du MPP est seulement composé de sept registres de 16 bits [14].

Le générateur d'adresses d'IMAGE garde cette simplicité, mais contient des éléments supplémentaires qui tiennent compte des propriétés des algorithmes de TNI de bas niveau. En ce qui concerne l'adressage des mémoires, nous pouvons en retenir trois principales qui sont les suivantes:

- répétition des opérations sur chaque pixel;
- accès aux pixels voisins du pixel traité;
- différentes résolutions (1,2,4,8 bits par pixel ou plus).

La figure 3.11 montre le schéma-bloc du générateur d'adresses et ses trois sections principales, chacune ayant un rôle précis: les deux registres-compteurs, l'additionneur et le registre de sortie. Le schéma détaillé est reproduit à l'annexe A.4.

Premièrement, puisque les opérations de TNI se répètent souvent pour chacun des pixels, l'adresse est emmagasinée dans un compteur qui peut être facilement incrémenté pour accéder successivement à tous les pixels d'une image. Un deuxième compteur peut servir pour pointer vers une autre image ou vers une table de variables. Les compteurs peuvent être chargés par la file de commandes ou directement à partir du champ 'microdata' du micro-mot. Ces compteurs de 15 bits de large permettent d'emmagasiner 32k bits par processeur. Cette densité de mémoire, à la limite de la technologie utilisée pour notre prototype, permet d'emmagasiner 4 images d'une grandeur de 512x512 pixels et d'une résolution de 8 bits avec une matrice de 16x16 PEs.

Deuxièmement, un simple additionneur permet d'additionner un décalage à l'adresse du compteur afin d'adresser les pixels voisins du pixel traité. Plusieurs algorithmes nécessitent les huit voisins les plus près comme dans le cas des convolutions avec une matrice de 3x3. On peut aussi facilement implanter des algorithmes de convolution avec un masque de grandeur arbitraire.

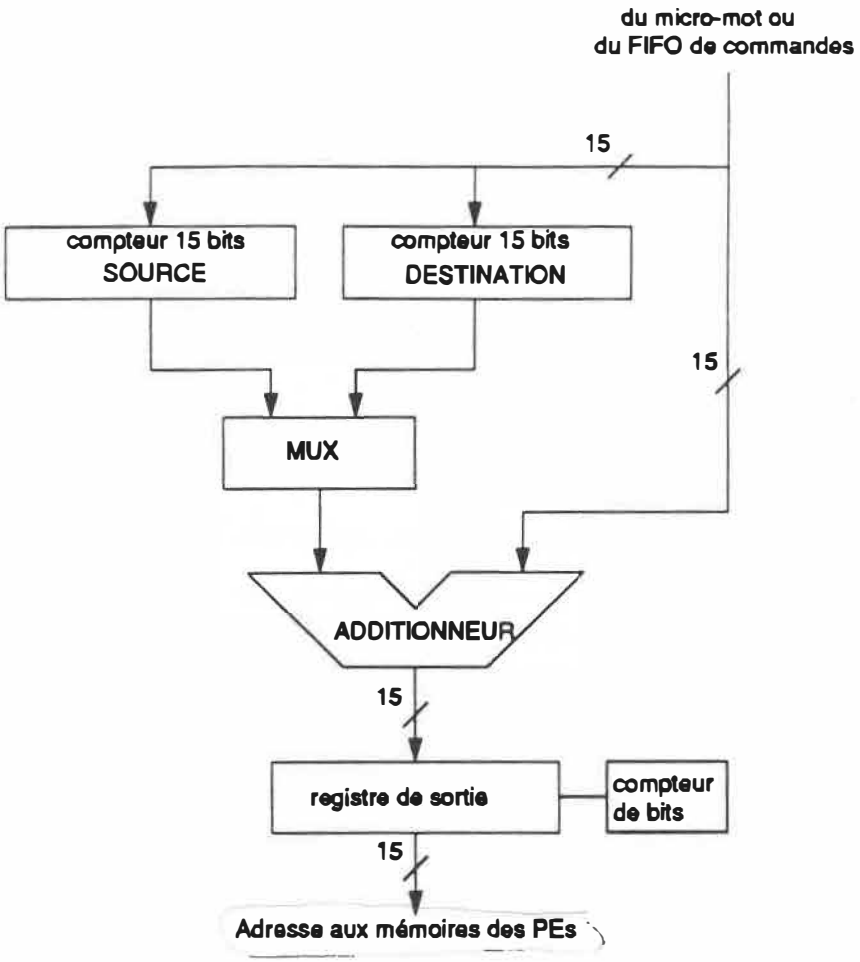


Figure 3.11 - Schéma-bloc du générateur d'adresses

-33	-32	-31
-1	0	1
31	32	33

Figure 3.12 - Décalage des adresses pour les opérations de voisinage (image de 32x32 pixels)

Dans ce genre de calcul, un registre-compteur pointe vers le pixel traité, tandis que l'additionneur permet d'adresser les pixels voisins. Prenons l'exemple d'un masque de grandeur 3x3 appliqué sur une image de 32 pixels par ligne, la figure 3.12 montre la valeur à additionner à l'adresse du pixel traité pour accéder aux pixels voisins.

Finalement, un registre, à la sortie du circuit, permet de mémoriser une des adresses générées par commande du micro-code. Cette adresse peut être maintenue pour des conditions particulières telles que la détection d'une forme, une division par zéro, etc. Cette adresse peut ensuite être relue par l'ordinateur hôte afin de prendre les mesures nécessaires puisqu'il peut déterminer où cette condition s'est produite dans l'image.

Le registre de sortie contient aussi un circuit supplémentaire permettant de traiter facilement et efficacement des images de différentes résolutions. Tel que montré à la figure 3.13, si un registre-compteur pointe vers le pixel ***n***, le fait d'incrémenter ce registre pointera vers le bit #1 de ce même pixel plutôt que vers le pixel suivant. Le compteur de bit, lorsque juxtaposé à l'adresse calculée, permet d'accéder les bits successifs d'un pixel sans modifier les registres compteurs. Les figures 3.14 (a), (b), (c) et (d) montre comment l'adresse résultante est formée si les résolutions des pixels sont respectivement 1, 2, 4 et 8 bits. Par exemple, si l'image traitée est d'une résolution de 4 bits par pixel, l'adresse résultante est formée des 13 bits les plus significatifs sortant de l'additionneur et des deux bits les plus bas du compteur de bits. De cette façon, on incrémente uniquement le

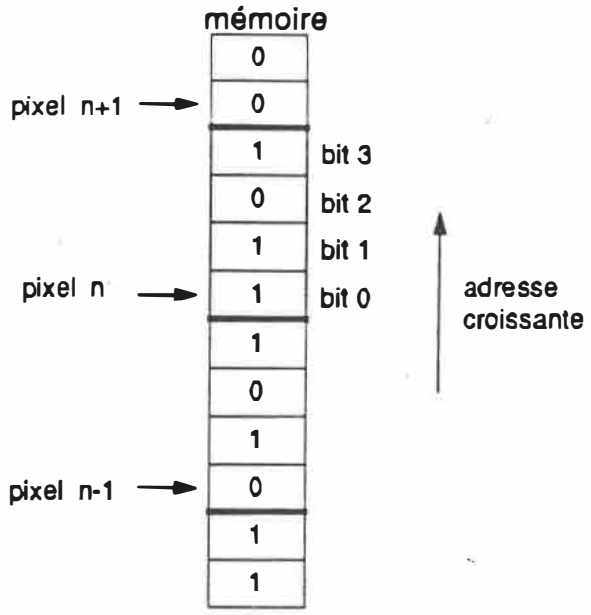


Figure 3.13 - Organisation des pixels en mémoire (4 bits/pixel)

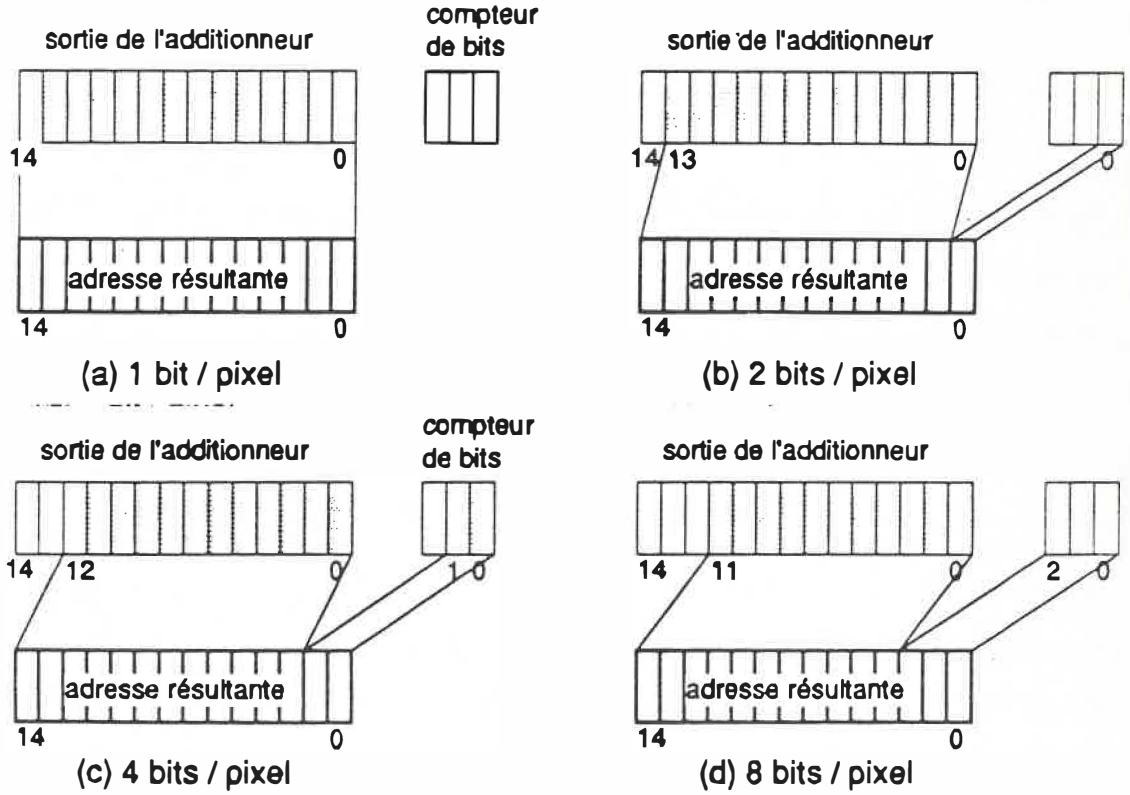


Figure 3.14 - Configurations de la sortie du générateur d'adresses pour différentes résolutions

compteur de bits pour accéder les bits d'un même pixel. L'utilisation de ce circuit est démontrée au chapitre cinq dans un exemple de seuillage d'une image.

Le générateur d'adresses, la micro-machine et les interfaces du MVP et du bus VME qui viennent d'être décrits sont les circuits qui alimentent et commandent la matrice de processeurs. Cette matrice forme le coeur de la machine IMAGE et est l'objet du chapitre suivant.

CHAPITRE 4 - LA MATRICE DE PROCESSEURS

4.1 Vue globale

La matrice de processeurs forme le coeur de la machine IMAGE et est essentiellement composée de circuits intégrés IMAGE-2 et de quelques circuits périphériques tels que montrés à la figure 4.1.

Les processeurs IMAGE-2 sont interconnectés entre eux pour former une matrice carrée de processeurs. Chaque processeur a un lien bidirectionnel avec chacun de ses huit voisins immédiats et chaque processeur est aussi relié à une mémoire externe de 32 kilobits. Cette architecture permet d'effectuer des opérations de TNI efficacement, mais on peut améliorer la flexibilité de la machine et la performance de la plupart des algorithmes par l'ajout de quelques circuits simples. Cette matrice est donc principalement composée de c.i. IMAGE-2 et leurs mémoires externes, de circuits de bordure, du registre K et de la table de données.

4.2 Le circuit intégré IMAGE-2

La figure 4.2 montre le schéma-bloc du processeur élémentaire (PE) qui forme le coeur du système IMAGE. La matrice de processeurs est composée de circuits intégrés IMAGE-2, chacun contenant dix processeurs. Parmi ceux-ci, deux sont réservés pour la tolérance aux défaillances de telle sorte que seulement huit processeurs sont utilisés. Le rapport technique d'IMAGE-2 [2] décrit le processeur élémentaire en détails, c'est pourquoi seulement les caractéristiques principales sont expliquées ici.

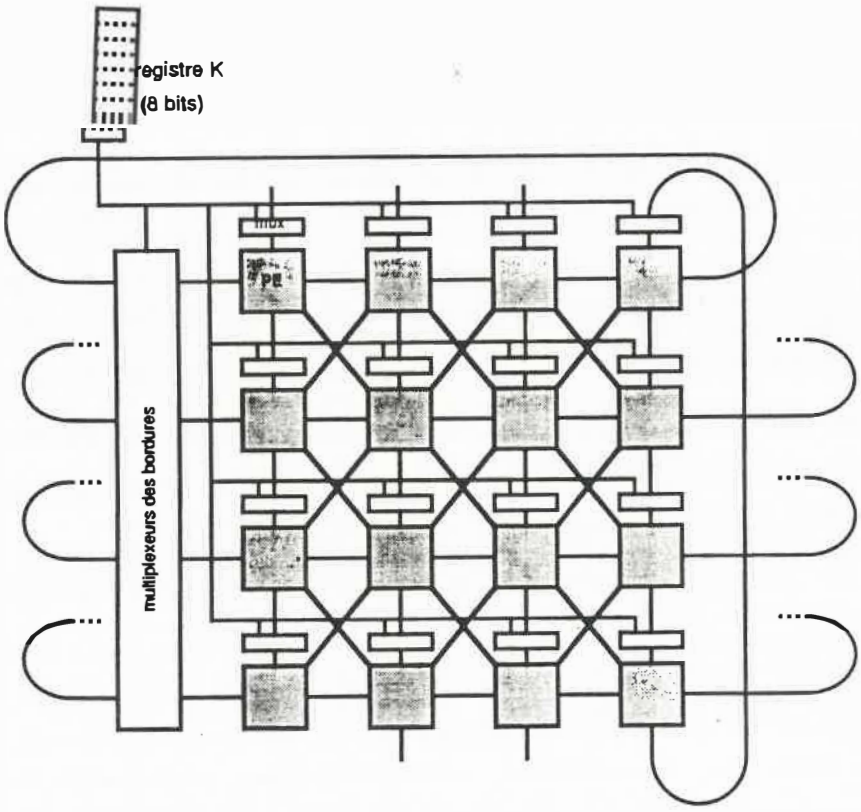


Figure 4.1 - La matrice de processeurs IMAGE-2

D'abord, l'architecture est de type sériel [13], i.e. toutes les données sont traitées et communiquées un bit à la fois. Cette architecture réduit le nombre d'interconnexions entre les PEs et les mémoires et simplifie les processeurs. Chaque PE compte deux banques de quatre registres de 12 bits, chaque bit étant adressable indépendamment. Un étage de multiplexeurs à la sortie des registres permet d'acheminer les données vers l'unité de calcul GFLU (Générateur de Fonctions Logiques Universel) ou vers les ports de sorties. Le GFLU peut générer n'importe laquelle des 2^{16} fonctions logiques de quatre entrées binaires et opère en parallèle avec une unité de calcul de la retenue pour les opérations d'addition et de soustraction. Une unité d'inhibition permet d'empêcher l'écriture dans certains registres pour des opérations conditionnelles. Les conditions peuvent être même imbriquées à plusieurs niveaux grâce à une pile de quatre bits qui peut maintenir le résultat des opérations précédentes.

Le PE communique avec le monde extérieur à l'aide de deux canaux de communication. Un premier canal, le canal de pipeline, est relié à un c.i. de mémoire externe propre à chaque processeur pour emmagasiner une section d'image et d'autres données nécessaires aux calculs. L'autre canal est réservé aux communications inter-processeurs. L'entrée de ce canal est composé d'un multiplexeur à huit entrées permettant de recevoir une donnée de l'un des huit processeurs voisins, tandis que la sortie est acheminée aux huit processeurs voisins. Ce schème d'interconnexions est représenté à la figure 2.6.

Une autre caractéristique importante du processeur est le séquencement en pipeline des opérations internes. L'architecture du PE permet d'effectuer plusieurs opérations en parallèle pour accélérer la vitesse des calculs et des

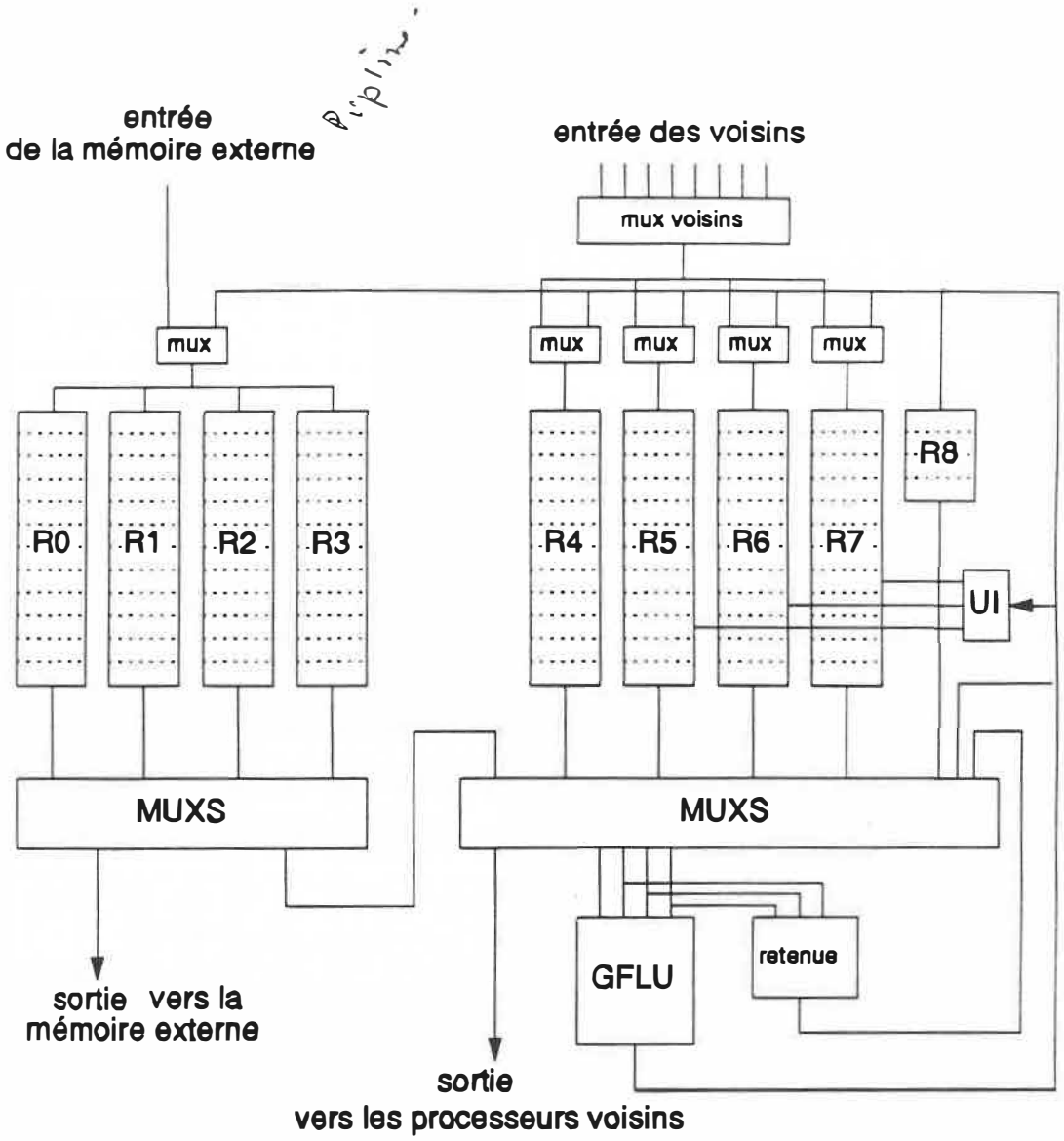


Figure 4.2 - Schéma-bloc d'un processeur élémentaire

communications. Les opérations du GFLU, la lecture et l'écriture dans les registres, les entrées et sorties des données prennent tous un cycle d'horloge, mais le parallélisme interne permet d'effectuer celles-ci simultanément. Par exemple, l'addition de deux nombres de huit bits situés dans les registres internes qui nécessiterait normalement 8 lectures, 9 écritures et 8 additions prend seulement 10 cycles telle que le montre la figure 4.3. Et simultanément, on pourra effectuer une opération de lecture ou d'écriture avec la mémoire externe puisque les deux blocs de registres sont indépendants.

IMAGE-2 a aussi servi de banc d'essai pour étudier une nouvelle technique de tolérance aux défaillances proposée par C.Cyr [22]. Cette technique permet de tolérer jusqu'à deux processeurs défectueux par c.i. IMAGE-2 et de reconfigurer automatiquement les canaux d'entrées et sorties des processeurs. Le micro-séquenceur est responsable d'envoyer des vecteurs de test pour que les circuits internes puissent automatiquement détecter et déconnecter les processeurs défectueux. Les circuits de reconfiguration s'assurent aussi que précisément huit processeurs par c.i. sont utilisés même s'ils sont tous non-défectueux. Si plus de deux processeurs sont défectueux, le c.i. doit être remplacé.

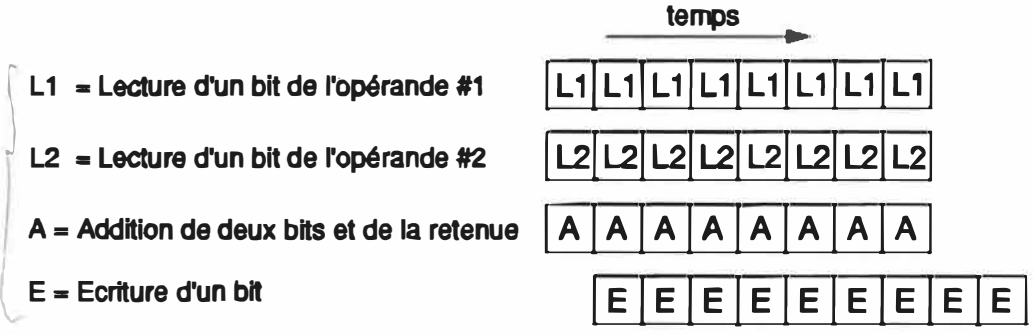


Figure 4.3 Exécution en pipeline d'une addition 8 bits

La matrice de processeurs est composée de c.i. IMAGE-2, chacun contenant huit PEs interconnectés en ligne. La figure 4.4 montre une photographie du c.i dans laquelle on peut distinguer les dix processeurs dont deux servent à la tolérance aux défaillances. Les tampons d'entrées et de sorties sont respectivement au haut et au bas de la figure 4.4, tandis que les circuits de contrôle occupent les flancs droit et gauche. La surface du c.i. est de 0,2 cm^2 et utilise la technologie à trois microns et à double couches de métal de Northern Telecom.

4.3 La mémoire externe

4.3.1 Généralités

Telle que mentionné précédemment, la machine IMAGE est basée sur une architecture de type IUDM (Instruction Unique, Données Multiples). Dans une architecture IUDM, chaque processeur élémentaire a sa propre mémoire de données contenant des opérandes utilisées par les instructions. De cette façon, chaque processeur n'a un accès direct qu'avec sa propre mémoire, c'est-à-dire qu'il faudra passer par le canal de communication inter-processeur pour transférer des données d'une mémoire à une autre.

Puisque notre architecture parallèle est axée vers le traitement numérique d'images, il faudra trouver un moyen de sectionner l'image et de la répartir parmi les processeurs. Etant donné le grand nombre de pixels dans une image, chaque processeur devra traiter plusieurs pixels. Par exemple, pour une image de 512x512 pixels traitée par une matrice de 256 processeurs, chaque processeur devra emmagasiner 1024 pixels.

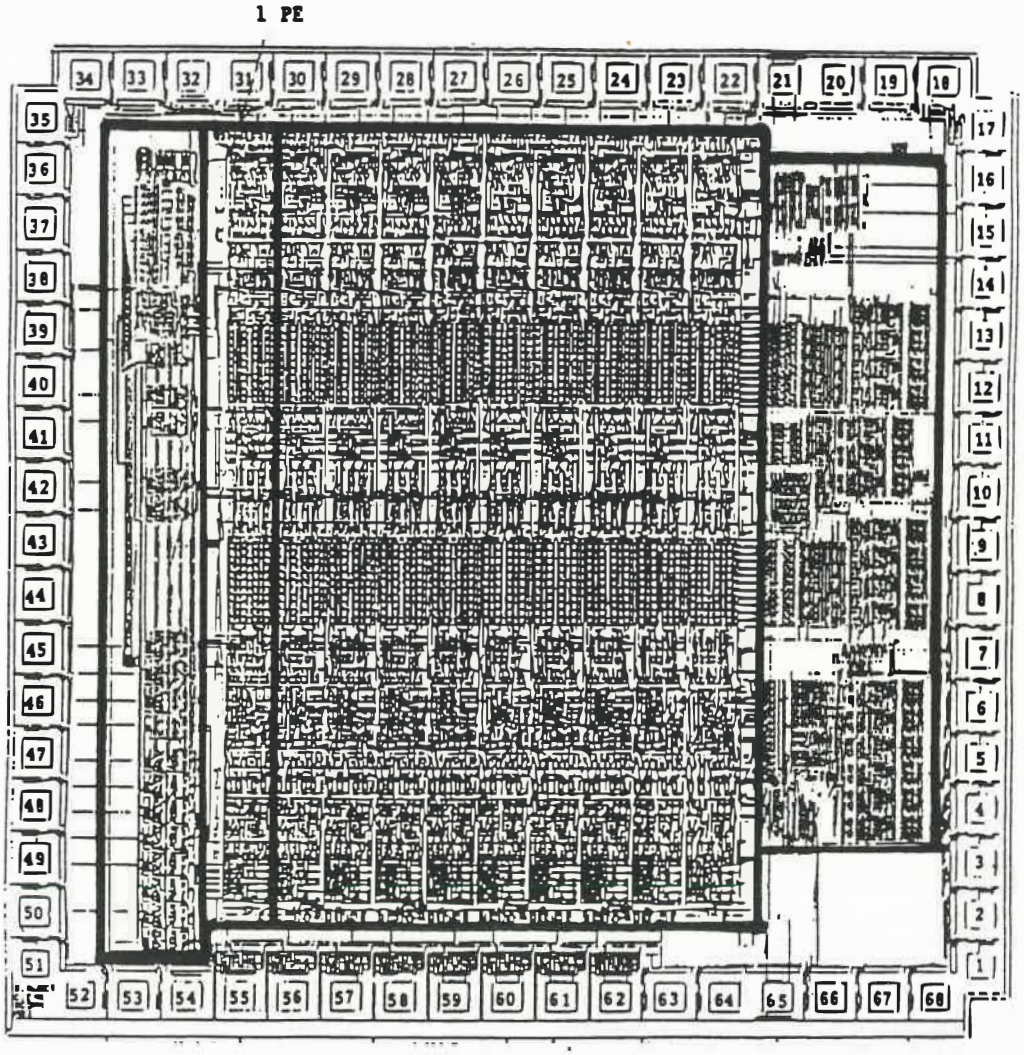


Figure 4.4 Photomicrographie du c.i. IMAGE2

La machine IMAGE utilise trois modes d'adressage pour distribuer l'image dans les mémoires des processeurs, nous les appellons respectivement le mode pyramidal, le mode fenêtre et le mode ligne.

4.3.2 Mode fenêtre

En mode fenêtre, la matrice de processeurs est déplacée à travers l'image pour traiter toute une image. La figure 4.5 montre la correspondance entre les processeurs et les pixels. Les nombres représentent l'adresse où le pixel est emmagasiné dans la mémoire du PE correspondant. L'exemple montre une matrice de 16 processeurs traitant une image de taille 16x16. L'image est divisée en damier, chaque carrée contenant le même nombre de pixels qu'il y a de processeurs. On traite une image complète en effectuant l'opération désirée sur chaque région successivement. Puisque chaque processeur traite un pixel de la région, des pixels adjacents sont emmagasinés dans les mémoires de processeurs adjacents. Si un processeur requiert des pixels voisins, ceux-ci devront être transférés par l'intermédiaire des canaux de communication inter-processeurs.

L'avantage de cette technique est qu'on peut traiter seulement les régions d'intérêt dans l'image. Par exemple, si un objet est détecté dans une certaine portion de l'image, le traitement peut être concentré dans cette région pour optimiser l'usage des processeurs au lieu de traiter toute l'image inutilement.

L'inconvénient se situe au niveau des bordures. Plusieurs algorithmes de TNI nécessitent la valeur des pixels voisins du pixel traité. Si, par exemple, les processeurs nécessitent la valeur du pixel situé à l'OUEST du pixel traité, les processeurs situés à la bordure OUEST de la matrice devront aller chercher le pixel

dans la région voisine, c'est-à-dire à une autre adresse. Et ce pixel se trouve dans la mémoire du processeurs situé sur la même ligne, mais à l'autre extrémité de la matrice.

4.3.3 Mode pyramidal

Dans le mode pyramidal, chaque processeur est assigné à une région de l'image [23]. La figure 4.6 montre la relation entre les processeurs et les pixels de l'image et comment les pixels sont emmagasinés dans les mémoires externes des PE. Dans cet exemple, une matrice de 16 processeurs se partagent une image de 16x16 pixels.

Cette technique est particulièrement adaptée pour les opérations de voisinage puisque la plupart du temps, les pixels voisins du pixel traité sont directement accessibles par le même PE sans nécessiter de communications inter-processeurs. Evidemment, lorsqu'on doit traiter les pixels en bordure, certains pixels devront être empruntés au processeur voisin, mais généralement, le nombre de pixels en bordure d'une région est beaucoup moindre que le nombre total de pixels dans une région. Par exemple, pour une image de 512x512 pixels et une matrice de 16x16 processeurs, comme dans le cas de la machine IMAGE, chaque processeur traite 1024 pixels, i.e une image de taille 32x32, dont seulement 124 (31x4), i.e. environ 12%, sont en bordure.

L'inconvénient est que toute l'image est traitée en tout temps. Si seulement une petite portion de l'image nécessite du traitement, seulement les processeurs assignés à cette portion de l'image opèrent utilement. Ce mode

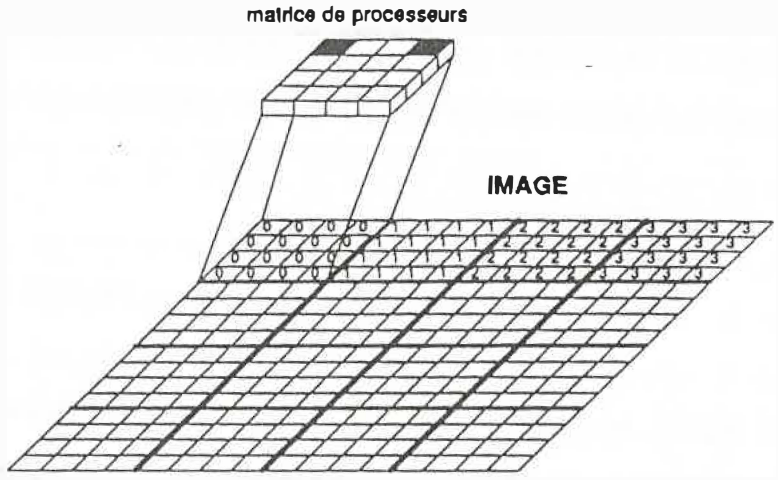


Figure 4.5 Mode d'adressage en fenêtre

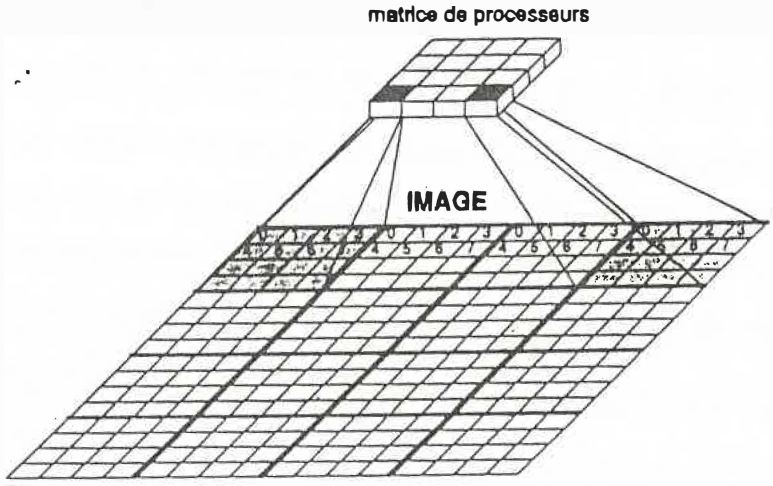


Figure 4.6 Mode d'adressage pyramidal

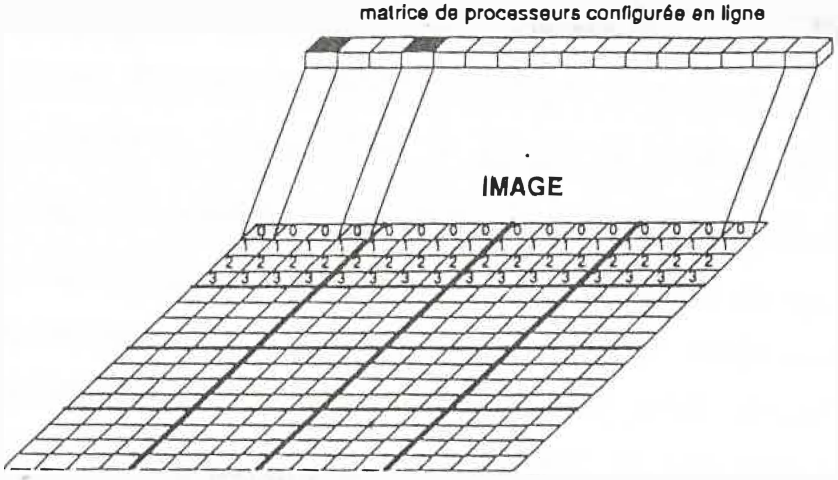


Figure 4.7 Mode d'adressage en ligne

d'adressage performe bien dans les algorithmes qui traitent l'image comme un tout, comme par exemple dans les opérations d'image à image et les opérations de prétraitement comme le filtrage.

4.3.4 Mode ligne

Dans le mode ligne, les processeurs sont interconnectés entre eux de façon à former une ligne tel que montré à la figure 4.7. Les adresses successives correspondent à des lignes successives dans l'image, s'il y a autant de processeurs qu'il y a de pixels dans une ligne. Ce mode facilite aussi l'exécution de plusieurs algorithmes de traitement numérique de signaux qui sont de nature unidimensionnels.

Pour chacun de ces modes d'adressage, les pixels doivent apparaître dans un certain ordre à la sortie de la mémoire à deux ports de l'interface du MVP avant d'être emmagasinés dans les mémoires de la matrice de processeurs. Cette séquence est contrôlée par un PAL (Programmable Array Logic) qui génère les adresses du port synchronisé avec l'horloge d'IMAGE. Cette séquence, peut être déduite à l'aide des figures 4.5, 4.6 et 4.7. Prenons par exemple le cas du mode d'adressage pyramidal pour une matrice de 4 X 4 processeurs. A l'adresse 0 du processeur OUEST, on emmagasine le premier pixel de la première ligne d'image, le pixel #0. A l'adresse 0 du prochain processeur à droite, on charge le 5ème pixel

de l'image, le pixel #4, tel que représenté à la figure 4.6. La séquence d'adresse à la mémoire à deux ports générée par le PAL sera donc:

0, **4**, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, etc.

Les addresses correspondantes du générateur d'adresses sont:

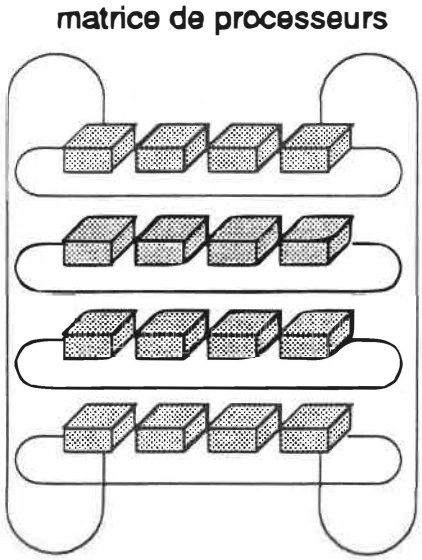
0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, etc.

4.4 Circuits périphériques

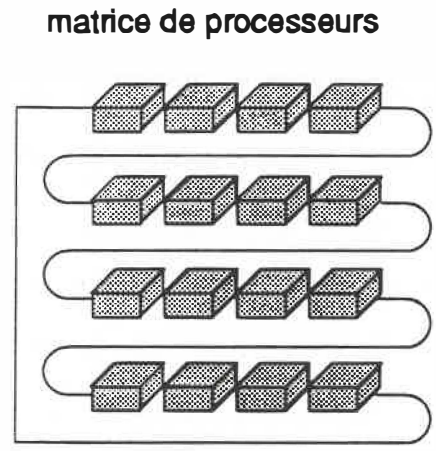
Les circuits périphériques permettent d'augmenter la flexibilité de l'architecture et la performance de plusieurs algorithmes. Ces circuits permettent entre autres de configurer la matrice en mode plein, pyramidal ou ligne et de transmettre une valeur à tous les processeurs simultanément ou seulement aux processeurs en bordure de la matrice.

4.4.1 Le traitement des bordures

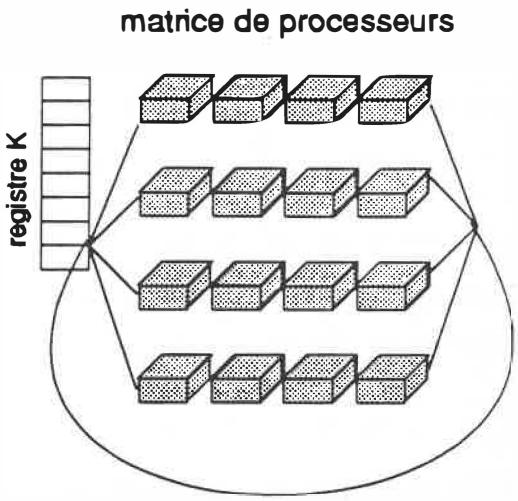
Dans une architecture matricielle à deux dimensions, les bordures de l'image et de la matrice de processeurs peuvent être traitées de différentes façons. Les parties a), b) et c) de la figure 4.8 montrent seulement les trois configurations de base, mais on peut combiner celles-ci pour en créer de nouvelles. Le choix du mode d'adressage déterminera en grande partie l'interconnexion des processeurs de bordure. Les figures A.5 et A.6 à l'annexe A montre les schémas détaillés des multiplexeurs de bordures.



(a) Configuration cylindrique



(b) configuration cylindrique décalée
ou en ligne



(c) Configuration avec le registre K

Figure 4.8 Configuration des bordures

4.4.1.1 Configuration cylindrique

Cette configuration est nécessaire pour les algorithmes faisant des opérations de voisinage dans le mode d'adressage en fenêtre. Effectivement les processeurs de bordure doivent être directement reliés aux processeurs opposés sur la même colonne ou la même ligne. Tel que montré sur la figure 4.9(a), par exemple, si un processeur de bordure effectue un accès EST, le voisin EST est en fait le processeur le plus à l'ouest sur la même ligne. Et de même pour les accès NORD-SUD.

Dans le mode d'adressage pyramidal, la configuration des circuits de bordure correspond en fait à la configuration des bordures de l'image puisque les processeurs en bordure de la matrice sont aussi en bordure de l'image.

4.4.1.2 Configuration cylindrique décalée

Cette configuration est en fait utilisée dans le mode ligne. Effectivement, le dernier processeur de la première ligne (en haut à droite) est relié au premier de la deuxième ligne, et ainsi de suite jusqu'au dernier processeur en bas à gauche. Cette configuration convertie la matrice bidimensionnelle de NxN PE en une ligne de N^2 PE tel que montrée à la figure 4.8(b).

4.4.1.3 Registre K

Cette configuration présente la bordure comme une constante tel que montré à la figure 4.8(c). Particulièrement utile dans le mode pyramidal, l'usager peut vouloir considérer la bordure comme étant une constante. Mais l'attrait principal de cette configuration est qu'il permet, rapidement, d'activer ou de désactiver seulement les processeurs de bordure [24]. Par exemple, on peut désactiver les processeurs EST de cette façon:

1- On charge 1 dans le registre K.
2- Chaque processeur envoie 0 au processeur à l'OUEST qui est chargé dans le registre d'état.

Ainsi, tous les processeurs reçoivent 0 du processeur situé à l'EST, sauf les processeurs de bordures qui reçoivent la valeur du registre K, soit 1. Cette valeur est emmagasinée dans le registre d'état du processeur permettant ainsi d'inhiber l'écriture dans certains registres. Cette opération peut ensuite être répétée pour chacune des bordures.

4.4.2 Les multiplexeurs d'entrée et le registre K

Tel que montré à la figure 4.1, un multiplexeur d'entrée a été rajouté à l'entrée NORD de chaque processeur afin d'accomoder des fonctions supplémentaires. La pénalité encourue par cet ajout est d'un c.i. supplémentaire pour chaque paire de c.i. IMAGE-2. Ce coût est minime en comparaison avec le gain de performance atteint pour plusieurs algorithmes de TNI. Un bit de contrôle du micro-code décide, à chaque cycle, si l'entrée NORD des processeurs devrait être connecté au registre K ou à la sortie du processeur situé au NORD.

Ce multiplexeur permet donc à tout les processeurs de recevoir simultanément une donnée provenant du registre K. Ce registre de 8 bits peut être chargé de trois façons:

- Directement du bus VME par l'ordinateur hôte.
- D'une valeur de la table de données.
- D'une donnée provenant du processeur du coin supérieur gauche de la matrice.

Cette valeur, de 1 à 8 bits, peut être transmise simultanément à tout les processeurs. Mentionnons par exemple le cas du calcul d'une convolution où les coefficients du masque sont emmagasinés dans la table de données. Chaque coefficient prend 9 cycles pour être transmis à tous les processeurs.

Le registre K peut emmagasiner le résultat global d'un calcul de la matrice qui peut soit être retourné à tous les processeurs pour un calcul subséquent, renvoyé à l'ordinateur hôte ou écrit dans la table de données. Mentionnons par exemple le cas du calcul MAX et MIN d'une image: l'intensité maximale ou minimale peut-être utilisée par l'ordinateur hôte pour réajuster le gain de l'étage d'entrée du numérisateur du MVP, ou elle peut être utilisée par tous les PE pour réajuster les intensités afin d'avoir un meilleur contraste.

4.5 La table de données

La table de données est le principal médium de communication qui permet à l'ordinateur hôte et à la micro-machine d'échanger des données avec la matrice de processeurs, à l'exclusion des images qui sont transférées par l'interface du MVP. Telle que montrée à la figure 4.9, la table de données est essentiellement une mémoire de quatre kilo-octets dont l'adresse provient de l'une de trois sources: du bus d'adresse de l'ordinateur hôte par le bus VME, du champ micro-data du micro-mot ou du champ de paramètres du FIFO de commandes. L'ordinateur peut ainsi y écrire directement une forme à reconnaître par exemple, ou y relire un histogramme calculé par la matrice de processeurs. Le registre K sert à convertir les données du format parallèle de la table de données au format sériel des processeurs IMAGE-2.

La capacité de la table de données est suffisamment importante pour plusieurs algorithmes de TNI. Par exemple, un histogramme calculé sur une image de grandeur 512 x 512 dont la résolution est de 8 bits par pixel nécessite 768 octets. En effet, dans le pire cas, on aura 512 x 512 = 262 144 pixels d'une même intensité, ce qui nécessite 3 octets. Puisque 256 intensités sont possibles avec une résolution de 8 bits, l'histogramme occupera 256 x 3 = 768 octets. Cette table peut aussi servir à emmagasiner une forme qui peut rapidement être distribuée à tout les processeurs. Le table est suffisamment grande pour transférer une image qui remplirait complètement la mémoire externe des processeurs qui est de même capacité.

Le chapitre 5 montre, par des exemples d'algorithmes, comment la table de données et les autres circuits périphériques sont utilisés.

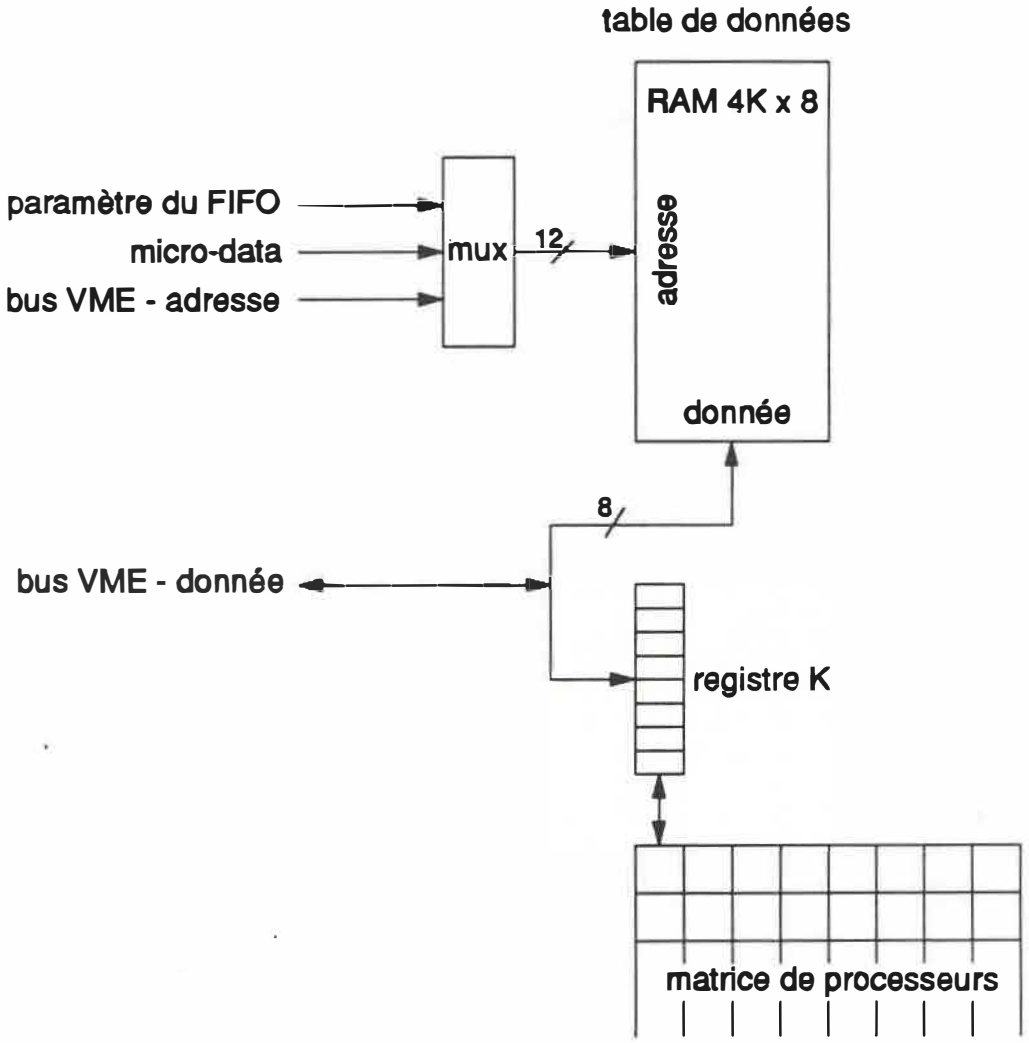


Figure 4.9 La table de données

CHAPITRE 5 - IMPLANTATION D'ALGORITHMES

5.1 Introduction

Tel que mentionné précédemment, les algorithmes de TNI peuvent être classés en deux catégories, les algorithmes de bas niveaux et ceux de haut niveaux.

Le présent document décrit précisément l'architecture de la machine IMAGE qui a été conçu pour effectuer des opérations de TNI de bas niveau. Ce chapitre montre, par l'implantation d'algorithmes connus et fréquemment utilisés, que l'architecture de la machine IMAGE contient des éléments qui permettent d'effectuer cette classe d'opérations efficacement tout en conservant une bonne flexibilité et une complexité modeste [25].

Lougheed [3] a tenté de déterminer les opérations de TNI les plus fréquemment utilisées afin de pouvoir évaluer les performances des machines. Il les classe dans quatre catégories:

1) les opérations locales et de voisinage;
2) les opérations sur plusieurs images;
3) les transformations géométriques;
4) l'extraction d'informations.

Si on restreint notre analyse aux opérations de bas niveaux, seulement les points 1) et 2) sont d'intérêt. D'autres auteurs [26],[27],[28] ont énuméré plus précisément les opérations élémentaires de bas niveaux qui se retrouvent dans la plupart des algorithmes. Ces opérations devront être effectuées efficacement par une

certaine architecture afin d'assurer une bonne performance globale de la machine pour le TNI.

Celles retenues par Levialdi [26] sont:

1) opérations de pixels;

2) opérations de voisinage;

3) opérations globales qui impliquent tous les pixels;

4) calculs statistiques.

Les opérations de pixels n'impliquent qu'un seul pixel à la fois, tandis que les opérations de voisinage impliquent les pixels voisins du pixel traité, généralement une fenêtre de 3x3 pixels. Une opération globale implique tous les pixels d'une image, comme c'est le cas par exemple pour une transformée rapide de Fourier (TRF). Finalement, les calculs statistiques sur une image font référence aux algorithmes qui comptent les pixels, par exemple pour calculer un histogramme d'intensité ou de profil, ou évaluer un périmètre après un étiquetage des pixels.

Les sections suivantes montrent de quelles façons ces types d'algorithmes sont implantés dans la machine IMAGE et les performances atteintes.

5.2 Opérations de pixels

Les opérations de pixels sont généralement simples et très variées. Elles ne nécessitent que la valeur du pixel traité dans une ou plusieurs images et ne modifient que ce pixel. Prenons l'exemple du seuillage d'une image. Le seuillage

consiste à transformer une image A d'une résolution arbitraire, par exemple 8 bits par pixel, à une image B binaire, soit:

$$\text{IF} \quad A_{xy} < \text{seuil} \quad \text{THEN} \quad B_{xy} = 0 \quad \text{ELSE} \quad B_{xy} = 1$$

Tel que montré à la figure 5.1, le seuil est d'abord chargé dans le registre K par l'intermédiaire de la table de données et distribué à tous les PE, tandis que les pointeurs aux images A (8 bits) et B (1 bit) sont respectivement chargés dans les registres d'adresse SOURCE et DESTINATION du générateur d'adresses. Le premier pixel est lu de l'image SOURCE, le seuillage est effectué et le résultat emmagasiné dans l'image DESTINATION. Les deux compteurs sont ensuite incrémentés. Grâce à l'architecture pipeline des PE, l'opération de seuillage, (i.e. une comparaison de deux valeurs de 8 bits) peut s'effectuer en même temps qu'une lecture ou une écriture à la mémoire externe. La figure 5.2 montre le parallélisme des opérations à l'intérieur des PE. Sans parallélisme, le seuillage nécessiterait normalement 17 cycles par pixel, i.e. huit cycles de lecture, huit cycles de seuillage et un cycle d'écriture. Mais puisque la comparaison de l'intensité du pixel avec le seuil à l'intérieur des PE peut s'effectuer en même temps que la lecture du prochain pixel, cette opération mise en parallèle ne demande plus que neuf cycles par pixel en moyenne.

Les registres du générateur d'adresses sont modifiés en parallèle avec l'exécution de l'algorithme dans les PE, ce qui n'influence aucunement le temps d'exécution du programme. La figure 5.3 montre l'organisation de la mémoire si chaque PE traite 1024 pixels. L'image A occupe les 8192 (1024 x 8 bits) premier bits, suivie de l'image B occupant 1024 bits (1024 x 1 bit). La séquence des

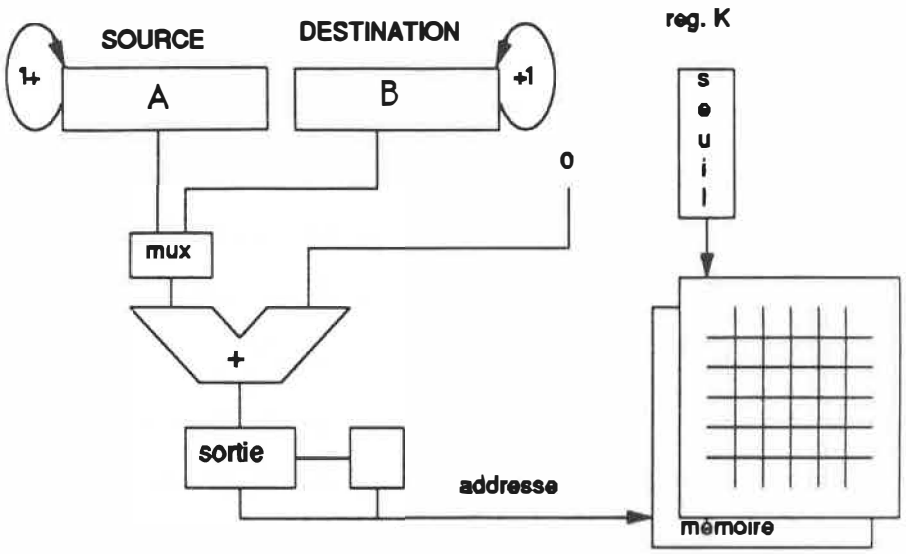


Figure 5.1 Opération du seuillage

L = Lecture d'un bit du pixel traité
S = Seuillage de chaque bit
E = Ecriture du résultat

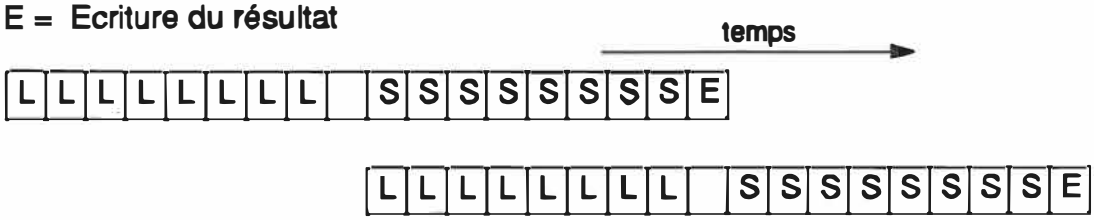


Figure 5.2 Opération de seuillage avec pipeline

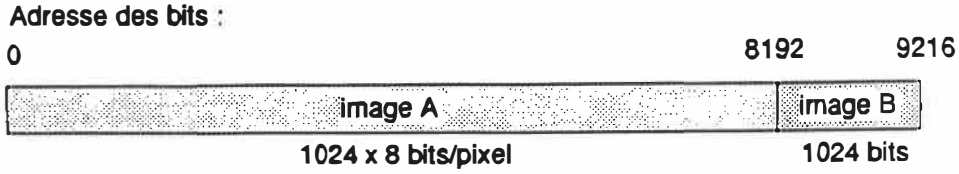


Figure 5.3 Organisation de la mémoire des PE pour un seuillage

adresses est montrée au tableau 5.1(a), chaque ligne étant un cycle d'horloge. Les deux premières colonnes représentent le contenu des registres SOURCE et DESTINATION chargés respectivement des valeurs 0 et 8192 au départ. La troisième colonne représente l'autre entrée de l'additionneur du générateur d'adresses, qui reste à zéro dans ce cas. La colonne suivante est l'état du compteur de bits du générateur d'adresses, suivie d'une colonne spécifiant l'opération équivalente effectuée par le décaleur à la sortie du générateur d'adresses. Lors de la lecture du pixel traité de 8 bits, l'adresse source n'est pas incrémentée pour chaque bit, mais elle est plutôt décalée vers la gauche de trois bits (i.e. multiplié par 8). Le compteur de bit occupe les trois bits les moins significatifs et est incrémenté pour accéder chaque bit du pixel traité d'où l'opération 8 * SRC + CB. La dernière colonne représente l'adresse résultante qui est envoyée aux mémoires externes des PE.

Avec l'aide de l'additionneur, on aurait pu utiliser qu'un seul registre du générateur d'adresse sans perte de performance. Effectivement, pendant qu'un pixel est traité, on peut lire le suivant en incrémentant le registre d'adresse. Le tableau 5.1(b) montre le séquencement des adresses dans ce cas. Le registre SOURCE pointe vers l'image A, et on additionne 8191 pour accéder le pixel correspondant dans l'image B. Le registre DESTINATION est libre et peut servir à d'autres fonctions.

Le temps total pour effectuer un seuillage sur une image de taille 512 x 512 avec 256 processeurs est d'environ 0,92 ms, soit:

9 cycles/pixel X 100 ns/cycle X 1024 pixels/processeur

Tableau 5.1(a) Séquence des adresses pour un seuillage en utilisant les deux registres d'adresse.

t	SRC	DEST	ADD	CPT.BIT	OPÉRATION	ADRESSE	INSTRUCTION
1	0	8192	0	0	8xSRC+CB	0	Lect. pix. 1 bit 0
2	0	8192	0	1	8xSRC+CB	1	Lect. pix. 1 bit 1
3	0	8192	0	2	8xSRC+CB	2	Lect. pix. 1 bit 2
4	0	8192	0	3	8xSRC+CB	3	Lect. pix. 1 bit 3
5	0	8192	0	4	8xSRC+CB	4	Lect. pix. 1 bit 4
6	0	8192	0	5	8xSRC+CB	5	Lect. pix. 1 bit 5
7	0	8192	0	6	8xSRC+CB	6	Lect. pix. 1 bit 6
8	0	8192	0	7	8xSRC+CB	7	Lect. pix. 1 bit 7
9							
10	1	8192	0	0	8xSRC+CB	8	Lect. pix. 2 bit 0
11	1	8192	0	1	8xSRC+CB	9	Lect. pix. 2 bit 1
12	1	8192	0	2	8xSRC+CB	10	Lect. pix. 2 bit 2
13	1	8192	0	3	8xSRC+CB	11	Lect. pix. 2 bit 3
14	1	8192	0	4	8xSRC+CB	12	Lect. pix. 2 bit 4
15	1	8192	0	5	8xSRC+CB	13	Lect. pix. 2 bit 5
16	1	8192	0	6	8xSRC+CB	14	Lect. pix. 2 bit 6
17	1	8192	0	7	8xSRC+CB	15	Lect. pix. 2 bit 7
18	1	8192	0	0	8xSRC+CB	8192	Ecriture pix. 1
19	1	8193	0	0	8xSRC+CB	16	Lect. pix. 3 bit 0

Tableau 5.1(b) Séquence des adresses pour un seuillage en utilisant un seul registre d'adresse.

t	SRC	DEST	ADD	CPT.BIT	OPÉRATION	ADRESSE	INSTRUCTION
1	0		0	0	8xSRC+CB	0	Lect. pix. 1 bit 0
2	0		0	1	8xSRC+CB	1	Lect. pix. 1 bit 1
3	0		0	2	8xSRC+CB	2	Lect. pix. 1 bit 2
4	0		0	3	8xSRC+CB	3	Lect. pix. 1 bit 3
5	0		0	4	8xSRC+CB	4	Lect. pix. 1 bit 4
6	0		0	5	8xSRC+CB	5	Lect. pix. 1 bit 5
7	0		0	6	8xSRC+CB	6	Lect. pix. 1 bit 6
8	0		0	7	8xSRC+CB	7	Lect. pix. 1 bit 7
9							
10	1		0	0	8xSRC+CB	8	Lect. pix. 2 bit 0
11	1		0	1	8xSRC+CB	9	Lect. pix. 2 bit 1
12	1		0	2	8xSRC+CB	10	Lect. pix. 2 bit 2
13	1		0	3	8xSRC+CB	11	Lect. pix. 2 bit 3
14	1		0	4	8xSRC+CB	12	Lect. pix. 2 bit 4
15	1		0	5	8xSRC+CB	13	Lect. pix. 2 bit 5
16	1		0	6	8xSRC+CB	14	Lect. pix. 2 bit 6
17	1		0	7	8xSRC+CB	15	Lect. pix. 2 bit 7
18	1		8191	0	SRC+ADD	8192	Ecriture pix. 1
19	2		0	0	8xSRC+CB	16	Lect. pix. 3 bit 0

5.3 Opérations de voisinage

Dans une opération de voisinage, la nouvelle valeur du pixel traité dépend de l'intensité de ce pixel et des pixels avoisinant ce point. Ces opérations sont utilisées pour la détection d'arêtes, la réduction du bruit, le lissage, etc.

La plupart de ces algorithmes utilisent un masque qui sert à calculer une moyenne pondérée des pixels avoisinant. Les plus populaires de ces masques sont de dimension 3x3, tels que les masques de Sobel, Prewitt et du Laplacien [29], et utilisent des coefficients qui sont des puissances de deux pour faciliter les calculs.

Prenons l'exemple du masque de la figure 5.4 pour détecter les arêtes verticales d'une image. Dans le cas général, les coefficients sont emmagasinés dans la table de données et transférés aux processeurs par le registre K. Si les coefficients sont simples, ils peuvent être codés directement dans le micro-code comme des additions ou des soustractions décalées. Effectivement, puisque les processeurs sont sériels par bit, lire un pixel de 8 bits et le multiplier par deux est équivalent à lire les sept bits les moins significatifs et emmagasiner ceux-ci dans les 7 bits les plus significatifs, c'est-à-dire un décalage vers la gauche. Dans ce cas, la multiplication ne prend aucun cycle supplémentaire. Une convolution, utilisant le masque de la figure 5.4 nécessite donc 6 lectures, 5 additions ou soustractions et une écriture pour chaque pixel traité. Chaque lecture, écriture, addition ou soustraction prend huit cycles. Telle que montrée à la figure 5.5(a), une exécution séquentielle nécessiterait 96 cycles (12 opérations de 8 cycles) tandis que la figure 5.5(b) montre l'exécution pipeline de cette opération qui compte seulement 64 cycles (8 X 8).

-2	0	2
-2	0	2
-2	0	2

Figure 5.4 Exemple de masque de convolution

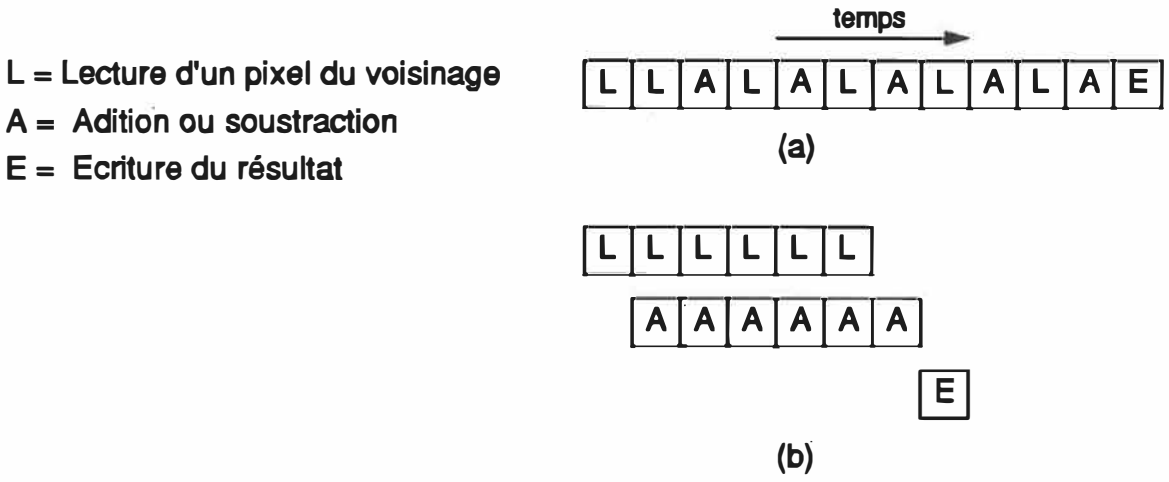


Figure 5.5 Opération de convolution sur un pixel
(a) sans pipeline (b) avec pipeline

La séquence d'adresses nécessaire pour exécuter une convolution est facilement réalisable grâce à l'additionneur du générateur d'adresse telle que montrée au tableau 5.2. Chaque ligne de ce tableau représente maintenant 8 cycles puisque toutes les opérations se font sur 8 bits, i.e le compteur de bits est incrémenté de 0 à 7. Dans ce genre de calcul, un registre-compteur pointe le pixel traité, tandis que l'additionneur permet d'adresser les pixels voisins. Par exemple, si chaque pixel traite une région de 32x32 pixels avec un masque de 3x3, la figure 5.6 montre les valeurs à additionner à l'adresse du pixel traité pour accéder les pixels voisins dans un mode d'adressage pyramidal. Par exemple, si on traite le pixel à l'adresse mémoire 150, le pixel situé au NORD est à l'adresse mémoire 150 - 32 = 118. Les pixels en bordures, qui comptent généralement pour un faible pourcentage des pixels traités, doivent subir un traitement légèrement différent parce que les pixels voisins sont situés dans la mémoire d'un processeur voisin. Le pixel sera donc reçu du canal de communication inter-processeur plutôt que de la mémoire externe.

-33	-32	-31
-1	0	1
31	32	33

Figure 5.6 Décalage des adresses des pixels voisins

Une telle opération de convolution prendrait donc, pour chaque pixel, 64 cycles fois 100 ns/cycle, i.e. 6.4 us. Si chaque processeur traite 1024 pixels, le temps total sera de 6.4 ms.

Tableau 5.2 Séquence des adresses pour une opération de convolution. Adresse source = 150 et 151, destination = 8342 et 8344 (taille de la région = 32 x 32)

SRC	DEST	ADD	CPT.BIT	OPÉRATION ADR.	INSTRUCTION
150	8342	0	0-7	8xSRC+CB+ADD150	Lect. central
150	8342	-33	0-7	8xSRC+CB+ADD117	Lect.NORD-OUEST
150	8342	-32	0-7	8xSRC+CB+ADD118	Lect. NORD
150	8342	-31	0-7	8xSRC+CB+ADD119	Lect. NORD-EST
150	8342	-1	0-7	8xSRC+CB+ADD149	Lect. OUEST
150	8342	1	0-7	8xSRC+CB+ADD151	Lect. EST
150	8342	31	0-7	8xSRC+CB+ADD181	Lect. SUD-OUEST
150	8342	32	0-7	8xSRC+CB+ADD182	Lect. SUD
150	8342	33	0-7	8xSRC+CB+ADD183	Lect. SUD-EST
150	8342				
150	8342	0	0-7	8xSRC+CB+ADD8342	Ecriture
151	8343	0	0-7	8xSRC+CB+ADD 151	Lect. central
151	8343	-33	0-7	8xSRC+CB+ADD118	Lect.NORD-OUEST
151	8343	-32	0-7	8xSRC+CB+ADD119	Lect. NORD
151	8343	-31	0-7	8xSRC+CB+ADD120	Lect. NORD-EST
151	8343	-1	0-7	8xSRC+CB+ADD150	Lect. OUEST
151	8343	1	0-7	8xSRC+CB+ADD152	Lect. EST
151	8343	31	0-7	8xSRC+CB+ADD182	Lect. SUD-OUEST
151	8343	32	0-7	8xSRC+CB+ADD183	Lect. SUD
151	8343	33	0-7	8xSRC+CB+ADD184	Lect. SUD-EST
151	8343				
151	8343	0	1	8xSRC+CB+ADD8343	Ecriture

5.4 Transformations globales

Les transformations globales impliquent tous les pixels de l'image traitée et génèrent une nouvelle image. Parmi celles-ci, notons la transformée rapide de Fourier (TRF) [30]. Cette opération est relativement complexe pour des processeurs simples tels qu'IMAGE-2, mais elle peut tirer avantage de l'architecture de la machine IMAGE puisque la TRF est facilement mise en parallèle. Sans entrer dans les détails de son implantation, mentionnons que les coefficients $e^{2\pi i/N}$ sont emmagasinés dans la table de données et les nombreux registres internes de douze bits des PE peuvent calculer des nombres à virgules flottantes efficacement. Les circuits de l'interface du MVP utilisés pour réordonner les pixels selon le mode d'adressage de la matrice (voir section 4.3.4) peuvent aussi être utilisés pour lire les pixels dans un ordre bit renversé afin de facilité le calcul de la TRF.

5.4 Calculs statistiques

Les algorithmes qui effectuent des calculs statistiques sur une image comptent les pixels qui ont une caractéristique commune pour en retirer un résultat global. L'un des plus utilisé est l'histogramme d'intensité. L'histogramme d'intensité consiste à compter le nombre de pixels de chaque intensité dans une image. L'implantation de cet algorithme sur des processeurs mis en parallèle a été étudié par plusieurs auteurs [11,31]. Ces méthodes sont résumés par [32] qui en retient deux principales, l'approche 'bin-spreading' et 'bin-collecting'. Dans la première, chaque processeur compte le nombre de pixels d'une certaine intensité dans toute l'image, tandis que dans la deuxième, chaque PE calcule un histogramme local de sa sous-image et ces histogrammes sont ensuite combinés

dans un seul processeur. Les deux approches sont facile à implanter dans IMAGE, l'approche 'bin-spreading' nécessite que toute l'image soit lue par chaque processeur, on devra alors configurer la matrice en mode ligne de sorte que les processeurs puissent transférer leur sous-image respective à tous les autres processeurs. L'approche 'bin-collecting', par contre, n'utilise les canaux de communication qu'à la toute fin de l'algorithme, mais nécessite 512 octets de la mémoire locale de chaque PE pour emmagasiner un histogramme local.

L'algorithme représenté à la figure 5.7 combine ces deux méthodes et ne nécessite aucun octet supplémentaire de la mémoire externe. Pour chaque intensité de pixel, chaque PE compte le nombre de pixels dans sa sous-image. Les comptes partiels sont ensuite accumulés dans le PE situé au coin NORD-OUEST de la matrice, i.e celui dont la sortie est connectée au registre K. La somme totale est ensuite écrite dans la mémoire de données où sera construit l'histogramme globale. Chaque PE devra donc exécuter les opérations suivantes pour chaque pixel, et pour chaque intensité possible:

1- lecture du pixel;
2- comparaison avec l'intensité calculée;
3- incrémentation conditionnelle d'un compteur.

Les étapes 1 et 2 sont exécutées en parallèle et prennent 10 cycles, tandis que l'incrémentation prend aussi 10 cycles en moyenne pour un total de 20 cycles par pixel. Pour une image de 512x512 pixels, chaque PE d'une matrice de 256 processeurs comptent 1024 pixels. Le temps total sera donc de 20 cycles X 1024 pixels X 256 intensités = 5242880 cycles i.e 0,52 seconde. Le temps d'accumulation des comptes partiels ne compte que pour un faible pourcentage du

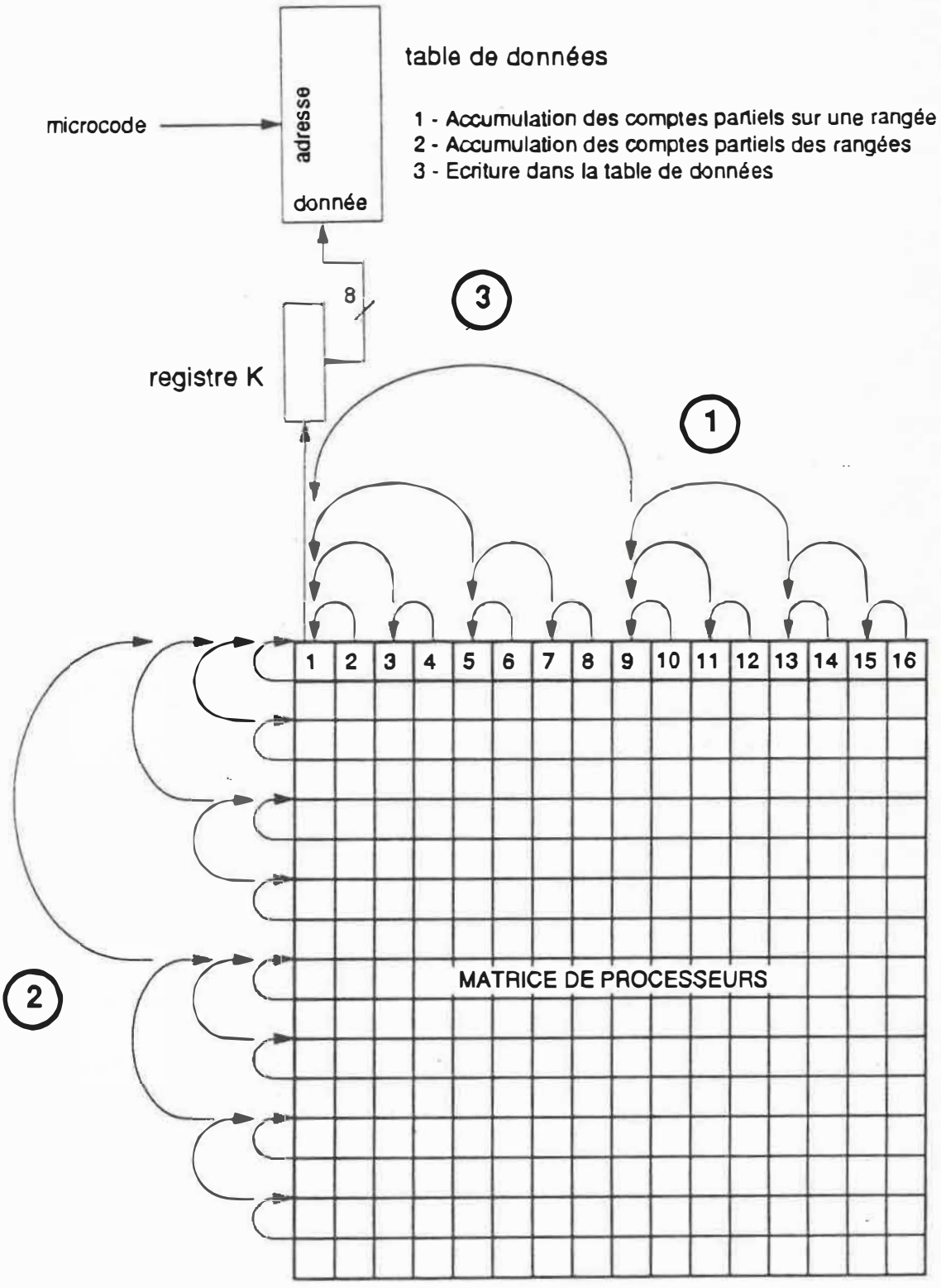


Figure 5.7 - Calcul d'un histogramme

temps total est s'effectue en trois étapes telles que représentées à la figure 5.7. D'abord, on accumule les comptes de chaque PE d'une même rangée dans la colonne OUEST de processeurs. Cette accumulation s'effectue en quatre phases de la façon suivante. Premièrement, chaque processeur additionne son compte avec celui de son voisin EST. Ce compte a une valeur maximal de 1024 dans cet exemple, l'addition prendra donc 12 cycles (2 nombres de 11 bits) plus deux cycles de délais de communication pour le transfert des données. Le PE #1 a donc le compte total des PE #1 et #2, le PE #3 a le total des PE # 3 et #4 et ainsi de suite. On répète cette opération, mais avec les huit processeurs des colonnes 1,3,5,7,9,11,13 et 15. Cette addition prendra cette fois-ci 13 cycles plus 4 cycles pour les communications puisque les données doivent traverser deux processeurs. Les troisième et quatrième phases prennent respectivement 22 (14+8) et 31 (15+16) cycles pour un total de 84 cycles pour l'étape 1. L'étape 2 est similaire alors qu'on accumule toutes les sommes des rangées dans le PE du coin supérieur gauche. Cette opération nécessite (16+2) + (17+4) + (18+8) + (19+16) = 100 cycles. Finalement, la dernière étape consiste à écrire cette somme dans la table de données. Puisque le total maximal est 262 144 pour une image de taille 512 X 512, on doit écrire trois octets pour chacune des 256 intensités, ce qui prend (3X8) + 2 = 26 cycles. Le temps total pour l'accumulation des comptes d'une intensité est de:

84 + 100 + 26 = 210 cycles, i.e 210 X 100ns = 5 ms.

Le temps total du calcul d'un histogramme est donc d'environ:

520 ms + 5 ms = 0,53 seconde.

Les deux approches initialement mentionnées nécessiteraient sensiblement le même temps puisque chaque processeur devra de toute façon exécuter les trois opérations mentionnées précédemment 262 144 fois, ce qui compte pour la plus grande part du temps de calcul pour une grande image.

CHAPITRE 6 - MODÉLISATION ET VÉRIFICATION

6.1 Aspects de la simulation

Avec l'évolution de l'électronique et de l'informatique, les techniques de développement de nouveaux circuits se sont grandement améliorées. Dans le passé, les concepteurs de circuits électroniques élaboraient des prototypes sur des plaquettes ou des supports à câblage (wire-wrap), mais ces techniques se sont vite avérées inefficaces et inadéquates avec l'évolution de la complexité et des fréquences d'opération des circuits.

De plus en plus, les concepteurs de circuits électroniques, que ce soit au niveau du circuit intégré, au niveau de la carte ou au niveau du système, utilisent des simulateurs pour valider leurs circuits. Les simulateurs permettent des vérifications qui sont souvent pratiquement impossibles à réaliser avec les circuits physiques, telles que la simulation de circuit défectueux, l'analyse de noeuds internes, le relevé de données paramétriques, etc.

Particulièrement dans le cas des circuits intégrés, il est maintenant possible de modéliser le fonctionnement du c.i. et de valider le design en évitant l'effort, les dépenses et le temps nécessaire à la production de plusieurs prototypes. Le simulateur permet d'étudier le comportement de circuits beaucoup plus complexes et à des niveaux d'abstraction beaucoup plus hauts que le permet l'analyse traditionnelle avec papier et crayon, sans compter les possibilités d'erreurs qui sont d'autant plus réduites.

6.2 Outils de modélisation et de simulation

6.2.1 Description du matériel

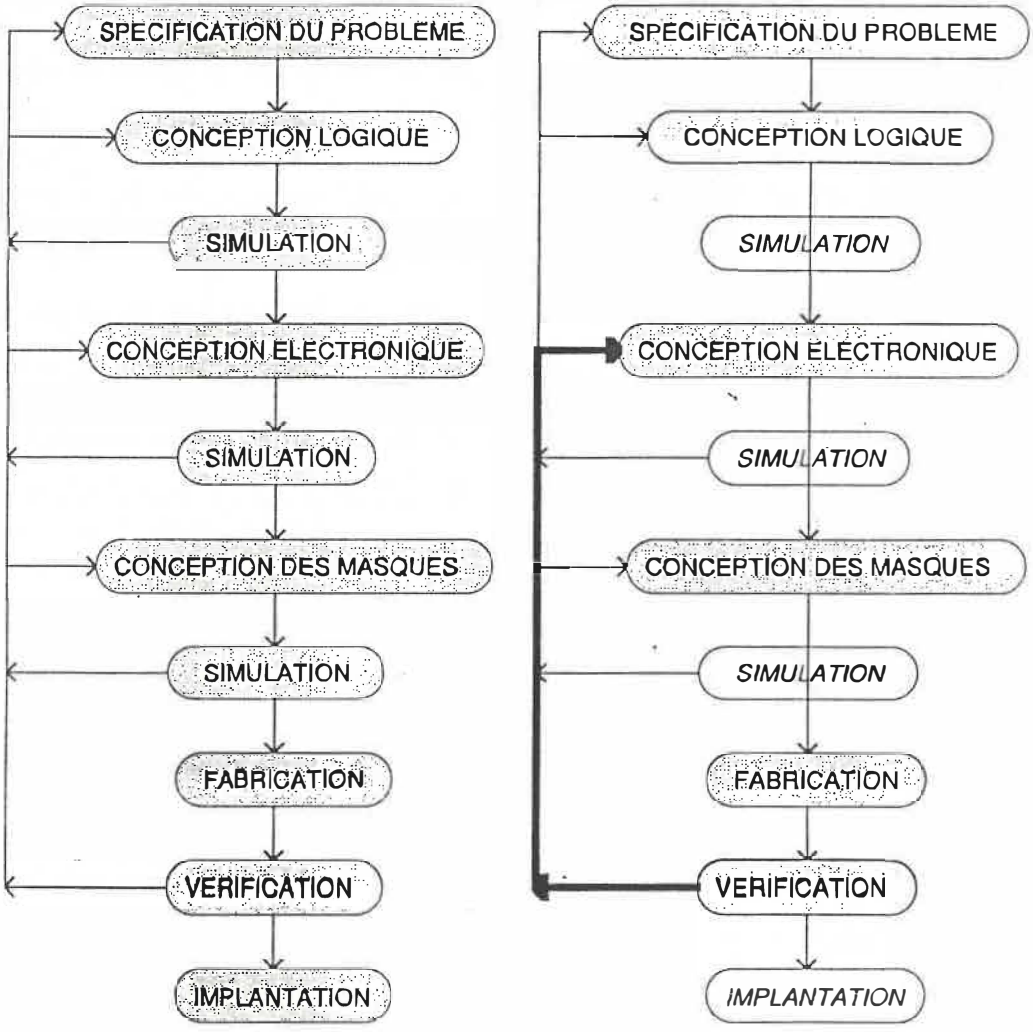
L'Ecole Polytechnique de Montréal a acquis récemment des outils de conception assistée par ordinateur de première qualité, les logiciels de MENTOR GRAPHICS et les postes de travail APOLLO. Ces outils de qualité industrielle ont déjà fait leurs preuves et sont utilisés dans plusieurs pays ainsi que chez plusieurs compagnies de la région montréalaise telles que Matrox, Marconi et Philips.

Les logiciels de Mentor Graphics permettent de modéliser et de simuler, dans un environnement intégré, des circuits électroniques divers. Un éditeur de schéma sophistiqué permet de modéliser des circuits complexes grâce à l'utilisation de niveaux hiérarchiques qui permettent de travailler à différents niveaux d'abstraction. Ceci rend l'analyse de cellules complexes pratiquement aussi aisées que l'analyse des cellules simples.

Les logiciels de Mentor Graphics jouissent aussi d'une grande diversité de modèles qui peuvent être utilisés dépendant du type d'analyse qu'on veut faire.

6.2.2 Processus de développement

A l'Ecole Polytechnique, ces outils servent surtout présentement au développement de nouveaux circuits intégrés. Avec ces outils, un cheminement qui augmenterait grandement les chances de succès d'un circuit intégré est représenté,



(a) cas général

(b) IMAGE-2, première version

Figure 6.1 - Processus de développement d'un circuit intégré

d'une façon simplifiée, à la figure 6.1(a) [33,34].

Une fois le problème clairement spécifié, on effectue une modélisation logique de haut niveau suivie d'une simulation. Si les résultats des simulations sont satisfaisants, on transforme le modèle à un niveau plus bas qui correspond plus fidèlement au comportement électrique des circuits. Si les résultats de ces simulations sont encore acceptables, on passe à la conception des masques qui sera suivie de simulations analogiques pour tester les parties critiques du circuit. Finalement, après la fabrication des c.i., on vérifie l'implantation.

Dans le cas du premier prototype du c.i. IMAGE-2, les étapes de simulation ont été sautées ou partiellement complétées faute d'outils adéquats. La figure 6.1(b) montre le chemin suivi. Dû au manque de simulations, la vérification a montré une grande quantité d'erreurs et on s'est retrouvé sur le chemin du retour vers la conception et la modélisation tel que montré par la flèche en caractère gras.

Dans le cas du c.i. IMAGE-2, les étapes de développement du deuxième prototype sont schématisées à la figure 6.2. Premièrement, le schéma au niveau des transistors a été extrait à partir des masques du premier prototype. Ces schémas ont été ensuite modélisés avec les logiciels de Mentor Graphics pour y être simulés. Le résultat de ces simulations nous ont amenés à faire des corrections nécessaires sur les dessins des masques. Finalement, la deuxième version du c.i. a été fabriquée et vérifiée. Les sections suivantes décrivent le modèle utilisé et les procédures de vérification du c.i. IMAGE-2.

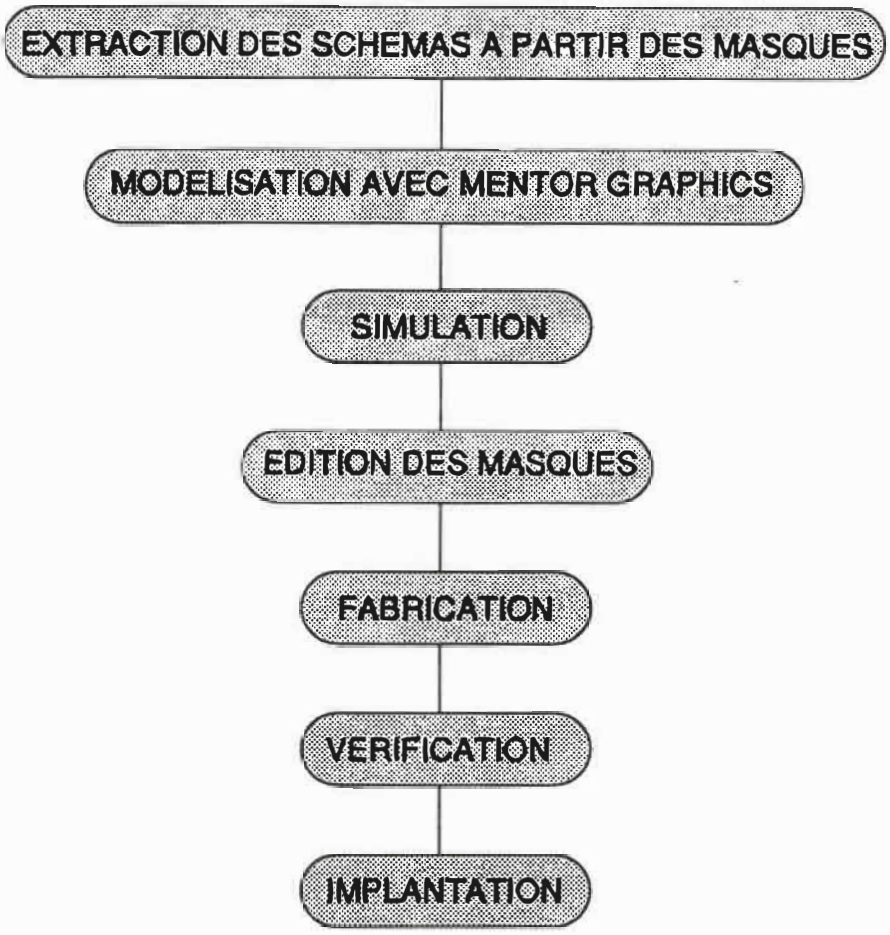


Figure 6.2 - Processus de développement de la deuxième version d'IMAGE-2

6.3 Modélisation du c.i. IMAGE-2

6.3.1 Buts

Etant donnée la complexité d'IMAGE-2 qui compte environ 11 000 transistors, la modélisation et la simulation permettent d'augmenter considérablement les chances de succès au niveau fonctionnel du c.i.. Daniel Audet avait déjà simulé plusieurs sections critiques du circuit avec le simulateur analogique SPICE disponible pour les postes de travail SUN, mais SPICE ne peut simuler que quelques dizaines de transistors tout au plus. Par contre, le simulateur de circuits logiques de Mentor Graphics permet de faire des simulations de circuits beaucoup plus complexes et beaucoup plus rapidement.

Cette modélisation allait aussi nous permettre de développer une documentation complète d'IMAGE-2. Jusqu'à ce jour, seulement des dessins faits à la main un peu éparpillés étaient disponibles, et ils n'avaient pas été maintenus à jour. Une implantation sur ordinateur du modèle permet de consulter et de maintenir à jour les schémas du circuit intégré facilement [2].

6.3.2 Types de modèles

Les modèles forment le coeur de toutes simulations et sont la clé du succès d'une simulation. Idéalement, on voudrait un modèle qui reproduit le plus exactement possible le comportement réel d'une composante tout en étant le plus rapide possible. Evidemment, ces deux contraintes s'opposent. Le modèle a donc un effet direct sur la qualité et le durée d'une simulation.

Ce sont les simulateurs analogiques MSPICE et MSIMON qui reproduisent le mieux le comportement électronique d'un circuit CMOS (Complementary Metal Oxide Semiconductor) parmi les outils de Mentor Graphics. Ces simulateurs sont très précis, mais peu pratiques lorsque vient le temps de simuler le comportement de plusieurs dizaines de transistors. C'est pourquoi on utilise un simulateur logique qui peut être utilisé avec plusieurs types de modèles dépendamment de la complexité du circuit ainsi que de la précision et de la performance désirées.

La figure 6.3 représente graphiquement les différents modèles logiques suivants [35]:

(a) le **modèle d'interrupteur** qui utilise les composantes les plus primitives telles que les transistors.

(b) le **modèle de porte** qui utilise des portes logiques élémentaires telles que ET, OU, XOR, etc.

(c) le **modèle fonctionnel** qui représente une fonction distincte d'un circuit telles qu'un additionneur, une bascule et une mémoire.

(d) le **modèle de comportement** qui décrit un circuit à l'aide d'un langage de programmation (C ou Pascal) et qui définit le comportement algorithmique d'un circuit.

(e) le **modèle VHDL** (VHSIC Hardware Description Langage) (VHSIC - Very High Speed Integrated Circuit) qui permet de décrire une composante, un circuit ou un système en terme de sa structure, de son comportement ou de son flot de données.

6.3.3 Description du modèle utilisé

Les circuits MOS que l'on retrouve dans le c.i. IMAGE-2 sont difficilement modélisables à l'aide de la logique traditionnelle à portes ou à relais. Plusieurs structures particulières ne sont pas représentables uniquement par ces modèles de circuits logiques, c'est pourquoi plusieurs simulateurs utilisent des techniques pour contourner ces problèmes. Hayes [36] résume bien les particularités des circuits MOS:

1) le transistor MOS agit comme interrupteur bidirectionnel à trois terminaux;

2) les circuits MOS contiennent des éléments de charge qui peuvent, dans certaines circonstances, tirer un signal à 1 ou à 0;

3) la logique câblée est grandement utilisée dans les circuits MOS alors qu'elle est souvent considérée comme une anomalie dans la logique traditionnelle;

4) les circuits logiques traditionnels utilisent les deux états 0 et 1 alors que les circuits MOS utilisent d'autres états comme Z, i.e haute-impédance;

5) les délais sont normalement représentés par des délais de lignes, alors que les délais des circuits MOS prennent plutôt la forme d'une capacité rattachée à un noeud.

Afin de contourner quelques-uns de ces problèmes, le simulateur Quicksim de Mentor Graphics utilise trois niveaux logiques et quatre intensités de signaux. Il inclut aussi des modèles de transistors MOS (fig. 6.4) qui agissent comme des interrupteurs bidirectionnels à trois terminaux. De plus un délai de propagation peut être rattaché aux points de connexion des composantes.

Les trois niveaux logiques sont 0,1 et X (inconnu). Les quatre intensités de signaux sont S(fort (strong)), R(résistif), Z(haute-impédance) et I(indéterminé). Le symbole S) est utilisé pour une sortie active; R) pour la charge d'une résistance tire-haut ou tire-bas; Z) pour une ligne flottante; et I) pour une force indéterminée.

La combinaison de ces niveaux logiques et des ces intensités crée 12 états de signaux tels que représentés à la table 6.1. La table 6.2 montre comment les collisions entre différents états logiques sont résolues. Par exemple, un signal à un niveau logique 0 de forte intensité (0S) en collision avec un niveau 1 résistif (1R) résultera en un niveau 0 fort (0S).

Table 6.1 - Etats des signaux de simulation

INTENSITES	valeurs logiques 0	1	X
S	0S	1S	XS
R	0R	1R	XR
Z	0Z	1Z	XZ
I	0I	1I	XI

Table 6.2 - Résolution des contentions des signaux

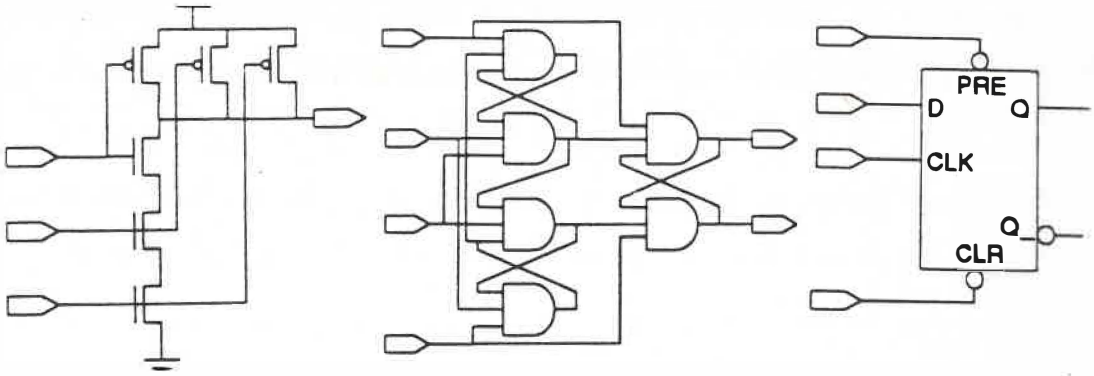
0Z	0Z											
XZ	XZ	XZ										
1Z	XZ	XZ	XZ									
0R	0R	0R	0R	0R								
XR	XR	XR	XR	XR	XR							
1R	1R	1R	1R	1R	1R	1R						
0I	0I	0I	0I	0I	0I	0I	0I					
XI	XI	XI	XI	XI	XI	XI	XI	XI				
1I	1I	1I	1I	1I	1I	1I	1I	1I	1I			
0S	0S	0S	0S	0S	0S	0S	0S	0S	0S	0S		
XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	
1S	1S	1S	1S	1S	1S	1S	1S	1S	1S	1S	1S	1S
	0Z	**XZ**	**1Z**	**0R**	**XR**	**1R**	**0I**	**XI**	**1I**	**0S**	**XS**	**1S**

Dans le cas du c.i. IMAGE-2, seulement les six composantes représentées à la figure 6.4 étaient nécessaires pour décrire le c.i., soit les transistors MOS de type P et N, les résistances tire-haut et tire-bas et les deux alimentations Vcc et MASSE.

6.3.4 Qualité de la modélisation

Les modèles utilisés sont relativement simples et permettent des simulations rapides du comportement logique des circuits CMOS. Par contre, dans le cas du c.i. IMAGE-2, on s'est permis certaines libertés dans l'arrangement des transistors qui sont difficilement simulables à l'aide d'un simulateur logique. En particulier, le c.i. IMAGE-2 est parsemé de circuits logiques à préchargement qui, lorsque simulés, ne reflètent pas toujours le comportement réel. Les modèles utilisés ne détecteront pas non plus les problèmes de partage de charges, commun dans un design MOS tel qu'IMAGE-2.

L'un des problèmes dans l'utilisation des outils de Mentor Graphics est l'absence de logiciel de transfert entre les postes de travail SUN et les postes de travail APOLLO pour transformer la description d'un circuit de l'éditeur de masques KIC (sur SUN) à l'éditeur de schéma NETED (sur APOLLO-MENTOR). Il a donc fallu extraire le schéma des circuits à partir du dessin des masques avec KIC (SUN) et les entrer à la main sur un poste de travail APOLLO. Cette procédure est une source importante d'erreurs puisqu'il n'y a aucun moyen de vérifier la concordance des circuits dans les deux postes de travail.



(a) modèle d'INTERRUPTEUR d'une porte NON-ET

(b) modèle de PORTE d'une bascule D

(c) modèle FONCTIONNEL d'une bascule D

```
IF CLKRISE THEN
IF PRE and CLR = 1
THEN
IF D = 0 THEN
Q = 0 AND QBAR = 1
```

(d) modèle de COMPORTEMENT d'une bascule D

```
PROCESS ( d0, d1, sel )
BEGIN
  IF sel = '0' THEN
    q <= d1 ;
  ELSE
    q <= d0 ;
  END IF ;
END PROCESS ;
```

(e) modèle en VHDL d'un multiplexeur

Figure 6.3 - Différents modèles de simulation

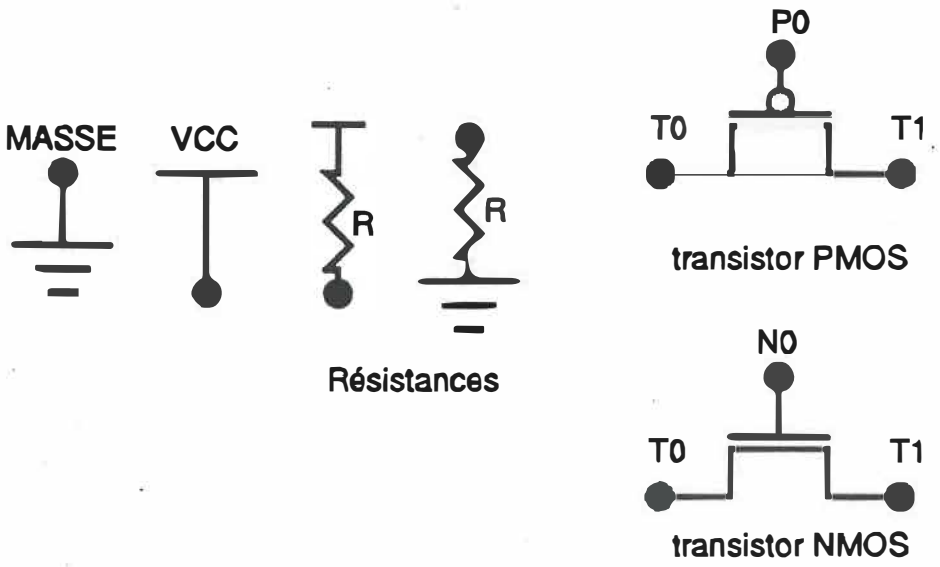


Figure 6.4 - Eléments du modèle utilisé pour IMAGE-2

Malgré tout, les outils de simulations de Mentor Graphics se sont avérés indispensables et efficaces pour la détection et la correction des erreurs du c.i. IMAGE-2. D'ailleurs, la fabrication et la vérification du c.i. qui ont suivi cette étape de modélisation et de simulation a été un grand succès.

6.4 Vérification

6.4.1 Méthode utilisée

Traditionnellement, la vérification d'un circuit intégré s'effectue à l'aide de vecteurs de test qui vérifient la plupart des noeuds du circuit. Dans le cas d'IMAGE-2, cette méthode était difficilement utilisable pour plusieurs raisons. Premièrement, malgré toutes les précautions prises, le modèle du c.i. créé avec les outils de Mentor Graphics pouvait contenir des erreurs, ce qui aurait rendu les vecteurs de test, générés à l'aide de ce modèle, inutilisables. Deuxièmement, étant donné la complexité du c.i., le nombre et la longueur des vecteurs auraient été trop grand.

Afin de tester plus efficacement IMAGE-2, des outils simples ont été développés qui permettent de générer facilement des séquences de bits pour vérifier des sections du c.i.. Ces séquences sont facilement modifiables si les schémas ne correspondent pas exactement au c.i. fabriqué. Afin d'effectuer les tests de cette façon, il a fallu développer un environnement matériel et logiciel qui est expliqué dans les prochaines sections.

6.4.2 Environnement matériel

Lorsque nos travaux ont été exécutés, les équipements de test disponibles au laboratoire de ITGE étaient composés des postes de travail SUN, d'un terminal HP 9133 programmable en langage BASIC, d'un générateur de vecteurs HP 8151, d'un analyseur de signaux HP 8152 et d'un oscilloscope 100 MHz. Il a de plus fallu développer une plaquette de test en circuit imprimé qui sert d'interface entre le circuit à tester et les sondes du générateur, de l'analyseur et de l'oscilloscope.

6.4.2.1 La plaquette de test

La plaquette de test, dessinée à la figure 6.5, sert d'interface entre le c.i. à tester et les appareils de test. Tel que montré à la figure 6.5, chacune des broches du c.i. peut se connecter aux sondes des appareils de mesures ou bien être forcée à un niveau logique ou à un voltage déterminé, tel que suggéré par J. Barette. Les différents modes de connexion se font à l'aide de cavaliers. Les différentes configurations possibles sont les suivantes: une connexion ouverte, à l'alimentation Vcc ou à la terre, et à une résistance tire-haut, tire-bas ou aux deux simultanément. De plus, la plaquette de test est munie de deux plaquettes de prototype qui peuvent contenir des circuits supplémentaires.

6.4.2.2 Le générateur de vecteurs et l'analyseur de signaux.

Le générateur de vecteurs HP 8151 permet d'envoyer une séquence de bits à la plaquette de test. Ses caractéristiques principales sont:

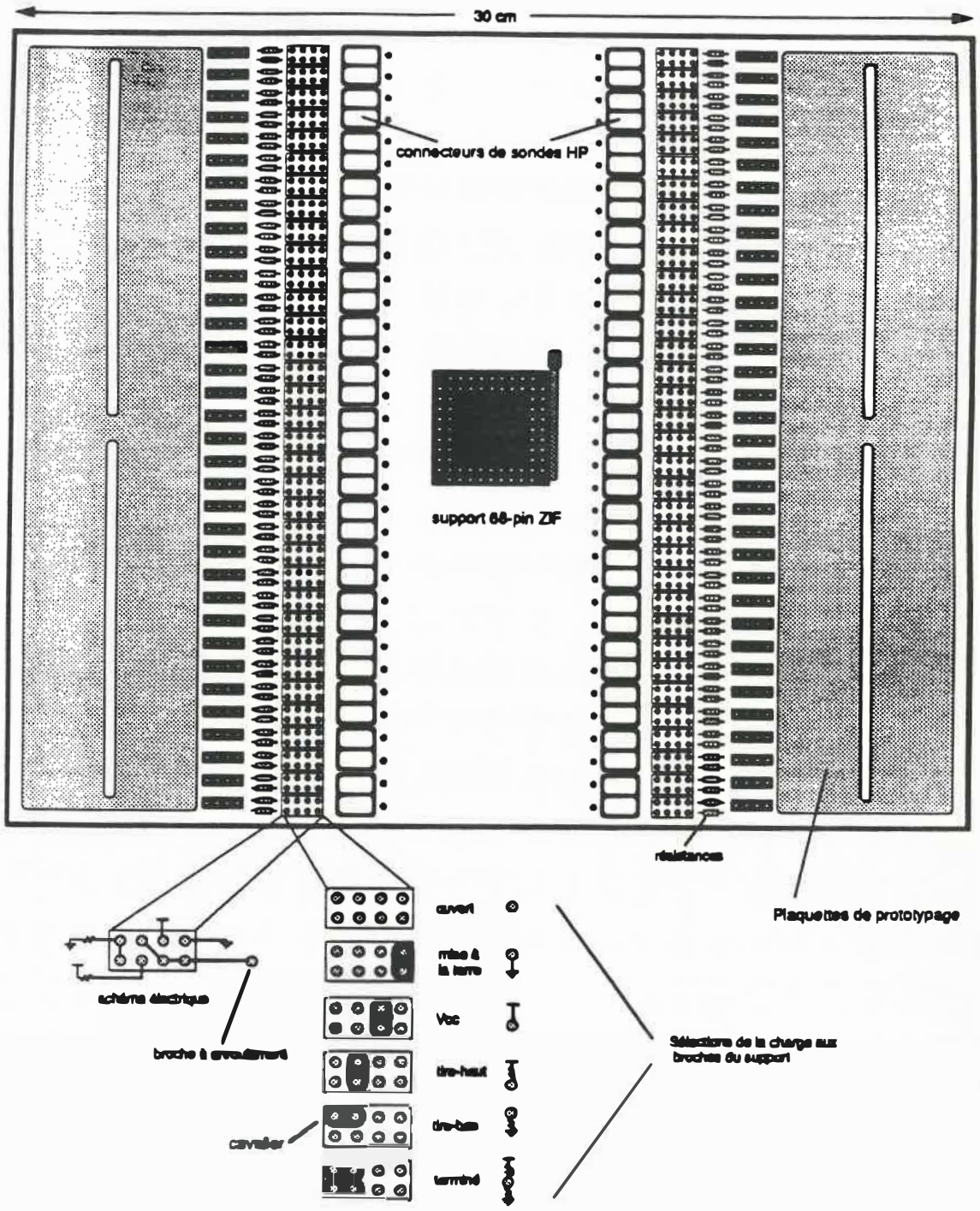


Figure 6.5 - La plaquette de test

1) un interface HP-IB;

2) 5 canaux de 4 bits;

3) 4 signaux d'horloge;

4) une fréquence d'opération maximale de 50 MHz;

5) une mémoire de 1024 mots de 32 bits.

Malgré la flexibilité du générateur, il a fallu ajouter des circuits supplémentaires pour effectuer nos tests efficacement. Premièrement, les canaux d'horloge ne permettent pas de générer deux signaux d'horloge déphasés de 90 degrés sans recouvrement tels que requis par IMAGE-2, et deuxièmement, le générateur n'a que 20 signaux de sortie alors qu'IMAGE-2 compte 36 broches de commande.

Afin de générer suffisamment de signaux pour IMAGE-2, chaque canal du générateur génère les signaux pour deux broches en deux étapes. La première étape consiste à mémoriser un premier vecteur de 16 bits envoyé du générateur à l'aide de registres supplémentaires ajoutés sur les plaquettes de prototype. Dans un deuxième temps, 20 nouveaux bits sont envoyés directement aux broches du c.i. suivi d'un coup d'horloge à IMAGE-2 pour exécuter la commande. Le schéma-bloc de ce circuit est représenté à la figure 6.6.

L'analyseur de signaux est connecté directement aux huit sorties d'IMAGE-2 et est synchronisé avec l'horloge interne du générateur de signaux.

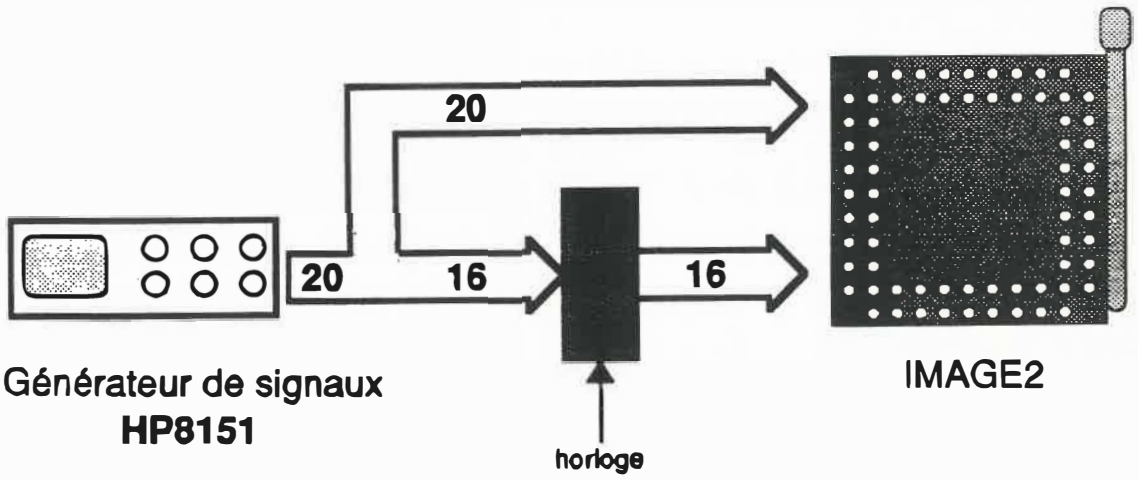


Figure 6.6 - Démultiplexage des signaux de contrôle sur la plaquette de test

6.4.3 Environnement logiciel

Afin d'utiliser efficacement les appareils de test, des logiciels ont été développés pour l'ordinateur SUN et pour le terminal HP9133. La génération de vecteurs de test suit le cheminement montré à la figure 6.7(a).

6.4.3.1 Le micro-assembleur DECO

Le micro-assembleur DECO (annexe B) permet de développer et de modifier rapidement des algorithmes de test. Chaque bit ou groupe de bits de commande d'IMAGE-2 est remplacé par un symbole mnémonique (annexe B.2) plus facile à utiliser. DECO prend comme intrant un fichier de texte qui doit suivre des règles de syntaxe simple et dont le nom se termine par '.COD'. Pour plus de détails concernant les mnémoniques et les règles de syntaxe, le lecteur est invité à consulter le Rapport Technique du Système Image [16]. L'extrant est un fichier '.BIT' qui contient des séquences de bits qui seront envoyés à IMAGE-2.

Le programme suivant est un exemple de fichier '.COD'.

```
# Programme qui génère 0,1,0,1,... à la sortie des processeurs
#  GC  4 août 1989
#
2  {  mxout g      gflu ffff      }
1  {                              }
2  {               gflu 0000      }
1  {  }
2  {               gflu ffff      }
1  {  }
2  {               gflu 0000      jz }
1  {                              jz }
```

Les lignes commençant par '#' sont des lignes de commentaires. Chaque ligne de code doit commencer par '1' ou '2' spécifiant la phase durant laquelle l'opération est exécutée puisqu'IMAGE-2 fonctionne sur deux phases et reçoit des commandes sur chacune de celles-ci. Par exemple le code d'opération du GFLU doit être envoyé sur la phase 2 de l'horloge, FFFF signifiant que la sortie du GFLU est 0, tandis que 0000 met la sortie à 1. Le symbole mnémonique 'mxout' nécessite un paramètre qui spécifie le signal qui sera envoyé à la sortie des processeurs, dans ce cas-ci, on veut le résultat du GFLU ('g'). Le symbole mnémonique 'jz' (jump zéro) est un code d'opération du micro-séquenceur Am2910, il signifie qu'on doit retourner à zéro et recommencer l'exécution du programme. Les symboles du micro-séquenceur doivent être répétés sur les deux

phases parce que celui-ci fonctionne à la demi de la fréquence d'IMAGE2. Le fichier *.BIT suivant représente les vecteurs envoyés au c.i. IMAGE-2.


```
00100010000101000101000000000010101100000000000000000000000000000000000000000000000000000000000111
00100010000101000101000000000010101100000000000000000000000000000000000000000000000000000000000111
00100010000010010001000000000010101100000000000000000000000000000000000000000000000000000000000111
00100010000000000000101000010001010110000000000000000000000000000000000000000000000000000000000111
00100010000000000000010000100010101000000000000000000000000000000000000000000000000000000000000111
00100010000000000000010000100010001000000000000000000000000000000000000000000000000000000000000111
00100010000000000000010000100010001000000000000000000000000000000000000000000000000000000000000111
00100010000000000000010000100010001000000000000000000000000000000000000000000000000000000000000111
01110111100010000000010000010000000000000000000000000000000000000000000000000000000000000000000111
00000000011001111111100000001000110000000000000000000000000000000000000000000000000000000000000111
01110111100011111111110000010000000000000000000000000000000000000000000000000000000000010000000000000
00000000011000000000000000010001100000000000000000000000000000000000000000000000000000010000000000000
01110111100011111111110000010000000000000000000000000000000000000000000000000000000000010000000000000
```


6.4.3.2 Le programme SUN2HP

Le programme SUN2HP (annexe C) est responsable de convertir le fichier '.BIT' en un format compatible au générateur et à la plaquette de test. D'une part il réarrange les bits afin qu'ils apparaissent correctement à la sortie du générateur de signaux pour les registres des plaquettes de prototype, et d'autre part il ajoute les bits nécessaires pour la génération des signaux d'horloges. L'intrant est un fichier '.BIT' tandis que l'extrant est envoyé au terminal HP9133 par un lien sériel.

6.4.3.3 Le programme de transfert HP2GEN

Le programme HP2GEN (annexe D), écrit en langage BASIC pour le terminal HP9133, fait l'interface entre le réseau des ordinateurs SUN et le générateur de vecteurs HP8151. Il lit les séquences de bits provenant du programme SUN2HP et les renvoie directement au générateur HP8151 par le lien HP-IB. Il démarre ensuite automatiquement la séquence de test.

6.4.4 Vérification du système IMAGE

Après la construction du système IMAGE, de nouveaux symboles mnémoniques ont été ajoutés au micro-assembleur DECO qui décrivent les nouveaux bits de contrôle des autres circuits du système telles que le générateur d'adresses, l'interface MVP et la micro-machine. Le Rapport Technique du Système Image [16] décrit ces nouvelles mnémoniques et leurs utilisations.

Puisque la machine IMAGE est directement reliée à un ordinateur SUN, on utilise le programme SUN2IMAGE (annexe E) pour transférer les fichiers '.BIT' générés par le micro-assembleur directement à la mémoire de micro-code. Une fois dans la mémoire de micro-code, le programme peut être immédiatement exécuté par la micro-machine. La séquence de développement de nouveaux algorithmes avec la machine IMAGE est représentée à la figure 6.7(b).

6.5 Performance et qualité des tests

Les tests effectués avec le générateur et l'analyseur de signaux permettent d'envoyer n'importe quelle séquence de bits au c.i. IMAGE-2, ils permettent ainsi de vérifier toutes les fonctions du c.i.. Considérant que le but de ces tests était de vérifier la validité du design au niveau logique, ces méthodes se sont avérées très efficaces puisqu'elles permettaient de vérifier de manière incrémentale chaque section du c.i.. Avec plus de ressources et de temps, la prochaine étape, maintenant que le design est complètement documenté, serait d'effectuer des tests paramétriques telles que la vitesse d'opération maximale, les formes d'ondes optimales des signaux d'horloges, les temps de propagations, etc.

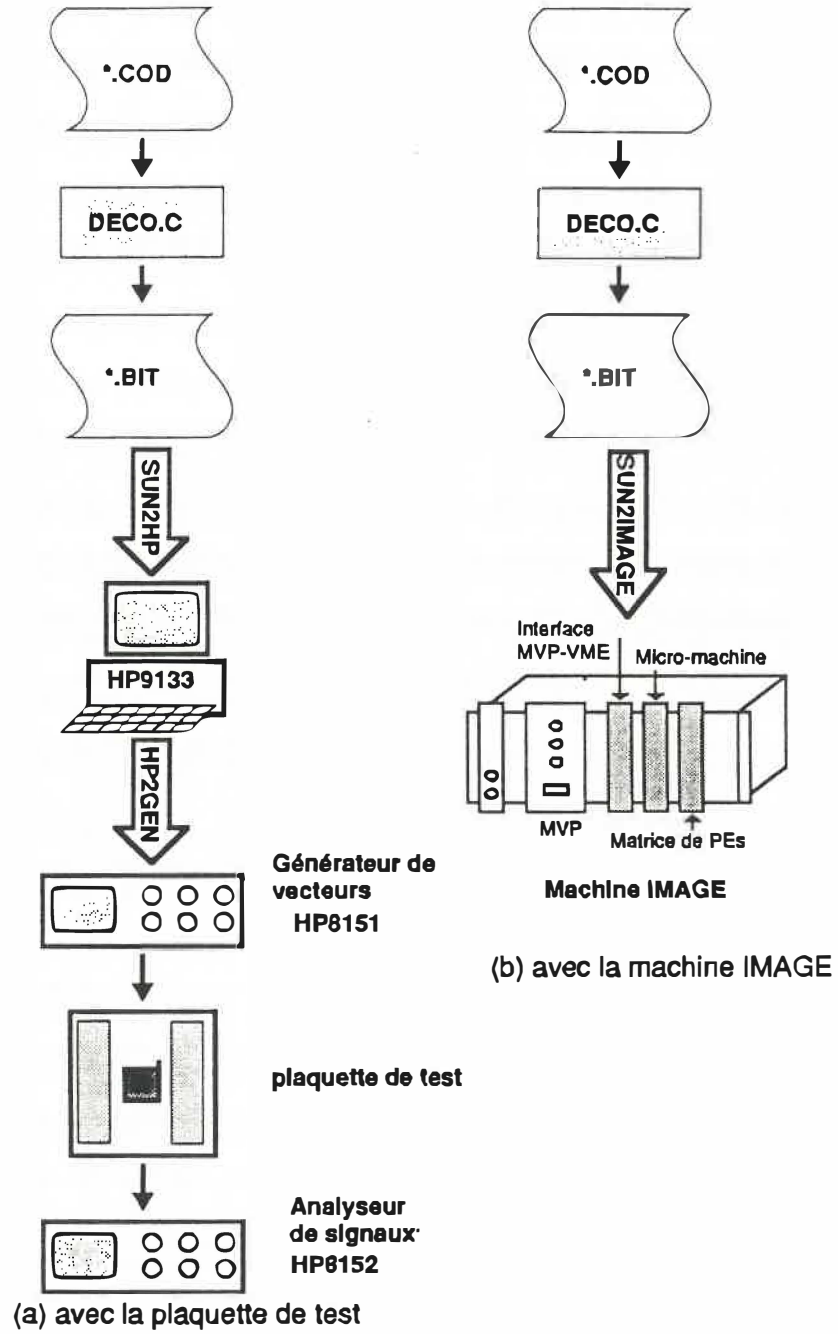


Figure 6.7 - Processus de vérification d'Image2

CONCLUSION

La machine IMAGE a démontré que l'utilisation de circuits ITGE peut offrir des performances temps réel dans des applications de traitement numérique d'images. Cependant, les performances de la machine pourraient être grandement améliorées en utilisant une technologie plus actuelle. Effectivement, IMAGE-2 compte environ 11 000 transistors alors que certains circuits en contiennent plus d'un million. Grâce à l'architecture régulière des processeurs IMAGE-2, la technologie actuellement disponible pour le laboratoire d'ITGE de l'Ecole Polytechnique permettrait d'incorporer quatre fois plus de processeurs par circuit intégré, i.e. 32. Une matrice plus grande ne demanderait que des modifications mineures aux autres circuits de la machine.

Les outils de conception et de vérification ont aussi évolués depuis le début du projet. L'éditeur de masque KIC a déjà été dépassé par deux générations (MAGIC et SDA), tandis que les appareils de test ont été remplacé par un seul outil (IMS) beaucoup plus performant. Cependant, les principes et méthodes utilisés restent valables indépendamment de la technologie.

Le choix l'ordinateur hôte SUN s'est montré judicieux puisqu'il permet de communiquer avec la machine IMAGE aisément grâce à l'interface UNIX au niveau logiciel et au bus VME, rapide et flexible, au niveau matériel. On peut aussi développer des outils et des logiciels plus facilement grâce à l'environnement de travail sophistiqué qu'on retrouve dans le système d'exploitation de l'ordinateur SUN.

La machine IMAGE, quoique très flexible est difficile à programmer. Le micro-assembleur permet de générer des séquences de micro-code à un très bas niveau, mais seulement si on connaît très bien l'architecture du c.i. IMAGE-2 et des autres circuits de la machine. La prochaine étape serait donc de développer un assembleur qui contiendrait les instructions de base les plus utilisées. Au niveau matériel, la machine contient présentement un seul c.i. IMAGE-2, mais l'implantation d'une matrice de 8 x 8 processeurs est en cours. Ceci devrait nous permettre d'implanter des algorithmes de TNI et d'évaluer les performances réelles de la machine et son efficacité.

Le champ d'application de la machine IMAGE peut être étendu à d'autres domaines demandant des calculs qui peuvent facilement être mis en parallèles dans un environnement IUDM. Parmi ces domaines, notons le traitement numérique de signaux, dont le TNI n'est en fait qu'un cas particulier.

Avec l'intérêt grandissant qu'on accorde aux architectures parallèles, des circuits performants sont maintenant disponibles sur le marché, c'est pourquoi il est maintenant plus utile de concentrer ses efforts dans l'architecture de la machine plutôt que dans l'architecture des processeurs. De plus, les ordinateurs parallèles s'orientent vers les architectures en pipelines telles que les Transputer de INMOS qui sont devenus des éléments de base d'architectures expérimentales.

BIBLIOGRAPHIE

[1] P.E. Danielsson and S. Levialdi, "Computer Architectures for Pictorial Information Systems," *IEEE Computer*, Nov. 1981, pp.53-67.

[2] D. Audet, G. Chouinard et C.Cyr, *Rapport technique du circuit intégré IMAGE-2 EPRT-88/01*, Ecole Polytechnique de Montréal, oct. 1988.

[3] R.M. Lougheed and C.W. Swonger, "An analysis of Computer Architectural Factors Contributing to Image Processor Capacity," *Proceedings of the SPIE Conference on Architectures and Algorithms for Digital Image Processing*, Vol. 596, 1985.

[4] D. Antonsson, B. Gudmundsson, T. Heldblom, B. Kruse, A. Linge, P. Lord, and T. Ohlsson, "PICAP - A System Approach to Image Processing," *IEEE Transactions on Computers*, Vol. C-31, No. 10, Oct. 1982, pp.997-1000.

[5] A.P. Reeves, "Parallel Computer Architectures for Image Processing," *Proceedings of the 1981 International Conference on Parallel Processing*, 1981, pp.199-206.

[6] K. Preston, M.J. Duff, S. Levialdi, P. E. Norgren and J.I. Toriwaki, "Basis of Cellular Logic with Some Applications in Medical Image Processing," *Proceedings of the IEEE*, vol. 67, no. 5, May 1979, pp. 826-856.

[7] M.J.E. Golay, "Hexagonal Parallel Pattern Transformation," *IEEE Transactions on Computers*, Vol. C-18, No. 8, Aug. 1969, pp.733-740.

[8] B. Kruse, "A Parallel Picture Processing Machine," *IEEE Transactions on Computers*, Vol. C-22, No.12, Dec. 1973, pp.1075-1087.

[9] R.B. Lougheed and D.L. McCubbrey, "The Cytocomputer: A Practical Pipelined Image Processor," *Conference of the 7th Annual Symposium on Computer Architecture*, 1980, pp. 271-277.

[10] M. Annaratone, E. Arnould, T. Gross, H.T. Kung, M. Lam, O. Menzilcioglu, and J.A. Webb, "The Warp Computer: Architecture, Implementation, and Performance," *IEEE Transactions on Computers*, Vol. C-36, No.12, Dec. 1987, pp. 1523-1538.

[11] H.J. Siegel, L.J. Siegel, F.C. Kemmerer, P.T. Mueller, H.E. Smalley, S.D. Smith, "PASM: A Partionable SIMD/MIMD System for Image Processing and Pattern Recognition," *IEEE Transactions on Computers*, Vol. C-30, No.12, Dec. 1981, pp. 934-945.

[12] M.J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, Vol. 54, Dec. 1966, pp.1901-1909.

[13] S.H. Unger, "A Computer Oriented Towards Spatial Problems," *Proceedings of the IRE*, Vol. 46, 1958, pp.1744-1750.

[14] M.J. Duff, "Review of the Clip Image Processing System," *AFIPS Conference Proceedings*, Vol. 47, 1978, pp. 1055-1060.

[15] K.E. Batcher, "Bit-Serial Parallel Processing Systems," *IEEE Transactions on Computers*, Vol. C-31, No. 5, May 1982, pp.377-384.

[16] K.E. Batcher, "The MPP Staging Memory," *International Conference on Parallel Processing*, 1984, pp.496-498.

[17] MVP-VME, *Reference Manual for the MVP-VME video digitizer*, 5468-MF-00, MATROX Electronic Systems.

[18] G. Chouinard, F. Bertaud et J.L. Houle, *Rapport technique du système IMAGE. EPM/RT-89-18*, Ecole Polytechnique de Montréal, Oct. 1989.

[19] *VMEbus Specifications Manual*, VMEbus manufacturers groups, rev B, Aug. 1982.

[20] L.W. Tucker and G.G. Robertson, "Architecture and Applications of the Connection Machine," *IEEE Computer*, aug. 1988, pp. 26-38.

[21] J. Kittler and M.J.B. Duff, *Image Processing Architectures*, Research Studies Press, England, 1985.

[22] C. Cyr, Y. Savaria, D. Audet, and J.L. Houle, "A Novel Self-Testing and Reconfiguration Scheme for Yield Improvement of two Dimensional Logic Arrays," *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers*, Oct. 1987, pp. 494-500.

[23] T. Ericsson and P.E. Danielsson, "LIPP - A SIMD Multiprocessor Architecture for Image Processing," *Proc. of the 10th Annual Int. Symp. on Computer Architecture*, 1983, pp.395-400.

[24] J.H. Siegel, "Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks," *IEEE Transactions on Computers*, Vol.C-26, No.2, Feb. 1977, pp.153-161.

[25] G. Chouinard, E.-A. Benaata, J.-L. Houle and Y. Savaria, "Performance Measurements on IMAGE-2, a Parallel Computer for Image Processing," *Eight Int. Symp. on Applied Informatics*, Feb. 1990.

[26] S. Levialdi, "Computer Architectures for Image Analysis," *International Conference on Pattern Recognition*, 1988, pp. 1148-1158.

[27] L.P. Cordella, M.J.B. Duff, and S. Levialdi, "An Analysis of Computational Cost in Image Processing: A Case Study," *IEEE Transactions on Computers*, Vol. C-27, No. 10, Oct. 1978, pp. 904-910.

[28] A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays," *IEEE Computer*, Jan. 1983, pp. 14-20.

[29] A. Rosenfeld and A.C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.

[30] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, New Jersey, 1975.

[31] M. Yasrebi, S. Deshpande and J.C. Browne, "A Comparison of Circuit Switching and Packet Switching Data Transfers Using Two Simple Image Processing Algorithms," *International Conference on Parallel Processing*, Aug. 1983, pp. 25-28.

[32] M. Yaserbi, J.C. Browne, D.P. Agrawal, "Analysis and Design of Parallel Algorithms and Implementations for Some Image Processing Operations," *International Conference on Parallel Processing*, 1987, pp. 901-908.

[33] N. Weste and K. Eshraghian, *Principle of CMOS VLSI Design*, Addison-Wesley, 1985.

[34] "*An Introduction to Digital Simulation*", Mentor Graphics Corp., Apr. 1989. 104 p.

[35] J.P. Hayes, "A Unified Switching Theory with Applications to VLSI Design," *Proceedings of the IEEE*, Vol. 70, No.10, Oct. 1982, pp.1140-1151.

Annexe A - Schémas des circuits de la machine IMAGE

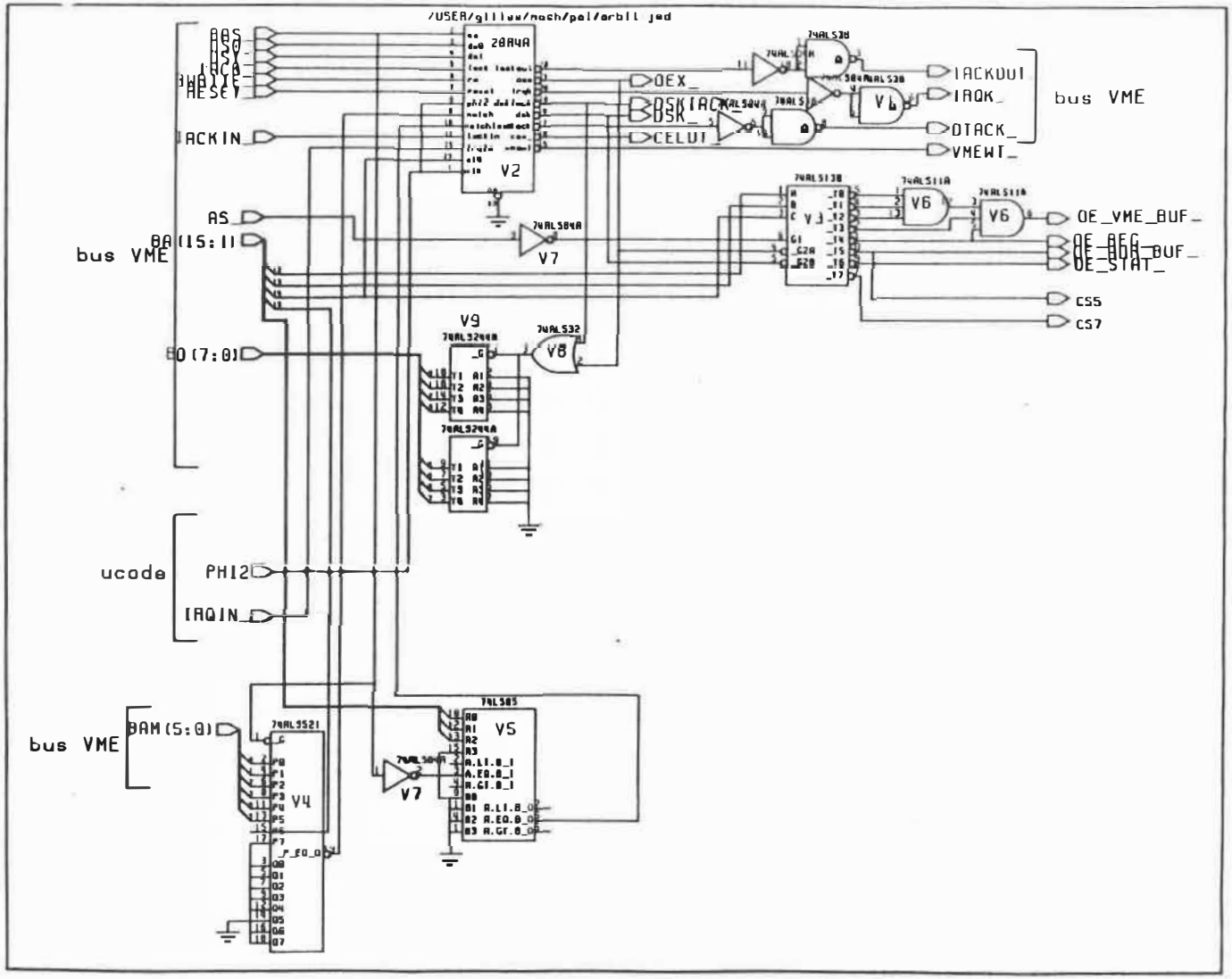


Figure A.1 - Schéma de l'interface du bus VME

/

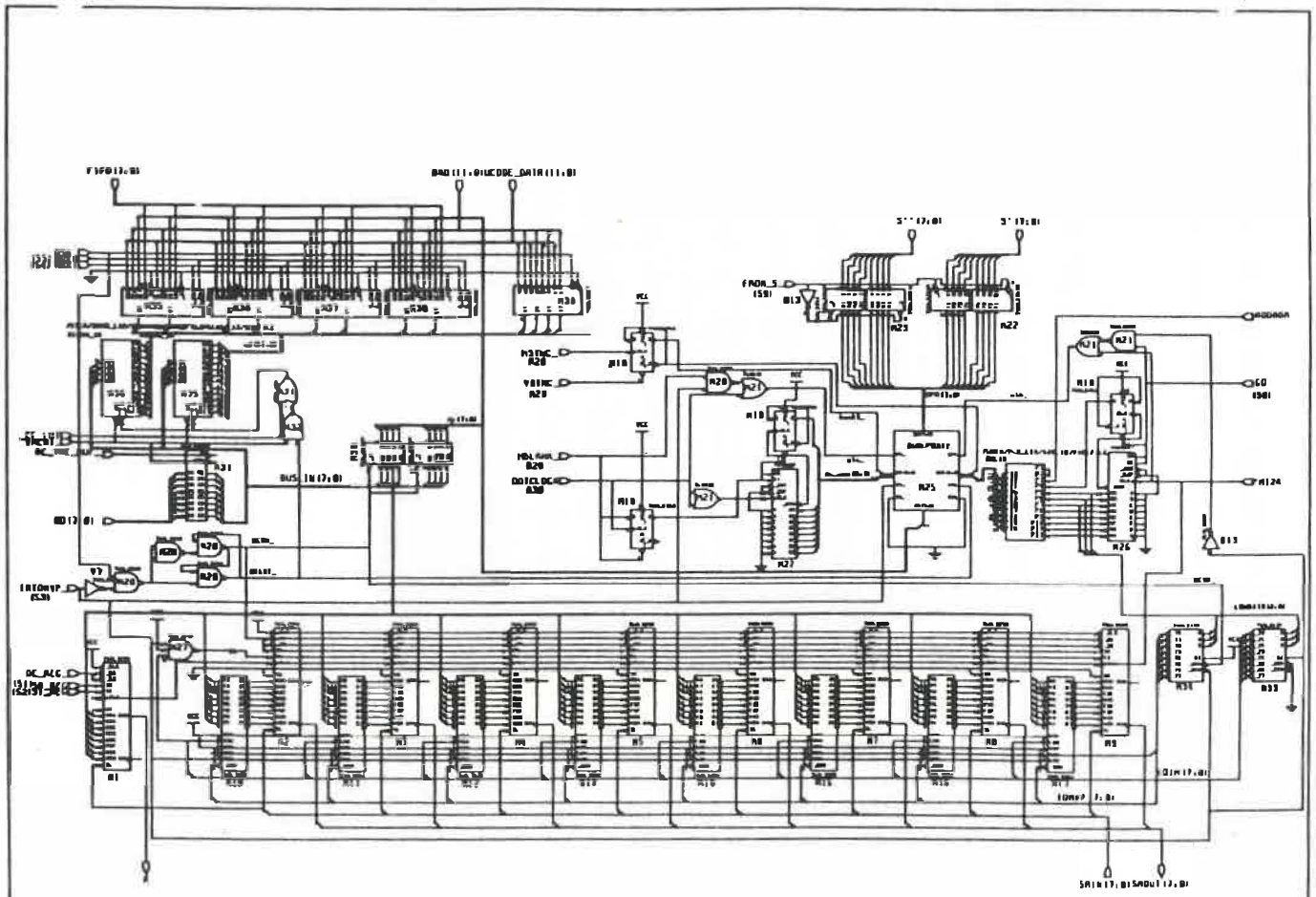


Figure A.2 - Schéma de l'interface du MVP

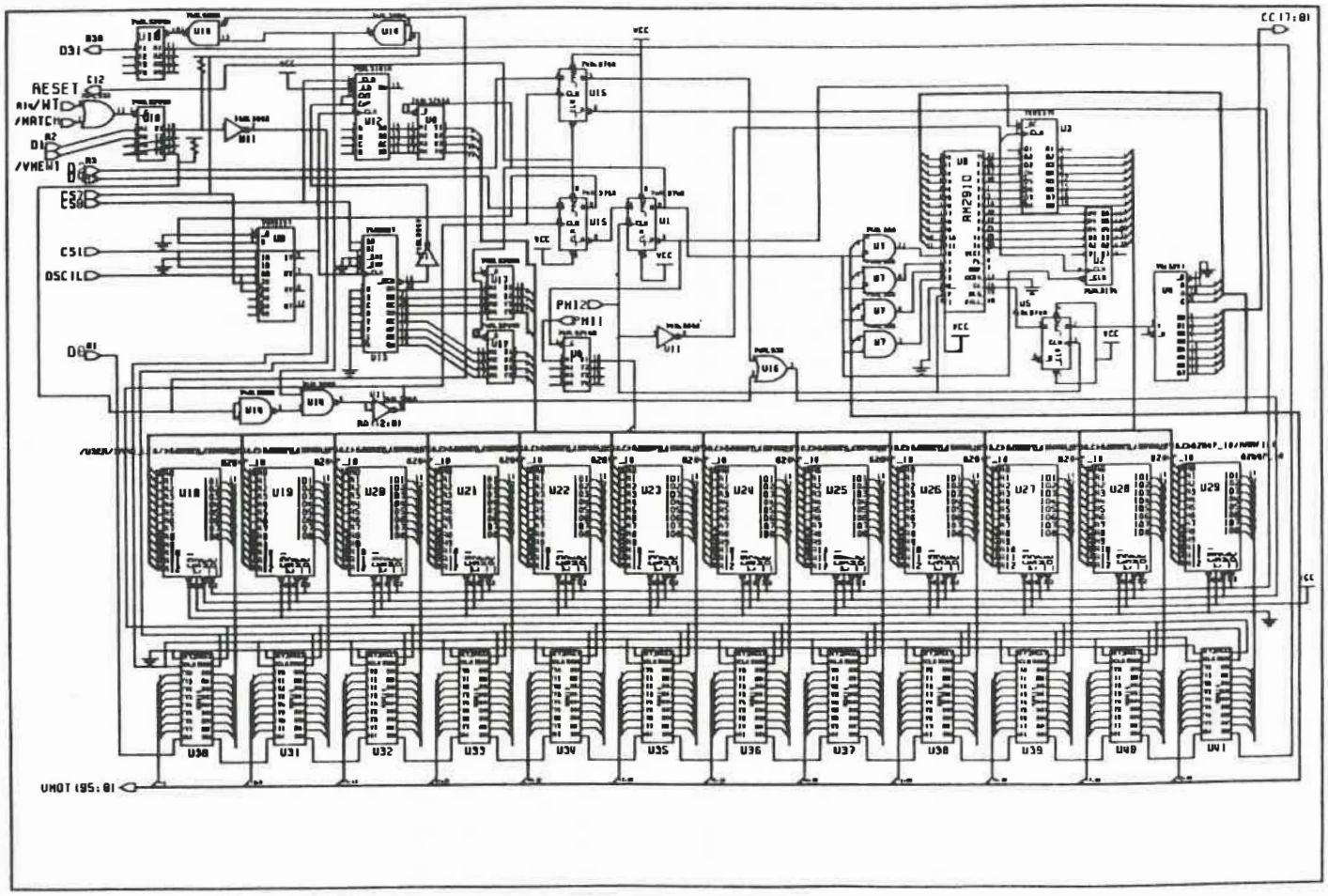


Figure A.3 - Schéma de la micro-machine

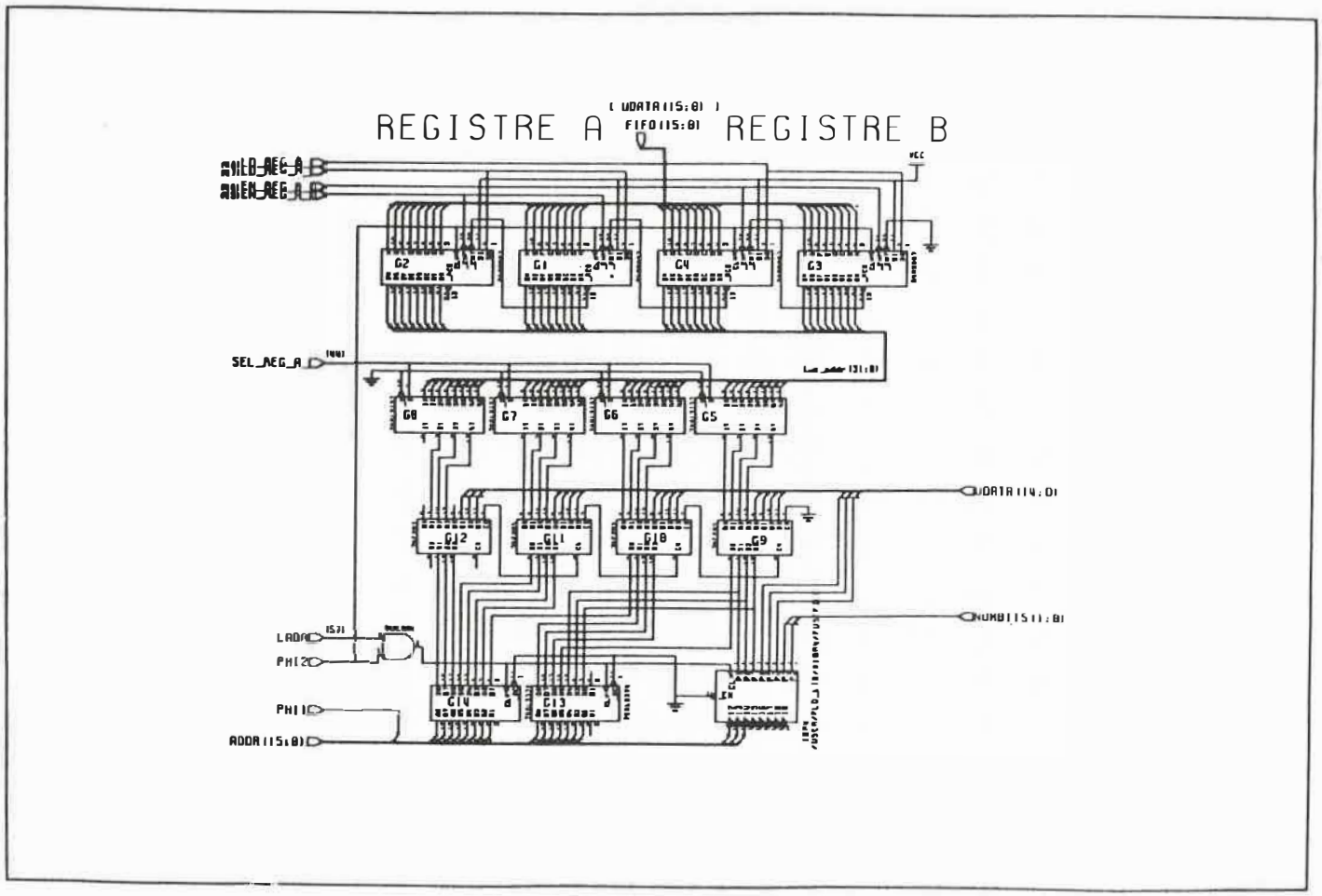


Figure A.4 - Schéma du générateur d'adresses

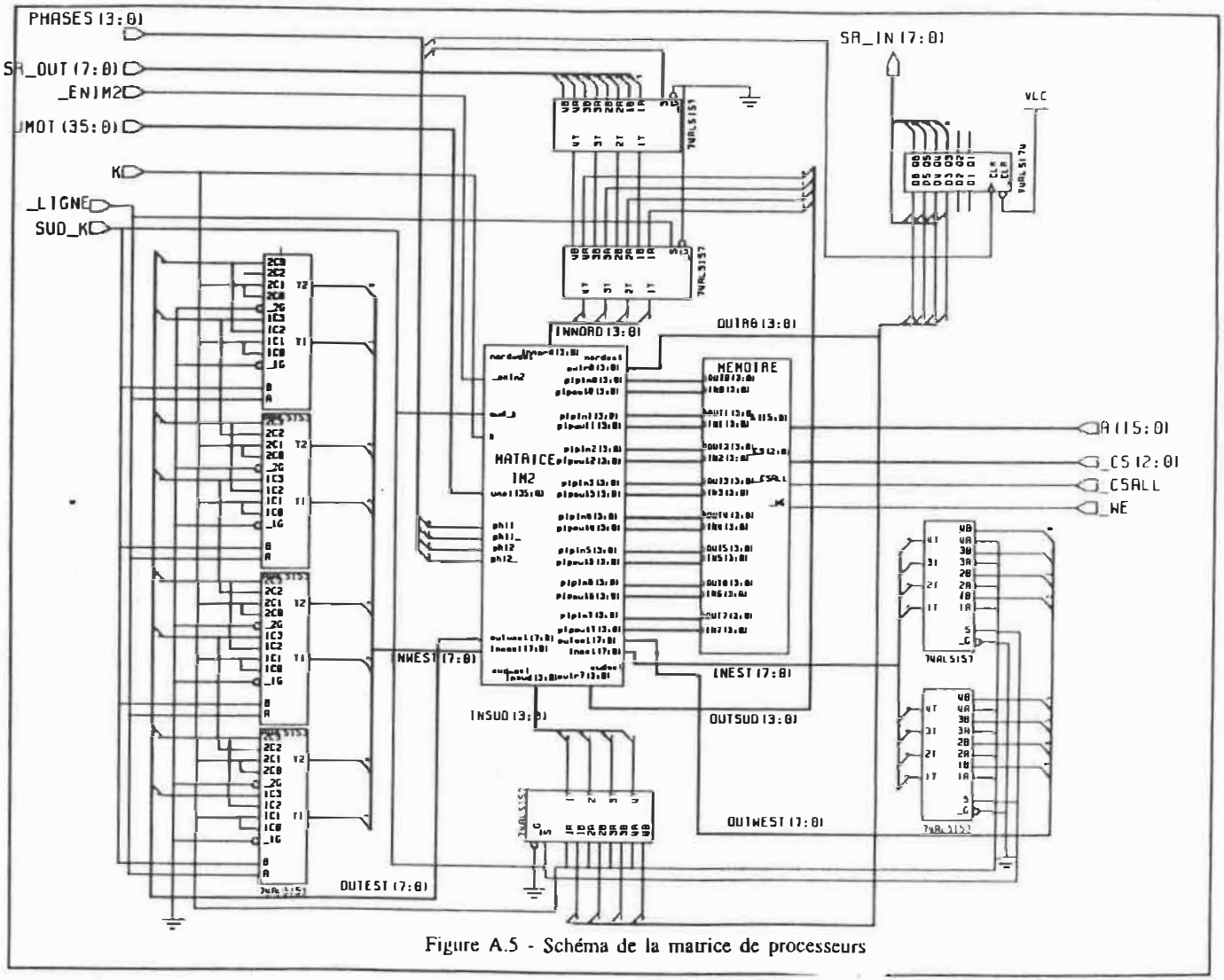


Figure A.5 - Schéma de la matrice de processeurs

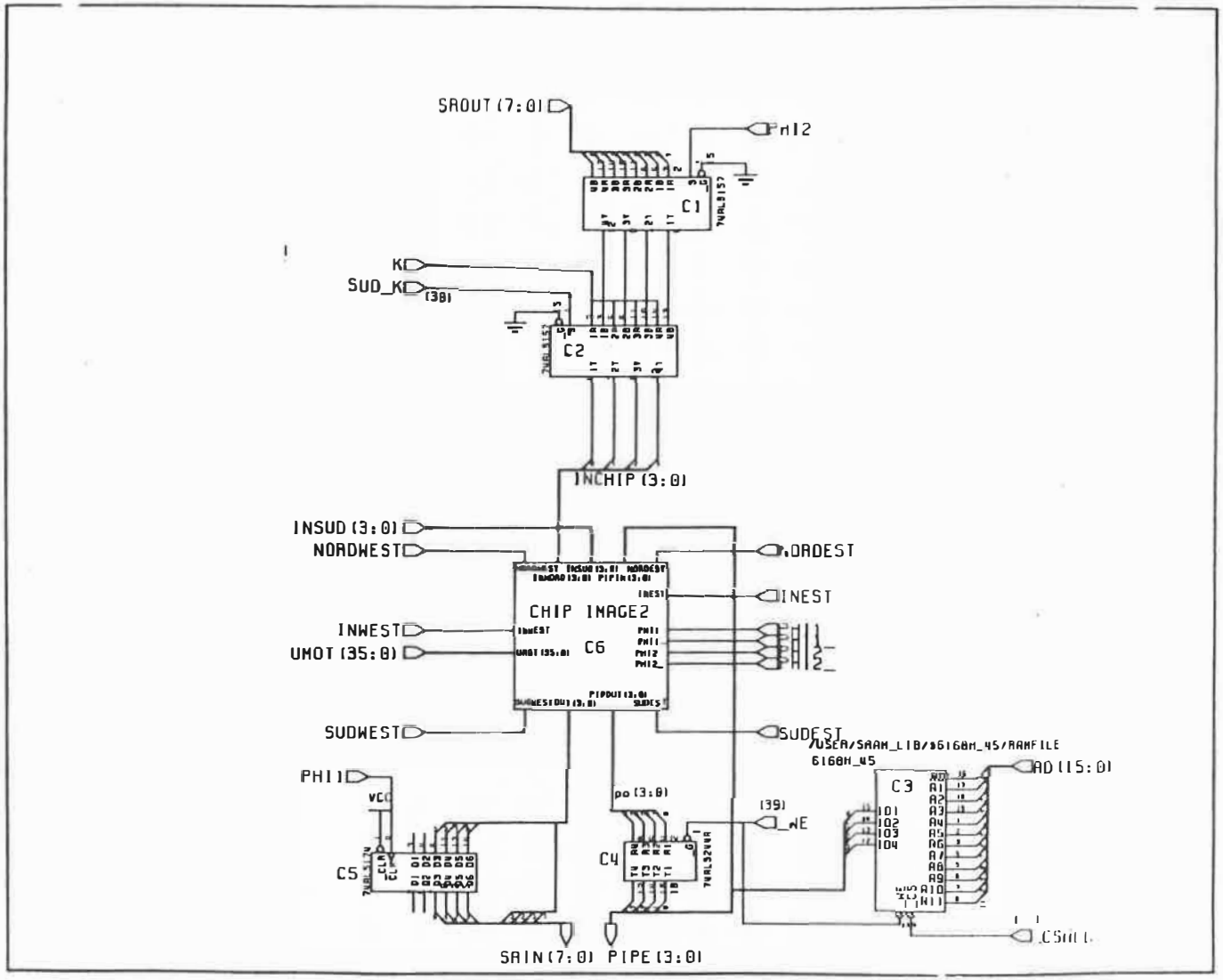


Figure A.6 - Schéma d'une cellule d'IMAGE-2

Annexe B.1 - Le micro-assembleur DECO

```
/*************************************************/
/*                                               */
/*                   DECO.C                      */
/*                                               */
/* Assembleur de code de Image2.                 */
/* Gilles Chouinard                              */
/* mai 89, F. Bertaud                            */
/* A partir d'un fichier de commandes ASCII, ce  */
/* programme genere la sequence de mot de commande */
/* qui est envoye au chip image a phi2 et phi1   */
/* successivement.                               */
/*                                               */
/* Le fichier d'entree doit etre un fichier *.cod .*/
/* Le fichier de sortie est un fichier *.bit.    */
/*                                               */
/*************************************************/

#include <stdio.h>

#define tolower(a) a
#define MEM 0
#define TEST 2
#define PIP 1
#define MXTEST 1
#define MXOUT 0
#define HIZ 2  /* en fait , l'output donne 0 puisque le noeud ne */
               /* se decharge pas.                               */

FILE *fopen(),*fp2,*fp3,*fp4;
char fin = 0,un,fincomment,inline[50],nom[80],i,j,nombre,*stg="phi";
char  nombre_de_mots,match,phi1[96],phi2[96],pipe,nopipe,phase,hiz,sel9;

char   mxa_mode=MEM,mxb_mode=MEM,mxd_mode=MEM,mxout_mode=MEM,mxtest_mode
= MXOUT;
char am2910,phi1_conf[4][96],phi2_conf[4][96],premier = 1;

/* J'initialise les mots initiaux a NOP . */

char phi1_init[96] = {0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,1,
                      1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1};


char phi2_init[96] = {0,1,1,1,0,1,1,1,1,0,0,0,1,0,0,0,0,0,
                      0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1};


unsigned long hexa,temp,hexa16;
```

```
char mnemo[70][12] =
 {
 "2ra",       /* 0-11 */
 "2wa",       /* 0-11 */
 "2rp",       /* 0-11 */
 "2wp",       /* 0-11 */
 "2load",     /* 0-7  */
 "2mset",     /* 0,1 */
 "2ldcond",   /* 0,1 */
 "2gflu",     /* 0x0000 - 0xFFFF */
 "2undefined 0",
 "2ldr8",     /* 0,1 */
/* 1-10 sur phi2 */
 "1add",      /* aucun parametre */
 "1sub",      /* aucun parametre */
 "1vois",     /* 0-7             */
 "2mxa",      /* 1,5,6,7,x  ; 1 = MXI      */
 "2mxb",      /* 3,4,5,6,7,x; 3 = MXIII    */
 "2mxd",      /* 4,5,6,7,p,x; p = PIPEIN   */
 "2mxout",    /* 3,4,5,6,x  ; 3 = MXIII    */
             /* c,8,?,g    ; pour le mode test */
 "2mxpipe",   /* 2,4 */
 "1mxr4",     /* g,v */
 "1mxr5",     /* g,v */
 "1mxr6",     /* g,v */  /* 20 */
 "1mxr7",     /* g,v */
 "2mx1",      /* 0,1,2,3 */
 "2mx2",      /* 0,1,2,3 --> mx3 = 0,2,1,3 */
 "2mxcg",     /* g,c sur phi2 */
 "1mxin",     /* g,p */
 "1mxr8",     /* m,8 */
 "2noload",   /* 0-7  sur phi2 */
 "2mxtest",   /* o,t,2 */

/* Generateur d'adresses et RAM. */

 "xirqin",    /* 0,1 */   /* inutilise. */
 "xmux0",     /* 0,1 */  /* 30 */
 "xmux1",     /* 0,1 */
 "ximtomvp",  /* 0,1 */
 "1s0_reg",   /* 0,1 */
 "1s1_reg",   /* 0,1 */
 "xen_im2",   /* 0,1 */
 "xligne",    /* 0,1 */
 "xsud_k",    /* 0,1 */
 "xcs",       /* 0,1,2 */  /* 38 */
 "xcsall",    /* 0,1 */
 "xwe",       /* 0,1 */   /* 40 */
 "1sel_reg_a", /* 0,1 */
 "1en_reg_a", /* 0,1 */
 "1en_reg_b", /* 0,1 */
 "1ld_reg_a", /* 0,1 */
 "1ld_reg_b", /* 0,1 */
 "xudata",     /* 0-7fff*/
 "1numbits",  /* 0,1,2 */  /* 47 */
```

```
 "xgo",         /* 0-1 */ /* doit commencer a phi1. */
 "xtomvp",        /* 1 = image to mvp; 0 = mvp to image */
 "l1adr",      /* 1 = latch addresse; 0 = stop latching addresse. */
/*50*/

 "xcc",         /* 0-7 */  /* 51 */
 "xadr",        /* 0-FFF; adresse des branch du 2910  */ /* 52 */

 /* Instruction du Am2910 */

 "xjz",       /* 53 */
 "xcjs",
 "xjmap",
 "xcjp",
 "xpush",
 "xjsrp",
 "xcjv",
 "xjrp",
 "xrfct",
 "xrpct",
 "xcrtn",
 "xcjpp",
 "xldct",
 "xloop",
 "xcont",
 "xtwb",        /* 68 */
 };

main(argc,argv)
char argc,*argv[];

{
if (argc != 2){ printf("ERREUR: le format est 'Deco <nom>.cod'\n");

                printf("        et ca cree un fichier <nom.bit>\n");

               }
else
{
fin = 0;
  strncpy(nom,argv[1],70);
  strcat(nom,".cod");
if ((fp2 = fopen(nom,"r")) == NULL)
  {
   fin = 1;
   printf("je ne peux ouvrir le fichier %s.cod\n",argv[1]);
  }
  else
  {

  strncpy(nom,argv[1],70);
  strcat(nom,".bit");

  if ((fp3 = fopen(nom,"w")) == NULL)
   {
```

```
      fin = 3;
      printf("Ne peux ouvrir la destination %s\n",nom)

    }
   } /* fin du else */

if ((fp4 = fopen("deco.init","r")) == NULL)
  {
   fin = 1;
   printf("je ne peux ouvrir le fichier deco.init.\n");
   exit(0);
  }

for (j=0;j<8;j++)
    for (i=0;i<97;i++) fprintf(fp3,"%c",getc(fp4));
fclose(fp4);


if (!fin)
{
for (i=0;i<96;i++)
 {
 phi1[i] = (phi1_init[i] == 1);
 phi2[i] = (phi2_init[i] == 1);
 }

while ( ((un = getc(fp2)) != EOF) && !fin)
  switch (un)
   {
    case ' ':break;
    case '\n':break;
    case '$':break;

    case '#':                /* commentaire */
      fincomment = 0;
      while ((!fincomment) && ((un = getc(fp2)) != EOF) )
        {
          if (un == '\n') fincomment = 1;
            else fincomment = 0;
        }
      if (!fincomment) fin = 1;
      break;
    case '1':   /* mot de commande sur phi1 */
      if (premier)
        { printf("Erreur : on DOIT commencer sur la phase 2 !!!\n");
          fin = 1;
        }
      premier = 0;
      if (fin = (phase == 1) )
         printf("Erreur : 2 phases 1 se suivent\n");
      phase = 1;
      getword();
      /* for (i=77;i<96;i++) phi1[i] = phi2[i];   copie les bits du
Am2910 */
      if (fin) printf("Erreur au mot #%d\n",nombre_de_mots);
```

```
        else
         {
            for (i=0;i<96;i++) fprintf(fp3,"%d",phi1[i]);
            fprintf(fp3,"\n");
         }
       break;
     case '2':  /* mot de commande sur phi2 */
        premier = 0;
        if (fin = (phase == 2) )
           printf("Erreur : 2 phases 2 se suivent\n");
       phase = 2;
       getword();
 /*      for (i=77;i<96;i++) phi2[i] = phi1[i];*/ /* copie les bits du
Am2910 */
       if (fin) printf("Erreur au mot #%d\n",nombre_de_mots);
        else
         {
            for (i=0;i<96;i++) fprintf(fp3,"%d",phi2[i]);
            fprintf(fp3,"\n");
         }
       break;

     default: printf("Erreur au mot %d : '#,1 ou 2' non trouves.\n",
                      nombre_de_mots);
              fin = 1;
              break;
   }
fin = 0;
}
if (fin != 1) fclose(fp2);
if (!fin)
 {
   nombre_de_mots++;
   if (phase == 2)
           {
           for (i=0;i<96;i++) fprintf(fp3,"%d",phi1[i]);
           fprintf(fp3,"\n");
           }
   if (phase == 1)
           {
      /*     printf("ERREUR: On doit terminer sur phi2 !!!\n"); */
           for (i=0;i<96;i++) fprintf(fp3,"%d",phi2[i]);
           fprintf(fp3,"\n");
           }
   fclose(fp3);
 }
}

}

/*********************************************************/
/* getword() parcours une ligne et compile les           */
/*           mnemonique une par une.                     */
/*********************************************************/
```

```
getword()
{
am2910 = 0;
while ((un=getc(fp2)) != '{')
  if (un==EOF) { printf("'{' non trouve. \n");
                  fin =1;
                }
nombre_de_mots++;
while ( (fscanf(fp2,"%s",inline) != EOF)
        && (inline[0] != '}') )
 {
  i = 0;
  while (inline[i] != '\0')
    {
     inline[i] = tolower(inline[i]);
     i++;
    }
  match = 0;
  while ( (match < 70) && strcmp(mnemo[match++]+1,inline) );;
  if ( (mnemo[--match][0]-48 != phase) && (mnemo[match][0] != 'x') )
    {
      printf("erreur:%s a phi%d\n",inline,phase);
      fin = 1;
    }
  switch (match)
   {
    /* ra  , les bits sont inverses, 0 <--> 1 , i.e. 6 <--> c */
     /* et phi1 et phi2 interchanges. */
    case 0: nombre = fscanf(fp2,"%d",&hexa);
                if ((nombre != 1) || (hexa > 16))
                                    {
                                         printf("parametre    de    'ra'
defectueux.\n");
                                     fin = 1;
                                    }
                else
                    {
                     phi2[0] = !((hexa & 1) != 0);
                     phi2[1] = ((hexa & 2) != 0);
                     phi2[2] = ((hexa & 4) != 0);
                     phi2[3] = ((hexa & 8) != 0);
                    }
                break;
    /* wa , les bits sont inverses, 0 <--> 1 , i.e. 6 <--> c */
     /* et phi1 et phi2 interchanges. */
            case 1: nombre = fscanf(fp2,"%d",&hexa);
                if ((nombre != 1) || (hexa > 16))
                                    {
                                         printf("parametre    de    'wa'
defectueux.\n");
                                     fin = 1;
                                    }
                else
                 {
                  temp = 1;
```

```
                for (i=0;i<4;i++)
                  {
                   phi1[i] = !((hexa & temp) != 0);
                   temp = temp<<1;
                  }
                }
               break;
    /* rp , les bits sont inverses, 0 <--> 1 , i.e. 6 <--> c */
     /* et phi1 et phi2 interchanges. */
    case 2: nombre = fscanf(fp2,"%d",&hexa);
               if ((nombre != 1) || (hexa > 16))
                                {
                                     printf("parametre    de    'rp'
defectueux.\n");
                                 fin = 1;
                                }
               else
                {
                    phi2[4] = !((hexa & 1) != 0);
                    phi2[5] = ((hexa & 2) != 0);
                    phi2[6] = ((hexa & 4) != 0);
                    phi2[7] = ((hexa & 8) != 0);
                }
               break;
    /* wp , les bits sont inverses, 0 <--> 1 , i.e. 6 <--> c */
     /* et phi1 et phi2 interchanges. */
         case 3: nombre = fscanf(fp2,"%d",&hexa);
               if ((nombre != 1) || (hexa > 16))
                                {

                                     printf("parametre    de    'wp'
defectueux.\n");
                                fin = 1;
                                }
               else
                {
                 temp = 1;
                 for (i=0;i<4;i++)
                   {
                    phi1[4+i] = !((hexa & temp) != 0);
                    temp = temp<<1;
                   }
                }
               break;
    /* load */
    case 4:nombre = fscanf(fp2,"%d",&hexa);
               if ((hexa > 7) || (nombre != 1))
                                {
                                    printf("parametre    de    'load'
defectueux.\n");
                                 fin = 1;
                                }
               else switch(hexa)
               {
               case 0 : phi1[9] = 0; break;
```

```
                case 1 : phi2[9] = 1; break;
                case 2 : phi1[10] = 0; break;
                case 3 : phi2[10] = 1; break;
                case 4 : phi2[11] = 1; break;
                case 5 : phi1[11] = 1; break;
                case 6 : phi2[12] = 0; break;
                case 7 : phi1[12] = 1; break;
                }
                break;
    /* mset */
    case 5 :nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mset'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == '1') phi1[8] = 0;
                     else phi1[8] = 1;
                break;
     /* ldcond */
    case 6 :nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                   printf("parametre    de    'ldcond'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == '1') phi2[8] = 0;
                     else phi2[8] = 1;
                break;
    /* gflu */
    case 7:nombre = fscanf(fp2,"%4x",&hexa);

                if ((hexa > 0xffff) || (nombre != 1))
                                {
                                    printf("parametre    de    'gflu'
defectueux.\n");
                                 fin = 1;
                                }
                else
                        {
                         phi1[13+3] = !((hexa & 0x0001)!=0);
                         phi1[13+7] = !((hexa & 0x0002)!=0);
                         phi1[13+1] = !((hexa & 0x0004)!=0);
                         phi1[13+5] = !((hexa & 0x0008)!=0);
                         phi2[13+3] =  ((hexa & 0x0010)!=0);
                         phi2[13+7] =  ((hexa & 0x0020)!=0);
                         phi2[13+1] =  ((hexa & 0x0040)!=0);
                         phi2[13+5] =  ((hexa & 0x0080)!=0);
                         phi1[13+2] = !((hexa & 0x0100)!=0);
                         phi1[13+6] = !((hexa & 0x0200)!=0);
                         phi1[13+0] = !((hexa & 0x0400)!=0);
```

```
                        phi1[13+4] = !((hexa & 0x0800)!=0);
                        phi2[13+2] =  ((hexa & 0x1000)!=0);
                        phi2[13+6] =  ((hexa & 0x2000)!=0);
                        phi2[13+0] =  ((hexa & 0x4000)!=0);
                        phi2[13+4] =  ((hexa & 0x8000)!=0);
                       }

                break;
     /* ldr8 */
    case 8 :
    case 9 :nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                    printf("parametre    de    'ldr8'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == '0') phi1[21] = 0;
                     else phi1[21] = 1;
                break;

    case 10:  phi2[21] = 0;  /* add */
                 break;

    case 11:  phi2[21] = 1;  /* sub */
                 break;
    /* vois */
    case 12:nombre = fscanf(fp2,"%d",&hexa);
                if ( (hexa > 7) || (nombre != 1) )
                                {
                                    printf("parametre    de    'vois'
defectueux.\n");
                                fin = 1;
                                }
                else
                 {
                  phi1[23] = ((hexa & 0x1) != 0);
                  phi1[22] = ((hexa & 0x2) != 0);
                  phi2[22] = ((hexa & 0x4) != 0);
                 }
                break;
    /* mxa ,phi1 et phi2 interchanges */
    case 13: nombre = fscanf(fp2,"%s",inline);
                 hexa = inline[0];
                 if (nombre != 1)
                       {
                        printf("parametre de 'mxa' defectueux.\n");
                        fin = 1;
                       }
                 else switch (tolower(hexa))
                 {
                      case '5' : phi2[24] = 1;
                                 phi1[24] = 1;
                                 phi2[29] = 0;
```

```
                                    mxa_mode = MEM;
                                    break;

                          case '6' : phi2[24] = 1;
                                    phi1[24] = 0;
                                    phi2[29] = 0;
                                    mxa_mode = MEM;
                                    break;

                          case '7' : phi2[24] = 0;
                                    phi1[24] = 0;
                                    phi2[29] = 0;
                                    mxa_mode = MEM;
                                    break;

                          case '1' : phi2[24] = 0;
                                    phi1[24] = 1;
                                    phi2[29] = 0;
                                    mxa_mode = MEM;
                                    break;

                          case 'x' : phi2[24] = 1;
                                    phi1[24] = 0;
                                    phi2[29] = 1;
                                    mxa_mode = HIZ;
                                    break;

                            default:   printf("parametre   de   'mxa'
defectueux.\n");
                                    fin = 1;
                                    break;

                          }
                     break;
    /* mxb , phi2 et phi1 interchanges */
    case 14: nombre = fscanf(fp2,"%s",inline);
                hexa = inline[0];
                if (nombre != 1)
                     {
                      printf("parametre de 'mxb' defectueux.\n");
                      fin = 1;
                     }
                else switch (tolower(hexa))
                     {
                     case '4' : phi2[25] = 0;
                                    phi1[25] = 1;
                                    phi2[29] = 0;
                                    mxb_mode = MEM;
                                    break;

                     case '5' : phi2[25] = 1;
                                    phi1[25] = 1;
                                    phi2[29] = 0;
                                    mxb_mode = MEM;
                                    break;
```

```
                        case '6' : phi2[25] = 1;
                                   phi1[25] = 0;
                                   phi2[29] = 0;
                                   mxb_mode = MEM;
                                   break;

                        case '7' : phi2[25] = 0;
                                   phi1[25] = 0;
                                   phi2[29] = 0;
                                   mxb_mode = MEM;
                                   break;

                        case '3' : phi2[25] = 0;
                                   phi1[25] = 0;
                                   phi2[29] = 1;
                                   mxb_mode = PIP;
                                   break;

                        case 'x' : phi2[25] = 1;
                                   phi1[25] = 0;
                                   phi2[29] = 1;
                                   mxb_mode = HIZ;
                                   break;

                          default:   printf("parametre   de   'mxb'
defectueux.\n");
                                   fin = 1;
                                   break;

                        }
                      break;
     /* mxd , phi2 et phi1 interchanges */
    case 15: nombre = fscanf(fp2,"%s",inline);
                hexa = inline[0];
                if (nombre != 1)
                     {
                      printf("parametre de 'mxd' defectueux.\n");
                      fin = 1;
                     }
                else switch (tolower(hexa))
                     {
                     case '4' : phi2[26] = 0;
                                phi1[26] = 1;
                                phi2[29] = 0;
                                mxd_mode = MEM;
                                break;
                     case '5' : phi2[26] = 1;
                                phi1[26] = 1;
                                phi2[29] = 0;
                                mxd_mode = MEM;
                                break;
                     case '6' : phi2[26] = 1;
                                phi1[26] = 0;
                                phi2[29] = 0;
```

```
                                    mxd_mode = MEM;
                                    break;
                            case '7' : phi2[26] = 0;
                                    phi1[26] = 0;
                                    phi2[29] = 0;
                                    mxd_mode = MEM;
                                    break;
                            case 'p' : phi2[26] = 1;
                                    phi1[26] = 1;
                                    phi2[29] = 1;
                                    mxd_mode = PIP;
                                    break;
                            case 'x' : phi2[26] = 1;
                                    phi1[26] = 0;
                                    phi2[29] = 1;
                                    mxd_mode = HIZ;
                                    break;
                              default:    printf("parametre    de    'mxd'
defectueux.\n");
                                    fin = 1;
                                    break;

                           }
                         break;
      /* mxout : 27-2-89, phi2 et phi1 interchanges, mxout maintenant */

                          /* valide sur phi2 */
     case 16: nombre = fscanf(fp2,"%s",inline);
                    hexa = inline[0];
                    if (nombre != 1)
                          {
                           printf("parametre de 'mxout' defectueux.\n");
                           fin = 1;
                          }
                    else switch (tolower(hexa))
                          {
                          case '4' : phi2[27] = 0;
                                    phi1[27] = 1;
                                    phi2[29] = 0;
                                    mxout_mode = MEM;
                                    break;
                          case '5' : phi2[27] = 1;
                                    phi1[27] = 1;
                                    phi2[29] = 0;
                                    mxout_mode = MEM;
                                    break;
                          case '6' : phi2[27] = 1;
                                    phi1[27] = 0;
                                    phi2[29] = 0;
                                    mxout_mode = MEM;
                                    break;
                          case '3' : phi2[27] = 0;
                                    phi1[27] = 0;
                                    phi2[29] = 0;
                                    mxout_mode = MEM;
```

```
                                    break;
                               case 'x' : phi2[27] = 1;
                                          phi1[27] = 1;
                                          phi2[29] = 1;
                                          mxout_mode = HIZ;
                                          break;
                               case '8' : phi2[27] = 0;
                                          phi1[27] = 1;
                                          phi2[29] = 0;
                                          mxout_mode = TEST;
                                          break;
                               case '?' : phi2[27] = 1;
                                          phi1[27] = 1;
                                          phi2[29] = 0;
                                          mxout_mode = TEST;
                                          break;
                               case 'g' : phi2[27] = 1;
                                          phi1[27] = 0;
                                          phi2[29] = 0;
                                          mxout_mode = TEST;
                                          break;
                               case 'c' : phi2[27] = 0;
                                          phi1[27] = 0;
                                          phi2[29] = 0;
                                          mxout_mode = TEST;
                                          break;
                                default:    printf("parametre    de    'mxout'
defectueux.\n");
                                        fin = 1;
                                        break;

                               }
                          break;

  /* mxpipe, phi1 et phi2 interchanges */
    case 17: nombre = fscanf(fp2,"%d",&hexa);
                    if (nombre != 1)
                          {
                           printf("parametre de 'mxpipe' defectueux.\n");

                           fin = 1;
                          }
                    else switch (hexa)
                          {
                          case 2   : phi1[29] = 1;
                                     break;
                          case 4   : phi1[29] = 0;
                                     break;
                           default:    printf("parametre    de    'mxpipe'
defectueux.\n");
                                   fin = 1;
                                   break;

                          }
                          break;
```

```
     /* mxr4 */
    case 18:nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != 'g') && (i != 'v')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mxr4'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == 'g') phi1[32] = 1;
                     else phi1[32] = 0;
                break;

    case 19:nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != 'g') && (i != 'v')) || (nombre != 1))
                                 {
                                    printf("parametre    de    'mxr5'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == 'g') phi2[32] = 0;
                     else phi2[32] = 1;
                break;

    case 20:nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != 'g') && (i != 'v')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mxr6'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == 'g') phi1[33] = 1;
                     else phi1[33] = 0;
                break;

    case 21:nombre = fscanf(fp2,"%s",inline);
                i = tolower(inline[0]);
                if (((i != 'g') && (i != 'v')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mxr7'
defectueux.\n");
                                fin = 1;
                                }
                else if (i == 'g') phi2[33] = 0;
                     else phi2[33] = 1;
                break;

    /* mx1, phi1 et phi2 interchanges */
    case 22: nombre = fscanf(fp2,"%d",&hexa);
                  if (nombre != 1)
                        {
                         printf("parametre de 'mx1' defectueux.\n");
                         fin = 1;
```

```
                    }
               else switch (hexa)
                    {
                    case 0:    phi2[31] = 0;
                               phi1[31] = 0;
                               break;
                    case 1:    phi2[31] = 1;
                               phi1[31] = 0;
                               break;
                    case 2:    phi2[31] = 0;
                               phi1[31] = 1;
                               break;
                    case 3:    phi2[31] = 1;
                               phi1[31] = 1;
                               break;
                      default:   printf("parametre    de    'mx1'
defectueux.\n");
                               fin = 1;
                               break;
                    }
                break;

    /* mx2, phi1 et phi2 interchanges */
    case 23: nombre = fscanf(fp2,"%d",&hexa);
               if (nombre != 1)
                    {
                     printf("parametre de 'mx2' defectueux.\n");
                     fin = 1;
                    }
               else switch (hexa)
                    {
                    case 0:    phi2[30] = 0;
                               phi1[30] = 0;
                               break;
                    case 1:    phi2[30] = 0;
                               phi1[30] = 1;
                               break;
                    case 2:    phi2[30] = 1;
                               phi1[30] = 0;
                               break;
                    case 3:    phi2[30] = 1;
                               phi1[30] = 1;
                               break;
                      default:   printf("parametre    de    'mx2'
defectueux.\n");
                               fin = 1;
                               break;
                    }
                break;

    case 24:nombre = fscanf(fp2,"%s",inline);
               i = tolower(inline[0]);
               if (((i != 'g') && (i != 'c')) || (nombre != 1))
                         {
```

```
                                    printf("parametre    de    'mxcg'
defectueux.\n");
                                fin = 1;
                                }
                    else if (i == 'c') phi2[34] = 0;
                       else phi2[34] = 1;
                    break;

 /* mxin, corrige 17 juillet 1989. */
    case 25:nombre = fscanf(fp2,"%s",inline);
                    i = tolower(inline[0]);
                    if (((i != 'g') && (i != 'p')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mxin'
defectueux.\n");
                                fin = 1;
                                }
                    else if (i == 'p') phi1[34] = 0;
                           else phi1[34] = 1;
                    break;

    case 26:nombre = fscanf(fp2,"%s",inline);
                    i = tolower(inline[0]);
                    if (((i != '8') && (i != 'm')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mxr8'
defectueux.\n");
                                fin = 1;
                                }
                    else switch (i)
                        {
                        case '8'  : phi1[27] = 0;
                                    phi2[27] = 1;
                                    phi1[29] = 1;
                                    sel9 = 1;
                                    break;
                        case 'm'  : sel9 = 0;
                                    break;

                          default:    printf("parametre    de    'mxr8'
defectueux.\n");
                                fin = 1;
                                break;

                        }
                     break;
     /* noload */
     case 27:nombre = fscanf(fp2,"%d",&hexa);
                if ((hexa > 7) || (nombre != 1))
                                {
                                   printf("parametre    de    'noload'
defectueux.\n");
                                 fin = 1;
                                }
                else switch(hexa)
```

```
                {
                case 0 : phi1[9] = 1; break;
                case 1 : phi2[9] = 0; break;
                case 2 : phi1[10] = 1; break;
                case 3 : phi2[10] = 0; break;
                case 4 : phi2[11] = 0; break;
                case 5 : phi1[11] = 0; break;
                case 6 : phi2[12] = 1; break;
                case 7 : phi1[12] = 0; break;
                }
                break;
    /* mxtest , phi1 et phi2 interchanges */
    case 28: nombre = fscanf(fp2,"%s",inline);
                hexa = inline[0];
                if (nombre != 1)
                      {
                       printf("parametre de 'mxtest' defectueux.\n");

                       fin = 1;
                      }
                else switch (tolower(hexa))
                      {
           /* ok */   case 'o' : phi2[28] = 1;
                                 phi1[28] = 1;
                                 mxtest_mode = TEST ;   /* TEST etait
MXOUT*/
                                 break;
          /* ok */    case '2' : phi2[28] = 0;
                                 phi1[28] = 1; /* etait 1 */
                                 mxtest_mode = TEST;
                                 break;
                      case 't' : phi2[28] = 1;
                                 phi1[28] = 0;
                                 mxtest_mode = TEST;
                                 break;
                      case 'x' : phi2[28] = 0;
                                 phi1[28] = 0; /* etait 1 */
                                 mxtest_mode = TEST;
                                 break;
                       default:    printf("parametre    de    'mxtest'
defectueux.\n");
                               fin = 1;
                               break;

                      }
                  break;


     /* irqin, sur phi1 ou phi2 */
     case 29: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                              {
                                 printf("parametre    de    'irqin'
defectueux.\n");
```

```
                                fin = 1;
                                }
                        else if (phase == 1)
                                phi1[56] = (i == '1');
                             else
                                phi2[56] = (i == '1');

                        break;

        /* mux0, sur phi1 ou phi2 */
        case 30: nombre = fscanf(fp2,"%s",inline);
                        i = inline[0];
                        if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mux0'
defectueux.\n");
                                fin = 1;
                                }
                        else if (phase == 1)
                                phi1[55] = (i == '1');
                             else
                                phi2[55] = (i == '1');

                        break;

        /* mux1, sur phi1 ou phi2 */
        case 31: nombre = fscanf(fp2,"%s",inline);
                        i = inline[0];
                        if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                    printf("parametre    de    'mux1'
defectueux.\n");
                                fin = 1;
                                }
                        else if (phase == 1)
                                phi1[54] = (i == '1');
                             else
                                phi2[54] = (i == '1');

                        break;

        /* imtomvp, sur phi1 ou phi2 */
        case 32: nombre = fscanf(fp2,"%s",inline);
                        i = inline[0];
                        if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                  printf("parametre    de    'imtomvp'
defectueux.\n");
                                fin = 1;
                                }
                        else if (phase == 1)
                                phi1[53] = (i == '1');
                             else
                                phi2[53] = (i == '1');
```

```
                    break;

     /* s0_reg , sur phi1 */
     case 33: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                   {
                                      printf("parametre    de    's0_reg'
defectueux.\n");
                                   fin = 1;
                                   }
                    else phi1[51] = (i == '1');

                    break;

     /* s1_reg , sur phi1 */
     case 34: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                   {
                                      printf("parametre    de    's1_reg'
defectueux.\n");
                                   fin = 1;
                                   }
                    else phi1[52] = (i == '1');

                    break;

     /* en_im2, sur phi1 ou phi2 */
     case 35: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                   {
                                      printf("parametre    de    'en_im2'
defectueux.\n");
                                   fin = 1;
                                   }
                    else if (phase == 1)
                                   phi1[36] = (i == '1');
                                else
                                   phi2[36] = (i == '1');

                    break;

     /* ligne, sur phi1 ou phi2 */
     case 36: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                   {
                                      printf("parametre    de    'ligne'
defectueux.\n");
                                   fin = 1;
                                   }
                    else if (phase == 1)
                                   phi1[37] = (i == '1');
```

```
                    else
                        phi2[37] = (i == '1');

                break;

    /* sud_k, sur phi1 ou phi2 */
    case 37: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                            {
                                printf("parametre    de    'sud_k'
defectueux.\n");
                            fin = 1;
                            }
                else if (phase == 1)
                        phi1[38] = (i == '1');
                     else
                        phi2[38] = (i == '1');

                break;

    /* cs , sur phi1 ou phi2 */
    case 38: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if  (nombre != 1)
                            {
                                 printf("parametre    de    'cs'
defectueux.\n");
                            fin = 1;
                            }
                else if (phase == 1)
                        switch (i)
                       {
                       case '0' : phi1[41] = 0;
                                  phi1[42] = 0;
                                  phi1[43] = 0;
                                  break;

                       case '1' : phi1[41] = 1;
                                  phi1[42] = 0;
                                  phi1[43] = 0;
                                  break;

                       case '2' : phi1[41] = 0;
                                  phi1[42] = 1;
                                  phi1[43] = 0;
                                  break;

                      case '3' : phi1[41] = 1;
                                  phi1[42] = 1;
                                  phi1[43] = 0;
                                  break;

                       case '4' : phi1[41] = 0;
                                  phi1[42] = 0;
```

```
                    phi1[43] = 1;
                break;

        case '5' : phi1[41] = 1;
                    phi1[42] = 0;
                    phi1[43] = 1;
                break;

        case '6' : phi1[41] = 0;
                    phi1[42] = 1;
                    phi1[43] = 1;
                break;
        case '7' : phi1[41] = 1;
                    phi1[42] = 1;
                    phi1[43] = 1;
                break;
         default:   printf("parametre   de   'cs'
defectueux.\n");
                    fin = 1;
                    break;
        }

         else
         switch (i)
        {
        case '0': phi2[41] = 0;
                    phi2[42] = 0;
                    phi2[43] = 0;
                break;

        case '1' : phi2[41] = 1;
                    phi2[42] = 0;
                    phi2[43] = 0;
                break;

        case '2' : phi2[41] = 0;
                    phi2[42] = 1;
                    phi2[43] = 0;
                break;

        case '3' : phi2[41] = 1;
                    phi2[42] = 1;
                    phi2[43] = 0;
                break;

        case '4' : phi2[41] = 0;
                    phi2[42] = 0;
                    phi2[43] = 1;
                break;

        case '5' : phi2[41] = 1;
                    phi2[42] = 0;
                    phi2[43] = 1;
                break;
```

```
                        case '6' : phi2[41] = 0;
                                   phi2[42] = 1;
                                   phi2[43] = 1;
                                   break;
                        case '7' : phi2[41] = 1;
                                   phi2[42] = 1;
                                   phi2[43] = 1;
                                   break;
                         default:   printf("parametre   de   'cs'
defectueux.\n");
                                   fin = 1;
                                   break;

                        }

                break;
     /* csall, sur phi1 ou phi2 */
     case 39: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                        {
                           printf("parametre   de   'csall'
defectueux.\n");
                        fin = 1;
                        }
                else if (phase == 1)
                        phi1[40] = (i == '1');
                     else
                        phi2[40] = (i == '1');

                break;

     /* we, sur phi1 ou phi2 */
     case 40: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                        {
                             printf("parametre   de   'we'
defectueux.\n");
                        fin = 1;
                        }
                else if (phase == 1)
                        phi1[39] = (i == '1');
                     else
                        phi2[39] = (i == '1');

                break;

     /* sel_reg_a, sur phi1 */
     case 41: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                        {
                         printf("parametre   de   'sel_reg_a'
defectueux.\n");
```

```
                                fin = 1;
                                }
                    else phi1[44] = (i == '1');

                    break;

     /* en_reg_a, sur phi1 */
     case 42: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                  printf("parametre    de    'en_reg_a'
defectueux.\n");
                                fin = 1;
                                }
                    else phi1[45] = (i == '1');

                    break;

     /* en_reg_b, sur phi1 */
     case 43: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                  printf("parametre    de    'en_reg_b'
defectueux.\n");
                                fin = 1;
                                }
                    else phi1[46] = (i == '1');

                    break;

     /* ld_reg_a, sur phi1 */
     case 44: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                  printf("parametre    de    'ld_reg_a'
defectueux.\n");
                                fin = 1;
                                }
                    else phi1[47] = (i == '1');

                    break;

     /* ld_reg_b, sur phi1 */
     case 45: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                  printf("parametre    de    'ld_reg_b'
defectueux.\n");
                                fin = 1;
                                }
                    else phi1[48] = (i == '1');
```

```
                    break;

    /* udata , sur phi1 ou phi2 */
    case 46:nombre = fscanf(fp2,"%4x",&hexa);

                if ((hexa > 0x7fff) || (nombre != 1))
                                {
                                   printf("parametre    de    'udata'
defectueux.\n");
                                 fin = 1;
                                }
                else if (phase == 1)
                               {
                                phi1[60] = ((hexa & 0x0001)!=0);
                                phi1[61] = ((hexa & 0x0002)!=0);
                                phi1[62] = ((hexa & 0x0004)!=0);
                                phi1[63] = ((hexa & 0x0008)!=0);
                                phi1[64] = ((hexa & 0x0010)!=0);
                                phi1[65] = ((hexa & 0x0020)!=0);
                                phi1[66] = ((hexa & 0x0040)!=0);
                                phi1[67] = ((hexa & 0x0080)!=0);
                                phi1[68] = ((hexa & 0x0100)!=0);
                                phi1[69] = ((hexa & 0x0200)!=0);
                                phi1[70] = ((hexa & 0x0400)!=0);
                                phi1[71] = ((hexa & 0x0800)!=0);
                                phi1[72] = ((hexa & 0x1000)!=0);
                                phi1[73] = ((hexa & 0x2000)!=0);
                                phi1[74] = ((hexa & 0x4000)!=0);
                                }
                           else
                               {
                                phi2[60] = ((hexa & 0x0001)!=0);
                                phi2[61] = ((hexa & 0x0002)!=0);
                                phi2[62] = ((hexa & 0x0004)!=0);
                                phi2[63] = ((hexa & 0x0008)!=0);
                                phi2[64] = ((hexa & 0x0010)!=0);
                                phi2[65] = ((hexa & 0x0020)!=0);
                                phi2[66] = ((hexa & 0x0040)!=0);
                                phi2[67] = ((hexa & 0x0080)!=0);
                                phi2[68] = ((hexa & 0x0100)!=0);
                                phi2[69] = ((hexa & 0x0200)!=0);
                                phi2[70] = ((hexa & 0x0400)!=0);
                                phi2[71] = ((hexa & 0x0800)!=0);
                                phi2[72] = ((hexa & 0x1000)!=0);
                                phi2[73] = ((hexa & 0x2000)!=0);
                                phi2[74] = ((hexa & 0x4000)!=0);
                                }
                break;

     /* numbits, sur phi1  */
    case 47: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (nombre != 1)
                                {
```

```
                                printf("parametre    de    'numbits'
defectueux.\n");
                                fin = 1;
                                }
                    else switch (i)
                         {
                         case '0' : phi1[49] = 0;
                                    phi1[50] = 0;
                                break;

                         case '1' : phi1[49] = 1;
                                    phi1[50] = 0;
                                break;

                         case '2' : phi1[49] = 0;
                                    phi1[50] = 1;
                                break;

                         case '3' : phi1[49] = 1;
                                    phi1[50] = 1;
                                break;

                          default:    printf("parametre    de    'numbits'
defectueux.\n");
                                    fin = 1;
                                    break;
                         }

                    break;
/* go */
     case 48: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                     printf("parametre    de    'go'
defectueux.\n");
                                fin = 1;
                                }
                    else if (phase == 1)
                           phi1[58] = (i == '1');
                         else
                           phi2[58] = (i == '1');

                    break;
/* tomvp */
     case 49: nombre = fscanf(fp2,"%s",inline);
                    i = inline[0];
                    if (((i != '0') && (i != '1')) || (nombre != 1))
                                {
                                   printf("parametre    de    'tomvp'
defectueux.\n");
                                fin = 1;
                                }
                    else if (phase == 1)
                           phi1[59] = (i == '1');
```

```
                    else
                       phi2[59] = (i == '1');

                break;
/* ladr */
     case 50: nombre = fscanf(fp2,"%s",inline);
                i = inline[0];
                if (((i != '0') && (i != '1')) || (nombre != 1))
                             {
                                 printf("parametre    de    'ladr'
defectueux.\n");
                             fin = 1;
                             }
                else phi1[57] = (i == '1');
                phi2[57] = 0;  /* doit etre a 0 a phi2. */
                break;


    /* cc */
    case 51: nombre = fscanf(fp2,"%d",&hexa);
             if ((nombre != 1) || (hexa > 7))
                             {
                                  printf("parametre    de    'cc'
defectueux.\n");
                              fin = 1;
                             }
             else
              if (phase == 1)
               {
                   phi1[77] = ((hexa & 1) != 0);
                   phi1[78] = ((hexa & 2) != 0);
                   phi1[79] = ((hexa & 4) != 0);
               }
               else
               {
                   phi2[77] = ((hexa & 1) != 0);
                   phi2[78] = ((hexa & 2) != 0);
                   phi2[79] = ((hexa & 4) != 0);
               }

             break;

    /* adr */
    case 52:nombre = fscanf(fp2,"%3x",&hexa);

             if ((hexa > 0xfff) || (nombre != 1))
                             {
                                  printf("parametre    de    'adr'
defectueux.\n");
                              fin = 1;
                             }
             else
               if (phase == 1)
                     {
                      phi1[80] = ((hexa & 0x0001)!=0);
```

```
                        phi1[81] = ((hexa & 0x0002)!=0);
                        phi1[82] = ((hexa & 0x0004)!=0);
                        phi1[83] = ((hexa & 0x0008)!=0);
                        phi1[84] = ((hexa & 0x0010)!=0);
                        phi1[85] = ((hexa & 0x0020)!=0);
                        phi1[86] = ((hexa & 0x0040)!=0);
                        phi1[87] = ((hexa & 0x0080)!=0);
                        phi1[88] = ((hexa & 0x0100)!=0);
                        phi1[89] = ((hexa & 0x0200)!=0);
                        phi1[90] = ((hexa & 0x0400)!=0);
                        phi1[91] = ((hexa & 0x0800)!=0);
                       }
                  else
                       {
                        phi2[80] = ((hexa & 0x0001)!=0);
                        phi2[81] = ((hexa & 0x0002)!=0);
                        phi2[82] = ((hexa & 0x0004)!=0);
                        phi2[83] = ((hexa & 0x0008)!=0);
                        phi2[84] = ((hexa & 0x0010)!=0);
                        phi2[85] = ((hexa & 0x0020)!=0);
                        phi2[86] = ((hexa & 0x0040)!=0);
                        phi2[87] = ((hexa & 0x0080)!=0);
                        phi2[88] = ((hexa & 0x0100)!=0);
                        phi2[89] = ((hexa & 0x0200)!=0);
                        phi2[90] = ((hexa & 0x0400)!=0);
                        phi2[91] = ((hexa & 0x0800)!=0);
                       }

                 break;
/* udata , sur phi1 ou phi2 */
default:  if (match > 52 && match < 69 && !am2910 )
                        {
                        hexa = match - 53;
                         if (phase == 1)
                        {
                        phi1[92] = ((hexa & 0x0001)!=0);
                        phi1[93] = ((hexa & 0x0002)!=0);
                        phi1[94] = ((hexa & 0x0004)!=0);
                        phi1[95] = ((hexa & 0x0008)!=0);
                        am2910 = 1;
                        break;
                        }
                         else
                        {
                        phi2[92] = ((hexa & 0x0001)!=0);
                        phi2[93] = ((hexa & 0x0002)!=0);
                        phi2[94] = ((hexa & 0x0004)!=0);
                        phi2[95] = ((hexa & 0x0008)!=0);
                        am2910 = 1;
                        break;
                        }
                        }
             else
             {
             printf("erreur a la commande --> %s <--\n",inline);
```

```
                fin = 1;
                break;
                }
    }
  }
/*
printf("mode de A B D OUT = %d %d %d %d\n",mxa_mode,mxb_mode,mxd_mode,
        mxout_mode);
*/

/* Finalement, on verifie d'autres conflits illegaux. */

if ( (!mxa_mode || !mxb_mode || !mxd_mode || !mxout_mode) &&
     (  (mxa_mode==PIP)  ||  (mxb_mode==PIP)  ||  (mxd_mode==PIP)  ||
(mxout_mode==PIP)) )
    { printf("Conflit entre le mode MEM et PIPE.\n"); fin = 1; }
if (sel9 && ( (mxa_mode != HIZ) || (mxb_mode != HIZ) || (mxd_mode !=
HIZ) ) )
    { printf("Conflit entre le mode R9 et MUXABD.\n"); fin=1; }
if ( (mxout_mode == TEST) && (mxtest_mode == MXOUT) )
    { printf("Le mxtest choisi mxout et mxout controle muxtest2.\n");
      fin = 1;
    }
if ( (mxout_mode == MEM) && (mxtest_mode == MXTEST) )
    {  printf("Le  mxtest  choisi  mxtest2  et  mxout  controle  les
registres.\n");
      fin = 1;
    }
}

/* Fin du programme. */
```

Annexe B.2 - La table de symboles mnémoniques

TABLE DES MNEMONIQUES

C.I. IMAGE

	mnémonique	param	description
2	ra	0-15	choisi le bit à lire des registres 4
2	wa	0-15	choisi le bit à écrire des registres 4-7
2	rp	0-15	choisi le bit à lire des registres 0-3
2	wp	0-15	choisi le bit à écrire des registres 0-3
2	load	0-7	active l'écriture du registre choisi.
2	noload	0-7	désactive l'écriture du registre choisi.
2	mset	0-1	1=reset le bit de condition, i.e load inconditionnellement 0=condition inchangée.
2	ldcond	0-1	1=set le bit de condition si la sortie du gflu = 1. 0=condition inchangée.
2	ldr8	0-1	1 = charge et décale la pile. 0 = pile inchangée.
2	gflu	0000-FFFF	code de la fonction du gflu.
1	add		demande à l'unité de retenue de générer une retenue d'addition.
1	sub		demande à l'unité de retenue de générer une retenue de soustraction.
1	vois	0-7	choisi l'un des 8 voisins. 1 = nord.
2	mxa	1,5,6,7	5,6,7 = sortie des registres 5,6,7 1 = sortie du mux1.
2	mxb	3,4,5,6,7	4,5,6,7 = sortie des registres 4,5,6,7 3 = sortie du mux3.
2	mxd	4,5,6,7,p	4,5,6,7 = sortie des registres 4,5,6,7 p = entrée pipeline du pe.
2	mxout	3,4,5,6	si le mxtest est en mode 'o' (out), alors la sortie du pe donne: 4,5,6 = sortie des registres 4,5,6 3 = sortie du mux3.
		c,8,?,g	si le muxtest est en mode 't' (test), alors la sortie du pe donne: c = carry. 8 = sortie du registre 8 (pile). ? = bit du registre conditionnel. g = sortie du gflu.
2	mxpipe	2,4	2 = sortie de mux2. 4 = sortie du registre 4.
2	mxr4	g,v	g = le mux d'entrée du registre 4 choisi la sortie du gflu. v = le mux d'entrée du registre 4 choisi le mux des voisins.
2	mxr5	g,v	g = le mux d'entrée du registre 5 choisi la sortie du gflu. v = le mux d'entrée du registre 5 choisi le mux des voisins.
2	mxr6	g,v	g = le mux d'entrée du registre 6 choisi la sortie du gflu.

			v = le mux d'entrée du registre 6 choisi le mux des voisins.
2	mxr7	g,v	g = le mux d'entrée du registre 7 choisi la sortie du gflu. v = le mux d'entrée du registre 7 choisi le mux des voisins.
2	mx1	0,1,2,3	le mux1 de sortie des registres de pipeline choisi le registre 0,1,2 ou 3.
2	mx2	0,1,2,3	le mux2 de sortie des registres de pipeline choisi le registre 0,1,2 ou 3. le mux3 est contrôlé par la même commande, i.e. que : si mux2 choisi r0, alors mux3 choisi r0, si mux2 choisi r1, alors mux3 choisi r2, si mux2 choisi r2, alors mux3 choisi r1, si mux2 choisi r3, alors mux3 choisi r3.
2	mxcg	c,g	c = le mux du carry choisi la sortie de l'unité de carry. g = le mux du carry choisi la sortie du gflu.
1	mxin	g,p	g = le mux d'entrée des registres de pipeline choisi la sortie du gflu. p = ou l'entrée pipeline du pe.
1	mxr8	m,8	m = l'entrée du gflu prend la sortie des muxs a,b,c,d. 8 = l'entrée du gflu prend les 4 bits du stack.
2	mxtest	o,t,2	le mux de sortie choisi: o = sortie de muxout. t = sortie du mux de test. 2 = sortie de mux2.

INTERFACES et GENERATEUR D'ADRESSES

	mnémonique	param	description
x	irqin	0-1	Non-implanté.
x	mux0	0-1	Contrôle la source de l'adresse de la LUT.
x	mux1	0-1	mux1 mux0 0 0 FIFO 0 1 Sortie du dual-port 1 0 Adresse du VME 1 1 udata du micro-code
x	imtomvp	0,1	0 = transfert IMAGE -> MVP.
1	s0_reg		Contrôle le registre K.
1	s1_reg		s1 s0 0 0 Maintient 0 1 Décalage à droite 1 0 Décalage à gauche 1 1 Chargement
x	ligne		Non-implanté. Active la configuration ligne de la matrice.
x	sud_k	0-1	0 = entrée NORD est branché sur le registre K. 1 = entrée NORD branché sur l'interface MVP.
x	cs	0-7	Choisi laquelle des huit lignes de processeurs aura sa mémoire locale activée. Utilisé lors du transfert d'images MVP -> IMAGE.
x	csall_	0-1	0 = active toutes les mémoires.
x	we	0-1	0 = Met les mémoires en mode écriture.
1	sel_reg_a_		0-1 0 = L'adresse est calculée avec le registre A. 1 = L'adresse est calculée avec le registre B.
1	en_reg_a	0-1	0 = Incrémente le registre A.
1	en_reg_b_		0-1 0 = Incrémente le registre B.
1	ld_reg_a_		0-1 0 = Charge le registre A avec udata(15:0).
1	ld_reg_b_		0-1 0 = Charge le registre B avec udata(15:0).
x	udata	0-7FFF	Charge le champ 'udata' du micro-mot.
1	numbits	0-3	Non-implanté
x	go	0-1	1 = Active le compteur d'adresse de l'interface MVP.
x	tomvp	0-1	Configure les buffers de S' et S'' correctement pour le transfert d'images: 1 = IM -> MVP.
1	ladr	0-1	Emmagasine dans le registre d'adresse la dernière adresse calculée.

MICRO-SEQUENCEUR

	mnémonique	param	description
x	adr	0000-7FFF	Adresse des instructions de branchement
x	cc	0-7	Choisi l'entrée du multiplexeur de condition pour les branchements conditionnels.
x	jz		Voir la description du Am2910 en annexe.
x	cjs		"
x	jmap		"
x	cjp		"
x	push		"
x	jsrp		"
x	cjv		"
x	jrp		"
x	rfct		"
x	rpct		"
x	crtn		"
x	cjpp		"
x	ldct		"
x	loop		"
x	cont		"
x	twb		"

Annexe C - Le programme SUN2HP

```
/*********************************************************
/*                                                       */
/*                          SUN2HP.C                     */
/*                                                       */
/* Sun2Hp.c envoie un fichier .bit cree par decode.c     */
/* au terminal hp dans le bon format pour le generateur  */
/* vecteurs.                                             */
/*                                                       */
/*  Gilles Chouinard 6-sept-88                           */
/*                                                       */
/* au gen : "x xxx 1234 5678 9abc defg hijk";            */
/* le 9eme bit du mot de commande de IMAGE2 va au 1er bit*/
/* dans le generateur i.e. au connecteur C1-0 de la      */
/* plaquette de test.                                    */
/* pareillement, le 28eme bit va a la position du        */
/* du generateur                                         */
/* et sera latche, donc out1 doit etre envoye avant out2.*/
/*                                                       */
/* Le terminal HP est branche sur le reseau et envoie une*/
/* commande                                              */
/* SUN2HP *.bit et lit la sortie de ce fichier.          */
/*                                                       */
/*********************************************************/
#include <stdio.h>

unsigned int out2[16] = { 9,10,32,33,11,12, 8,34, 0, 1, 2,
3,24,25,26,27};
u n s i g n e d i n t o u t 1 [ 1 6 ] =
{20,18,19,16,17,15,14,13,23,22,30,31,28,29,21,35};
u n s i g n e d i n t
pos[20]={6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24,26,27,28,29};
unsigned int num1 = 0;

char phi1,nombre,end_of_file,nom[100],i,line[100];
char gen1[40] = "0 001 xxxx xxxx xxxx xxxx 0000";
char gen2[40] = "0 xx0 xxxx xxxx xxxx xxxx 0000";
FILE *fp1;
main(argc,argv)
char argc,*argv[];
{
 if (argc != 2)
   {
     printf("Il faut un nom de fichier .bit sans extension.\n");
     exit(0);
   }
strncpy(nom,argv[1],70);
strcat(nom,".bit");       /* ajoute l'extension .bit au nom du fichier.
*/
if ((fp1 = fopen(nom,"r")) == NULL) {printf("Je ne peux ouvrir
%s\n",nom);
                                     exit(0);
                                    }
end_of_file = 0;
```

```
phi1 = 1;       /* on commence a phi1. */
while (!end_of_file)
{
 if ((nombre = fscanf(fp1,"%s",line)) == 1)
 {
 for (i=0;i<16;i++)
  {
   if ((gen1[pos[i]] = line[out1[i]]) == '1') num1++;
   if ((gen2[pos[i]] = line[out2[i]]) == '1') num1++;
  }
  num1++;
  if (phi1) /* on rajoute les phases au debut des vecteurs. */
    {
      gen2[2] = '1';
      gen2[3] = '0';
    }
  else
    {
      gen2[2] = '0';
      gen2[3] = '1';
    }
  num1++;
   printf("%s\n",gen1);   /* on envoie le vecteur memorise en premier.
*/
   printf("%s\n",gen2);

 }
 else end_of_file = 1;
 phi1 = !phi1;
}
printf("B\n");
printf("Nombre de 1 = %d\n",num1);
}

/* fin du programme. */
```

Annexe D - Le programme HP2GEN

```
1     ! PROGRAMME POUR TRANSFERER LES BITS DU VAX AU GENERATEUR.
2     ! GILLES CHOUINARD 13 SEPTEMBRE 1988
3     !
4     !              VAXTOGEN
6     !
10    INTEGER Ab
20    DIM Bufin$[10000] BUFFER,A(10,40)
30    COM A$(1000)[50],In$[100],Inp$[100],Init$(20)[40]
40    Init$(1)="FOR 0 001 0000 1001 0010 0000 1000"
50    Init$(2)="FOR 0 100 0000 1001 0010 0000 1000"
60    Init$(3)="FOR 0 001 0000 0100 0010 0000 1000"
70    Init$(4)="FOR 0 010 0000 0100 0010 0000 1000"
80    Init$(5)="FOR 0 001 0010 0000 0010 0010 1000"
90    Init$(6)="FOR 0 100 0010 0000 0010 0010 1000"
100   Init$(7)="FOR 0 001 0010 0000 0010 0010 0000"
110   Init$(8)="FOR 0 010 0000 0000 0010 0010 0000"
120   Init$(9)="FOR 0 001 0000 0000 0010 0010 0000"
130   Init$(10)="FOR 0 100 0000 0000 0010 0010 0000"
140   Init$(11)="FOR 0 001 0000 0000 0010 0010 0000"
150   Init$(12)="FOR 0 010 0000 0000 0010 0010 0000"
160   Init$(13)="FOR 0 001 0000 0000 0010 0010 0000"
170   Init$(14)="FOR 0 100 0000 0000 0010 0010 0000"
180   Init$(15)="FOR 0 001 0000 0000 0010 0010 0000"
190   Init$(16)="FOR 0 010 0000 0000 0010 0010 0000"
191   Inp$=""
200   Debut=0
210   ASSIGN @Ser TO 9
220   RESET @Ser
230   ASSIGN @Buf TO BUFFER Bufin$;FORMAT ON
240   CONTROL 9,0;0
250   CONTROL 9,3;9600
260   CONTROL 9,4;3
270   Oldinp$=Inp$
280   INPUT "Fichier .bit >>",Inp$
290   IF Inp$="" THEN
300      Inp$=Oldinp$
310   END IF
320   Inp$=LWC$(Inp$)
330   In$="send2hp "&Inp$
331   Num1=0
340   PRINT
350   PRINT "DEBUT DE LA RECEPTION DE ";Inp$&".bit"
360   OUTPUT @Ser;In$
370   TRANSFER @Ser TO @Buf;CONT
380   L=0
390   Fin=1
400   WHILE Fin=1
410     ENTER @Buf;A$(L)
420     IF A$(L)[1;1]="B" THEN
430       Fin=0
440     ELSE
450       L=L+1
460       PRINT "*";
470     END IF
480   END WHILE
490   ABORTIO @Ser
500   FOR K=1 TO L-1
510    FOR I=1 TO 30
520     IF A$(K)[I;1]<>"0" THEN
```

```
521         Num1=Num1+1
530         A$(K)[I;1]="1"
540       END IF
550      NEXT I
560     A$(K)[2;1]=" "
570     A$(K)[6;1]=" "
580     A$(K)[11;1]=" "
590     A$(K)[16;1]=" "
600     A$(K)[21;1]=" "
610     A$(K)[26;1]=" "
611     IF Num1<6 THEN
612     Num1=0
614     END IF
615     Num1=Num1-6
620     NEXT K
630     RESET @Buf
631     Num1=Num1-24
632     PRINT
633     PRINT "NOMBRE DE 1 = ";Num1
640     PRINT
650 PRINT CHR$(13);"FIN DE LA RECEPTION : ";L-2;" vecteurs";CHR$(7)
660   !
670   ! ON ENVOIE MAINTENANT LES VECTEURS AU GENERATEUR
680   !
690   IF Debut=0 THEN
700   OUTPUT 713;"TSA 0"
710   FOR K=1 TO 16
720     OUTPUT 713;Init$(K)
730   NEXT K
740   Debut=1
750   END IF
760   OUTPUT 713;"TSA 16"
770   A$(2)[28;1]="1"
780   FOR K=2 TO L-1
790     OUTPUT 713;"FOR"&A$(K)
800   NEXT K
810   OUTPUT 713;"FAD 16 LAD ";VAL$(16+L-3)
820   SEND 7;CMD 1
830   PRINT "FIN DE LA TRANSMISSION AU GENERATEUR";CHR$(7);CHR$(7)
840   BEEP 800,.5
850   GOTO 280
860    END
```

Annexe E - Le programme SUN2IMAGE

```
/*********************************************************/
/*                                                       */
/*                      SUN2IMAGE.C                      */
/*                                                       */
/* Programme de chargement du code de commande provenant */
/* des fichiers *.bit et allant dans la RAM de microcode */
/* via les registres de diagnostiques de la micro-machine*/
/* Les bits sont envoyes seriellement sur un data du bus */
/* VME. D autres bits data sont utilises pour commander  */
/* mode des registres.                                   */
/*                                                       */
/* Gilles Chouinard Bertaud F. --19-05-89 --             */
/*                                                       */
/* commentaires :                                        */
/* L adresse qui doit etre selectionnee est              */
/* celle qui correspond au cs7 du decodeur 138           */
/* de la carte interface vme (differ. de 1 adr. de la lut*/
/*                                                       */
/*********************************************************/
#include <stdio.h>

#include "../lib/imginstall.c"

extern struct IMG_MEMORY_MAP *g_cur_board;

        /* verifier que IMG_MEMORY_MAP englobe cette */

        /* adresse */


#define CHGSE_0  0

#define CHGSE_1  0

#define LATCH_818  0

#define TRANSTORAM   0

#define ad_registre  0

FILE *fopen(),*fp1;
char fin = 0, nom[70] ;

main(argc, argv)
char argc,*argv[];
{
long *ptr;
int nbr_bits;
char un ;

if (argc != 2)
   {
   printf("ERREUR: le format est 'SUN2IMAGE <nom>.BIT'\n");
```

```
   }
else
{
strncpy(nom,argv[1],70);
strcat(nom,"cod");
if (fin =((fp1 = fopen(nom,"r")) == NULL))
   printf("je ne peut pas ouvrir le fichier %s.bit\n",argv[1]);

if (!fin)
{
ptr = &(img_install("/dev/img", 0)->mem[ad_registre]));
nbr_bits = 0 ;
while ( ((un = getc(fp1)) != EOF) && !fin)
   switch (un)
   {
   case ' '  : break; /* on s occupe juste du cas des ' ' en debut
*/

                     /*  de  fichier  au  cas  improbable  ou  il  y  en
aurait*/


   case '\n' : if (fin = (nbr_bits != 96))
                  printf("ERREUR:la ligne ne contient pas 96 bits!\n");

               else
                  {
                   nbr_bits = 0;
                   *ptr = LATCH_818;
                   *ptr = TRANSTORAM;
                  }
               break;


   case '0'  :          /* traitement des 96 bits a venir formant le
code*/

                     /* d une phase
*/

              nbr_bits+1 ;
              *ptr = CHGSE_0;
              break;

   case '1'  : nbr_bits+1 ;
               *ptr = CHGSE_1;
                break;

   }
}
}
img_uninstall();
```

```
}
/* fin programme */
```

Annexe F - Le programme SUN2HP

```
/*****************************************************************/
/*                                                               */
/*                            SUN2HP.C                           */
/*                                                               */
/* Sun2Hp.c envoie un fichier .bit cree par decode.c             */
/* au terminal hp dans le bon format pour le generateur de vecteurs. */
/*                                                               */
/*  Gilles Chouinard 6-sept-88                                   */
/*                                                               */
/* au gen : "x xxx 1234 5678 9abc defg hijk";                    */
/* le 9eme bit du mot de commande de IMAGE2 va au 1er bit dans    */
/* le generateur i.e. au connecteur C1-0 de la plaquette de test. */
/* pareillement, le 28eme bit va a la position d du generateur    */
/* et sera latche, donc out1 doit etre envoye avant out2.        */
/*                                                               */
/* Le terminal HP est branche sur le reseau et envoie une commande */
/* SUN2HP *.bit et lit la sortie de ce fichier.                  */
/*                                                               */
/*****************************************************************/

#include <stdio.h>

unsigned int out2[16] = { 9,10,32,33,11,12, 8,34, 0, 1, 2, 3,24,25,26,27};
unsigned int out1[16] = {20,18,19,16,17,15,14,13,23,22,30,31,28,29,21,35};
unsigned int pos[20]={6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24,26,27,28,29};
unsigned int num1 = 0;

char phi1,nombre,end_of_file,nom[100],i,line[100];
char gen1[40] = "0 001 xxxx xxxx xxxx xxxx 0000";
char gen2[40] = "0 xx0 xxxx xxxx xxxx xxxx 0000";
FILE *fp1;
main(argc,argv)
char argc,*argv[];
{
 if (argc != 2)
   {
     printf("Il faut un nom de fichier .bit sans extension.\n");
     exit(0);
   }
strncpy(nom,argv[1],70);
strcat(nom,".bit");      /* ajoute l'extension .bit au nom du fichier. */
if ((fp1 = fopen(nom,"r")) == NULL) {printf("Je ne peux ouvrir %s\n",nom);
                                     exit(0);
                                    }
end_of_file = 0;
phi1 = 1;       /* on commence a phi1. */
while (!end_of_file)
{
 if ((nombre = fscanf(fp1,"%s",line)) == 1)
 {
 for (i=0;i<16;i++)
  {
   if ((gen1[pos[i]] = line[out1[i]]) == '1') num1++;
```

```
    if ((gen2[pos[i]] = line[out2[i]]) == '1') num1++;
   }
   num1++;
   if (phi1) /* on rajoute les phases au debut des vecteurs. */
     {
       gen2[2] = '1';
       gen2[3] = '0';
     }
   else
     {
       gen2[2] = '0';
       gen2[3] = '1';
     }
   num1++;
    printf("%s\n",gen1);  /* on envoie le vecteur memorise en premier. */
    printf("%s\n",gen2);

 }
 else end_of_file = 1;
 phi1 = !phi1;
}
printf("B\n");
printf("Nombre de 1 = %d\n",num1);
}

/* fin du programme. */
```

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
3 9334 00290911 5