Conception d'architectures intégrées de traitement d'image de bas niveau

Ahmed Boubekeur

THESE

présentée par

Ahmed BOUBEKEUR

pour obtenir le grade de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(Arrêté ministériel du 23 novembre 1988)

(Spécialité : Micro-électronique)

CONCEPTION D'ARCHITECTURES INTEGRES DE TRAITEMENT D'IMAGE DE BAS NIVEAU

Date de soutenance : le 4 Mars 1992

Composition du Jury :

Madame	Gabrièle SAUCIER	
Messieurs	Jacques MOSSIERE	Président
	Manfred GLESNER	Rapporteur
	Alain GUYOT	Rapporteur
	Jacques TRILHE	

Thèse préparée au sein du Laboratoire Conception de Systèmes Intégrés

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

46, AVENUE FELIX VIALLET 38031 GRENOBLE CEDEX

TEL : 76.57.45.00

PRESIDENT DE L'INSTITUT
MONSIEUR G.LESPINARD

ANNEE 1990-1991

PROFESSEURS DES UNIVERSITES

ENSERG	BARIBAUD	MICHEL
ENSIEG	BARRAUD	ALAIN
ENSPG	BAUDELET	BERNARD
UFR PGP	BAUDIN	GERARD
ENSIEG ILL	BEAUFILS	JEAN-PIERRE
ENSERG	BLIMAN	SAMUEL
ENSHMG	BOIS	PHILIPPE
ENSPG	BONNET	GUY
ENSEEG	BONNETAIN	LUCIEN
ENSIEG	BRISSONNEAU	PIERRE
CUEFA	BRUNET	YVES
ENSHMG	CAILLERIE	DENIS
ENSPG	CAVAIGNAC	JEAN-FRANCOIS
ENSPG	CHARTIER	GERMAIN
ENSERG	CHENEVIER	PIERRE
UFR PGP	CHERADAME	HERVE
ENSIEG	CHERUY	ARLETTE
ENSERG	CHOVET	ALAIN
ENSHMG	COGNET	GERARD
ENSEEG	COLINET	CATHERINE
ENSIEG	CORNUT	BRUNO
ENSIEG	COULOMB	JEAN-LOUIS
ENSIMAG	CROWLEY	JAMES
ENSEEG	DALARD	FRANCIS
ENSHMG	DARVE	FELIX
ENSIMAG	DELLA DORA	JEAN
ENSERG	DEPEY	MAURICE
ENSPG	DEPORTES	JACQUES
ENSEEG	DEROO	DANIEL
ENSEEG	DESRE	PIERRE
ENSEEG	DIARD	JEAN-PAUL
ENSERG	DOLMAZON	JEAN-MARC
ENSEEG	DURAND	FRANCIS
ENSPG	DURAND	JEAN-LOUIS
ENSHMG	FAUTRELLE	YVES
ENSIEG	FOGGIA	ALBERT
ENSIMAG	FONLUPT	JEAN
ENSIEG	FOULARD	CLAUDE
ENSEEG	GALERIE	ALAIN

UFR PGP	GANDINI	ALESSANDRO
ENSPG	GAUBERT	CLAUDE
ENSERG	GENTIL	PIERRE
ENSIEG	GENTIL	SYLVIANE
ENSERG	GUERIN	BERNARD
ENSEEG	GUYOT	PIERRE
ENSIEG	IVANES	MARCEL
ENSERG	JANOT	MARIE-THERESE
ENSIEG	JAUSSAUD	PIERRE
ENSPG	JOST	REMY
ENSPG	JOUBERT	JEAN-CLAUDE
ENSIEG	JOURDAIN	GENVIEVE
UFR PGP	LACHENAL	DOMINIQUE
ENISEG	LACOUME	JEAN-LOUIS
ENSIEG	LADET	PIERRE
ENSIEG	LIENARD	JOEL
ENSHMG	LESIEUR	MARCEL
ENSHMG	LESPINARD	GEORGES
ENSPG	LONGUEQUEUE	JEAN-PIERRE
ENSHMG	LORET	BENJAMIN
ENSEEG	LOUCHET	FRANCOIS
ENSEEG	LUCAZEAU	GUY
ENSIMAG	LUX	AUGUSTIN
ENSIEG	MASSE	PHILIPPE
ENSIEG	MASSELOT	CHRISTIAN
ENSIMAG	MAZARE	GUY
ENSIMAG	MOHR	ROGER
ENSHMG	MOREAU	RENE
ENSIEG	MORET	ROGER
ENSIMAG	MOSSIERE	JACQUES
ENSHMG	OBLED	CHARLES
ENSERG	PANANAKAKIS	GEORGES
ENSEEG	PAULEAU	YVES
ENSIEG	PERRET	ROBERT
ENSHMG	PIAU	JEAN-MICHEL
ENSERG	PIC	ETIENNE
ENSIMAG	PLATEAU	BRIGITTE
ENSERG	POUPOT	CHRISTIAN
ENSEEG	RAMEAU	JEAN-JACQUES
ENSPG	REINISCH	RAYMOND
UFR PGP	RENAUD	MAURICE
ENSIMAG	ROBERT	FRANCOIS
ENSIEG	ROYE	DANIEL
ENSIEG	SABONNADIERE	JEAN-CLAUDE
ENSERG	SAGUET	PIERRE
ENSIMAG	SAUCIER	GABRIELE
ENSPG	SCHLENKER	CLAIRE
ENSPG	SCHLENKER	MICHEL
UFR PGP	SILVY	JACQUES
ENSHMG	SIRIEYS	PIERRE
ENSEEG	SOHM	JEAN-CLAUDE
ENSIMAG	SOLER	JEAN-LOUIS
ENSEEG	SOUQUET	JEAN-LOUIS
ENSHMG	TICHKIEWITCH	SERGE
ENSHMG	TROMPETTE	PHILIPPE
ENSIMAG	VERJUS	JEAN-PIERRE
ENSPG	VINCENT	HENRI
ENSERG	ZADWORNY	FRANCOIS

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

ENSPG	BLOCH	DANIEL.....	RECTEUR	21.12.1993
ENSIMAG	LATOMBE	J.CLAUDE.....	DETACHEMENT	01.05.1993
ENSHMG	PIERRARD	J.MARIE.....	DETACHEMENT	01.05.1991
ENSIMAG	VEILLON	GERARD.....	DISPONIBLE	01.10.1993

SURNOMBRE

ENSHMG	BOUVARD	MAURICE	30.09.1990

PERSONNES AYANT OBTENU LE DIPLOME D'HABILITATION A DIRIGER DES RECHERCHES

BALESTRA	FRANCIS
BALME	LOUIS
BECKER	MONIQUE
BIGEON	JEAN
BINDER	ZDENECK
BOE	LOUIS-JEAN
BOUVIER	GERARD
CHASSERY	JEAN-MARC
CHOLLET	JEAN-PIERRE
COEY	JEAN-PIERRE
COMMAULT	CHRISTIAN
CORNUEJOLS	GERARD
COURNIL	MICHEL
DALLERY	YVES
DESCOTES-GENON	BERNARD
DUGARD	LUC
DURAND	MADELEINE
FERRIEUX	JEAN-PAUL
GAUTHIER	JEAN-PAUL
GHIBAUDO	GERARD
HAMAR	SYLVIANE
HAMAR	ROGER
HORAUD	PATRICE
KUENY	JEAN-LOUIS
LATOMBE	CLAUDINE
LE HUY	HOANG
LE GORREC	BERNARD
LOZANO-LEAL	ROGELIO
MAHEY	PHILIPPE
MEUNIER	GERARD
MICHEL	GERARD
MONMUSSON-PICQ	GEORGETTE
MULLER	JEAN
MULLER	JEAN-MICHEL
NGUYEN TRONG	BERNADETTE
NIEZ	JEAN-JACQUES
PASTUREL	ALAIN
PERRIER	PASCAL
PLA	FERNAND
RECHENMANN	FRANCOIS
ROGNON	JEAN-PIERRE
ROUGER	JEAN
TCHUENT	MAURICE
TRYSTRAM	DENIS

DIRECTEURS DE RECHERCHE DU CNRS

ALEMANY	ANTOINE
ALLIBERT	COLETTE
ALLIBERT	MICHEL
ANSARA	IBRAHIM
ARMAND	MICHEL
AUDIER	MARC
BERNARD	CLAUDE
BINDER	GILBERT
BONNET	ROLAND
BORNARD	GUY
CAILLET	MARCEL
CARRE	RENE
CHATILLON	CHRISTIAN
CLERMONT	JEAN-ROBERT
COURTOIS	BERNARD
CRISTOLOVEANU	SORIN
DAVID	RENE
DION	JEAN-MICHEL
DRIOLE	JEAN
DURAND	ROBERT
EUSCUDIER	PIERRE
EUSTATHOPOULOS	NICOLAS
FINON	DOMINIQUE
FRUCHARD	ROBERT
GARNIER	MARCEL
GIROD	JACQUES
GLANGEAUD	FRANCOIS
GUELIN	PIERRE
HOPFINGER	EMIL
JORRAND	PHILIPPE
JOUD	JEAN-CHARLES
KAMARINOS	GEORGES
KLEITZ	MICHEL
KOFMAN	WALTER
LANDAU	IOAN
LEJEUNE	GERARD
LEPROVOST	CHRISTIAN
MADAR	ROLAND
MARTIN	JEAN-MARIE
MERMET	JEAN
MICHEL	JEAN-MARIE
NAYROLLES	BERNARD
PASTUREL	ALAIN
PEUZIN	JEAN-CLAUDE
PHAM	ANTOINE
PIAU	MONIQUE
RENOUARD	DOMINIQUE
SENATEUR	JEAN-PIERRE
SIFAKIS	JOSEPH
SIMON	JEAN-PAUL
SUERY	MICHEL
TEODOSIU	CHRISTIAN
VACHAUD	GEORGES
VAUCLIN	MICHEL
WACK	BERNARD
YAVARI	ALI-REZA
YONNET	JEAN-PAUL

UNIVERSITE JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. MEDECINE
GRENOBLE 1

Président de l'Université :

M. NEMOZ Alain

ANNEE UNIVERSITAIRE 1990 - 1991

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ERE CLASSE

ADIBA Michel	Informatique
ANTOINE Pierre	Géologie I.R.I.G.M.
ARVIEU Robert	Physique Nucléaire I.S.N.
AURIAULT Jean Louis	Mécanique
BARRA Jean René	Statistiques - Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean Paul	Mathématiques Pures
BILLET Jean	Géographie
BLANCHI Jean Pierre	A.P.S.
BOEHLER Jean Paul	Mécanique
BOITET Christian	Informatique et Mathématiques Appliquées
BORNAREL Jean	
BRUANDET Jean François	Physique
CARLIER Georges	Biologie Végétale
CASTAING Bernard	Physique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEMAILLY Jean Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FINKE Gerde	Informatique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GERMAIN Jean Pierre	Mécanique
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences nucléaires
GILLARD Roland	Mathématiques
GUITTON Jacques	Chimie

../ ...

HERAULT Jeanny	Physique
HICTER Pierre	Chimie
JANIN Bernard	Géographie
JOLY Jean René	Mathématiques Pures
JOSELEAU Jean Paul	Biochimie
KAHANE André	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées
LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LOISEAUX Jean Marie	Sciences Nucléaires I.S.N.
LONGEQUEUE Nicole	Physique
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
NEMOZ Alain	Physique
OMONT Alain	Astrophysique
PELMONT Jean	Biochimie
PERRIER Guy	Géophysique
PIERRE Jean Louis	Chimie Organique
RENARD Michel	Thermodynamique
RICHARD Jean Marc	
RIEDTMANN Christine	Mathématiques
RINAUDO Marguerite	Chimie C.E.R.M.A.V.
ROBERT Jean Bernard	
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALLADE Marcel	Physique
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie
VIDAL Micheal	

... / ..

PROFESSEURS DE 2EME CLASSE

APPARU Marcel	Chimie
ARMAND Gilbert	Géographie
ARNAUD Hubert	Géologie
ARTRU Marie Christine	Physique
ATTANE Pierre	Mécanique
BARATE Robert	Sciences Nucléaires
BARET Paul	Chimie
BARGE Jean	Mathématiques
BARLET Roger	Chimie
BERTIN José	Mathématiques
BLOCK Marc	Biologie
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORRIONE Dominique	Automatique informatique
BOULON Marc	Mécanique
BOUTRON Claude	Glaciologie
BOUVET Jean	Biologie
BROSSARD Jean	Mathématiques
BRUGAL Gérard	Biologie
CAMPILLO Michel	Géophysique
CAVAILLE Jean Yves	Chimie
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
CHOLLET Jean Pierre	Mécanique
COLOMBEAU Jean François	Mathématiques (ENSL)
COTTET Georges-Henri	Modélisation, calcul scientifique, statis.
COURT Jean	Chimie
CUNIN Pierre Yves	Informatique
DAVID Jean	Géographie
DEROUARD Jacques	Physique
DHOUAILLY Danielle	Biologie
DUFRESNOY Alain	Mathématiques Pures
DUPUY Claude	Chimie
DURAND Mireille	Sciences Nucléaires
FONTECAVE Marc	Chimie
FOURNIER Jean Marc	Physique
GASPARD François	Physique
GIDON Maurice	Géologie
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GOURC Jean Pierre	Mécanique
GUIGO Maryse	Géographie
GUMUCHIAN Hervé	Géographie
HACQUES Gérard	Mathématiques Appliquées
HAMMOU Abdelkader	Chimie
HERBIN Jacky	Géographie
HERINO Roland	Physique
HERZOG Michel	Biologie
JARDON Pierre	Chimie
JUTTEN Christian	Physique
KERCKHOVE Claude	Géologie
KOSAREW Siegmund	Math. fondamentales et appliquées
KLINGER Jurgen	Glaciologie
LAURENT Christine	Mathématiques
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
MERCHEZ Fernand	Physique
MILAS Michel	Chimie
MOREL Alain	Géographie
MORIN Pierre	Physique
NGUYEN HUY Xuong-	Informatique
OUDET Bruno	Mathématiques Appliquées

../ ...

PAUTOU Guy	Biologie
PECHER Arnaud	Géologie
PELLETIER Guy	Astrophysique
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Claude	Biologie
PIBOULE Michel	Géologie
PORTESEIL Jean Louis	Physique
PUECH Laurent	Physique
RAYNAUD Hervé	Mathématiques Appliquées
REGNARD Jean René	Physique
ROBERT Claudine	Didactique des disciplines scientifiques
ROBERT Danielle	Chimie
ROBERT Gilles	Mathématiques Pures
SAJOT Gérard	Physique
SARROT REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
TEMPERVILLE André	Mécanique
TISSUT Michel	Biologie
TOURNIER Evelyne	Informatique et Mathématiques appliquées
VALLADE Marcel	Physique
VALLON Michel	Glaciologie
VICAT Jean	Physique
VINCENS Maurice	Chimie
VINCENT Gilbert	Physique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie
WITOMSKI Patrick	

Avant-Propos

Je tiens à remercier tout d'abord Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG et directrice du laboratoire Conception de Systèmes Intégrés, pour m'avoir accueilli dans son laboratoire et pour avoir guidé mes recherches. Le travail présenté ici a été possible grâce à ses nombreuses remarques et suggestions.

Je remercie également :

Monsieur Jacques MOSSIERE, Professeur et directeur de l'ENSIMAG, de me faire l'honneur de présider ce jury,

Monsieur Alain GUYOT, Maître de conférence à l'ENSIMAG, d'avoir accepté d'être rapporteur de cette thèse,

Herr Prof. Manfred Glesner, Professor an der Universität Darmstadt, dafür, daß er sich als Gutacher dieser Dissertation zur Verfügung stellt und dafür extra nach Grenoble anreist,

Monsieur Jacques TRILHE, Docteur ingénieur à SGS-Thomson et manager du projet ESPRIT 824 WSI, d'avoir accepté de faire partie de mon jury. Je le remercie également pour sa disponibilité.

Je remercie tous ceux qui, de près ou de loin, ont contribué aux travaux présentés dans cette thèse. Plus particulièrement les membres du groupe WSI et de la synthèse de haut niveau. Enfin, je remercie également tous les membres du CSI pour l'agréable ambiance de travail.

RÉSUMÉ

Cette thèse propose des méthodes et outils de synthèse d'architectures intégrées massivement parallèles destinées au traitement d'image de bas niveau.

Dans une première partie, l'implantation sur tranche entière d'un réseau 2D de 6720 processeurs (PEs) monobits, dans le cadre d'un projet Européen ESPRIT (WSI 824), est présentée. L'accent a été mis sur les techniques de tolérance aux défauts de fin de fabrication, obligatoires pour un circuit de grande échelle. Une approche hiérarchique a été adoptée. Au niveau sous-système, une colonne de PEs de réserve destinée à remplacer les PEs défaillants a été implantée. Au niveau de la tranche, un réseau de commutateurs original, contrôlé à partir des plots d'entrée/sortie permet de contourner les sous-systèmes défaillants et de construire la cible définitive.

Dans une deuxième partie un système de synthèse de haut niveau permet de concevoir automatiquement des réseaux de PEs dédiés à une application données à partir d'une spécification comportementale de haut niveau. Les ressources de réseau en termes de registres et connexions (multiplexeurs) sont alors minimisées.

Mots-clés : WSI, reconfiguration, rendement, tolérance aux défauts, réseau d'interconnexion, réseau 2D, SIMD, traitement d'image, synthèse de haut niveau.

ABSTRACT

This thesis presents methods and tools for the design of massively parallel low level image processing integrated systems.

In a first part, the design of a programmable 2D array of 6720 single bit processors (PEs) implemented on a full wafer within an ESPRIT (824) project is addressed. Two-level defect tolerance techniques have been used. At a subsystem level, a spare column is provided to replace defective PEs. At the wafer level, an original switching network externally controlled is used to bypass defective subarrays.

In a second part, a high level synthesis tool is used to synthesize automatically from a behavioral specification a dedicated array from a specific application. Resources in terms of registers, connections (multiplexers) and memory points are minimized.

Keywords : WSI, reconfiguration, yield, defect-tolerance, switching network, 2D array, SIMD, image processing, high-level synthesis.

INTRODUCTION

L'objet de cette thèse est la réalisation sur silicium d'architectures massivement parallèles pour le traitement d'image de bas niveau

Dans un premier chapitre un rappel est fait sur ce que l'on entend de façon générale par le traitement d'image de bas niveau. Il est rappelé qu'il s'agit de réaliser les premiers traitement sur une image digitalisée (filtrage, seuillage, détection de contour) et que l'architecture la mieux adaptée est une architecture SIMD constituée par un réseau 2D de processeurs monobits; chaque processeur étant associé à un pixel de l'image. Les réalisations commerciales ainsi que les produits de recherche concernant ce type d'architecture les plus connues sont évoquées. Cette thèse ayant comme objectif d'étudier une intégration poussée au maximum, un état de l'art sur l'intégration tranche entière consistant à réaliser sur une tranche de 4" un seul circuit, est présentée.

Dans un deuxième chapitre, la réalisation centrale de cette thèse, à savoir, la conception et la fabrication d'une architecture SIMD programmable d'un réseau monobits de 6720 processeurs intégrée sur une tranche entière (connue sous le nom de ELSA pour "European Large SIMD Array") sont présentées. Le processeur élémentaire 1 bit est relativement classique et toute l'originalité repose dans la mise en place de techniques de tolérance aux défauts de fabrication car il est clair qu'un circuit de cette taille ne peut être exempt de

défauts. La tolérance aux défauts est implantée de façon hiérarchique. Une technique figée par colonnes de réserve est utilisée au niveau des sous-matrices. Au niveau supérieur, une technique plus simple et plus originale de construction d'un réseau 2D de taille maximale regroupant des sous-matrices opérationnelles est utilisée. L'accent est mis sur les techniques de configuration à ce deuxième niveau. En particulier, la description et la réalisation d'un réseau de commutateurs contrôlables à partir des plots d'entrée sortie ont constitué l'élément le plus original. On montrera la réalisation finale de ce démonstrateur WSI (wafer scale integration) de plusieurs millions de transistors.

Dans une troisième partie, une retombée de l'étude d'ELSA est présentée. ELSA est un démonstrateur couteux et lourd à mettre en œuvre, par suite de son aspect programmable. A partir de l'expérience de conception, un système automatique de génération de circuits à la demande réalisés par une matrice de processeurs monobits et dédié au traitement d'image de bas niveau est proposé. Il s'agit cette fois de prendre comme cible une application dédiée, et de remplacer le réseau de processeurs "universels" programmables par un réseau à connexion dédiée dont le processeurs a une structure "minimale" adéquate. Le contrôle est une suite de mots de commandes figées. Des techniques du type compilation de silicium ont été mises en œuvre pour optimiser la structure finale.

Cette méthode de conception rapide de circuits dédiés a pu être validée sur 3 démonstrateurs sur silicium mais n'est pas encore entièrement automatisée à ce jour.

Cette thèse propose donc d'une part une architecture "universelle", programmable pour le traitement d'image de bas niveau, intégrée sur tranche entière et d'autre part un environnement de conception de circuits dédiée à base de la même architecture. Remarquons que de façon générale, les architectures SIMD de ce type ont été relativement peu étudiées comparant à la très grande masse de travaux sur les architectures systoliques et neuro-mimétiques.

CHAPITRE I

ARCHITECTURE PARALLELE

I.1 TRAITEMENT D'IMAGE

Le traitement d'image est intensivement utilisé dans des domaines très variés tels que l'automatisation des procédés industriels de fabrication (Soudage, découpage, usinage de surface, montage de carte électronique, tri, prise et dépose de pièces, assemblage et réparation), l'inspection (inspection de soudures ou défauts de métaux, lecture automatique de clichés médicaux), photographie industrielle (prise de vue par satellite de la terre pour la prévision météorologique, analyse de site minier, analyse de site spatiale etc...).

Une image peut provenir de plusieurs sources. Il existe maintenant des caméras qui délivrent directement des images digitales au lieu par exemple du film classique qui nécessite une digitalisation par scanner. L'image dans ce cas représente les luminances des objets dans la scène prise par la caméra. Dans d'autres domaines tels que la médecine ou l'astronomie, il existe des instruments qui délivrent des images digitales à partir de rayons X ou d'ondes ultrasons. Les sondes spatiales délivrent des images digitales à partir de mesures de radiation micro-onde ou infrarouge.

En général, le but recherché ou la finalité du traitement d'image est d'améliorer l'image ou d'en extraire des informations.

Les opérations typiques qui sont rencontrées sont

- élimination des "flous" de l'image

- lissage des taches et bruit de l'image

- amélioration du contraste ou autre propriété visuelle de l'image

- segmentation d'une image en régions différentiant par exemple l'objet et le fond.

- élimination ou réduction de distorsion

- reconnaissance de forme, classification et prise de décision.

Les avantages du traitement digital de l'image sont une précision élevée et une grande flexibilité. Ses inconvénients majeurs sont un coût élevé et une vitesse de traitement souvent insuffisante, en particulier pour les applications en temps réel.

La dénomination utilisée pour le traitement d'image est liée au type d'entrée traité et au type de résultats produits. On distingue deux types d'entrées et de sorties à savoir les images et les descriptions. Le *traitement*

d'image reçoit en entrée une image et produit en sortie un résultat de type image également. Si la sortie est de type description alors on parlera de *reconnaissance de forme* ou de *vision par ordinateur*. On peut avoir comme entrée une description et comme résultat une image; dans ce cas on parle de *synthèse graphique*.

Une image est une représentation visuelle d'un objet ou d'une scène. Typiquement une scène est filmée par une caméra qui transmet des signaux, représentant la luminosité contenue dans cette scène, sous forme analogique. Ces signaux analogiques sont transformés en signaux numérisés, autrement dit *échantillonnées* dans le temps et *quantifiés* en amplitude. Le résultat de l'échantillonnage donne naissance à une *image digitale* (ou tout simplement image par la suite). Une image est donc un tableau bidimensionnelle (2D) de nombres. Une image est généralement de forme rectangulaire, mais la forme carré existe aussi et on trouve typiquement des tailles telles que 256×256 et 512×512. Chaque élément dans ce tableau est appelé un *pixel*. Chaque pixel est codé sur un nombre fini de bits représentant une quantité positive.

Les algorithmes rencontrés dans les domaines cités en haut sont très divers. Nous nous limitons dans ce travail au seul traitement d'image tel qu'il est défini ci-dessus; l'entrée est donc une image et la sortie est également une image. Ce type de traitement est aussi connue sous le nom de traitement d'image bas niveau.

I.1.1 Traitement d'image bas niveau

Ce sont les premiers traitements que reçoivent les images une fois échantillonnées et quantifiées. Cela comprend, en général, le filtrage et l'amélioration de l'image pour des traitements ultérieurs.

Une représentation est dite de bas niveau si elle fondée sur des tableaux de données qui correspondent directement aux points images. Une correspondance naturelle est établie entre une donnée dans le tableau et un "pixel" de l'image.

Un algorithme est de bas niveau s'il prend comme source (entrée) une image et produit une image de la même taille comme destination (sortie). Les opérations rencontrées dans le bas niveau peuvent être groupés en deux grandes catégories : ponctuelles et locales.

Les opérations *ponctuelles* sont caractérisées par un traitement qui prend une image comme entrée et rend un résultat sous forme d'image également et telles que la valeur d'un pixel de l'image résultat dépend uniquement de la valeur du pixel correspondant de l'image source. Ces opérations sont

abondamment utilisées dans les manipulations telle que l'amélioration du contraste par transformation du niveau de gris, compression de dynamique ou seuillage...Une image résultat peut être aussi obtenue à partir de plusieurs images; on peut être intéressé par exemple par la somme ou le produit de deux images point par point.

Dans la catégorie des opérations *locales*, la valeur d'un pixel de l'image résultat dépend des valeurs associées aux pixels dans le voisinage du pixel correspondant de l'image source. De telles opérations sont utilisées pour la réduction de bruit, la détection de frontières, le lissage...

I.1.2 Besoins architecturaux des systèmes de traitement d'image de bas niveau

Le traitement d'image bas niveau est dominé par le traitement de tableaux de données numériques. Ces tableaux ont la taille des images à manipuler et les éléments de ces tableaux sont des entiers binaires, le nombre de bits étant variable. Les calculs effectués sont essentiellement des opérations arithmétiques et logiques classiques, bien que des calculs complexes tels que la racine carré et fonctions trigonométriques puissent aussi être utilisés.

Les opérations se font entre deux tableaux de même taille, entre des éléments et leurs voisins, entre des éléments et un masque ou enfin entre des éléments et une constante globale. Les ressources matérielles doivent donc assurer une communication locale et une propagation globale de valeurs. Les architectures que nous étudions ici sont des architectures dites massivement parallèles que nous allons définir.

I.2 ARCHITECTURES MASSIVEMENT PARALLELES

Pour pouvoir résoudre certain problèmes de traitement d'image dans des temps raisonnables, des calculateurs très puissants sont nécessaires. Beaucoup d'efforts ont été faits pour augmenter les performances des ordinateurs. Ces efforts sont orientés en permanence vers l'amélioration voire même l'invention de composants de plus en plus performants. Cependant, l'accroissement des performances au niveau composant a des limites pour une technologie donnée. D'autres voies sont aussi explorées tel que l'accroissement de performances obtenue par les architectures et les algorithmes parallèles.

Dans notre application, beaucoup de calculs tels les opérations au niveau pixel sont indépendants les uns des autres et peuvent donc être exécutés en parallèle.

Le parallélisme peut être temporel ou spatial. Le parallélisme temporel peut se faire par l'enchaînement ("pipelining") de l'exécution des instructions et a donné lieu aux machines dites vectorielles.

Le parallélisme spatial consiste en la répartition des données sur un grand nombre de processeurs. Unger [Unge58] a proposé une architecture intéressante pour résoudre des problèmes de reconnaissance de formes. La structure consistait en une matrice rectangulaire de quelques centaines de processeurs élémentaires (PEs) opérants sous le contrôle d'un seul processeur central. Chaque processeur pouvait communiquer avec ses voisins les plus proches. Ce papier était le précurseur d'une famille d'architectures qui sera classée plus tard, en 1966 par Flynn, sous le nom de SIMD pour "Single Instruction Multiple Data" [Flyn66].

Flynn a en effet proposé une classification générale des architectures de calculateurs qui prend en compte comme paramètres : le flot d'instructions et le flot de données. Cela a permis d'obtenir quatres catégories :

SISD (flot d'Instructions Simple, flot de données Simple) ; dans cette classe, nous trouvons les machines séquentielles ou encore dites von Neumann, où une unité centrale traite une donnée à la fois.

MISD (flot d'Instructions Multiple, flot de données Simple) ; cette classe, peu réaliste implique plusieurs unités centrales qui opèrent avec des instructions multiples sur une seule donnée.

SIMD (flot d'Instructions Simple, flot de données Multiple).

MIMD (Multiple flot d'Instructions, Multiple flot de données) ; celle-ci comporte plusieurs processeurs autonomes qui exécutent des instructions différentes sur des données différentes.

Nous ne prétendons pas faire le point sur la classification d'architectures, car la taxinomie proposée par Flynn est suffisante et convient à notre approche. Remarquons qu'elle n'est plus adaptée pour rendre compte des nombreuses architectures plus récentes. Pour plus de détails on peut consulter [Skil88], [Tuck88] et [Dunc90].

Dans ce travail, nous nous concentrons uniquement sur les machines SIMD.

I.2.1 Machines SIMD : historique

Comme annoncé précédemment les architectures SIMD sont très efficaces pour résoudre les problèmes de traitement d'image de bas niveau [Rose81, 83, 85], [Batc82], [Reev82], [Litt89].

Historiquement, la première machine, appelé SOLOMON (Simultaneous Operation Linked Ordinal MOdular Network) fut construite chez Westinghouse par Slotnick [Slot62]. Elle contenait 16×16 processeurs. Pendant la même période, McCormick [McCo63], à l'université de l'Illinois, concevait une machine de 32×32 processeurs connue sous le nom de ILLIAC III (ILLInois Advanced Computers). Ces deux machines sont restées au stade expérimental. Un peu plus tard, Duff [Duff73], à UCL, étudiait une machine de 20×20 processeurs élément de base pour une série de machines connues plus tard sous le nom de CLIP (Cellular Logic for Image Processing). Bien que ces machines n'aient pas servi pratiquement, elles ont servi de base pour les développements futurs. La difficulté que les concepteurs devaient affronter alors était liée à la technologie des année 60 dont la fiabilité était insuffisante pour construire des machines aussi ambitieuses.

Slotnick qui travaillait sur SOLOMON rejoignit l'université de l'Illinois et construisit, en 1972, ILLIAC IV [Barn68], [Bouk72] qui constitua une véritable percée dans le monde des ordinateurs. Au cours de la même année STARAN [Batc72] fut aussi construit et livrée par GOODYEAR à la NASA.

En même temps en Angleterre, ICL, développait la machine DAP (Distributed Array Processor) [Redd73] comptant (32×32) processeurs qui mit l'accent sur le concept de mémoire distribuée. Une version de 64×64 fut commercialisée par Active Memory Technology [AMT81] au début des années 80.

ILLIAC IV restait en service jusqu'en 1983 où il fut remplacé par la machine MPP (Massively Parallel Processor) [Batc83]. La machine MPP, constituée d'une matrice de 128×128, a bénéficié des possibilités avancées d'intégration VLSI et ne fut fabriqué qu'à un seul exemplaire. Elle a définitivement démontré la faisabilité des ordinateurs parallèles et a apporté une expérience très riche dans le domaine d'exploration des algorithmes parallèles.

Il faut noter que les projets précédents ont porté sur de grandes machines et ont été très couteux. Dans une optique différentes des grandes machines, la société MARTIN MARIETTA a commandé à NCR un circuit à très haute

intégration qui peut être la base d'une machine SIMD performante. Le circuit, appelé GAPP (Geometric Arithmetic Parallel Processor), est commercialisé depuis 1985. Il contient un réseau de 6×12 processeurs élémentaires. MARTIN MARIETTA a construit un système embarqué qui contient 51 840 processeurs élémentaires ainsi qu'un système non embarqué coprocesseur d'un VAX qui contient 82 944 processeurs élémentaires, ceci est le plus grand réseau jamais construit [Clou88].

A la différence du GAPP qui est un circuit intégré commercialisé, plusieurs machines ont vu le jour ces dernières années. Citons à titre d'exemple la famille AIS, commercialisé par Applied Intelligent Systems. Selon le produit, le nombre de PEs varie de 64 PEs pour le AIS 3000 à 1K pour le AIS 5000. Les performances pour un AIS 4000 (512 PEs) sont de l'ordre de 1,4 ms pour une addition sur 8 bits. La famille DAP, commercialisé par Active Memory Technology, contient deux versions avec 1K et 4K processeurs avec des performances respectives de 140 et 560 MFLOPS. La "connection machine" de Thinking Machine dans sa version la plus récente (CM-2G) contient 64k processeurs et atteint une performance de 10 GFLOPS. Enfin, MasPar commercialise plusieurs machines allant de 1K (MP 1100) à 16k (MP 1200) avec des performances respectives de 82 MFLOPS et 1.3 GFLOPS.

I.2.2 Machines SIMD : architectures

Ce sont des machines qui comportent un très grand nombre de processeurs élémentaires qui exécutent tous la même instruction avec un contrôleur central et une horloge unique. Un modèle d'une telle machine est donnée dans la figure 1.1.

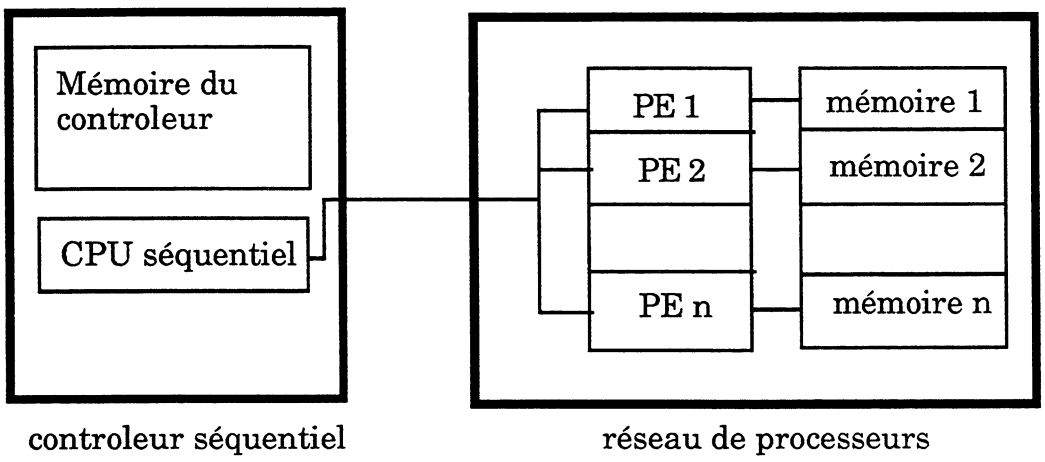


Figure 1.1 Modèle général d'une machine SIMD

Ces architectures sont caractérisées par le type de processeur, la mémoire, le réseau d'interconnexion et enfin la technologie. Certaines de ces machines sont destinées au calcul général tel que l'ILLIAC IV [Barn68], DAP [Reed73], MPP [Batc82]. D'autres sont spécialisées pour le traitement associatif tel que STARAN [Batc], PEPE [Levi8]. Du point de vue traitement, ces machines sont divisées en deux catégories à savoir les machines à base de processeurs monobits ("bit-slice") et les processeurs à mots longs ("word-slice"). Nous nous restreignons dans cette thèse aux machines monobits.

1.2.3 Structure du processeur

L'architecture du processeur de base est usuellement extrêmement simple. En général, son unité de calcul arithmétique et logique (UAL) n'effectue des opérations que sur des données de 1 bit, le calcul est alors en bit-série.

1.2.4 Organisation de la mémoire

Dans la plupart des machines organisées en réseaux, la mémoire globale peut être vue comme une mémoire à trois dimensions. Comme les PEs sont monobits la mémoire permet à chaque processeur d'accéder à 1 bit d'un mot de k×1 bits. Ce concept de mémoire à trois dimensions (figure 1.2) s'applique aux GAPP, MPP et le DAP. Cette mémoire peut être soit intégrée dans le PE lui même ou externe dans la plupart des cas.

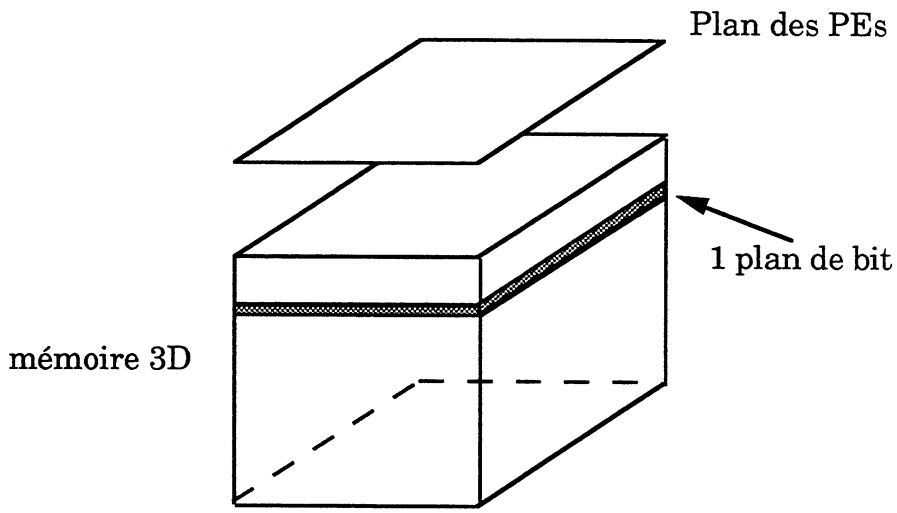


Figure 1.2 Le concept d'une mémoire 3D

Si la configuration des PEs est linéaire, alors la mémoire est bidimensionnelle (figure 1.3). Cette organisation a été implémentée dans Staran.

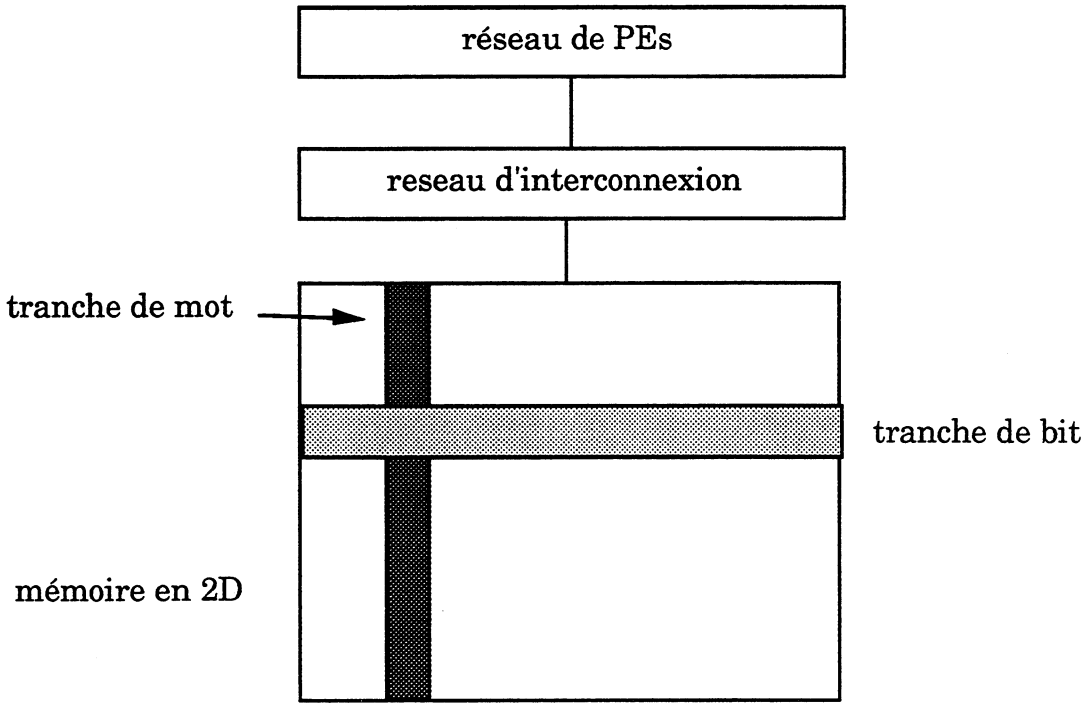


Figure 1.3 Le concept d'une mémoire 2D

Machine	ILLIAC IV	MPP	DAP	GAPP	C M
Nombre de PEs par chip	—	8	16	72	16
Type de PE	64 bits	1 bit	1 bit	1 bit	1 bit
mémoire interne /PE	2K bytes	—	—	128 bits	—
mémoire externe /PE	64 Kbits	64 Kbits	1 Mbit		64 Kbits

Table 1.1 Exemple de mémoire dans quelques machines (interne "on-chip", externe "off-chip")

1.2.5 Réseaux d'interconnexion et communication

Du fait du grand nombre de PEs et du parallélisme à granularité très fin, la topologie du réseau d'interconnexion joue un rôle très important dans les performances de telles machines. Les réseaux d'interconnexions peuvent être conçus spécifiquement pour un algorithme donné tel que la transformée de Fourier rapide ("FFT") par exemple, pour une classe d'algorithmes telle que le

calcul matriciel, ou pour implémenter n'importe quel type de communication sans algorithme ou opération cibles. Ce problème de réseaux d'interconnexion a fait l'objet de nombreuses recherches [Batc68], [Seig77] [Seig79], [NaSa81], [Flan82], [DuVo82], [SaSc89], [PrRe89], [ChCh90], [RaSa90], [BoRa91].

Citons à titre d'exemple la grille, la ligne-colonne, l'hypercube.

La *grille* est la structure la plus simple. Elle a été proposée par Unger et matérialisée par l'ILLIAC IV. Dans cette structure chaque processeur communique avec ses voisins les plus proches (figure 1.4). Cette structure est utilisée aussi dans le MPP et le DAP.

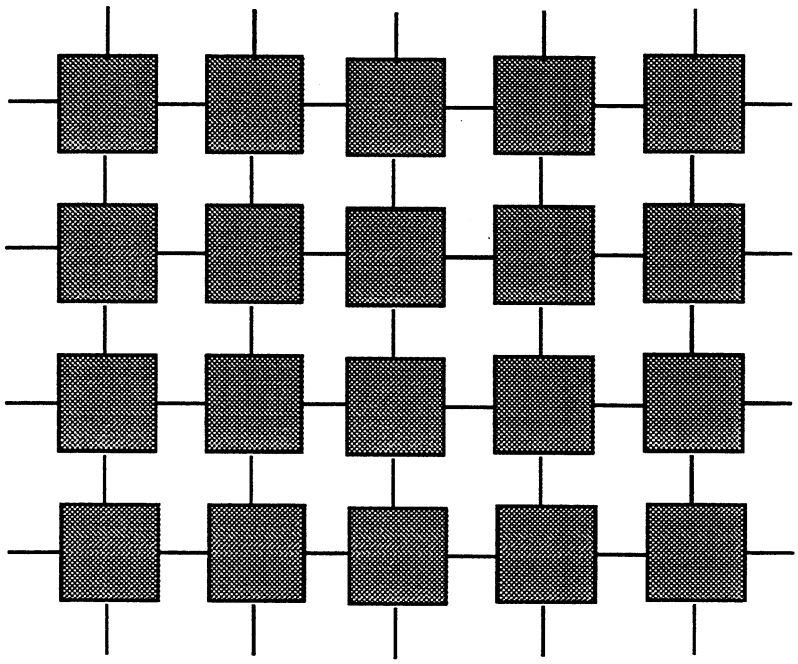


Figure 1.4 Structure de grille

Une variante de cette structure utilisée dans BLITZEN [Heav86], est connue sous le nom de "grille en X" qui consiste en fait à étendre la structure de base à quatre directions en englobant les directions à 45° c'est à dire le nord-est, le sud-est, le sud-ouest et le nord-ouest (figure 1.5).

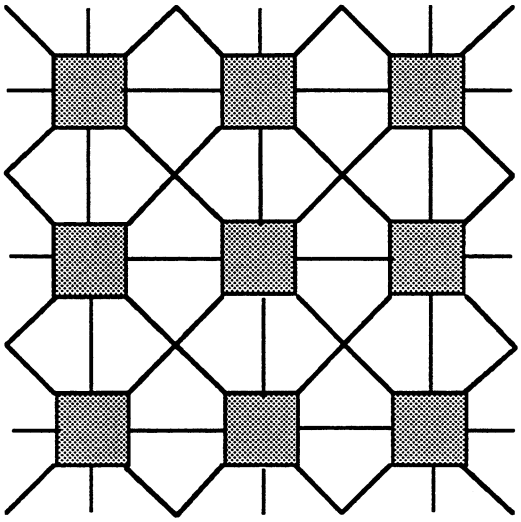


Figure 1.5 Interconnexion en X

L'interconnexion *ligne-colonne* est spécifique au GAPP [Clou84] et à ELSA [Harb86]. Il s'agit d'un réseau 2D organisé en lignes et colonnes. C'est en fait une grille mais séparée en deux plans orthogonaux : un plan nord-sud et un plan est-ouest. Cette séparation permet un déplacement simultané dans les deux plans. La figure 1.6 montre en détail cette structure. Les registres mis en jeux sont noircis.

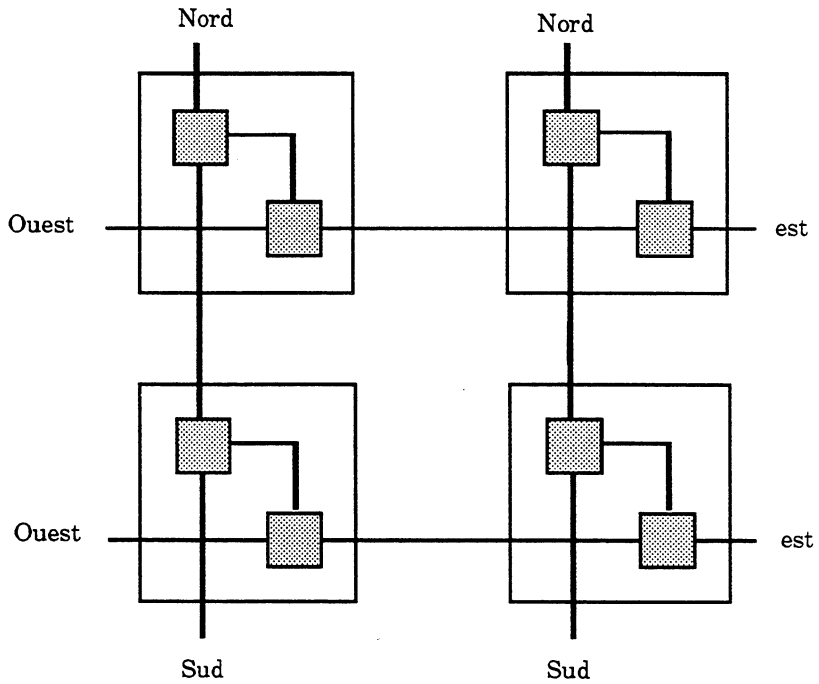


Figure 1.6 Interconnexion ligne-colonne

L'interconnexion *en hypercube* a été implémentée dans Staran et Aspro. La figure 1.7 illustre cette structure avec un cube de 8 éléments. Bien que plus

riche en interconnexions que les précédents, ce réseau ne permet pas non plus de faire directement tous les chemins.

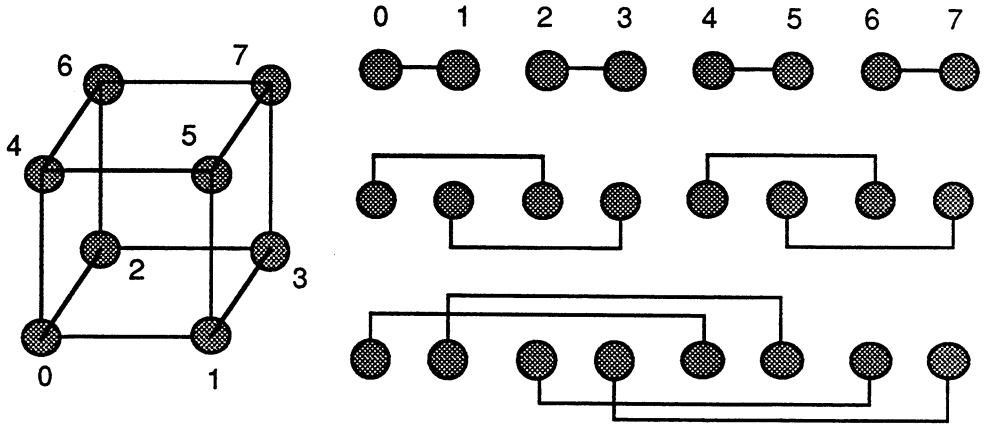


Figure 1.7 Exemple de 3-cube et de ses interconnexions

I.3. LA TECHNOLOGIE ET L'INTÉGRATION SUR TRANCHE ENTIERE

Les performances des machines massivement parallèles sont intimement liées à l'intégration technologique. Cette intégration peut être augmentée de 2 façons.

La première consiste à la finesse de la gravure. Mais ceci est limité d'une part par le processus technologique utilisé et d'autre part par la physique du dispositif lui-même. On n'est pas très loin de la taille limite au delà duquel le dispositif n'obéit plus aux lois physiques classiques.

La deuxième est d'augmenter la taille des puces. Cette solution a comme limite la taille de la tranche de silicium. Elle ouvre la porte à l'étude de l'intégration d'un système sur tranche entière, connue dans la littérature sous le nom WSI (de l'anglais "Wafer Scale Integration").

Les avantages attendus de l'approche WSI sont un coût minimum, une grande compaction, une fiabilité accrue, une augmentation de la vitesse et une réduction de la consommation [Moor85], [West87].

L'intégration tranche entière permet de réduire considérablement le nombre de boîtiers. La connectique est énormément réduite. La diminution du nombre de boîtiers entraîne une diminution des amplificateurs d'entrée/sortie. La puissance consommée diminue et la vitesse de fonctionnement du système augmente. De plus les risques de pannes d'un système sont directement liés à la

connectique élevé entre boîtiers. La fiabilité de systèmes "WSI" est alors accrue.

Cependant, un certain nombre de problèmes sont liés à l'intégration sur tranche entière. D'une par le problème fondamental est celui du rendement. Les circuits intégrés tranche entière ont un rendement de fin de fabrication très bas. D'autre part des connections trop longues sur la tranche posent des problèmes de charges capacitives.

Un autre problème concerne la dissipation de la chaleur produite par le circuit (un millier de watts pour la machine proposé par Trilogy et une centaine pour la machine étudié par l'université de Brunel [Lea91]). En effet, une élévation de température peut dégrader les performances des transistors jusqu'à les rendre inopérants.

Néanmoins le problème le plus résistant reste celui du rendement. Durant la fabrication d'un circuit intégré des défauts peuvent apparaître (poussière à une étape de fabrication, manque de métal sur une connexion, *défaut dans le cristal*, mauvaise croissance d'un oxyde de grille, ...) engendrant un mauvais ou un non fonctionnement d'un élément du circuit. Avec les taux de rendement habituels toute tranche de silicium comportera des éléments défectueux. Il convient donc sur les circuits WSI de mettre plus d'éléments qu'il est nécessaire, ces éléments redondants pallient aux éléments défectueux. Plusieurs méthodes ont été mises au point pour utiliser uniquement les éléments sains. Construire une architecture cible en n'utilisant que les éléments sains s'appelle la *configuration* d'une architecture cible.

La connexion d'éléments sains est effectuée soit par des dispositifs de connexion irréversible (technique à laser ou à faisceau d'électron) soit par des dispositifs de connexion réversible à programmation électrique.

Les premiers travaux ou tentatives pour intégrer les circuits WSI furent menées chez Westinghouse [SaLy64]. Parallèlement, Texas Instrument [Petr67] a mis au point, dès les années 1966, une technique connue sous le nom de "discretionary wiring". Celle ci permet de relier les éléments sains d'une tranche à l'aide d'un niveau de métal supplémentaire. En 1969, cette compagnie à réalisé avec cette technique un registre à décalage de 32kbit sur une tranche de 2 pouces. Cette technique fut abandonnée parce que peu pratique: coût du traitement individualisé de chaque tranche, pollution de la tranche pendant la phase de test, etc... Par la suite, Hughes [Calh69] et IBM [Bars77] tentaient d'améliorer cette technique avec quelques variantes. Plus tard en 1988, un circuit de Transformé de Fourier à été développé au

laboratoire FUJISTU [Yama89]. Ce circuit contient 170000 portes intégrées réalisé dans une technologie CMOS 2.3µm triple métal sur une tranche de 4 pouces. Parmi les 88 processeurs intégrés seulement 48 sont utilisés, soit 45% de processeurs sont redondants. La logique réalisant la configuration représente 30% de la surface totale

Des techniques utilisant des fusibles ont été mises au point en parallèle. Honeywell [Hunt76] a construit une mémoire composée de cellules de base connectées à un bus central traversant la tranche. Des fusibles ont été utilisés pour déconnecter d'une manière permanente les cellules mémoires défectueuses. Puis, Innova Inc. à réalisé une mémoire SRAM de 256 kByte en utilisant cette technique. Hughes [Filo77] a conjugué les deux approches ("discretionary wiring" et la technique des fusibles) pour réaliser une mémoire tolérant les défauts.

La société TRILOGY, crée aux début des années 80, a planifié la production de gros calculateurs compatibles IBM, intégrés sur tranche entière. Bien que son échec fut spectaculaire, les techniques développées par cette compagnie méritent d'être citées. Le calculateur contient à peu près 1500 circuits. Pour pallier les défauts de fabrication, une technique bien connue de triplication avec vote majoritaire ("Triplicated Massive Redundancy") [Peltz83] a été utilisée en plus des fusibles et des dispositifs de contournement.

Une technique appelé RVLSI (Restructurable VLSI) à été développée au MIT Lincoln Laboratory au-début des années 80 [Wya89] [Jess85]. Une tranche RVLSI est composé de cellules complètement isolées et d'un réseau d'interconnexion. Les connexions entre les cellules valides sont établies à l'aide d'un laser. De nombreux circuits ont été réalisés pendant la période 1980 à 1988. Le premier circuit réalisé avec cette technique est un intégrateur digital. Ce circuit contient 200% de redondance pour les cellules et 50% de redondance pour les interconnexions. Le circuit a été fabriqué dans une technologie CMOS 5µ sur une tranche de 3 pouces. La dernière réalisation est un réseau 2D de processeur et de mémoire. Le circuit contient 60% de redondance pour la mémoire et 150% de redondance pour le processeur. Il a été fabriqué dans une technologie CMOS 2µ sur une tranche de 5 pouces.

Au cours des années 80 des mémoires tranches entières ont été proposés. NTT [Kita80] en 1980 a fabriqué pour la reconnaissance de mots, une ROM à 4 Mbits sur une tranche de 3 pouces. En 1984, NTT a fabriqué pour le stockage d'images, une mémoire statique de 1.5 Mbits sur une tranche de 4" [Cohe84]. Peu après, une deuxième firme INOVA Microelectronics, a annoncé une

mémoire statique de 8 Mbits [Benn85]. Puis, la société S.G.S Thomson Microelectronics a réalisé en 1988 une mémoire statique de 4.5 Mbits [Nasr88] [Marr89].

Citons au titre de la recherche, le circuit WASP (WSI Associative String Processor) développé à l'université de Brunel (U.K) depuis 1984. Ce circuit est configuré par des commutateurs programmables électriquement et de manière réversible [Lea89]. Ces commutateurs à base de portes de transfert permettent de contourner les blocs défectueux. Plusieurs versions contenant 1.26M et 8.43M de transistors ont été fabriquées. La dernière version a été fabriquée en 1990 en technologie CMOS 2µ sur une tranche de 6 pouces.

CHAPITRE II

LE PROJET ELSA

L'objectif du projet ELSA est de pouvoir faire du traitement d'image de bas niveau en temps réel sur une image en couleur de haute résolution. Il s'agit en fait de concevoir une matrice SIMD permettant d'atteindre des performances de l'ordre de quelques dizaines de milliers d'instructions par seconde.

II.1 L'architecture d'ELSA

II.1.1 Le processeur élémentaire

ELSA est un réseau de processeurs élémentaires (PE). Au niveau architectural, chaque PE contient six multiplexeurs, cinq bascules, un additionneur/soustracteur et deux mémoires (figure 2.1). Nous allons décrire le rôle de ces différents éléments [Harb89, 90].

II.1.1.1 Les multiplexeurs

Les multiplexeurs contrôlent la sélection des signaux à l'entrée de l'additionneur/soustracteur, des RAMs, d'une bascule drapeau noté FLG et d'un registre de communication CM.

II.1.1.2 Les bascules

Les bascules stockent les données de sortie de cinq des multiplexeurs. Trois d'entre eux (N S , E W , C) stockent les entrées pour l'additionneur/soustracteur, un autre (CM) est utilisé comme un registre de communication et le dernier (FLG) sert de drapeau conditionnel. Les bascules échantillonnent les sorties de tous les multiplexeurs — à l'exception de la RAM — pendant la première phase de l'horloge de base et produisent un résultat pendant la deuxième phase. L'additionneur/soustracteur, quant à lui, reçoit directement des sorties séparées pendant la première phase de l'horloge.

II.1.1.3 Le drapeau conditionnel

La sortie du drapeau FLG est utilisée pour générer un signal conditionnel $FG = \overline{C2} + FLG$, C2 étant un signal de contrôle global. Quand FG est à 0 toutes les bascules, à exception de CM, ont leur chargement inhibées et gardent leurs valeurs précédentes.

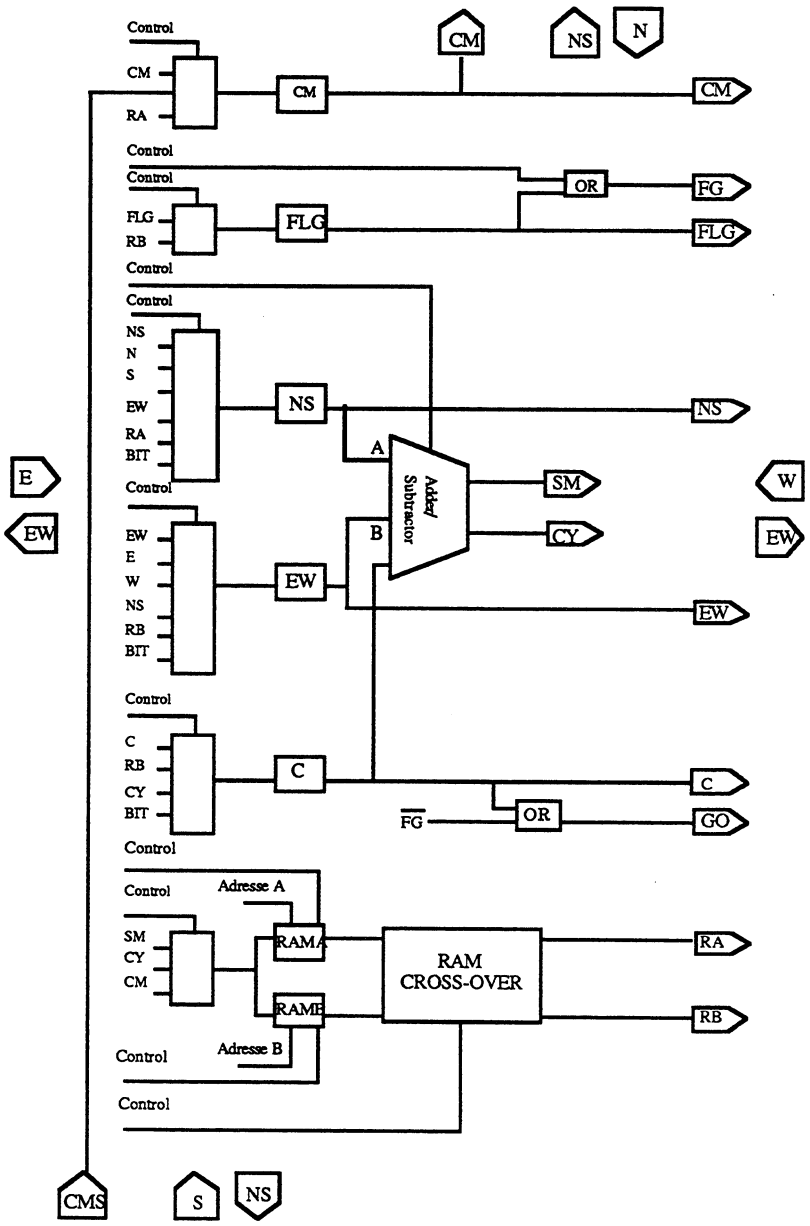


Figure 2.1 Le processeur élémentaire d'ELSA.

II.1.1.4 L'additionneur/soustracteur

L'additionneur/soustracteur reçoit trois entrées à savoir NS, EW, C et produit deux résultats la somme SM et la retenue CY. Il permet, selon les commandes, le calcul de NS+EW+C, NS-EW-C et EW-NS-C.

II.1.1.5 Les mémoires

Chaque PE dispose de deux RAMS statiques adressables indépendamment l'une de l'autre. Chaque RAM contient 64 mots d'un bit; elle est organisée en une matrice de 4 lignes et 16 colonnes. Au début de chaque période PHI1 un signal de précharge est appliqué aux RAM, permettant aux données d'être disponible pendant le reste de PHI1. L'écriture dans les RAMS est possible

Après avoir effectué des études de rendement des circuits intégrés sur tranche entière (cf. II.2), une structure cible à été choisie dans laquelle les PEs sont regroupés en sous-matrices et reliés à l'aide d'un réseau de reconfiguration. Ce réseau permet de connecter les sous-matrices entre-elles afin de réaliser la matrice SIMD fonctionnelle globale (figure 2.3).

Chaque sous-matrice contient une matrice de 7x12 PEs. Les signaux de commandes et les adresses de RAM sont décodés au niveau central et sont acheminés vers chacun des éléments par une matrice de signaux de commande. Aux extrémités de chaque sous-matrice, des amplificateurs sont implantés. Ils sont bidirectionnels pour les données et unidirectionnels pour le bus de communication. Le mode de ces amplificateurs (entrée ou sortie) est déterminé en décodant l'instruction en cours.

Le réseau dispose aussi d'une sortie globale appelée "GO" (de "global output"). Ce signal est un ET logique de tous les registres C des PEs. Sa valeur est un zéro logique si et seulement si tout le plan C contient que des zéros.

L'architecture d'ELSA est entièrement cascadable sans pénalisation au niveau des performances.

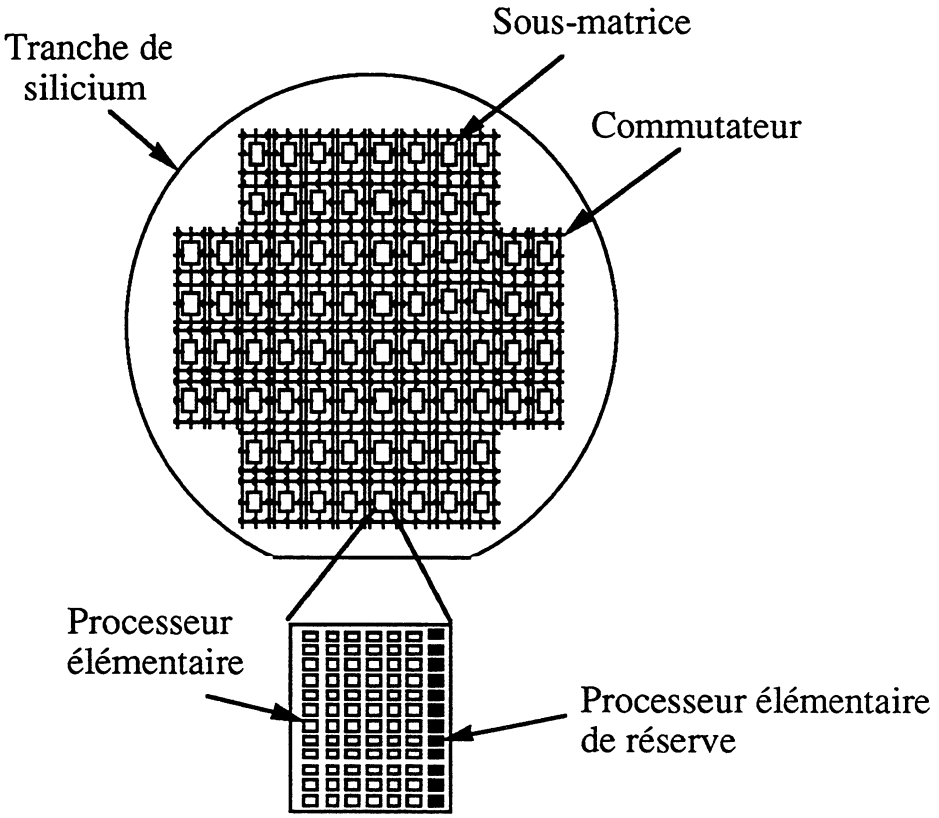


Figure 2.3 L'architecture d'ELSA

Après avoir effectué des études de rendement des circuits intégrés sur tranche entière (cf. II.2), une structure cible à été choisie dans laquelle les PEs sont regroupés en sous-matrices et reliés à l'aide d'un réseau de reconfiguration. Ce réseau permet de connecter les sous-matrices entre-elles afin de réaliser la matrice SIMD fonctionnelle globale (figure 2.3).

Chaque sous-matrice contient une matrice de 7x12 PEs. Les signaux de commandes et les adresses de RAM sont décodés au niveau central et sont acheminés vers chacun des éléments par une matrice de signaux de commande. Aux extrémités de chaque sous-matrice, des amplificateurs sont implantés. Ils sont bidirectionnels pour les données et unidirectionnels pour le bus de communication. Le mode de ces amplificateurs (entrée ou sortie) est déterminé en décodant l'instruction en cours.

Le réseau dispose aussi d'une sortie globale appelée "GO" (de "global output"). Ce signal est un ET logique de tous les registres C des PEs. Sa valeur est un zéro logique si et seulement si tout le plan C contient que des zéros.

L'architecture d'ELSA est entièrement cascadable sans pénalisation au niveau des performances.

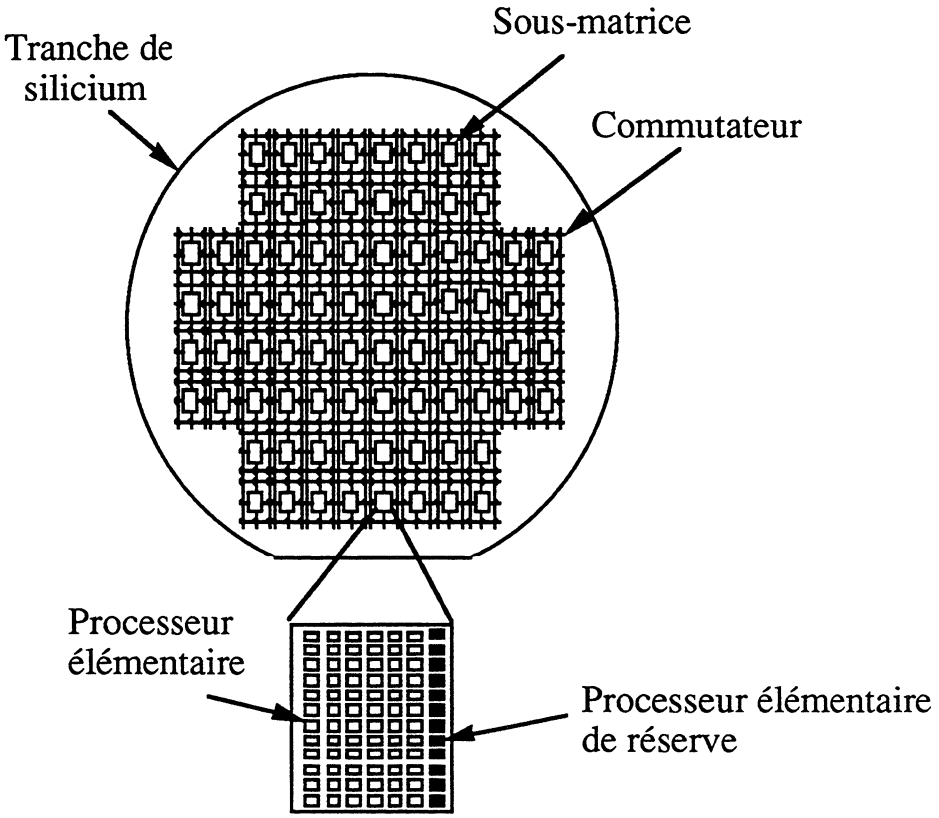


Figure 2.3 L'architecture d'ELSA

II.2 Rendement d'architecture intégrées sur tranche

II.2.1 Notion habituelle de rendement de circuit

Le rendement d'un circuit est la probabilité d'avoir aucun défaut sur le circuit. L'apparition de défaut sur un circuit est un phénomène aléatoire. Pour une technologie donnée et un type de conception, ce rendement dépend uniquement de la surface du circuit. De nombreux modèles ont été élaborés cherchant à donner la meilleure prédiction. Nous allons présenter les modèles les plus utilisés.

II.2.1.1 Modèle de Poisson

Ce modèle utilisant la distribution de Poisson à été proposé en premier par Hofstein et Heiman en 1963 [HoHe63]. Il suppose que les défauts sont répartis de manière uniforme sur la surface avec une densité moyenne de défaut D_m. Le rendement (R) s'écrit :

$$R = e^{-D_m S}$$

où S est la surface du circuit en cm^2 et D_m est en nombre de défauts par cm^2. Le produit $D_m S$ correspond au nombre moyen de défauts sur le circuit, D_m est généralement compris entre 1 et 8 défauts par cm^2. Ce modèle est très simple et très utilisé.

II.2.1.2 Modèle Binomial négatif

Pour de plus grandes surfaces, le modèle de Poisson donne des estimations un peu pessimistes par rapport aux rendements observés. Cette déviation est attribué aux amas de défauts ("Cluster") que l'on observe sur les tranches. Le modèle proposé par Stapper [Stap73], [StRo82] tient compte de ce phénomène. Le rendement s'écrit alors :

$$R = \frac{1}{\left(1 + \frac{D_m S}{ß}\right)^{ß}}$$

où ß ($0 \leq ß < \infty$) est un paramètre déterminé de manière empirique qui tient compte du phénomène d'amas. Plus ß s'approche de 0, plus le phénomène d'amas est important. Quand ß tend vers l'infini, ce phénomène est négligeable et on retrouve le modèle de poisson. Selon Stapper, expérimentalement on a $0.3 \leq ß \leq 5$.

D'autres modèles ont été développés tenant compte d'une répartition de défauts non uniforme. Murphy [Murp64] a proposé une fonction de densité de défauts $f(D)$ telle que:

$$R = \int_0^\infty f(D)e^{-DS}dD$$

avec $\int_0^\infty f(D)dD = 1.$

Plusieurs fonctions de distribution ont été proposées donnant des modèles différents.

II.2.1.3 Modèle de Murphy

Murphy en premier a proposé une fonction de distribution de défauts constante autour d'une valeur moyenne Dm.

$$\begin{cases} f(D) = \dfrac{1}{2D_m} & \text{pour } D \in [0, 2D_m] \\ f(D) = 0 & \text{pour } D \notin [0, 2D_m] \end{cases}$$

Le rendement s'écrit alors :

$$R = \frac{1 - e^{-2D_m S}}{2D_m S}$$

Il a ensuite proposé pour $f(D)$ une fonction linéairement croissante jusqu'à Dm, puis linéairement décroissante jusqu'à 2Dm telle que :

$$\begin{cases} f(D) = D/D_m^2 & \text{pour } D \in [0, D_m] \\ f(D) = (2 - D/D_m)/D_m & \text{pour } D \in [D_m, 2D_m] \\ f(D) = 0 & \text{pour } D \in [2D_m, \infty] \end{cases}$$

on obtient :

$$R = \left(\frac{1 - e^{-D_m S}}{D_m S} \right)^2$$

II.2.1.4 Modèle de Seeds

Seeds propose une distribution à décroissance exponentielle pour $f(D)$ à partir de laquelle il obtient le rendement suivant :

$$R = e^{-(2D_m S)^{1/2}}$$

De la moyenne faite sur les rendements de Murphy et de Seeds, résulte le modèle de Murphy-Seeds :

$$R = \frac{\left(\frac{1 - e^{-Dm\,S}}{Dm\,S}\right)^2 + e^{-(2Dm\,S)^{1/2}}}{2}$$

La figure 2.4 représente les modèles pour une valeur de $Dm=0,3$ défaut/cm^2.

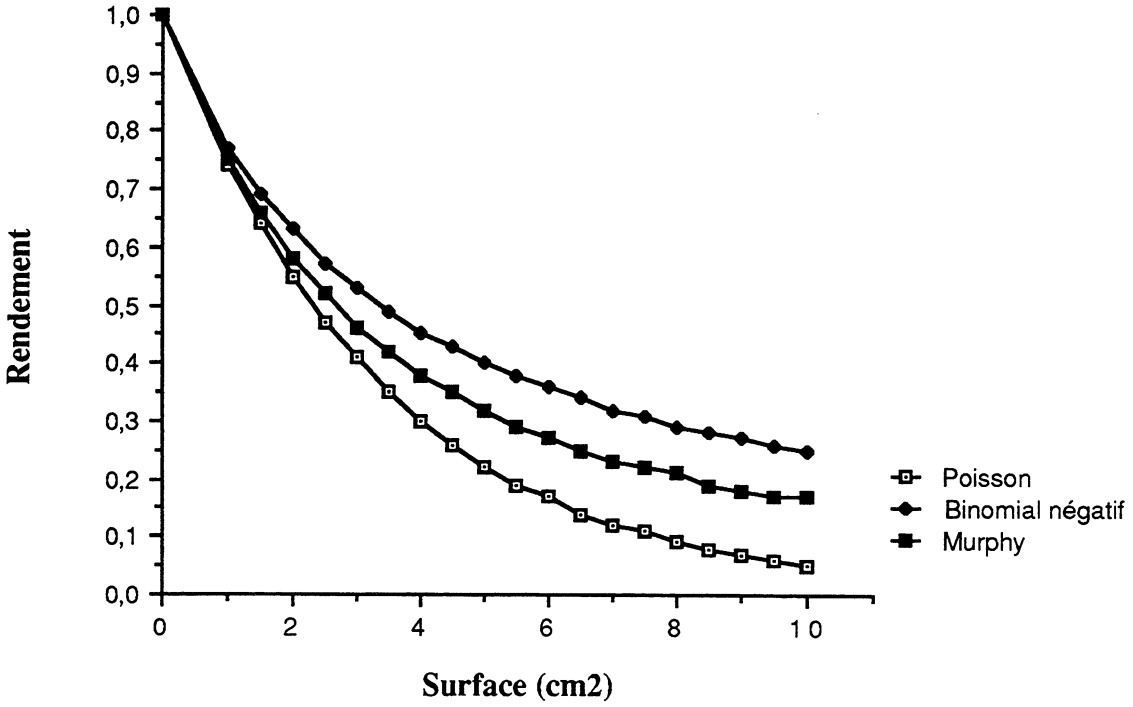


Figure 2.4. Rendement des différents modèles.

Notons qu'il existe encore d'autres modèles tenant compte du fait que les défauts apparaissent plus fréquemment au centre et sur les bords de la tranche, ceux-ci introduisent alors une fonction de répartition des défauts radiale [Walk87],[Gand87].

II.2.2 Stratégie d'augmentation de rendement

Les stratégies d'augmentation de rendement s'appuient sur des techniques de "reconfiguration" et de "configuration" d'une cible donnée. Les techniques de "reconfiguration" sont caractérisées par une cible de dimension figée et par la distinction entre éléments faisant partie d'une cible initiale et éléments de réserve. On appelle alors reconfiguration de la structure cible le remplacement d'éléments défaillants de la cible initiale par des éléments de réserve. Les

formules de rendement sont alors celles habituellement utilisées pour exprimer la fiabilité des architectures redondantes à réserve ("standby redundancy" ...).

Les techniques de "configuration" consistent à considérer tous les éléments implantés sur la tranche de façon indifférenciée, à identifier ceux qui sont opérationnels, puis à construire à partir d'un algorithme dit de "configuration" la plus grande cible possible (ici une matrice 2D). Le calcul du rendement doit tenir compte de la *moisson* de l'algorithme de configuration. Cette moisson ou "harvest", correspond au rapport entre le nombre d'éléments utilisés et le nombre d'éléments valides, elle indique que tous les éléments bons ne peuvent pas être utilisés.

Dans ELSA, deux stratégies ont été utilisées de façon hiérarchique. Au niveau de la sous-matrice, une technique de reconfiguration utilise une colonne de réserve pour remplacer les éléments défaillants d'une sous-matrice initiale. Au niveau de la tranche, une technique de configuration crée une cible finale de taille maximale à partir des sous-matrices opérationnelles.

Deux types d'évaluation de rendement ont donc été utilisés à ces deux niveaux. Le travail délicat a concerné le bon dimensionnement des sous-matrices en prenant en compte un partitionnement topologique supplémentaire du "réticule" destiné à permettre une photorépétition au cours de la fabrication.

II.2.2.1 Rendement des systèmes à cible figée avec redondance.

Nous allons évaluer le rendement des systèmes à cible figée en supposant une indépendance des défauts, pour cela nous allons utiliser le modèle de Poisson. Une évaluation tenant compte de la dépendance des défauts est très complexe [KoSt88], on se satisfera du modèle de Poisson car il donne des valeurs pessimistes du rendement.

Soit une architecture cible de k éléments et supposons que p éléments aient été implantés. Le rendement du circuit est alors la probabilité d'obtenir au-moins k éléments valides, ce rendement s'écrit :

$$R = \sum_{i=0}^{i=p-k} C_i^p \, R_e^{p-i} (1-R_e)^i$$

où Re est le rendement d'un élément et C_i^p est le nombre de combinaisons de i éléments parmi p. Notons Rsr = $(Re)^p$ et Cr = $\sum_{i=0}^{i=k} C_i^p \left(\frac{1}{Re} - 1\right)^i$, Rsr correspond au rendement sans redondance et Cr est un coefficient de redondance ≥ 0. Le rendement avec redondance s'écrit :

$$R = Rsr\ Cr$$

La figure 2.5 représente le rendement d'une architecture cible en fonction de la redondance, ces données sont obtenues pour une architecture cible de 5 éléments. Le rendement d'un élément est de 0,74, il correspond au rendement de Poisson d'un élément de surface S = $1cm^2$ avec une densité de défaut D_m = 0,3 défaut/cm^2. On remarque que l'ajout d'éléments redondants améliore toujours le rendement et qu'il existe un seuil à partir duquel l'augmentation n'est plus significative. Notons que pour chaque élément redondant ajouté correspond une augmentation de surface.

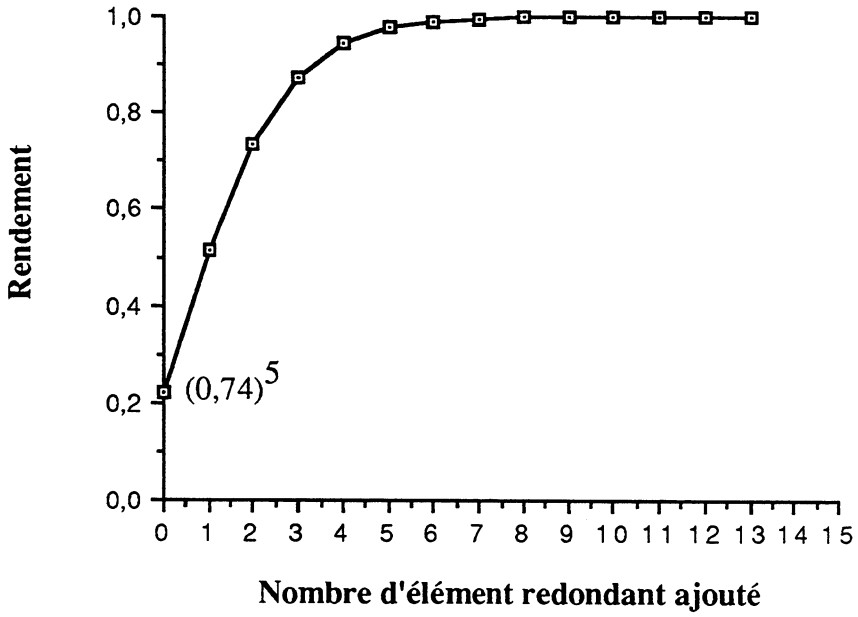


Figure 2.5. Rendement en fonction du nombre d'éléments redondants ajoutés.

II.2.2.2 Rendement des systèmes à moisson maximale.

La configuration d'un circuit est réalisée à l'aide d'un réseau de commutateurs. Dans certain cas, un commutateur défectueux peut entraîner l'échec de la configuration, il faut alors tenir compte du rendement de ce réseau.

Soit p le nombre d'éléments implantés sur la tranche, k la taille de l'architecture cible et q le nombre de commutateurs nécessaire à créer la cible. Notons Rc le rendement d'un commutateur et Re le rendement d'un élément. De la même manière que précédemment, le rendement de cette cible s'écrit :

$$R = (Rc)^q \sum_{i=k}^{i=p} C_i^p \, Re^i (1-Re)^{p-i}$$

Notons que le nombre de commutateurs nécessaire à créer une cible dépend de la taille de la cible.

La configuration dépend aussi de la performance de l'algorithme construisant l'architecture cible. Suivant l'algorithme utilisé, la moisson des éléments valides varie. La moisson de l'algorithme dépend de la taille de la cible, de la taille de l'architecture hôte, de la répartition des éléments défectueux, du nombre de commutateurs défectueux et de leur répartition. Déterminer de manière théorique le nombre de commutateurs nécessaire à créer une cible et la moisson de l'algorithme est un problème statistique très complexe [Lass90], nous les déterminerons par la suite de manière empirique. Pour pouvoir réaliser un cible de k éléments avec p éléments implantés et une moisson de h(p, k, ...), il faut k' éléments valides avec k'>k. En approximation, on peut dire que k'= k/h(p, k, ...).

Soit p le nombre d'éléments implantés sur la tranche, k la taille de l'architecture cible, q le nombre de commutateurs nécessaire à créer la cible et h(p, k, ...) la moisson de l'algorithme. Une approximation du rendement de la cible s'écrit :

$$R = (Rc)^q \sum_{i=k'}^{i=p} C_i^p \, Re^i (1-Re)^{p-i}$$

où Re et Rc sont respectivement le rendement d'un élément et d'un commutateur.

La figure 2.6 donne le rendement en fonction de la taille de la matrice cible pour différentes valeurs de la moisson. Ces données sont obtenues pour une matrice hôte de 49 éléments avec un réseau double rail, le tableau 2.1 donne les valeurs de p, Re, q et de $(Rc)^q$. Dans cet exemple, en première approximation nous supposons que 50% des commutateurs sont utiles pour la

configuration quelle que soit la taille de la cible et que la moisson de l'algorithme est une constante.

Taille de la matrice hôte (p)	Rendement d'un élément (Re)	nombre de commutateurs (q)	Rendement des commutateurs (Rc)^q
7x7	0,76	196	0,86

Tableau 2.1

On s'aperçoit que le rendement est majoré par la valeur 0.86 correspondant à (Rc)^q et qu'il existe une cible, appelé cible maximale, à partir duquel le rendement chute. La diminution de la moisson a comme effet de décaler la courbe vers la gauche, cela correspond à une diminution du rendement. Le tableau 2.2 donne la cible maximale en fonction de la moisson.

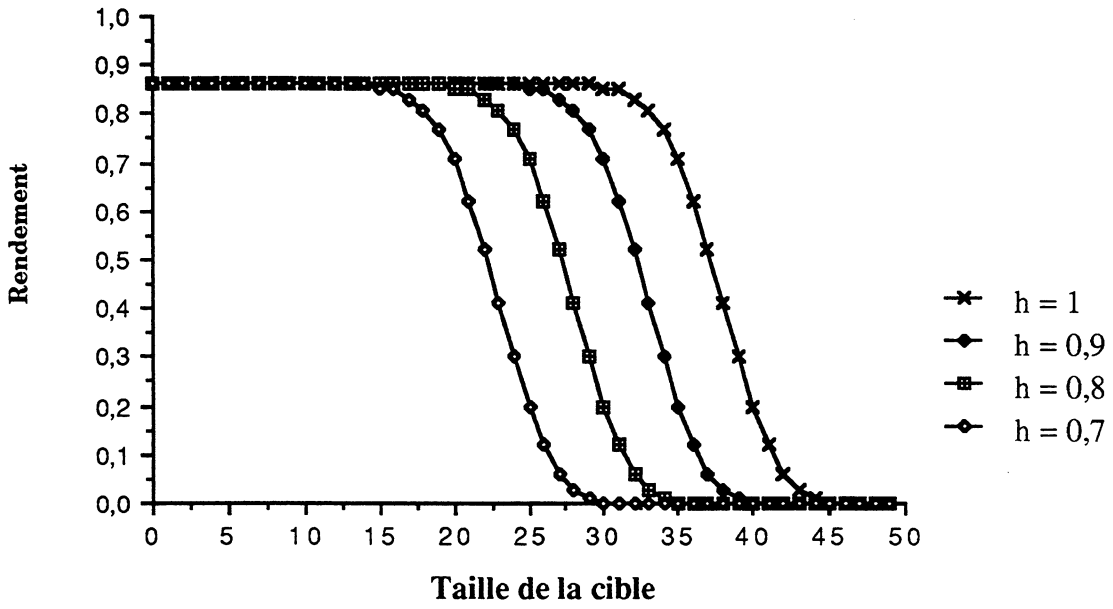


Figure 2.6. Le rendement en fonction de la taille de la cible.

	h = 1	h = 0,9	h = 0,8	h = 0,7
Taille de la cible maximale	29	24	19	14

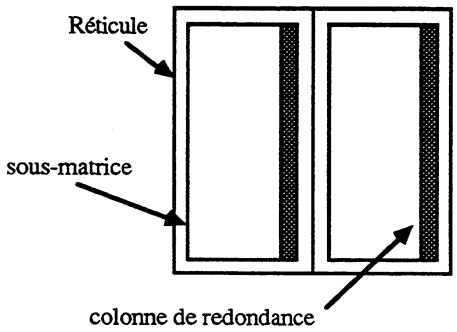
Tableau 2.2.

II.2.3 Rendement d'ELSA

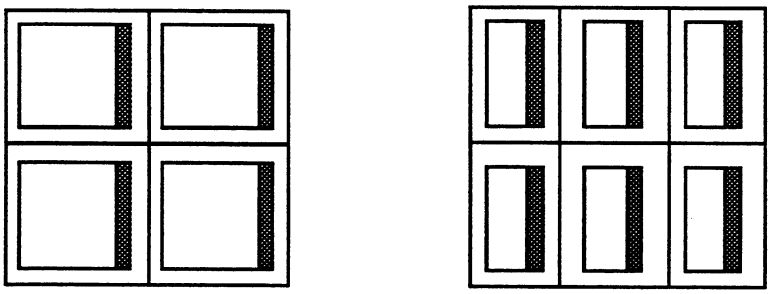
Nous allons utiliser le modèle pessimiste du paragraphe précédent afin d'évaluer le rendement d'ELSA.

Le rendement d'un PE est très élevé R = 0,99 pour D_m = 0,3 défaut/cm^2, il n'est donc pas nécessaire d'introduire de la redondance dans le PE. La tolérance aux défauts est alors à deux niveaux : au niveau sous-matrice (les PEs sont regroupés en sous-matrice) et au niveau tranche.

A partir des surfaces du PE (0,361 mm^2), du commutateur (0,256 mm^2) et du réticule (14 x 16 mm^2), nous pouvons évaluer les différentes partitions; c'est-à-dire le nombre de PEs dans une sous-matrice, le nombre de lignes redondantes dans une sous-matrice et le nombre de sous-matrices sur la tranche. Le tableau 2.3 et la figure 2.7 représente trois partitions étudiées au niveau de ce projet.



Partition I



Partition II Partition III

Figure 2.7. Différentes partitions du réticule.

	Taille de la sous-matrice	Nombre de sous-matrice dans le réticule	Nombre de sous-matrice sur la tranche	Nombre de PE sur la tranche
Partition I	6 x 24 + 1 colonne de redondance	2	40	6720
Partition II	6 x 12 + 1 colonne de redondance	4	80	6720
Partition III	3 x 12 + 1 colonne de redondance	6	120	5760

Tableau 2.3

II.2.3.1 Rendement des sous-matrices.

Les techniques de reconfiguration au niveau de la sous-matrice utilisent des colonnes de redondance; elles permettent de reconfigurer plus d'un PE par ligne.

Soit n le nombre de PE par ligne, le rendement d'une ligne avec 1 PE redondant s'écrit :

$$R_{ligne} = C_{n-1}^{n} R_{pe}^{n-1}(1-R_{pe}) + R_{pe}^{n}$$

$$R_{ligne} = n R_{pe}^{n} (1/R_{pe} -1) + R_{pe}^{n}$$

D'une façon générale le rendement d'une ligne avec k PE redondant s'écrit :

$$R = \sum_{i=0}^{i=k} C_{i}^{n} R_{e}^{n-i}(1-R_{e})^{i}$$

Soit m le nombre de colonnes de la sous-matrice, le rendement de la sous-matrice est :

$$R_{sous\text{-}matrice} = (R_{ligne})^{m}$$

Dans ELSA, une sous-matrice non défectueuse peut être utilisée si ses quatres commutateurs périphériques ne sont pas défectueux (cf. figure 2.3). Le rendement de la sous-matrice devient alors : $R_{sous\text{-}matrice} = (R_{ligne})^{m} (R_{commutateur})^{4}$.

Le tableau 2.4 représente le rendement d'une ligne et de la sous-matrice en fonction du nombre de colonne redondante pour chaque partition, ces rendements sont obtenus pour une valeur de $D_m = 0.3$ défaut/cm^2. Ces calculs montrent qu'il n'est pas nécessaire de mettre plus d'une colonne de redondance dans une sous-matrice car l'augmentation de rendement n'est plus significative.

	sans redondance	1 colonne de redondance	2 colonnes de redondance
Partition I	R_{ligne}=0,932 $R_{sous-matrice}$=0,19	R_{ligne}=0,998 $R_{sous-matrice}$=0,95	R_{ligne}=1 $R_{sous-matrice}$=0,99
Partition II	R_{ligne}=0,932 $R_{sous-matrice}$=0,42	R_{ligne}=0,998 $R_{sous-matrice}$=0,97	R_{ligne}=1 $R_{sous-matrice}$=0,99
Partition III	R_{ligne}=0,961 $R_{sous-matrice}$=0,62	R_{ligne}=0,999 $R_{sous-matrice}$=0,98	R_{ligne}=1 $R_{sous-matrice}$=0,99

Tableau 2.4

II.2.3.2 Rendement de la tranche

L'expression du rendement d'ELSA s'obtient par la relation suivante (cf II.2.2.2) :

$$R = (R_{sous-matrice})^p (R_c)^q \left(\sum_{i=0}^{i=kh} C_i^p (1/R_{sous-matrice-1})^i \right)$$

où : p = nombre de sous-matrices sur la tranche.

q = nombre de commutateur utile pour la reconfiguration.

h = moisson de l'algorithme de configuration.

k = nombre de sous-matrices redondantes sur la tranche.

Avant de calculer le rendement, nous devons évaluer le nombre de commutateurs utiles et la moisson de l'algorithme de configuration. Il est difficile de connaître les valeurs de q(p, k, ...) et de h(p, k, ...), mais en première approximation nous allons prendre une valeur moyenne obtenue après de nombreux tests. Nous supposons qu'en moyenne 50% des commutateurs sont utiles à la configuration et que la moisson de l'algorithme

en moyenne est de l'ordre de 0,75. Le tableau 2.5 donne le nombre de commutateurs utiles suivant la partition.

	nombre de commutateur sur la tranche	% des commutateurs utiles	nombre de commutateurs utiles	Rendement des commutateurs utiles
Partition I	160	50%	80	0,92
Partition II	320	50%	160	0,85
Partition III	480	50%	240	0,78

Tableau 2.5

La figure 2.8 représente le rendement d'ELSA en fonction de la taille de la matrice cible pour les trois partitions. Ce rendement est obtenu en prenant 50% des commutateurs utiles et h = 0,75.

On remarque que le rendement d'une tranche d'ELSA est majoré par la valeur 0,92. Cette valeur correspond au rendement des commutateurs utiles. En effet, dans le meilleur des cas, si on suppose que $R_{sous\text{-}matrice} = 1$, alors le rendement d'ELSA devient : $R = (R_{commutateur})^q$. Le tableau 2.6 donne la valeur maximum du rendement d'ELSA pour chaque partition. On remarque aussi que le rendement chute à partir de la cible maximale (cf. II.2.2). Le tableau 2.7 donne la taille de la matrice maximale en fonction de la partition du système. Cette évaluation du rendement montre que les partitions I et II permettent de configurer les plus grandes cibles. La partition II permet de gagner 13% en taille de cible par rapport à la partition I en ayant une diminution de rendement de 7%. Il est plus souhaitable de choisir la partition II pour ELSA, car elle nous permet de configurer de plus grande cible en ayant une diminution de rendement peu significative.

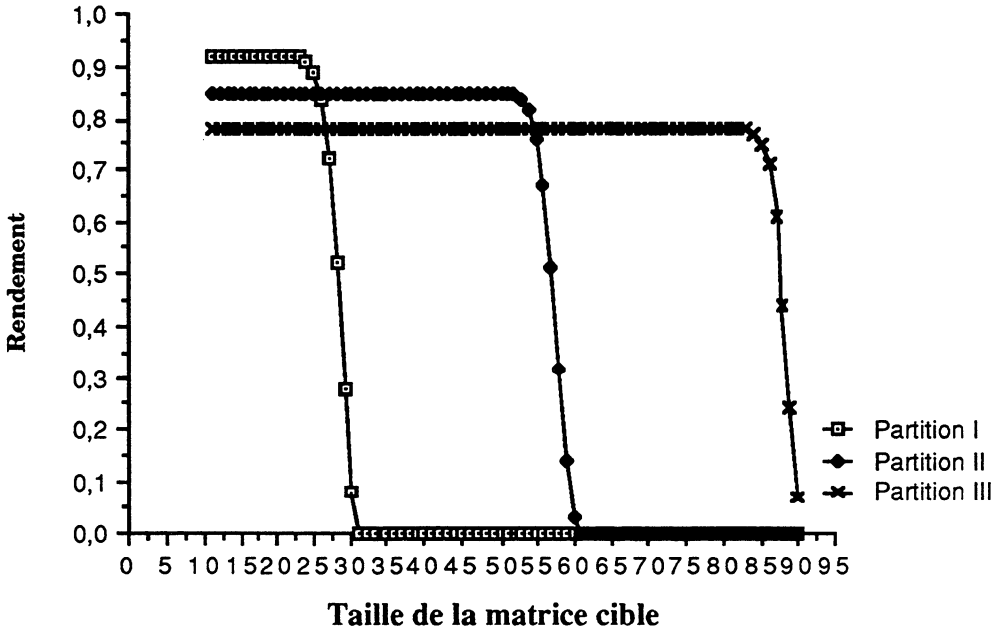


Figure 2.8. Rendement d'ELSA en fonction de la taille de la matrice cible.

Partition I	RELSA ≤ 0,92
Partition II	RELSA ≤ 0,85
Partition III	RELSA ≤ 0,78

Tableau 2.6

	Cible maximale en sous-matrice	Cible maximale en PE
Partition I	23	3312
Partition II	52	3744
Partition III	83	2988

Tableau 2.7

II.3 Implantation physique de la tolérance aux défauts

II.3.1 La reconfiguration au niveau de la sous-matrice

La reconfiguration est possible à ce niveau par l'ajout de colonnes supplémentaires de PEs. L'évaluation du rendement d'ELSA à montrer qu'il suffisait d'introduire une colonne redondante pour tolérer les PEs défectueux.

Des moyens de contournement ont donc été implémentés pour remplacer les PEs défectueux par les PEs redondants. Un dispositif de programmation à faisceau d'électron et des portes de transmission ont été utilisés. Ce dispositif permet la reconfiguration d'une ligne de PE. Chaque ligne contient un PE redondant et en programmant le dispositif par un faisceau d'électron ou par un laser, le PE défectueux est contourné et le PE redondant est inclus dans la sous-matrice. L'avantage de cette méthode est le peu de surface utilisé pour la reconfiguration de la sous-matrice.

La programmation du dispositif peut être réversible ou non. La programmation électriquement réversible est utilisé pour reconfigurer la sous-matrice pendant la phase du test, tandis que la programmation physique est utilisée pour figer définitivement la configuration. Lors d'un auto-diagnostic du PE, le résultat est stocké dans une bascule. Celle-ci commande les portes de transfert des circuits de contournement. La figure 2.9 représente l'ensemble du dispositif de programmation. Le signal SET permet d'exécuter l'auto-diagnostic du PE. Celui-ci est réalisé à partir des sorties des bascules "NS" et "EW" du PE. Le résultat de ce test est stocké dans la bascule contrôlant des portes de transfert. La valeur de cette bascule peut être forcée en utilisant un fusible laser ou un transistor à grille flottante.

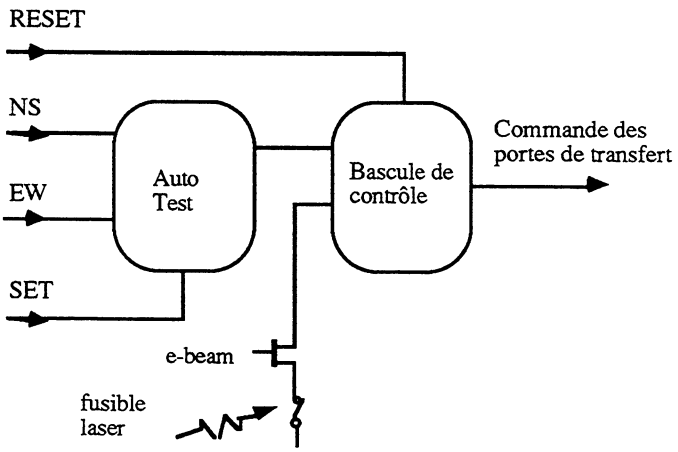


Figure 2.9. Dispositif de reconfiguration au niveau de la sous-matrice.

Le réseau de reconfiguration associé est représenté sur la figure 2.10. Ce réseau permet uniquement le contournement d'un processeur par ligne.

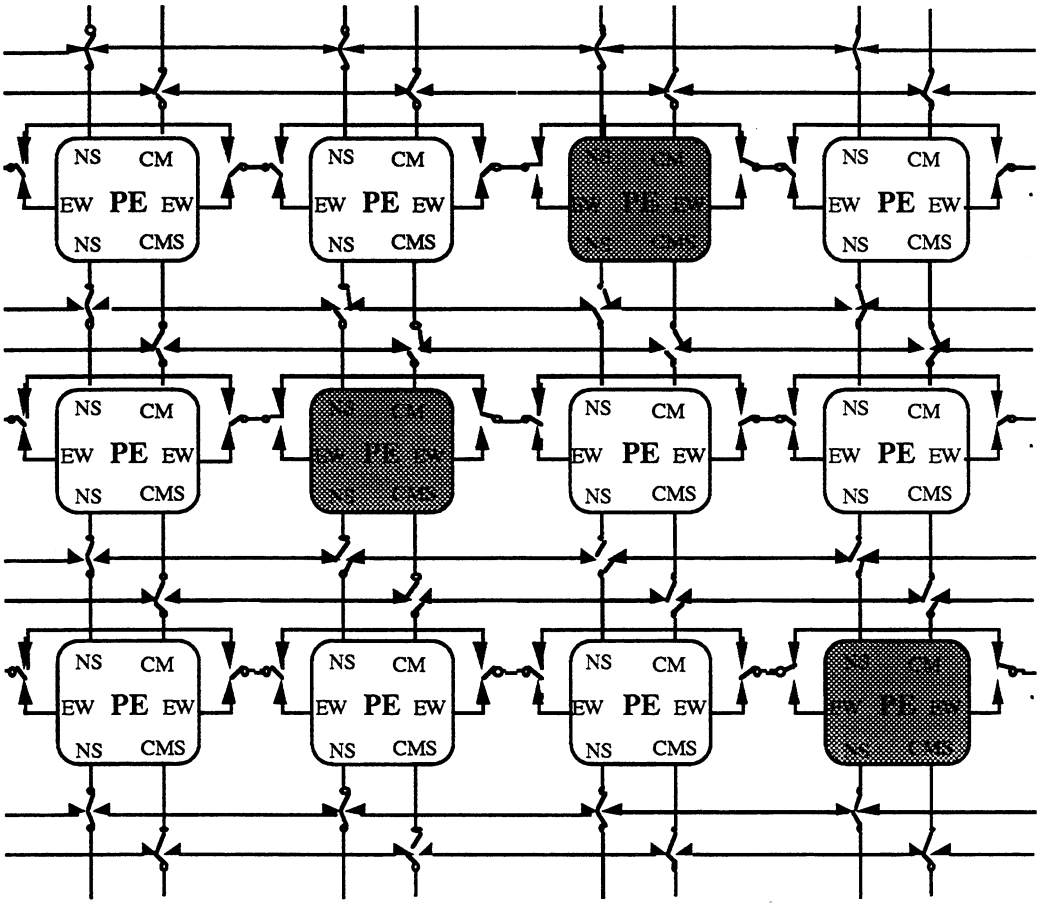


Figure 2.10. Schéma de la sous-matrice.

II.3.2 La configuration au niveau de la tranche

Les commandes globales et l'alimentation du circuit sont des ressources vitales. Si elles sont défectueuses, la tranche ne peut pas être utilisé. Au niveau de la tranche, une tolérance aux défauts du réseau des commandes globales (incluant l'horloge), du réseau d'alimentation et des sous-matrices doit être implantée.

a) Le réseau de commande global

Les commandes globales contrôlent tous les PEs sur toute la tranche. La solution choisie pour tolérer les défauts est d'intégrer le réseau comme une grille. La figure 2.11 illustre cette structure.

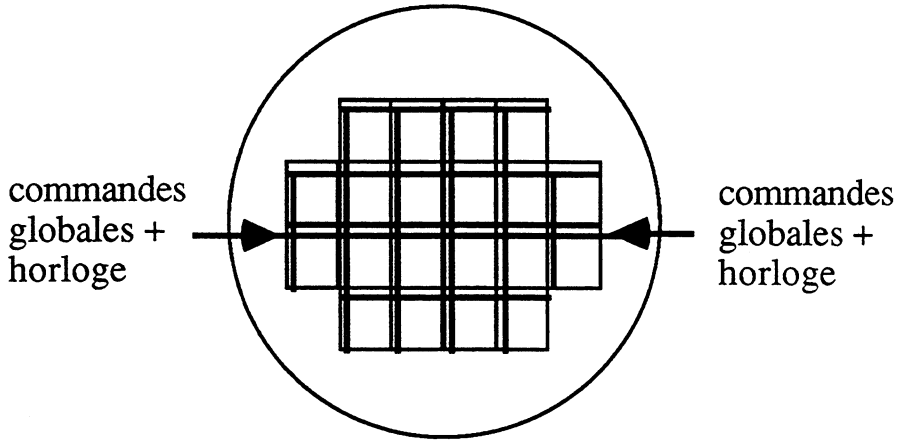


Figure 2.11. Le réseau de commandes globales.

Dû à sa structure hautement connexe, le réseau n'est pas sensible aux coupures. Afin de réduire la résistance et la capacité de ces lignes, le réseau est intégré en utilisant l'aluminium de la couche supérieur (résistance = 50 mΩ/carré).

b) Le réseau d'alimentation

Il est primordial que l'alimentation du circuit tolère les défauts notamment les court-circuits. Pour cela, en général, la méthode utilisée est de contrôler la distribution de l'alimentation dans les différents blocs par des commutateurs [Naaa89] [Rvls89]. Les commutateurs permettent la connexion de l'alimentation d'un bloc aux bus d'alimentation. Si un bloc est défaillant et provoque un court-circuit, il sera déconnecté du bus d'alimentation.

Comme, les bus d'alimentation sont des connexions en métal2 très large (de l'ordre de 25µ), ces bus sont robustes vis à vis coupures, il n'est donc pas nécessaire de prévoir de les configurer.

La figure 2.12 représente la distribution de l'alimentation d'ELSA, un bus d'alimentation est connecté à 1 ou 2 réticules. Dans ce circuit, aucun commutateur permettant de contrôler la distribution de l'alimentation est implanté. La distribution de l'alimentation n'est donc pas tolérante aux défauts. S'il existe un court-circuit dans un bloc défaillant, le réticule contenant ce bloc est perdu ainsi que le réticule connecté au même bus d'alimentation.

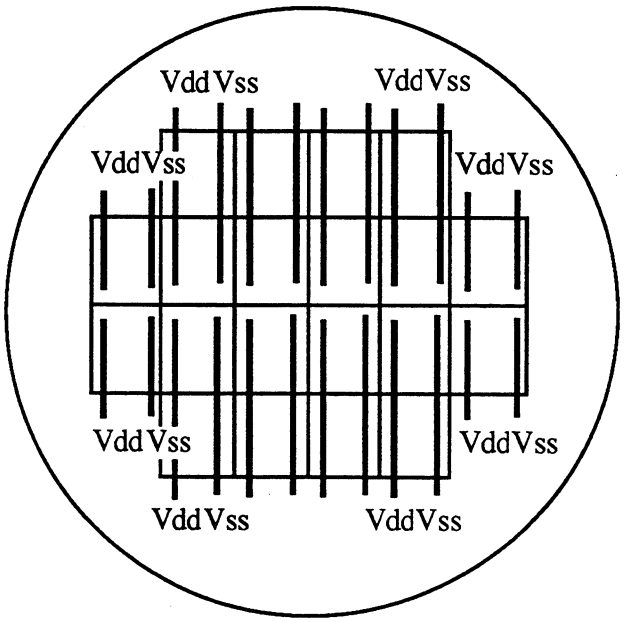


Figure 2.12. La distribution de l'alimentation dans ELSA.

c) Le réseau de configuration.

Au niveau de la tranche, chaque sous-matrice est testée individuellement selon une stratégie d'auto-test (cf II.6). A partir des ressources valides, il faut constituer sur la tranche une matrice 2D opérationnelle. Le rôle du réseau de configuration est d'établir les connexions entre les sous-matrices valides afin d'obtenir cette matrice 2D.

Trois structures de réseau ont été étudiées, à savoir le réseau double-rail, triple-rail et le réseau en étoile (figure 2.13). Les réseaux double-rail et triple-rail comportent le même commutateur à quatre directions. Le réseau en étoile ne comporte pas de commutateur, les signaux sont dirigés par des portes de transfert. La figure 2.13 (b) montre un exemple d'utilisation de ce réseau, une seule connexion logique peut traverser les interconnexions. Ce réseau se rapproche d'un réseau simple rail.

Les critères de sélection entre ces réseaux sont la tolérance à des exemples de cartographie de défauts, la surface et les performances temporelles.

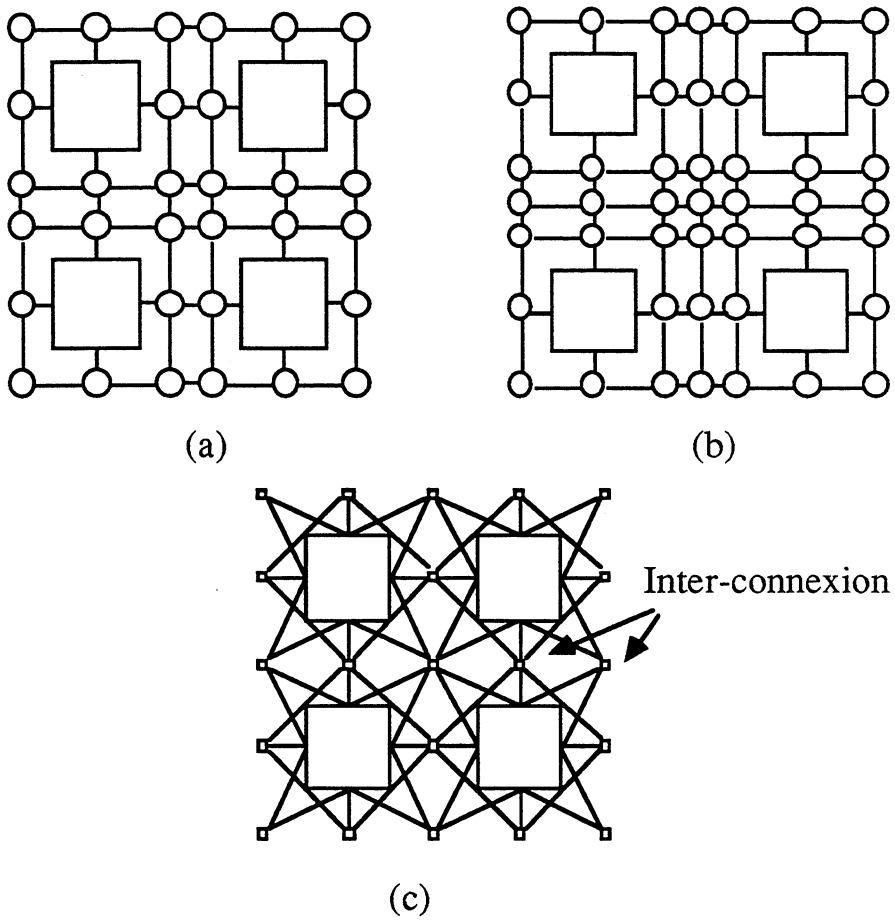


Figure 2.13. (a) Le réseau double-rail. (b) Le réseau triple-rail. (c) Le réseau en étoile.

Critère de tolérance à des cartographie de défauts :

De nombreuses cartographies de défaut ont été étudié, elles correspondent à une distribution uniforme de défaut et à des amas de défaut. En général, une distribution uniforme engendre une déformation de la matrice, par contre un amas engendre des croisements ou des chevauchements de connexion.

Les figures 2.14 (a) et (b) représentent un exemple de distribution uniforme de défaut. Les figures (a) et (b) représentent respectivement la configuration d'une matrice à l'aide du réseau double-rail et du réseau en étoile. (la configuration à l'aide du réseau triple-rail est équivalente à celle du réseau double-rail). On remarque pour une simple déformation de la matrice que les trois réseaux sont équivalents.

Les figures (c), (d) et (e) représentent respectivement la configuration d'une matrice à l'aide du réseau double rail, du réseau en étoile et du réseau triple rail pour le même amas de défaut. On s'aperçoit que le réseau double-

rail et triple-rail supportent le chevauchement des lignes 2 et 3 de la matrice, alors que le réseau en étoile ne le supporte pas. En effet, avec le réseau en étoile il est impossible de réaliser complètement la ligne 3 et la colonne 2 de la matrice. De manière générale, le réseau en étoile n'est pas souhaitable car il ne permet pas la configuration pour des amas de défaut.

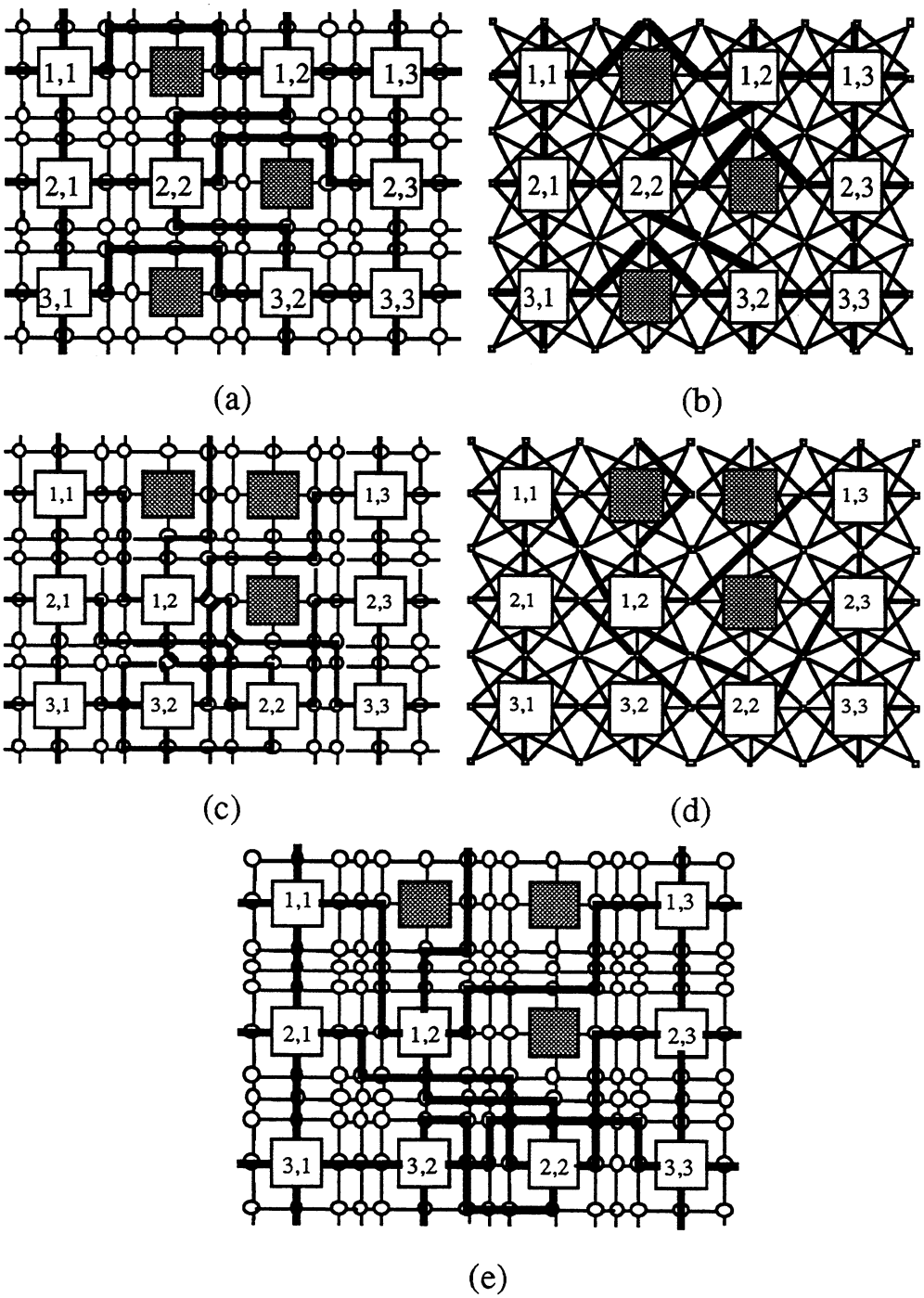


Figure 2.14. (a) et (b) configuration avec le réseau double-rail et le réseau en étoile pour une distribution uniforme de défaut. (c), (d) et (e) configuration

avec le réseau double-rail, le réseau en étoile et le réseau triple-rail pour un amas de défaut.

Critère de surface :

Il est évident que le réseau triple-rail à une surface plus importante que le réseau double-rail. Il reste à évaluer la surface du réseau en étoile par rapport aux autres réseaux.

La surface des réseaux dépend de la surface des commutateurs et de la connectique. Nous rappelons que les connexions sont des bus. A partir du plan de masse de la connectique du réseau en étoile (figure 2.15), on en déduit que la surface de sa connectique est au-moins égal à celle du réseau triple-rail. La surface réservée à la logique de contrôle (commutateur, porte de transfert) peut être négligée car une conception judicieuse positionne les commutateurs sous les bus de connexion.

On peut dire que le réseau en étoile a une surface aussi importante que le réseau triple-rail.

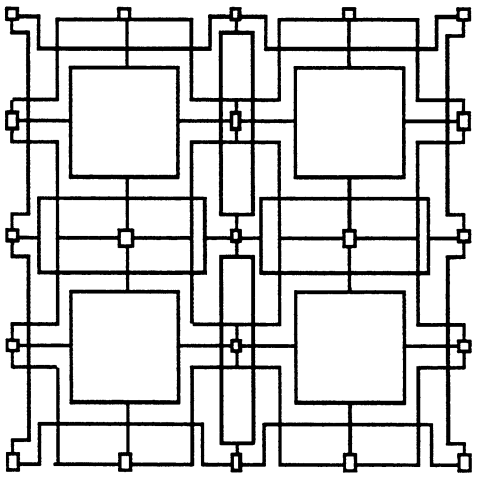


Figure 2.15. Plan de masse du réseau en étoile.

Critère de performance temporelle:

La vitesse d'un réseau dépend de la longueur de la connectique et du nombre de porte de transfert que doit traverser le signal. Il est évident que le réseau double-rail et plus rapide que le réseau triple-rail. Une comparaison entre le réseau en étoile et le réseau triple-rail montre que le nombre de porte de transfert traversé par le signal est toujours inférieur pour le réseau en

étoile (2 et 4 au lieu de 7 et 11), et de plus ces deux réseaux ont une connectique équivalente. On peut conclure que le réseau triple-rail est le plus lent parmi ces trois réseaux.

Les critères de sélection les plus importants dans ce projet sont la tolérance aux défauts et la surface. Nous avons vu que seule les réseaux double et triple rail tolèrent les amas de défauts. Le réseau triple rail tolère de plus gros amas que le réseau double-rail, mais au prix d'une surface de reconfiguration plus importante. Il est préférable d'utiliser le réseau double-rail car il est peu probable d'avoir de gros amas de défauts (cf. II.2.3).

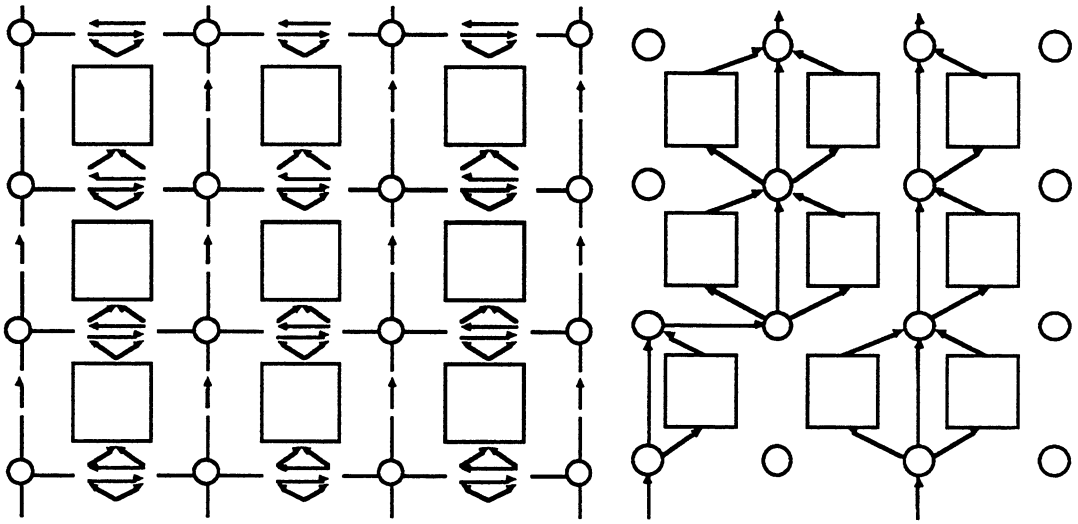
II.3.3 Les commutateurs programmables

Le réseau de configuration est composé de commutateurs. Dans ce projet, les commutateurs envisageables peuvent être réversibles ou non. Nous avons comme objectif d'utiliser dans un premier temps ce réseau à des fins de test. Les fusibles et les antifusibles ne peuvent pas être utilisés seuls, parce qu'ils donnent une configuration permanente. En effet, le taux de couverture du test n'étant jamais de 100%, il serait dangereux d'avoir comme seule possibilité la configuration définitive de la matrice.

L'utilisation de grilles flottantes développées dans ce projet ne s'est pas avéré pratique parce qu'un environnement "e-beam" sophistiqué est indispensable, et elles manquent de fiabilité. En conséquence, le choix d'une logique programmable comme commutateur semble donc la plus appropriée.

Un commutateur programmable à été développé par Blatt et Katevenis [Blatt 90] [Blatt 89] pour la reconfiguration d'une mémoire WSI. La figure 2.16 (a) représente le réseau de commutateurs, ces commutateurs contrôlent les connexions entre les mémoires. La figure 2.16 (b) montre un exemple de configuration à l'aide des commutateurs.

La programmation d'un commutateur est séquentielle. Tout d'abord, le commutateur à programmer est sélectionné, puis ces données de configuration lui sont envoyés. La sélection des commutateurs est faite par adresse. Chaque commutateur du réseau a une adresse différente, l'adresse du commutateur à programmer est propagée de commutateurs en commutateurs sur tout le réseau. Puis de même, les données de configuration sont propagées de commutateurs en commutateurs. Des possibilités de contournement sont intégrées pour tolérer les commutateurs défectueux.



(a) (b)

Figure 2.16. (a) Le réseau de commutateur contrôlant les connexions entre les mémoires. (b) Un exemple de reconfiguration à l'aide des commutateurs.

Citons une autre proposition permettant la configuration d'une matrice WSI. Les auteurs Kim et Reddy [KiRe89], proposent la modification de l'architecture du processeur élémentaire pour permettre la configuration. L'idée de base est de contourner un processeur lorsque celui-ci est défectueux. L'architecture proposée permet de contourner un processeur si au plus il existe un processeur défectueux par ligne. La figure 2.17 (a) montre l'architecture générale d'un processeur et la figure 2.17 (b) donne l'architecture générale du processeur modifiée permettant la reconfiguration.

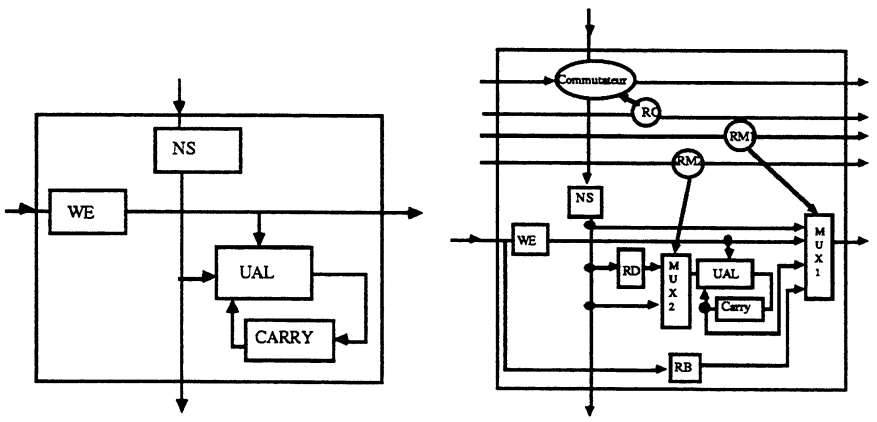


Figure 2.17. (a) L'architecture générale d'un processeur élémentaire. (b) L'architecture du processeur permettant la configuration.

Le processeur contient un commutateur à trois configurations (figure 2.18), des registres de contrôle (RC, RM1, RM2), un registre de contournement (RB), un registre pour resynchroniser les données (RD) et des multiplexeurs (MUX1, MUX2). Le multiplexeur "MUX1" permet de sélectionner soit l'utilisation normale du processeur, le contournement du processeur, le décalage Ouest-Est du registre "NS" ou permet un test. Le multiplexeur "MUX2" introduit la resynchronisation des données.

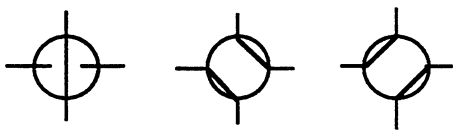


Figure 2.18. Les trois configurations du commutateur.

L'exemple ci-dessous (figure 2.19) montre la reconfiguration d'une matrice logique (3x3) sur une matrice hôte défectueuse (3x4) à l'aide de ce processeur.

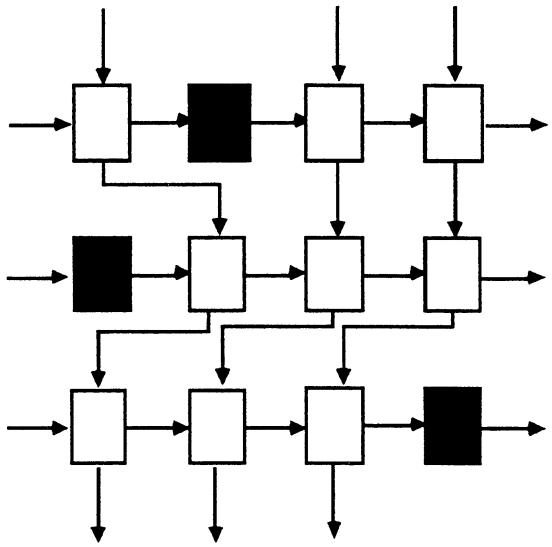


Figure 2.19. Reconfiguration d'une matrice logique (3 x3).

Les auteurs proposent d'étendre cette architecture en contournant au plus *p* processeurs défectueux par ligne. Cette approche n'est pas très bonne car la reconfiguration basée sur le contournement des processeurs défectueux n'est pas la meilleure approche (cf. III.1), et d'autre part le contrôle de la reconfiguration n'est pas tolérant aux défauts. En effet, la reconfiguration est déterminé par les trois registres de contrôle. Le chargement de ces registres s'obtient par décalages successifs, s'il existe un registre défectueux alors la reconfiguration de la ligne échoue.

II.4 Le réseau de commutateurs

II.4.1 Le commutateur de base

Le réseau de configuration retenu est un réseau double-rail de commutateurs. Celui-ci permet de connecter les sous-matrices valides entre elles. Les connexions entre les sous-matrices peuvent être directes; dans ce cas une sous-matrice est directement reliée à sa voisine. Elles peuvent être complexes; la connexion doit contourner un certain nombre de sous-matrices voire de commutateurs défectueux. L'élément de base du réseau est un commutateur à quatre directions (Nord, Sud, Est, Ouest) avec quatre configurations possibles données en figure. 2.20. A chaque configuration est associé un couple de valeurs (D0, D1) qui permet de choisir l'une des configurations.

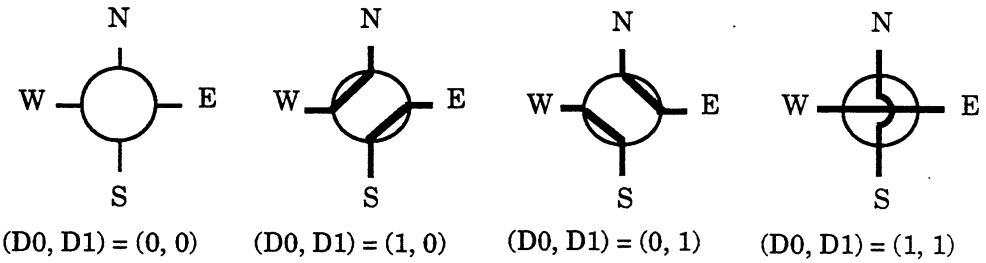


Figure. 2.20. Les quatre configurations du commutateur.

Le commutateur de base ou le bloc de TRANSFERT est un ensemble de portes de transfert réalisant les connexions entre les quatre directions Nord, Sud, Est, Ouest. Cette partie est programmable par les deux variables (D0, D1) à travers un décodeur (figure. 2.21).

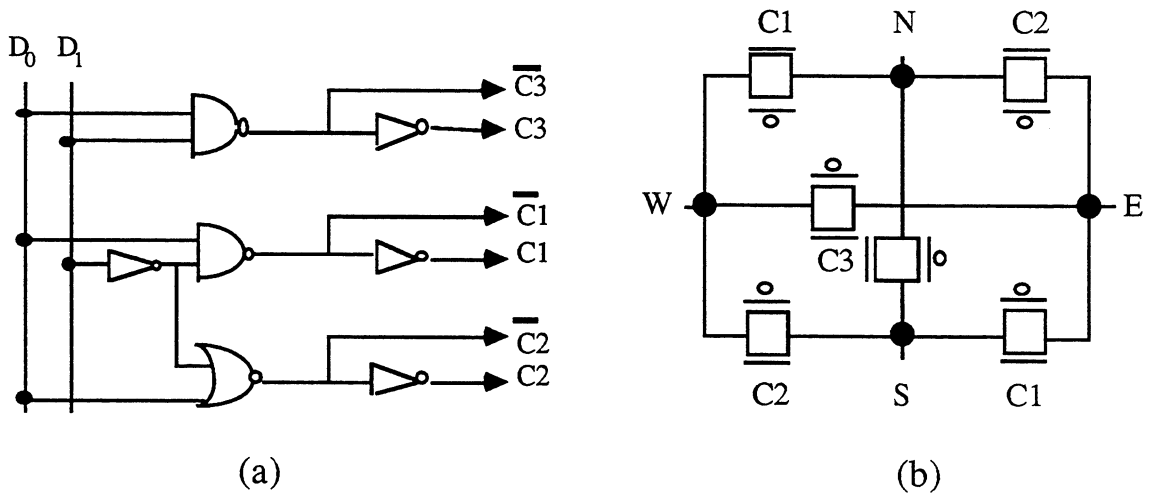


Figure 2.21.(a) Le contrôle de la partie TRANSFERT.(b) La partie TRANSFERT.

II.4 2 La programmation du commutateur

La matrice étant intégrée sur tranche entière, le réseau contient de l'ordre de 600 commutateurs dont une partie sera défectueuse.Il faut alors concevoir un réseau dont la programmation est tolérante aux défauts. Nous choisissons des commutateurs programmables à partir de la périphérie; c'est-à-dire seuls les commutateurs périphériques sont connectés à des plots d'entrée/sortie. Un commutateur interne au réseau, c'est-à-dire non périphérique, se programmera à travers d'autres commutateurs. Il faut alors créer un chemin de la périphérie au commutateur interne. Ce chemin sera appelé chemin de programmation du commutateur.

Pour acheminer les données de programmation du commutateur, ce chemin peut être vu comme un registre à décalage global ayant comme point d'entrée un commutateur périphérique et comme point de sortie le commutateur interne à programmer. Le commutateur doit contenir un registre à décalage local constituant une partie du registre à décalage global permettant l'acheminement de ces données. Ces données de programmation doivent pouvoir être stockées au-niveau de chaque commutateur dans des bascules pour maintenir la programmation. Nous avons vu précédemment que deux valeurs sont nécessaires pour déterminer la configuration du commutateur. Il suffit alors que le commutateur contienne deux bascules et un registre à décalage à deux bits. Ces deux bascules sont notées (D0, D1) et le registre à décalage est noté (R0, R1).

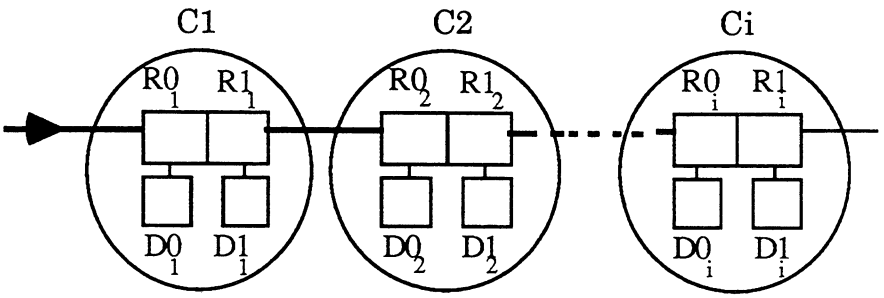


Figure 2.22. Chemin pour atteindre le commutateur i.

Supposons que nous voulons programmer le commutateur Ci à travers les commutateurs (C1, C2, ..., Ci-1) (figure. 2.22), C1 étant un commutateur périphérique. Les données nécessaires pour programmer ce chemin sont des séquences de bits obtenues par concaténation de couples d_i à partir des valeurs de la configuration de chaque commutateur ($d0_1$, $d1_1$), ($d0_2$, $d1_2$), ... , ($d0_i$, $d1_i$). Ces séquences de bits sont envoyées à travers les registres à

décalage successifs ($R0_1$, $R1_1$), ($R0_2$, $R1_2$), ... , ($R0_i$, $R1_i$), puis fixées définitivement dans les couples de bascules ($D0_1$, $D1_1$), ($D0_2$, $D1_2$), ... , ($D0_i$, $D1_i$). Le chemin est alors établie.

Nous allons illustrer plus précisément la programmation des commutateurs par l'exemple de la figure 2.23. La figure 2.23(a) représente les deux commutateurs à programmer et la figure 2.23(b) représente les bascules nécessaires.

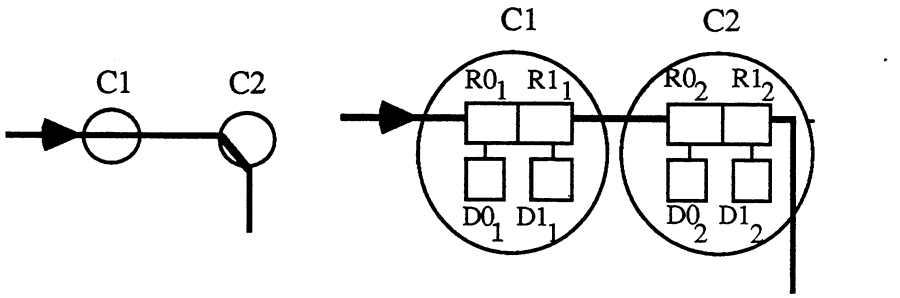


Figure 2.23 (a) Chemin a créer. (b) les registres et les bascules associés.

Afin d'atteindre C2, il faut configurer le commutateur C1 d'Ouest en Est, avec la configuration ($d0_1$, $d1_1$) = (1, 1). Dans un premier temps, ces deux valeurs (1, 1) sont envoyées en série dans ($R0_1$, $R1_1$), puis elles sont stockées dans ($D0_1$, $D1_1$). Cela correspond aux opérations ci-dessous:

$$R0_1\ R1_1 \leftarrow 11$$

$$D0_1\ D1_1 \leftarrow R0_1\ R1_1$$

Pour programmer le second commutateur, C2, dans la configuration ($D0_2$, $D1_2$) = (1, 0), nous avons besoin de traverser C1. Les valeurs (1, 0) sont envoyées à ($R0_1$, $R1_1$) et en même temps les anciennes valeurs de ($R0_1$, $R1_1$) sont décalées vers ($R0_2$, $R1_2$). Ensuite, les valeurs (1, 1) sont envoyées à ($R0_1$, $R1_1$) pour programmer C1, tandis que les valeurs ($R0_1$, $R1_1$) = (1, 0) est décalées vers ($R0_2$, $R1_2$). Les valeurs étant positionnées dans les deux registres à décalage, elles sont stockées dans les bascules pour programmer C2 et de nouveau C1. Cela correspond aux opérations suivantes:

instant 1	$R0_1\ R1_1 \leftarrow 10$	//	$R0_2\ R1_2 \leftarrow R0_1\ R1_1$
instant 2	$R0_1\ R1_1 \leftarrow 11$	//	$R0_2\ R1_2 \leftarrow R0_1\ R1_1$
instant 3	$D0_1\ D1_1 \leftarrow R0_1\ R1_1$	//	$D0_2\ D1_2 \leftarrow R0_2\ R1_2$

Deux signaux sont nécessaires pour effectuer ces opérations. Le premier sert à envoyer les données de programmation, ce signal est noté PROGRAM, et le second permet de valider le chargement de (R0, R1) dans (D0, D1), ce signal est nommé CTRL. Le chronogramme de la figure. 2.24 correspond à cette programmation. Le chargement des bascules est effectif sur le front descendant du signal CTRL.

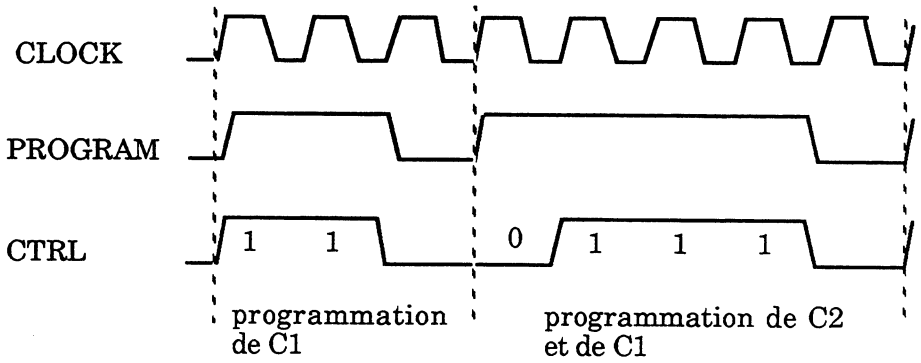


Figure 2.24. Chronogramme de la programmation de C1 et C2.

II.4.3 Architecture du commutateur

Nous avons vu que deux signaux CTRL et PROGRAM permettent la programmation du commutateur. Celui-ci est composé d'une partie contrôle et d'une partie commutation (figure 2.25). La partie commutation est constituée d'un ensemble de commutateurs de base qui permet d'acheminer dans la direction souhaité des données DATA de *N* bits. La partie contrôle permet de configurer cette partie commutation, autrement dit de choisir la direction d'acheminement des données en positionnant les commutateurs de base.

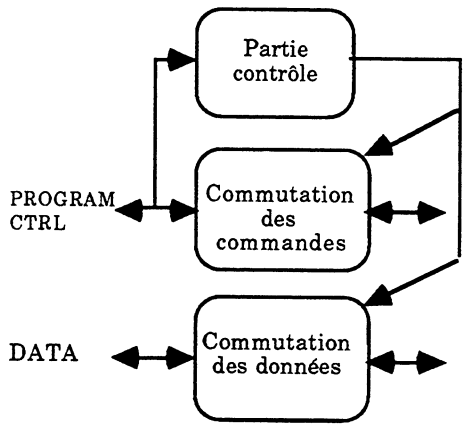


Figure 2.25. L'Architecture du commutateur.

Nous avons vu que le réseau peut être programmé uniquement par sa périphérie. Pour accéder aux signaux de contrôle CTRL et PROGRAM d'un

commutateur interne au réseau, il faut que ceux-ci se propagent de commutateur en commutateur le long du chemin de programmation. Dans l'exemple de la programmation des deux commutateurs de la figure 2.23, les signaux CTRL et PROGRAM de C2 sont activés à travers C1, c'est-à-dire le commutateur C1 achemine ces signaux d'Ouest en Est (figure 2.26).

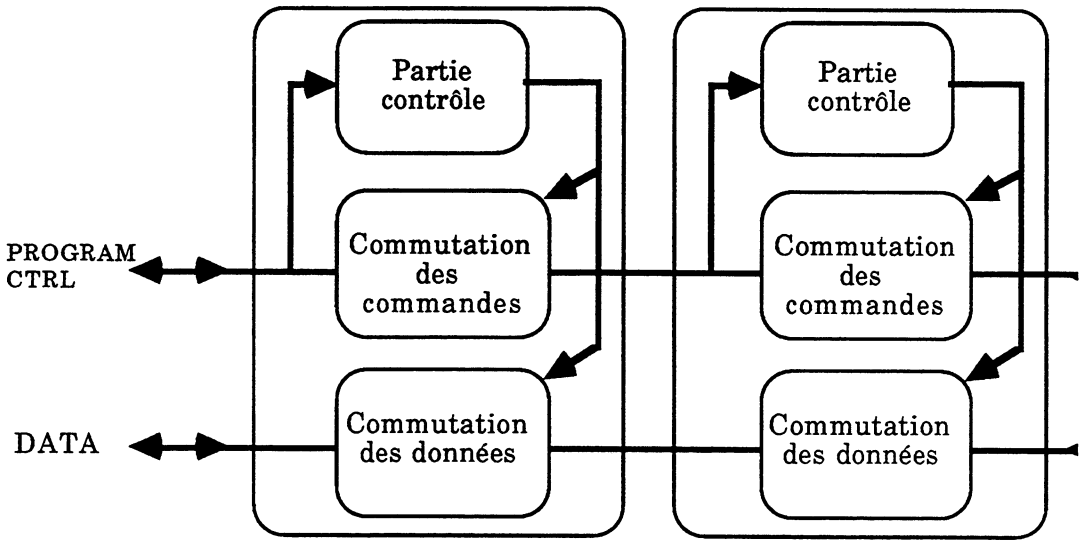


Figure 2.26. Les signaux de programmation sont propagés par la partie commutation.

La partie commutation achemine alors les données DATA ainsi que celle des signaux de programmation dans la bonne direction. La partie commutation doit contenir *N+2* blocs de TRANSFERT, *N* sont nécessaires pour diriger le bus de données DATA de *N* bits et 2 sont nécessaires pour diriger les signaux de programmation CTRL et PROGRAM. La figure 2.27 représente la partie commutation pour *N*=3, dans ELSA *N*=12.

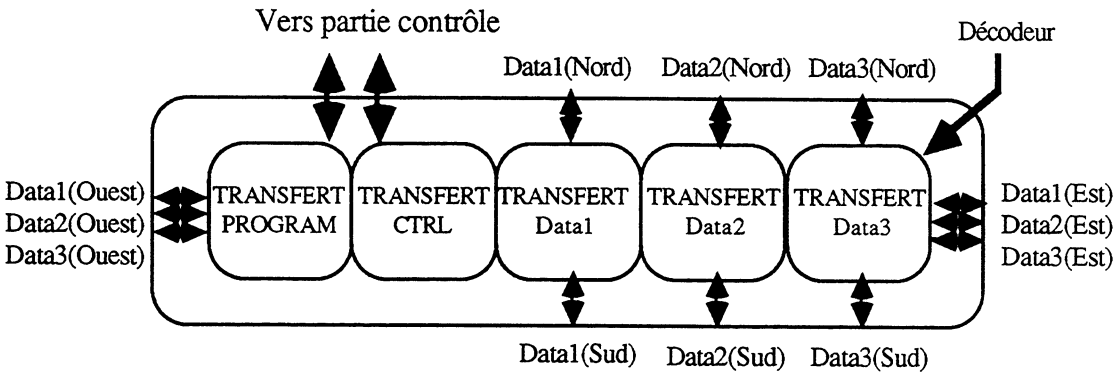


Figure 2.27. La partie commutation.

Nous avons vu que le réseau de commutateurs est programmable uniquement par sa périphérie. En conséquence, le commutateur doit pouvoir se programmer par les quatres directions Nord, Sud, Est, Ouest. Celui-ci reçoit alors les huit signaux CTRL(x) et PROG(x) (x={Nord, Sud, Est, Ouest}).

Pendant la programmation du réseau, c'est-à-dire pendant la programmation de plusieurs chemins, il se peut que des données de programmation interfèrent lorsqu'il y a croisement de chemins (figure 2.28).

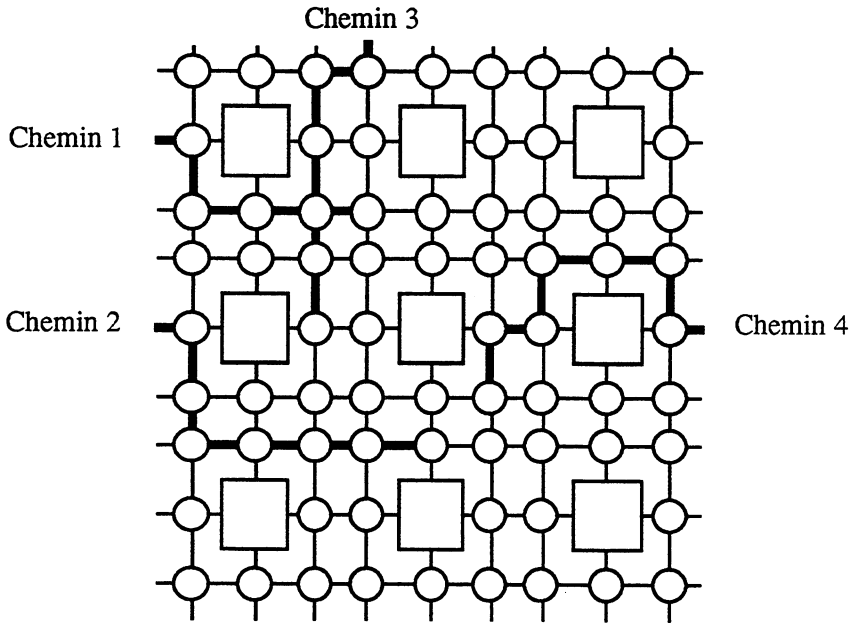


Figure 2.28. Programmation de plusieurs chemins.

Pour pallier le croisement des données, la partie contrôle contient un bloc nommé SELECT, qui veille à ce qu'une seule direction de programmation soit active à la fois. En conclusion, la partie contrôle contient un registre à décalage (R0, R1), deux bascules (D0, D1), un décodeur et un bloc SELECT (figure 2.29).

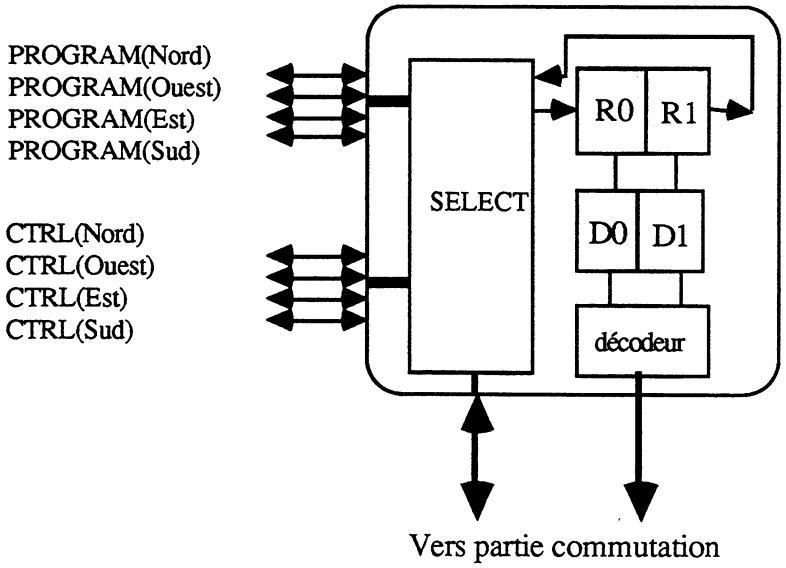


Figure 2.29. L'architecture de la partie contrôle.

Le rôle de cette partie contrôle est soit de permettre le chargement des deux bascules (D0, D1) afin de configurer le commutateur, soit de décaler les données de programmation. Les opérations de cette partie contrôle sont décrites ci-dessous :

(x et x' sont respectivement la direction d'entrée et de sortie du signal CTRL, ils appartiennent à {Nord, Sud, Est, Ouest})

Si *(CTRL(x) = 0)* ***Alors***
D0 ← R0 / chargement des données */*
D1 ← R1
Sinon
R0 ← PROGRAM(x) / Décalage des données */*
R1 ← R0
PROGRAM(x') ← R1
D0 ← D0 / maintien des anciennes données */*
D1 ← D1

II.4.4 Architecture du bloc SELECT

Nous avons vu que le bloc SELECT impose aux commutateurs de n'être programmés que par une seule direction. Pour cela, il faut d'une part valider la programmation si et seulement si un signal CTRL(x) est à 1, et d'autre part mémoriser l'origine du signal de programmation. Quatre bascules sont utilisées pour mémoriser l'origine du signal de programmation, ces bascules

sont appelées "Flags(x)" (x={Nord, Sud, Est, Ouest}) , le chargement de ces bascules est autorisé par la fonction F tel que:

$$F = \bar{a}\,\bar{b}\,\bar{c}\,\bar{d} + a\,\bar{b}\,c\,\bar{d} + \bar{a}\,b\,c\,d + a\,b\,\bar{c}\,d$$

ou: $\bar{a}$ est le complément de a.

a = CTRL(Nord)

b = CTRL(Sud)

c = CTRL(Est)

d = CTRL(Ouest)

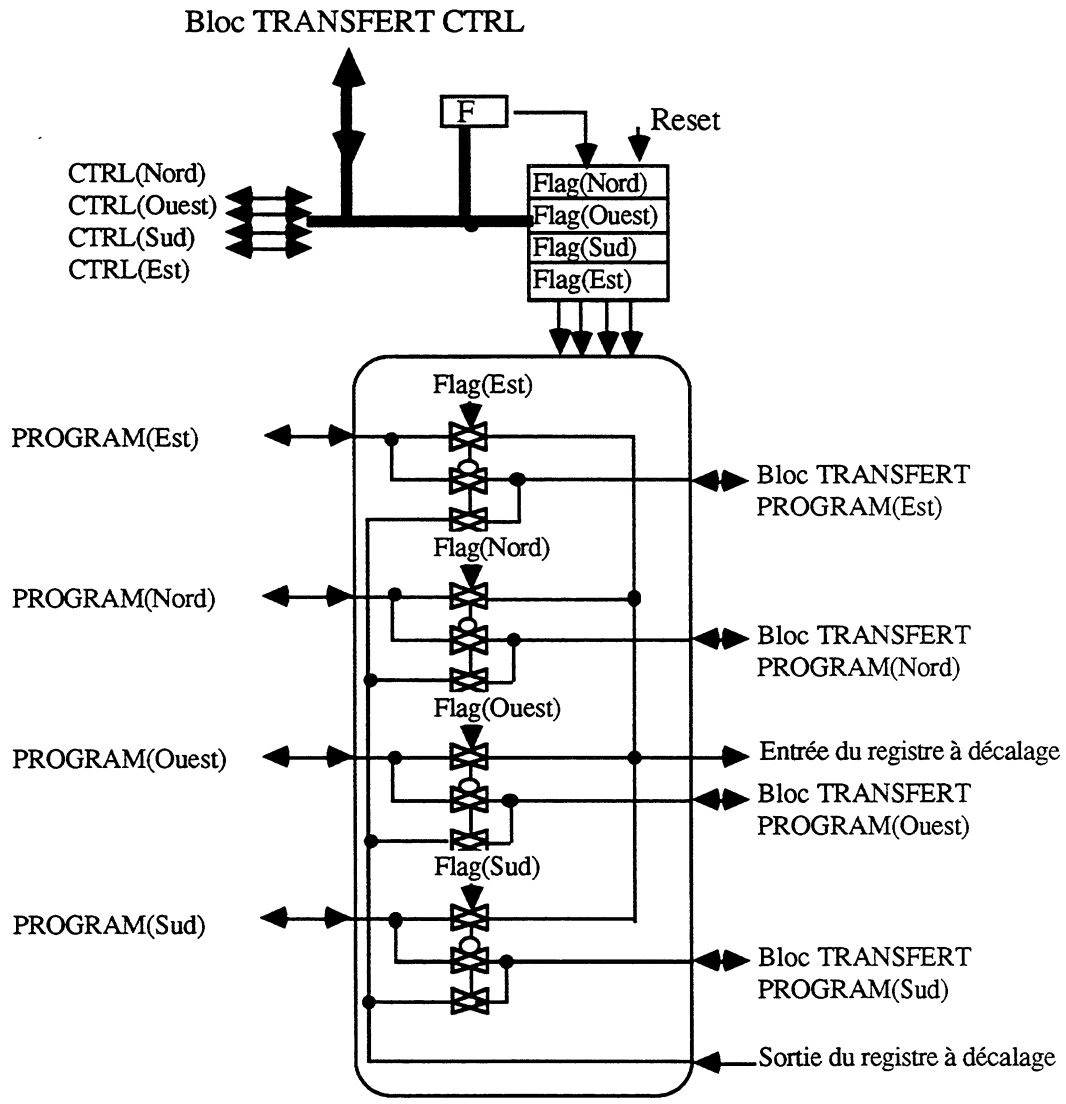


Figure 2.30. L' architecture du bloc SELECT

Le bloc SELECT doit permettre aussi aux données de programmation PROGRAM(x) d'accéder au registre à décalage (R0, R1), puis d'être acheminées par la partie commutation. Un ensemble de portes de transfert

commandées par les bascules "Flag(x)" contrôle la circulation des données de programmation. La figure 2.30 représente l'architecture de ce bloc SELECT. Celui-ci contient la fonction F, les quatre bascules "Flag(x)" et les portes de transfert. A chaque "Flag(x)" sont associées trois portes de transfert, notées portes(x) de transfert comme montré sur la figure 2.31. Cette figure montre aussi les liaisons entre le bloc SELECT, le registre à décalage, les blocs TRANSFERT CTRL et PROGRAM de la partie commutation.

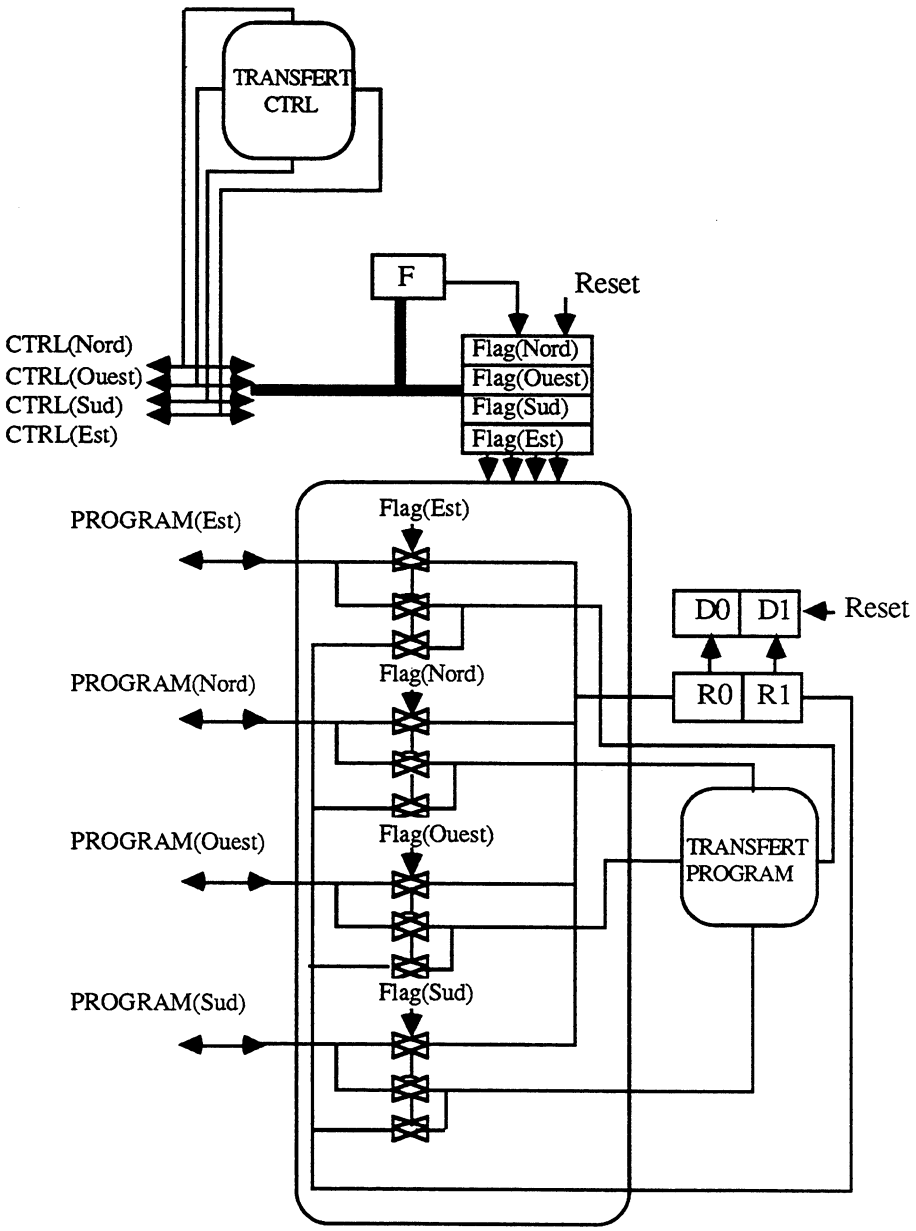


Figure 2.31. La relation entre le bloc SELECT et les autres éléments du commutateur.

Pour mieux comprendre le fonctionnement de ce bloc et sa relation avec les autres éléments, nous allons détailler l'exemple d'une programmation du commutateur par le côté Nord dans la configuration (D0, D1) = (1, 1).

Tout d'abord, le commutateur est initialisé par le signal Reset. Cette initialisation le positionne dans la configuration (0, 0), cela correspond à l'initialisation des deux bascules (D0, D1). En même temps, les signaux CTRL(x) sont déchargés et les "Flags(x)" du bloc SELECT sont initialisés à 0.

Pour programmer le commutateur, il faut que les données de programmation PROG(Nord) puisse être chargées dans les bascules (D0, D1), ce chargement est autorisé par le signal CTRL(Nord). Ce chargement est contrôlé par le bloc SELECT.

Le bloc SELECT autorise la programmation par la fonction F. Dans cet exemple, CTRL(Nord) est chargé à 1 tandis que les autres signaux CTRL(x) sont à 0. F autorise alors CTRL(Nord) à se charger dans le "Flag(Nord)". Ce "Flag(Nord)" passe de l'état 0 à 1. A partir cet instant les "Flag(x)" sont à 0 sauf le "Flag(Nord)". La figure 2.32 montre le chargement de CTRL(Nord) dans le "Flag(Nord)" associé.

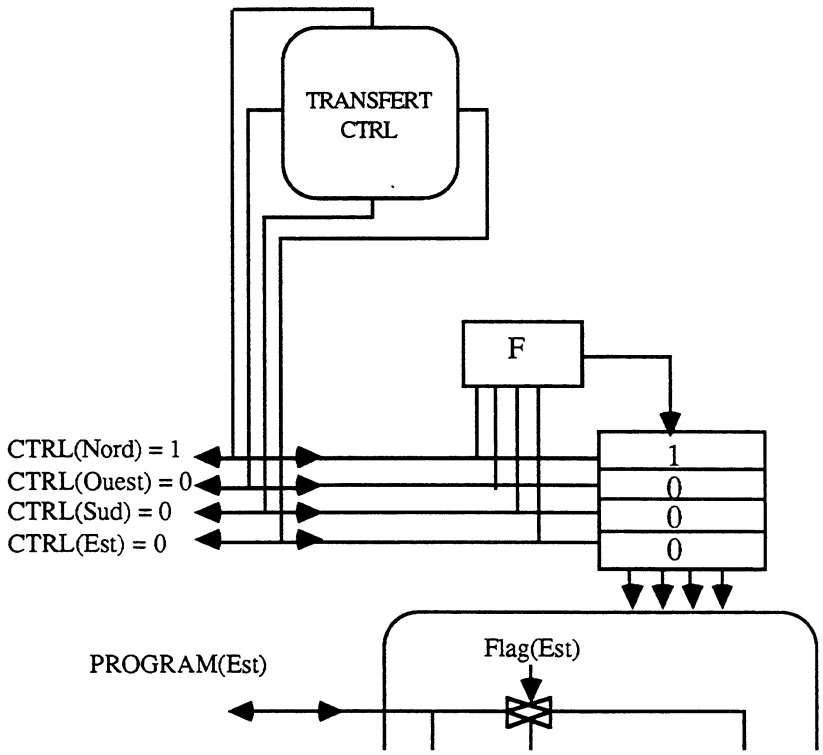


Figure 2.32. Le chargement du "Flag(Nord)".

Le rôle des "Flag(x)" est de permettre aussi la liaison entre le signal PROG(x) et le registre à décalage. Pour cela, chaque "Flag(x)" contrôle trois portes(x) de transfert. La première porte(x) autorise la liaison entre la donnée PROG(x) et le registre à décalage, la seconde autorise la liaison du bloc TRANSFERT PROGRAM de la partie commutation vers l'extérieur et la troisième autorise la liaison entre la sortie du registre à décalage et le bloc TRANSFERT PROGRAM de la partie commutative.

Dans cet exemple, lorsque "Flag(Nord)" est chargé à 1, la première et la troisième porte(Nord) de transfert sont fermées tandis que sa seconde est ouverte. La première porte(Nord) autorise la liaison entre PROG(Nord) et le registre à décalage. Par cette liaison, les données peuvent maintenant atteindre le registre à décalage (R0, R1). Deux coups d'horloge suffisent alors aux données (1, 1) pour se positionner dans le registre. La figure 2.33 montre la circulation de ces données PROG(Nord).

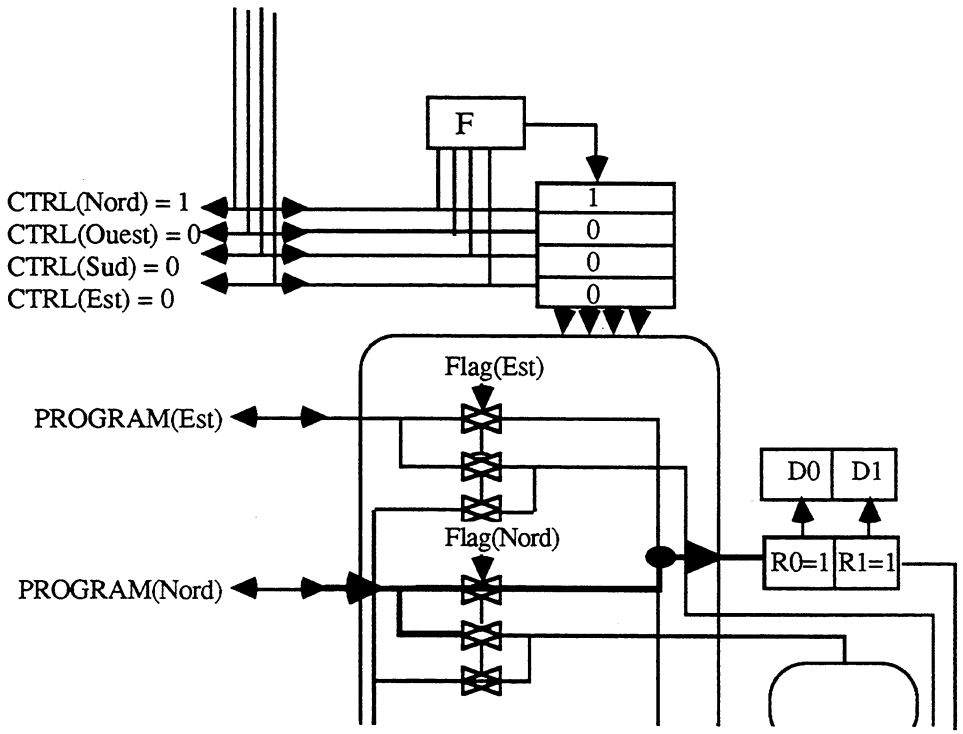


Figure 2.33. La circulation des données pour atteindre le registre (R0, R1).

Après avoir positionné les données dans le registre, il faut mettre le signal CTRL(Nord) à 0 pour programmer le commutateur. Cela permet le chargement des bascules (D0, D1) à partir de (R0, R1). Cette programmation positionne tous les blocs de TRANSFERT de la partie commutation dans la configuration (1, 1), en particulier les deux blocs de TRANSFERT CTRL et

PROG. Lorsque CTRL(Nord) est mis à 0 PROG(Nord) (resp. CTRL(Nord)) est relié à PROG(Sud) (resp. CTRL(Sud)). La figure 2.34 montre le chargement des bascules (D0, D1) et la configuration des deux blocs de TRANSFERT.

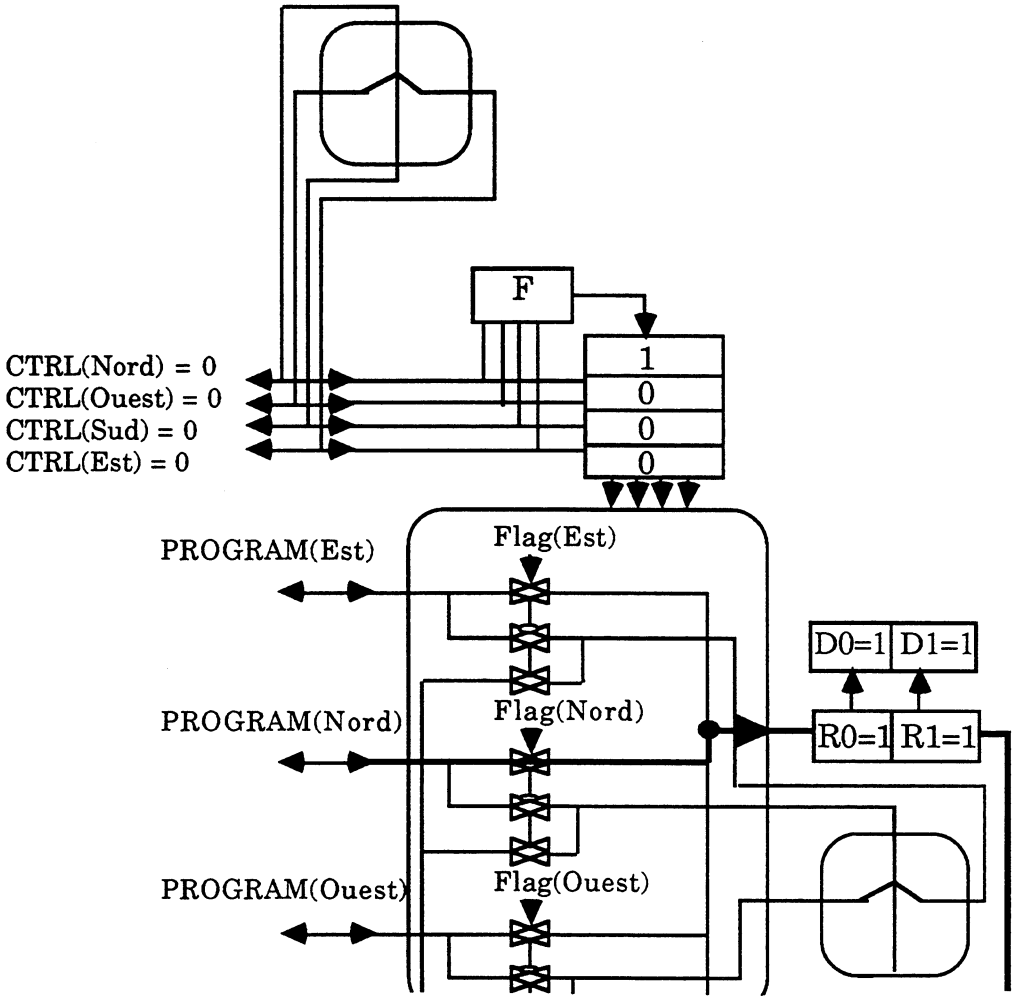


Figure 2.34. La configuration des blocs de TRANSFERT.

Au coup d'horloge suivant, les données du registre à décalage seront décalées et elles devront atteindre un autre commutateur. Ces données doivent suivre le chemin de programmation. Pour cela elles doivent être dirigées par la partie commutation. L'introduction des secondes et des troisièmes portes(x) de transfert autorisent cette liaison entre le registre à décalage et le bloc TRANSFERT PROG.

Dans cet exemple, la sortie du registre à décalage est relié à l'entrée Nord du bloc TRANSFERT PROGRAM via la troisième porte(Nord) de transfert (les

autres troisièmes portes(x) étant ouvertes, elles n'autorisent pas de liaison). La figure 2.35 montre la liaison avec le bloc TRANSFERT PROG.

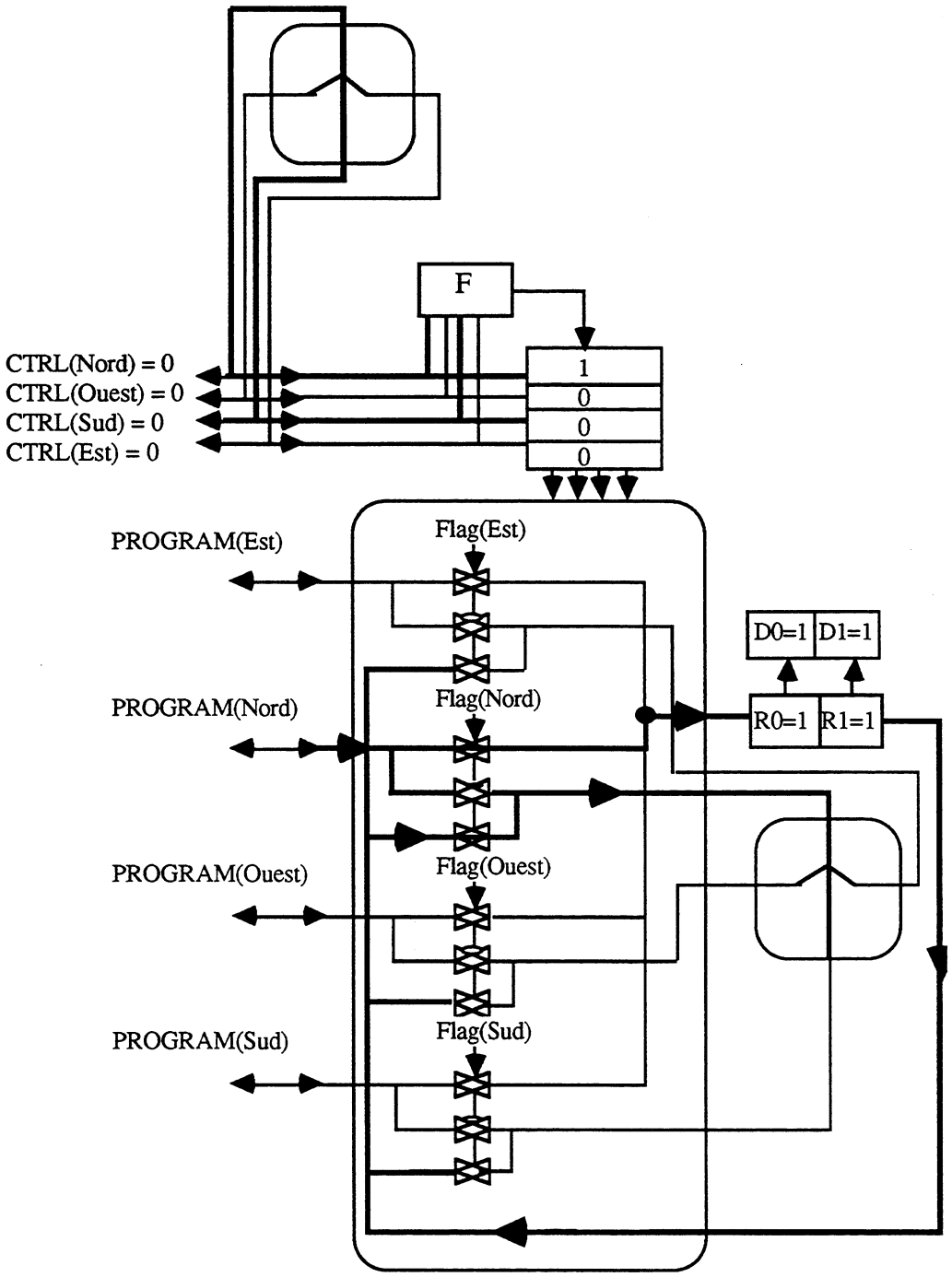


Figure 2.35. Le contrôle de la direction des données PROG(Nord).

Pour réaliser la liaison entre la sortie du bloc TRANSFERT PROGRAM et l'extérieur, c'est-à-dire les connecteurs PROG(x), nous avons introduit la seconde porte(x) de transfert. Suivant la configuration du commutateur la sortie du bloc de TRANSFERT peut-être au Sud, à l'Ouest ou à l'Est. Pour cela

toutes les secondes portes(x) sont fermées sauf la porte(Nord) de transfert. Dans cet exemple, la sortie est autorisée par le connecteur PROG(Sud) puisque l'on programme la configuration (1, 1). La figure 2.36 montre la liaison entre le connecteur PROGRAM(Sud) et le bloc TRANSFERT PROGRAM.

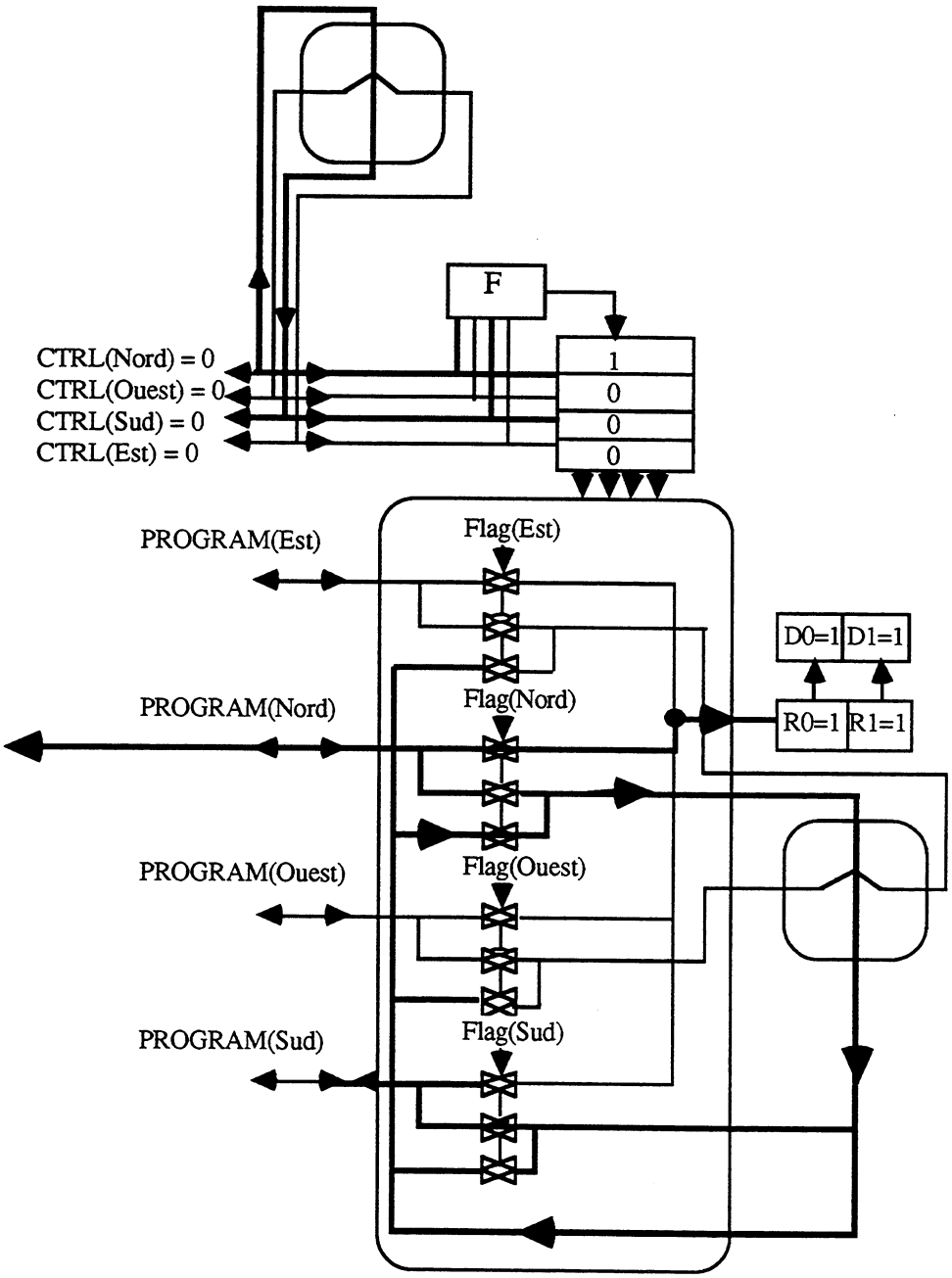


Figure 2.36. La liaison entre la sortie du bloc TRANSFERT PROGRAM et le connecteur de sortie PROGRAM(Sud).

Au coup d'horloge suivant, la donnée R1 peut alors sortir du registre à décalage, atteindre l'entrée Nord du bloc TRANSFERT PROGRAM via la troisième porte(Nord), sortir par le côté Sud de ce bloc et atteindre la sortie PROGRAM(Sud) du commutateur via la seconde porte(Nord)

II.5 La réalisation électrique et topologique

II.5.1 L'amplification des données

Après simulation du réseau de commutateur et sachant que le circuit doit fonctionner à une fréquence d'au-moins 10 Mhz, nous avons introduit des amplificateurs. Ceux-ci doivent accélérer la circulation des données entre les sous-matrices, c'est-à-dire diminuer le temps de charge et de décharge des bus de données.

La communication entre les sous-matrices étant bidirectionnelle, il faut que les amplificateurs le soient aussi. Dans ce cas il se pose alors le problème du contrôle du sens des amplificateurs. Pour mieux le comprendre, celui-ci est illustré par un exemple (figure 2.37). Cette figure représente une connexion logique entre les éléments (1, 1) et (1, 2); à travers celle-ci les données circulent de (1, 1) vers (1, 2) dans le sens logique Ouest → Est. On s'aperçoit que dans le segment physiques A les données circulent dans le sens physique Nord→Sud et dans le segment physique B les données circulent dans le sens physique opposé Sud→Nord. En conséquence, l faut que le contrôle du sens de B soit inversé par rapport à celui de A. Le contrôle du sens des amplificateurs doit tenir compte d'une part du sens logique des données et d'autre part de l'orientation physique de la connexion ou plus généralement de la programmation des commutateurs. C'est à partir du résultat de la configuration que l'on obtient ces informations.

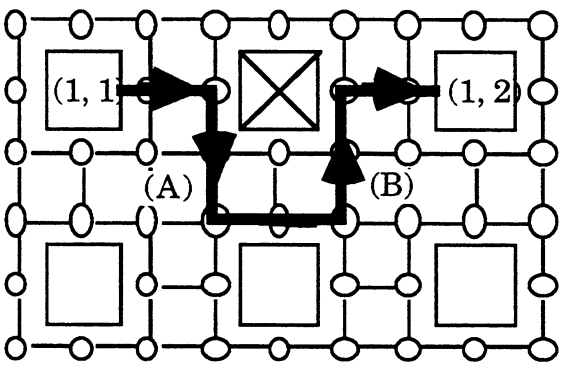


Figure 2.37. Un exemple de connexion logique.

Pour contrôler les amplificateurs, nous avons à notre dispositions les commandes globales de la matrice Cnord → sud et Couest → est qui gèrent la circulation logique des données de la matrice. Il suffit alors de connecter la bonne commande parmi {Cnord → sud, $\overline{\text{Cnord}}$ → sud, Couest → est, $\overline{\text{Couest}}$ → est} pour contrôler correctement le sens des amplificateurs. Nous avons décidé de contrôler cette sélection à partir des commutateurs. Pour cela chaque amplificateur est directement relié à un commutateur et, dans ce dernier, deux bits de contrôle sont ajoutés pour sélectionner le sens adéquat.

La figure 2.38 représente le contrôle du sens des amplificateurs à partir des commutateurs. Deux bits (R2, R3) sont ajoutés au registre à décalage du commutateur et deux bascules (D2, D3) lui sont associés pour stocker le contrôle du sens de l'amplificateur.

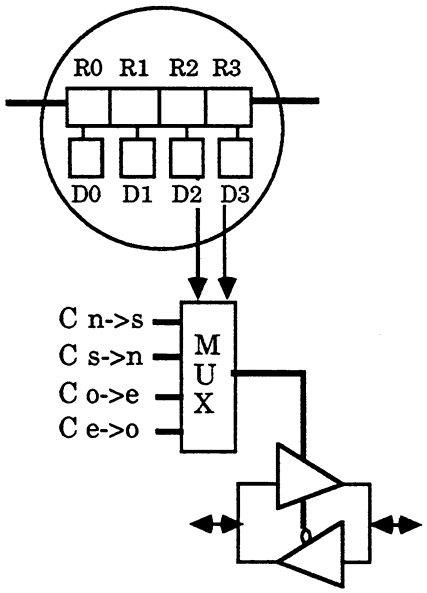


Figure 2.38. Le contrôle de l'amplificateur à partir du commutateur.

La tableau 2.6 ci-dessous donne le sens de l'amplificateur en fonction des deux bits (D2, D3).

(D2, D3)	Direction
00	Cnord → sud
01	$\overline{\text{Cnord}}$ → sud
11	Couest → est
10	$\overline{\text{Couest}}$ → est

Tableau 2.6

Les deux nouveaux bits sont programmés de la même manière que les deux bits contrôlants la configuration du commutateur. Nous allons illustrer cette programmation à partir d'un exemple. La figure 2.39(a) représente une suite (C1, C2, C3) de commutateurs à programmer. (C1, C3) sont des commutateurs à deux bits et C2 est un commutateur à quatre bits contrôlant l'amplificateur (figure. 2.39 (b)). On suppose que le sens de l'amplificateur doit être contrôlé par la commande Couest → est.

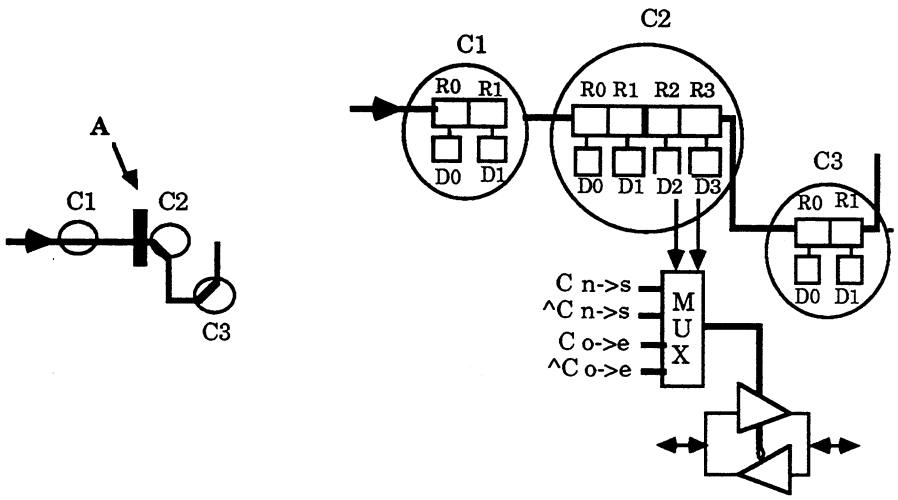


Figure 2.39. (a) Le chemin à programmer.(b) Le modèle associé.

Le chemin est programmé par le côté ouest de C1. La tableau 2.7 donne le flot de données à envoyer par CTRL_DATA(Ouest) de C1. Les bits en gras correspondent à la programmation de l'amplificateur. La programmation de C1 est faite à l'étape 1. A l' étape 2, C2 et C1 sont programmés. Pour programmer C2, 4 bits sont envoyées. Les deux premiers bits (11) permettent la sélection de la commande Couest → est et les deux derniers bits (01) configure le commutateur. A l'étape 3, les trois commutateurs sont programmés pour obtenir le chemin.

Etape	Données
1	11
2	**10**0111
3	10**10**0111

Tableau 2.7

Nous avons donc conçu un réseau muni d'amplificateurs bidirectionnelles entièrement programmables électriquement. Pour conserver une homogénéité dans la programmation du réseau, c'est-à-dire dans un premier temps une programmation électrique puis dans un second temps une programmation par laser ("hard"), nous avons introduit des dispositifs permettant de fixer définitivement le sens des amplificateurs. Ces dispositifs sont programmables par des coupures de métal 2 à l'aide d'un laser et ils sont intégrés dans le commutateur reliés à l'amplificateur.

Après avoir défini les caractéristiques de l'amplificateur, il nous reste à déterminer le type, le nombre et l'emplacement des amplificateurs. Après simulations de plusieurs types, nous avons retenu celui décrit sur la figure 2.40 car il est plus rapide en changement de sens. La tableau 2.8 donne le sens en fonction de la commande.

commande	sens
0	E/S(2) → E/S(1)
1	E/S(1) → E/S(2)

Tableau 2.8

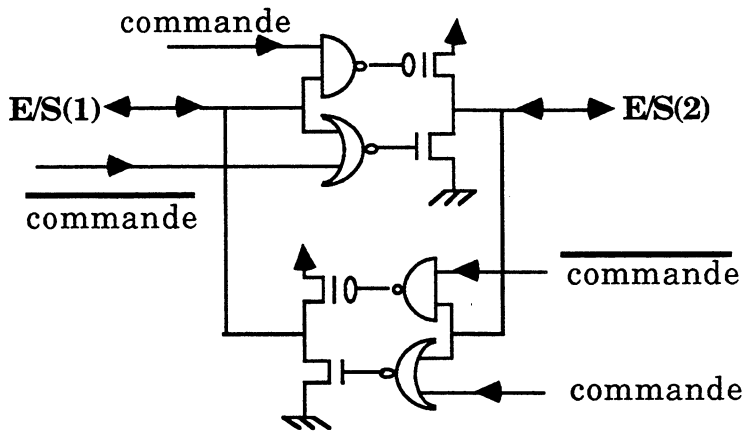


Figure 2.40. Schéma logique de l'amplificateur.

L'emplacement des amplificateurs dans la matrice doit être défini sachant que la réalisation de la matrice entière est obtenue par la répétition du réticule. La figure 2.41 montre l'emplacement choisi pour les amplificateurs dans le réseau, seulement 7 commutateurs au maximum peuvent être traversés sans amplificateur, ce qui correspond a un temps de charge de 10 ns et un temps de décharge de 10 ns.

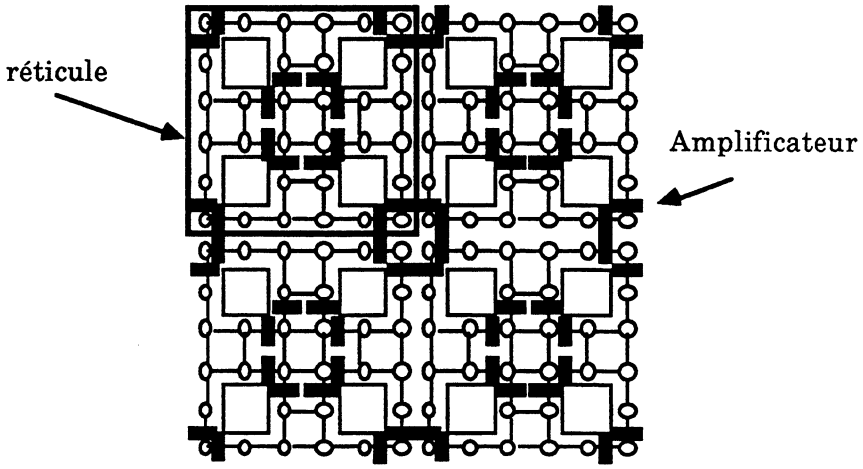


Figure 2.41. La disposition des amplificateurs.

II.5.2 Plan de masse

Rappelons que la matrice entière est obtenue par la répétition du réticule. Dans ce motif, nous avons vu que chaque sous-matrice est entourée de huit commutateurs et de quatre amplificateurs. Les huit commutateurs se décomposent en quatre commutateurs à deux bits et quatre commutateurs à quatre bits qui contrôlent les quatre amplificateurs. La figure 2.42 illustre l'implantation physique des amplificateurs.

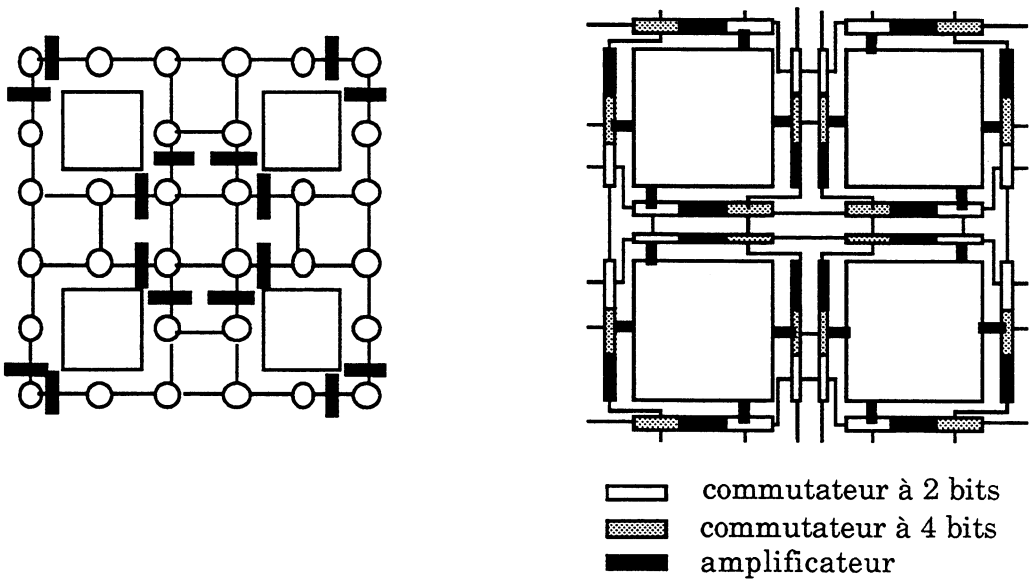


Figure 2.42. Implantation physique du réseau de commutateurs.

La réalisation du commutateur et de l'amplificateur est faite en vue de minimiser la surface du à la configuration. Nous en avons déduit une

contrainte de réalisation; minimiser la hauteur du commutateur et de l'amplificateur. Il faut que la longueur de deux commutateurs plus celle d'un amplificateur correspond au maximum à celle d'un côté de la sous-matrice. Les schémas logiques, électriques et topologiques des différents blocs composant le commutateurs à deux bits, le commutateurs à quatre bits et l'amplificateur sont données dans l'annexe I.

II.5.3 Réalisation de la tranche

Nous avons vu que l'organisation du réticule est faite sachant que la tranche est obtenue par la répétition du réticule, ceci est une donnée importante pour la réalisation des autres éléments (alimentation, signaux de contrôle, horloge). La disposition des réticules sur la tranche est présentée sur la figure 2.43. Nous avons placé 20 réticules de 14250µ x 16000µ sur une tranche de 4". Le circuit contient alors 4930700 transistors sur une surface de 45,6 cm^2

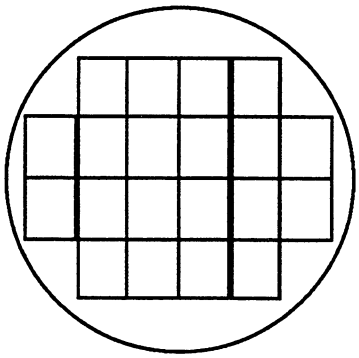


Figure 2.43. La disposition des réticules sur la tranche.

Nous avons vu au § II.3.2 la stratégie de tolérance aux défauts retenue pour l'alimentation, les signaux de contrôle et l'horloge. A partir de cette stratégie, nous en déduisons l'organisation de l'alimentation, des signaux de contrôle et de l'horloge au-niveau du réticule (figure 2.44).

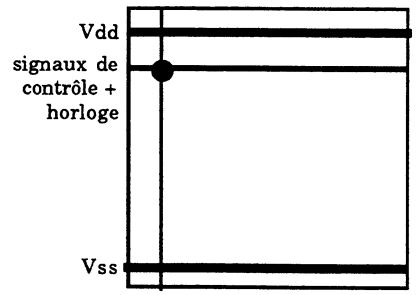


Figure 2.44. La distribution résultante dans le réticule.

La réalisation des plots ainsi que de leurs amplificateurs pose des problèmes particuliers. Il est impossible d'intégrer les plots sur la tranche en utilisant la répétition des masques du réticule puisque ceux-ci sont nécessaires et utiles uniquement sur le bord de la tranche. En conséquence, ces plots ne peuvent pas être dessinés sur les masques du réticule.

Une solution est d'utiliser des masques à l'échelle 1 de la tranche. Les plots étant des carrés de métal 2, nous allons réaliser un masque de métal2 à l'échelle 1 de la tranche sur lequel seront dessinés les plots avec leurs connexions.

Pour des raisons de coût et technologique, il n'est pas possible d'utiliser des masques à l'échelle 1 pour tous les niveaux technologiques. Les amplificateurs ne peuvent pas alors être dessinés sur des masques à l'échelle 1 de la tranche. Ils sont intégrés dans le réticule et seul les amplificateurs en bordure de la tranche sont utiles. Il est possible de déconnecter les amplificateurs inutiles en utilisant le masque de métal 2 à l'échelle 1 de la tranche.

La réalisation du niveau métal 2 à l'échelle 1 est faite après le métal 2 obtenu par la répétition du réticule. Pendant la métallisation à l'échelle de la tranche, il est possible de détruire du métal 2 déjà déposé. Nous utilisons cette possibilité pour déconnecter les amplificateurs inutiles à l'intérieur de la tranche.

Le dessin du masque de métal 2 à l'échelle 1 de la tranche est donné en annexe II.

II.6 TEST d'ELSA

Le test d'ELSA s'effectue en plusieurs étapes en accord avec la hiérarchie de la reconfiguration. Ces étapes sont : le test du PE, de la sous-matrice, du réseau de commutateurs et de la tranche configurée. Notons qu'il y a une complète indépendance entre le réseau de commutateurs et les PES, en conséquence le test du réseau peut être appliqué avant ou après le test du PE et de la sous-matrice. Après le test du PE, de la sous-matrice et du réseau il y a configuration de la tranche, et c'est en dernier que la tranche configuré est testée.

Les tests effectués sont fonctionnels, c'est-à-dire que l'on cherche à valider l'architecture et non pas à détecter toutes les pannes éventuelles. Le test du PE et de la sous-matrice sont réalisés à l'aide d'un microscope électronique.

Pour ces tests, seules les commandes globales sont utilisées. Toute la connectique nécessaire au test est raisonnable vis-à-vis de l'utilisation d'un microscope électronique. Par-contre, le test des commutateurs et de la tranche configurée sont réalisés de manière classique pour des raisons de pratiques (connectique trop importante pour un microscope électronique). Nous allons présenter la méthodologie de ces différents tests.

II.6.1 Test du PE

Le test du PE est réalisé sans utiliser les communications entre les PEs, seuls les commandes globales et l'entrée globale "BIT" sont utilisées pour charger les vecteurs de test. Le test est appliqué simultanément sur tous les PEs.

Le test de chaque partie du PE s'effectue toujours en deux étapes, en premier les vecteurs de test sont chargés, puis une comparaison est effectué. La comparaison porte toujours sur les états des deux registres NS et EW du PE. Si les états de ces deux registres sont différents alors le test a réussit sinon il a échoué. Le résultat est stocké dans une bascule de contrôle (BC) commandant les portes de transfert des circuits de contournement (cf II.3.1). La reconfiguration de la sous-matrice est alors effectuée à la suite de ce test. Il est possible que la reconfiguration de la matrice échoue. Par exemple, s'il existe plus d'un PE défectueux sur la ligne. Aucune indication d'une telle situation est donnée à la suite de ce test, il est alors important d'effectuer un test de la sous-matrice pour déterminer les sous-matrices défectueuses.

Le test s'applique sur les mémoires, les multiplexeurs, les registres et l'UAL. Nous allons présenter le test des mémoires qui correspond à la plus grande partie du test. Il s'effectue de la manière suivante:

II.6.1.1 Chargement des mémoires rama et ramb.

Le signal global "BIT" via l'UAL permet de charger les mémoires (l'entrée "SM" des mémoires est sélectionnée). Un vecteur (1010...) est écrit dans la première ligne de la mémoire A, puis un vecteur complément (0101 ...) est écrit dans la seconde ligne de la mémoire A. Cette procédure est répétée sur les lignes suivantes. Le même chargement est effectué sur la mémoire B, mais les vecteurs sont inversés; c'est-à-dire (0101 ...) sur la première ligne,(1010 ...) sur la seconde ligne etc ...

II.6.1.2 Comparaison entre les mémoires.

Les valeur d'adresse 0 des mémoires RAMA et RAMB sont chargées dans les registres "EW" et "NS". Elles sont comparées et le résultat est stocké dans la bascule de contrôle ("BC"). Cette comparaison est réalisée pour toutes les adresses. Si à une comparaison, les deux valeurs sont identiques alors le PE est défectueux et il est contourné.

Un autre chargement des mémoires ainsi qu'une comparaison sont effectués, mais avec des vecteurs inversés. Les premiers vecteurs de la RAMA et RAMB sont respectivement "0101 ... " et "1010 ..." au lieu de "1010 ..." et "0101 ...". Ce deuxième chargement permet de tester la mémorisation du 1 et du 0 dans les deux mémoires.

II.6.2 Test de la sous-matrice

Le test de la sous-matrice est basé sur des comparaisons entre les états des registres CM de la première ligne de la sous-matrice. Ces états peuvent être mémorisés dans des bascules qui sont observables avec un microscope électronique.

Les vecteurs de test sont obtenus à l'aide des commandes globales, de l'entrée globale "BIT" et du 0 appliqué pour les commutateurs aux quatres entrées Nord, Sud, Est et Ouest de la sous-matrice. Le test s'applique sur les mémoires, les communications entre les PEs, le registre FLG et l'UAL. A titre d'exemple, nous allons présenter le test des mémoires puis celui des communications entre les PEs.

II.6.2.1 Test des mémoires.

Ce test a pour but de vérifier toutes les mémoires de la sous-matrice reconfiguré, il vérifie en fait qu'il n'y a pas plus d'un PE défectueux sur une ligne. Le chargement des mémoires RAMA et RAMB du PE est le même que celui effectué lors du test du PE (cf. II.6.1). La vérification des valeurs des mémoires est réalisée à l'aide des registres CM. A chaque étape du test, la valeur de la mémoire RAMA est chargée dans le registre CM, puis elle est décalée par l'axe de communication CM jusqu'aux registres CM la première ligne de la sous-matrice. Les états des registres CM sont observés par un microscope électronique. Ce test est appliqué pour toutes les valeurs de la mémoire RAMA, puis il est appliqué sur la mémoire RAMB.

II.6.2.2 Test des communications

En premier, les registres NS et EW sont positionnés à 1 en utilisant l'entrée globale BIT. Puis par décalage successifs dans le sens Nord-Sud et Ouest-Est, le niveau 0 appliqué aux entrées de la sous-matrice est propagé dans les registres NS et EW. Une fois les registres NS et EW chargés à 0, l'UAL est utilisée pour vérifier ce chargement. Une addition entre NS, EW et C chargé à 1 est utilisée, si CY est à 0 alors les deux registres sont à 0 sinon au-moins un des 2 registres est à 1. Ce résultat est stocké dans la RAMB puis propagé aux registres CM de la première ligne de la sous-matrice. Les états des registres CM sont observés par un microscope électronique.

II.6.3 Test du réseau de commutateur

Nous rappelons que seuls les commutateurs périphériques sont connectés à des plots, et que la partie commutation d'un commutateur est de 12 bits. Le test du réseau correspond à la programmation de chemin de commutateurs puis à la vérification de l'établissement du chemin par envoie d'un signal sur 1 bit de la partie commutation. Le test s'effectue en trois phases. Pendant la première phase, des chemins pré-définis sont programmés et testés pour atteindre tous les commutateurs. La phase II correspond à la programmation de chemin pour atteindre si possible tous les commutateurs non testés pendant la phase I. La phase III teste les 11 bits restants de la partie commutation sur tous les commutateurs déclarés non-défectueux.

II.6.3.1 Phase I

Cette phase se décompose en 40 étapes, chaque étape correspond à la programmation de 20 chemins identiques en parallèle. A chaque étape, les chemins sont de plus en plus important. La première étape correspond à la programmation d'un chemin de 4 commutateurs puis à l'envoie d'un signal à l'entrée du chemin. Si le signal est observé à la sortie du chemin alors les 4 commutateurs ne sont pas défectueux sinon il existe au-moins un commutateur défectueux. Si le chemin n'est pas défectueux alors la deuxième étape augmente le chemin de deux nouveaux commutateurs. Ce chemin de 6 commutateurs est programmé et vérifié comme précédemment. Si ce chemin n'est pas défectueux alors la troisième étape ajoute deux nouveaux commutateurs, ce nouveau chemin est programmé et testé. Ce processus est répété jusqu'à la quarantième étape. La quarantième étape correspond au chemin de taille maximum.

II.6.3.2 Phase II

A la suite de la phase précédente, une cartographie des commutateurs non défectueux est obtenue. Il est possible que des commutateurs n'ont pas été testés durant la phase I. En effet, si à une étape donnée le test d'un chemin a échoué alors les étapes suivantes n'ont pas été appliquées.

Durant la phase II, de nouveaux chemins sont crées à partir de la cartographie des commutateurs non défectueux pour atteindre tous les commutateurs. Actuellement, la création des nouveaux chemins est manuelle.

II.6.3.3 Phase III

Durant les phases précédentes, un ensemble de chemin a été établi qui couvre tous les commutateurs du réseau. Les commutateurs ont été classifiés valides ou défectueux, mais uniquement un bit de la partie commutative à été testé. Durant la phase III, les chemins des phases précédentes sont appliquées et le test est appliqué sur les 11 bits restants du commutateur.

II.6.4 Test de la tranche configurée

A partir de la cartographie des sous-matrices et des commutateurs valides, la tranche est configurée en utilisant les algorithmes de configuration et de programmation du réseau. Le but de ce test est de vérifier la communication entre les sous-matrices. Une séquence de vecteur est appliqué sur un bord de la tranche, puis après décalages, ces vecteurs sont observés directement sur les plots du coté opposé de la tranche. Cette procédure est appliquée pour le sens Sud-Nord, Ouest-Est et CMS-CMN. Le nombre de décalage nécessaire pour traverser la tranche dépend de la taille de la matrice configurée.

Les procédures de test du PE, de la sous-matrice, du réseau de commutateurs et de la tranche configurée sont données annexe III

CHAPITRE III

SYNTHESE AUTOMATIQUE DE PROCESSEURS MASSIVEMENT PARALLELES

Dans ce chapitre nous présentons un prototype d'outil de synthèse de haut niveau pour réseau 2D de processeurs monobit dédiés au traitement d'image bas niveau.

III. 1 OUTIL DE SYNTHESE DE HAUT NIVEAU

Un outil de synthèse de haut niveau génère à partir d'une description comportementale l'architecture optimisée d'un circuit. Cette architecture est en général définie au niveau RTL (transfert de registre).

Un outil de synthèse de haut niveau est paramétré par le type de circuit cible. Parmi les circuits cibles ayant fait l'objet d'études détaillées on peut citer les circuits du type microprocesseur MACPITTS [Fox83], LAGER [Ra86], ALGIC [ShGL86], SPIL [PeAl86], et SYCO [Jerr89], et les circuits de traitement de signal FIRST [DeRe85], SECOND [SmDe88], CATHEDRAL [RMVG88].

III.1.1 Définitions

La synthèse de haut niveau est généralement divisée en plusieurs étapes illustrées dans la figure 3.1.

Nous donnerons dans les prochains paragraphes une description succincte des différentes tâches composant la synthèse de haut niveau. Ces tâches sont la translation, l'ordonnancement, l'allocation, la synthèse de contrôleur et la génération de la partie opérative. Elles sont décrites dans les paragraphes suivants.

La description algorithmique est l'entrée du système de synthèse. Elle est généralement décrite dans un langage de haut niveau tel que C ou LISP, ou un langage spécifique de description comportementale tel VHDL. Cette description est analysée par le compilateur et transformée en une représentation interne généralement sous forme d'un graphe de dépendance de données. Un exemple de description comportementale est illustré par la figure 3.2

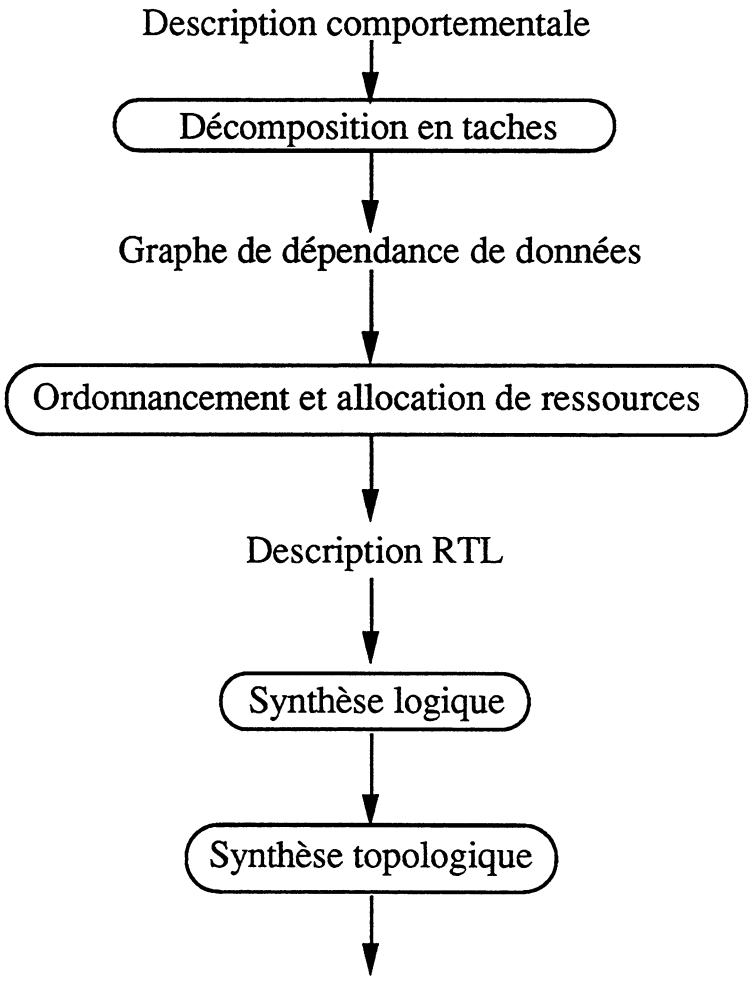


Figure 3.1 Etapes de la synthèse de haut niveau

```
x = a + b

y = e + 19

z = x * y
```

Figure 3.2 Exemple de description comportementale.

Le graphe de dépendance de données (GDD) est un graphe orienté dont les noeuds représentent les opérations effectuées sur les données et les arcs représentent la dépendance des données. Le GDD de l'exemple de la figure 3.2 est donné par la figure 3.3.

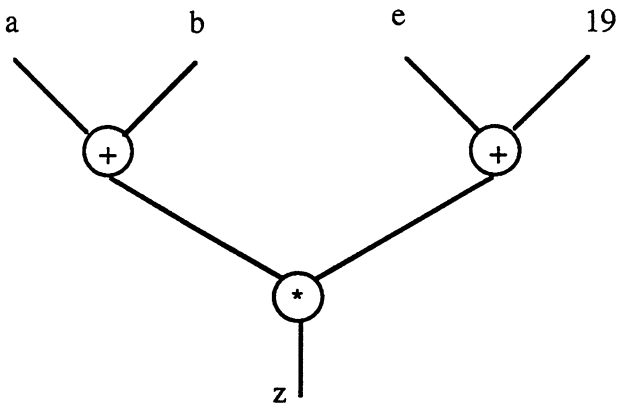


Figure 3.3 Exemple d'un GDD

Les deux étapes d'ordonnancement des taches et d'allocation de ressources sont les plus importantes. L'*ordonnancement* est l'affectation des opérations à des instants distincts appelés "pas de contrôle". L'*allocation* est l'assignation des opérations aux opérateurs et de variables aux registres. Elle se décompose en trois phases : le choix des opérateurs, la détermination du nombre d'opérateurs identiques et l'assignation finale de ressources. L'assignation de ressources est l'assignation des unités fonctionnelles aux unités physiques (opérateurs), des variables aux registres et des éléments de connexion aux transferts de données.

Bien que l'ordonnancement et l'allocation soient deux tâches différentes, elles ne sont pas forcément indépendantes. On trouve des systèmes, SILC [BaAL85], FACET [TsSi19], EMUCS [HiTh83], CHIPPE [KaPa84], qui commencent par faire un ordonnancement suivi - d'une manière plus ou moins indépendante - d'une allocation. D'autre, MAHA [PaPM86] privilégie l'allocation et réalise ensuite l'ordonnancement. Une troisième catégorie ELF [GiKn84] SLICER [PaGa87], HAL [PaKn86] effectue les deux tâches en même temps. Dans ce cas, les registres sont alloués au fur et à mesure de l'ordonnancement des opérations mais cette allocation est très difficile à réaliser. En effet, dans certains programmes effectuant une analyse globale, l'ordonnancement d'une opération dans des pas de contrôle peut dépendre de l'ordonnancement d'autres opérations. Ces dernières peuvent être dans des pas de contrôle soit précédents soit suivants le pas de contrôle courant. De ce fait, un pas de contrôle peut ne pas être complètement rempli jusqu'à ce que la dernière opération soit ordonnancée d'où la difficulté de l'allocation concurrente des registres. Dans les programmes de synthèse effectuant l'allocation des registres à partir d'un graphe dont les opérations ont déjà été ordonnancées, le processus d'allocation est plus facile à traiter.

Le fait de commencer par l'ordonnancement ou l'allocation n'est pas tout à fait arbitraire. L'un privilégie un critère au détriment de l'autre.

Une fois l'ordonnancement et l'allocation effectués, l'architecture est définie et peut être synthétisée au niveau layout.

Enfin, la dernière étape du processus de synthèse consiste en l'obtention du circuit à implémenter à partir des éléments réels, et grâce à un outil de placement/routage.

Suite à l'expérience acquise au cours de la conception d'ELSA, nous nous sommes intéressés à la synthèse automatisée de circuits dédiés au traitement d'image de bas niveau, en évitant la lourdeur de l'aspect programmable. L'architecture reste une matrice de processeurs monobits mais le réseau de processeurs "universels" programmables est remplacé par un réseau à connexion dédiée dont le processeur a une structure "minimale" adéquate. Les PEs contiennent une unité arithmétique et logique, des registres de 1-bit en nombre minimal, et une mémoire de dimension dédiée (cf.III.3). Le contrôle est une suite de mots de commandes figées.

III.1.2 Système proposé

Dans ce paragraphe nous donnons un aperçu général de notre système (figure 3.4). L'entrée consiste en la description algorithmique d'une application décrite dans un langage de haut niveau adapté au traitement d'image de bas niveau. Une première étape consiste à compiler le programme source. Cette compilation fournit en sortie un graphe de dépendance des données ordonnancées de macroinstructions sur les images. Ce graphe fera l'objet d'une expansion dans la prochaine étape. L'expansion consiste à remplacer les macroinstructions par leurs GDD en termes d'opérations ordonnancées. Ce graphe servira de base aux étapes suivantes à savoir l'assignation des registres, la génération de l'unité arithmétique et logique, l'amélioration de l'ordonnancement et l'allocation des multiplexeurs. L'objectif de l'optimisation étant de réduire la taille du circuit.

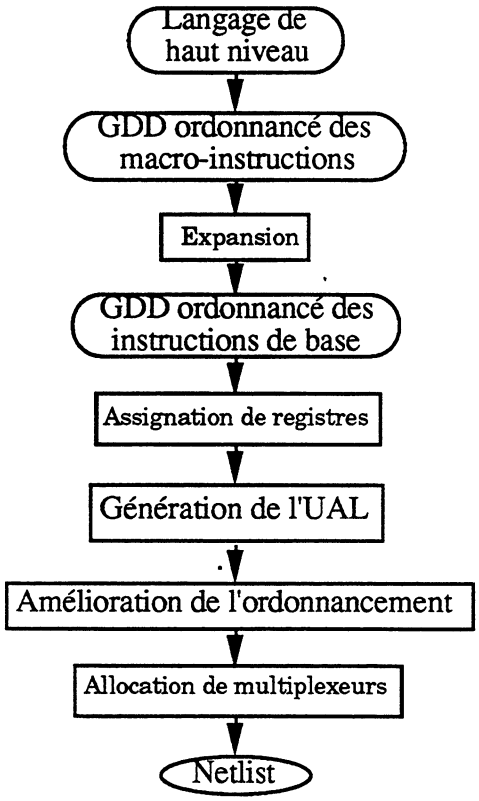


Figure 3.4 Etapes du système de synthèse

III. 2 DEFINITION DU LANGAGE CIBLE

Les algorithmes de traitement d'image de bas niveau sont décrits dans un langage de haut niveau. Il existe deux approches pour le choix du langage. La première consiste à développer un langage nouveau comme c'est le cas de Actus [Perr79] pour l'ILLIAC IV, le Parallel Pascal [Reev84] pour le MPP. La deuxième à étendre un langage déjà existant soit pour le calcul parallèle en général, comme c'est le cas de Matrix Pascal, FORTRAN 8X, soit pour répondre aux besoins spécifiques pour couvrir une gamme d'application comme le traitement d'image par exemple, comme le langage L (extension d'ALGOL) [Raetal83], PIXAL (extension d'ALGOL 60) [Leetal83], HCL (extension de C) [Pfei90].

La deuxième solution semble plus viable et raisonnable [Magg83], [Pfei90] pour des raisons évidentes. Le langage C a été adopté dans cette étude.

Nous rappelons les constructions de base du langage utilisé ainsi que la définition des macroinstructions.

III.2.1 Les expressions

Les expressions sont les éléments de base du langage. Une expression peut être construite à partir de l'un des quatre éléments suivants : les variables, les opérations, les constantes et les parenthèses.

Les variables peuvent être de trois types : entier, booléen et image. Les deux premiers ont la même signification que dans le langage C, tandis que la troisième requiert plus d'explication. Ce troisième type a été introduit dans un souci de garder les choses naturelles à l'utilisateur et d'avoir un langage expressif. La syntaxe pour déclarer une image est la suivante :

image nom_image (expression1, expression2, expression3);

Image, mot réservé, est le mot clé qui sert pour la déclaration. Nom_image est le nom donné pour référencer cette variable. Expression est un entier. Expression 1 est la taille x de l'image, expression 2 est la taille y de l'image et expression 3 est la profondeur de l'image.

Pour déclarer une image binaire (codée sur 1 bit), appelée "carte" de 128×128 pixels, on écrit:

image carte (128, 128, 1);

Seules les contantes entières sont supportées dans la version actuelle.

Les opérations de base telles que les opérations arithmétiques, logiques ou relationnelles sont fournies. Dans les opérations arithmétiques on trouve les opérations habituelles : l'addition (+), la soustraction (-), la multiplication (×), la division (/) l'incrémentation (++) et la decrémentation (--). Dans la classe des opérations relationnelles, on trouve inférieur à (<), supérieur à (>), égale à (==), différent de (!=), inférieur ou égal à (≤) et supérieur ou égal à (≥).

III.2.2 Les structures de contrôle

Les instructions classiques pour controler le déroulement d'un programme sont fournies à savoir le **if**, le **for** et le **while**. Leur syntaxe est la suivante :

if (expression) instruction 1 **else** instruction 2

while (expression) instruction

for (expression 1; expression 2; expression 3) instruction

Enfin, on trouve l'instruction **break** qui sert à sortir prématurément d'une boucle et l'instruction **return** qui sert à terminer un sous-programme.

III.2.2 Les macros de haut niveau

Nous avons ajouté plusieurs macros très utiles pour faciliter la tâche de l'utilisateur et c'est cette partie qui constitue la puissance de description liée à l'application. Ces macros sont divisées en trois catégories. La première concerne le mouvement de données, la deuxième les entrées sorties et la dernière le calcul proprement dit.

III.2.2.1 Les macros de mouvement de données

Les décalages les plus simples sont ceux effectués dans l'une des quatres directions (N, S, E, W). La figure 3.5 montre quelques exemples sur une image (3,3,1). Pour un décalage nord chaque ligne de la matrice est décalée d'une position vers le nord. On peut décaler cette matrice de deux ou plusieurs positions à condition de ne pas dépasser la taille physique de la matrice. La macro **shift** a été définie pour cet effet.

shift(A, B, C) A: l'image à décaler
B: direction du décalage ∈ {Est, West, North, South}
C : pas de décalage ∈ N*

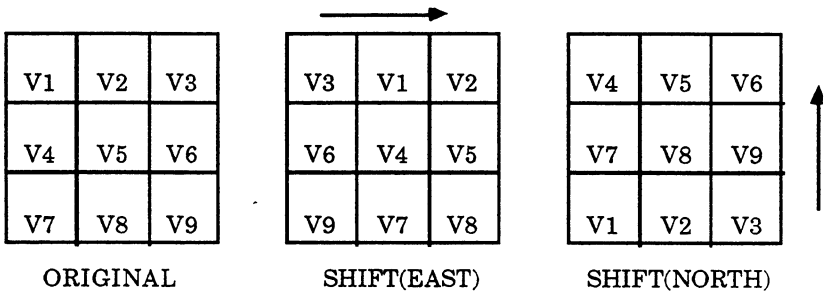


Figure 3.5. Exemples de décalages

Dans les décalages en diagonale les données se déplacent sur un axe qui fait un angle de 45 degrés par rapport à l'horizontale. Il y a quatre sens : nord-ouest, nord-est, sud-ouest et sud-est. Le décalage en diagonale se décompose en deux séries d'opérations. La première consiste à décaler les données dans une direction de base, la deuxième consiste en les décaler dans l'autre direction. Les deux directions de base sont par définition orthogonales. La figure 3.6 illustre ce cas.

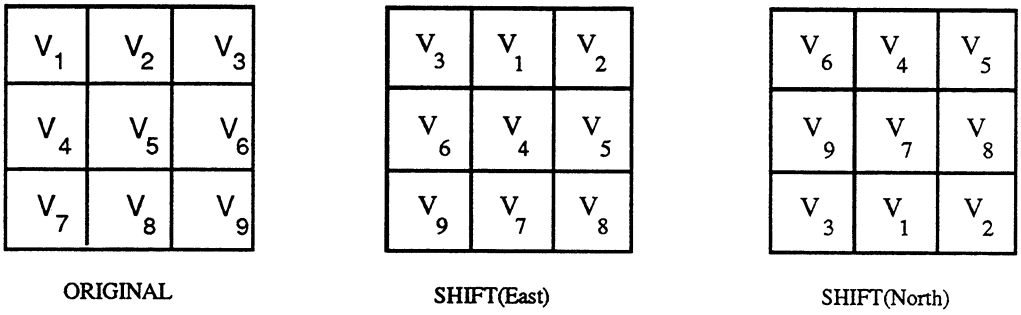


Figure 3.6. Exemples de décalages en diagonale

Le décalage en escalier, **stair-shift**, est une opération de décalage plus élaborée. Considérons une matrice avec N lignes et M colonnes; le décalage droit en escalier de cette matrice, consiste à décaler la première ligne de 0 position, autrement dit ne pas la décaler, la deuxième ligne de 1 position à droite et d'une manière générale décaler la ième ligne de i-1 positions. Quelques exemples de décalage en escalier dans divers sens sont donnés dans la figure 3.7. La description de cette macro est la suivante :

stair_shift(A, B, C) A: l'image à décaler
B: direction du décalage ∈ {Est, West, North, South}
C: pas de décalage ∈ N*

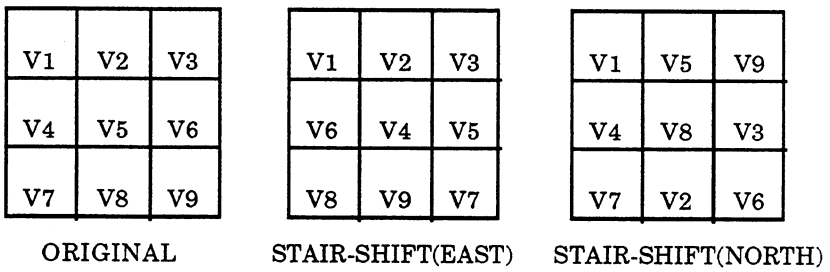


Figure 3.7 Exemples de décalage en escalier

III.2.2.2 Les macros d'entrées/sorties

Les données sont normalement présentes à un bord du réseau. Pour pouvoir entrer les données dans la matrice nous avons défini une macro spéciale. Cette macro est nommée **in**. Cette opération consiste à entrer la première ligne de la matrice dans la première ligne du réseau. Ensuite, cette ligne est décalée. La ligne suivante est entrée pour prendre la place de l'ancienne ligne déjà décalée. Ce processus sera répété jusqu'à ce que la matrice totale soit acquise. Le déroulement, étalé dans le temps et l'espace, de cette opération est illustré par la figure 3.8 et la description de la macro est la suivante :

in (A, B) A: une ligne de l'image à charger
B : nombre de lignes de A

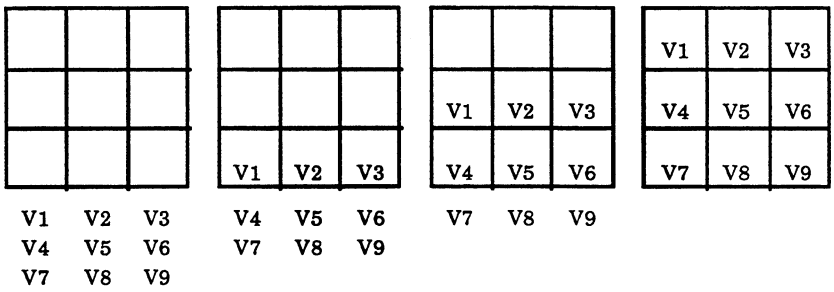


Figure 3.8. Le processus de chargement

III.2.2.3 Les macros de calcul

Toutes les opérations arithmétiques et logiques classiques sont implantées que ce soit sur les types de données classiques ou sur des images. En plus on a doté ce langage d'opérations plus élaborées telles que l'addition en ligne (**add_lin**) ou en colonne (**add_col**).

III.2.3 Exemple de description

Nous donnerons, ici l'exemple du "OU" logique entre deux images. Le principe est illustré en figure 3.9

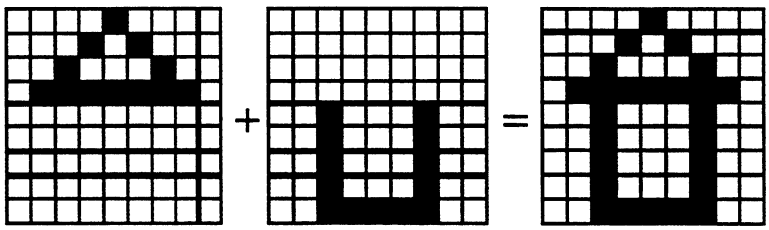


Figure 3.9. Opération "OU" logique entre deux images

```
/* programme pour un "OU" logique entre images A et B
   et récupérer résultat dans C */
main ()
image A(9, 9, 1), B(9, 9, 1), C(9, 9, 1);
{
      C = or (A, B)
}
```

On trouvera en annexe 4 d'autres exemples de description.

III. 3 DEFINITION DE L'ARCHITECTURE CIBLE

Comme il a été dit auparavant, dans le mode SIMD tous les processeurs élémentaires exécutent la même instruction au même moment sous contrôle central. Certains calculs requièrent qu'une partie des PEs seulement exécutent l'instruction courante. Cette possibilité d'autoriser certains PEs seulement à exécuter une instruction est appelée *masquage*. Le masquage se fait en dotant le PE d'un élément de masque, encore connu sous le nom d'élément d'inhibition.

La structure de base est appelée un processeur élémentaire. Un PE est constitué de deux partie essentielles : une partie de communication et une partie pour le calcul. Un PE peut être modélisé par un quadruplet P = <M, A, S, I>, où M représente la partie communication, A l'unité de calcul arithmétique et logique, S la partie mémorisation de données et I l'élément d'inhibition. Ce modèle couvre toutes les réalisations possibles d'un PE. L'ensemble des ressources implantées dans le PE est déterminé par la tâche que doit accomplir le PE.

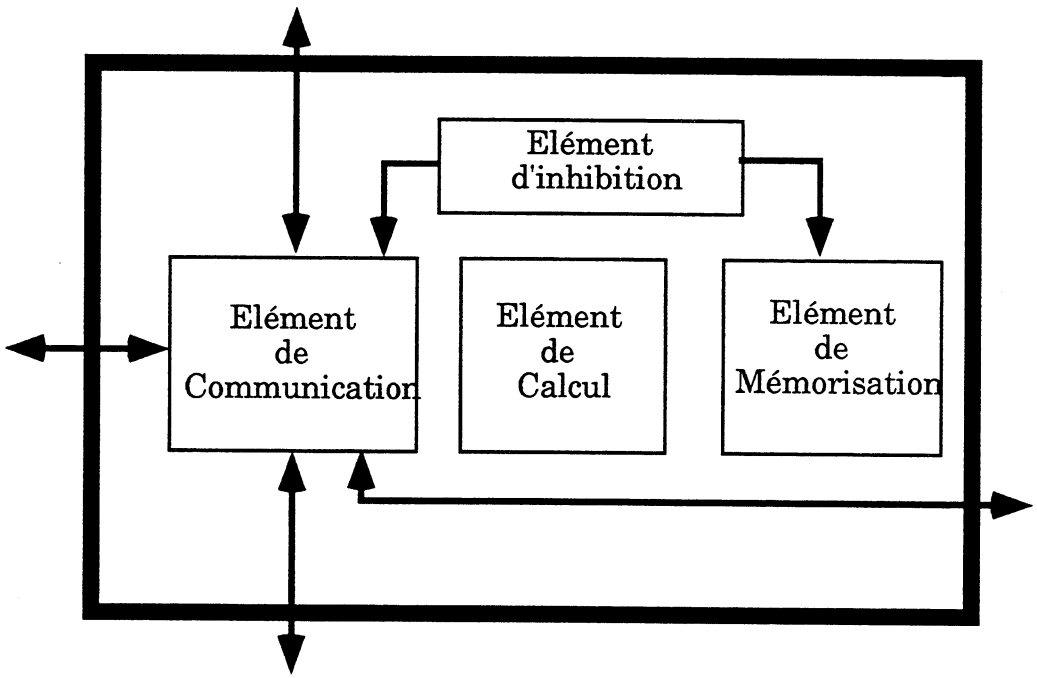


Figure 3.10 Schéma bloc d'un PE générique

III.3.1 Elément de communication

Cet ensemble sert à la communication entre un PE et ses quatre voisins. Il est composé de registres de communication qui peuvent être montés en

registres de décalages. Une partie de cet ensemble peut être utilisée comme registres sources pour la partie calcul ainsi que la partie mémorisation. Un registre, CM, est spécialisé pour la communication au niveau du réseau. On appelle l'ensemble des registres CM, au niveau du réseau, le *plan* CM

III.3.2 Elément de mémorisation

L'élément de mémorisation au niveau processeur élémentaire est une mémoire vive a accès aléatoire (RAM). Elle est constituée de mots de 1-bit et un espace adressable allant de 0 jusqu'à N. La mémoire peut recevoir des données soit de la partie communication soit de la partie calcul. On peut considérer cette mémoire comme une troisième dimension pour le plan contenant la matrice des PEs. Chaque unité sur cette dimension correspond à un plan qui contient le bit de même rang de la mémoire de tous les PEs. Si la mémoire est de taille N×1 bit, on dit qu'on a N plans mémoire (Figure 3.11). Au niveau adressage chaque plan est considéré comme une seule case mémoire.

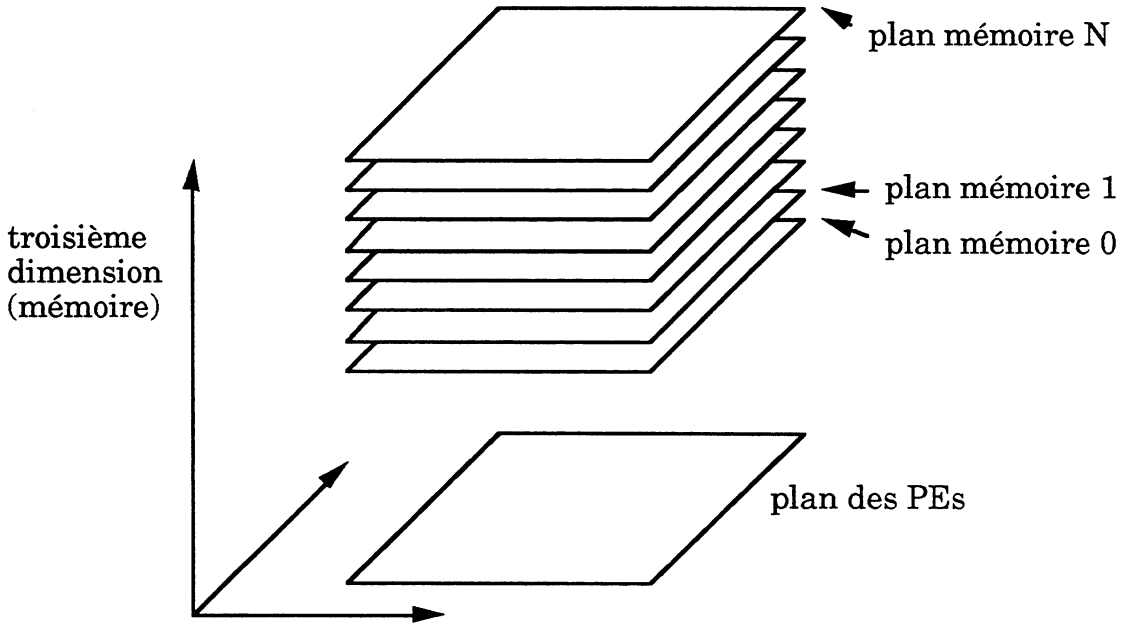


Figure 3.11 Exemple de la mémoire N×1 bits

III.3.2.1 Etude de l'implantation physique

Une étude comparative a été faite entre deux techniques d'implantation physique des élément de mémorisation. La première réalisation utilise le générateur de RAM du logiciel SOLO2030. La deuxième réalisation utilise des bancs de registres. Les résultats sont résumés dans le tableau 3.1, la technologie utilisée est du CMOS 1.5 micron à deux niveaux de métal.

Taille de mémoire (bits)	Bancs de registres (microns^2)	Générateur de RAM (microns^2)
4	364×364	—
8	540.5×540.5	—
16	798.2×798.2	—
32	1130×1130	928×928
64	1584×1584	1187×1187

Table 3.1 Table de comparaison entre le générateur de RAM et les bancs de registre. Les (—) ne peuvent être généré.

A partir des données de ce tableau on peut conclure que si on a besoin de mémoires de taille supérieure à 32×1 bit, il est souhaitable de faire appel au générateur de RAM. Sinon, il convient d'utiliser les mémoires conçues avec les bancs de registres car le générateur de RAM ne dispose pas de mémoire dont la taille est inférieure à 32×1 bit.

III.3.3 Elément de calcul

Cet élément est essentiellement une UAL monobit. Elle peut faire des calculs arithmétiques ou logiques sur des données provenant de la partie communication. Le calcul se fait en série et bit par bit.

III.3.3.1 Etude comparative des différentes UALs

L'UAL est un circuit combinatoire qui selon les valeurs d'un certain nombre de signaux de commande effectue les opérations arithmétiques et logiques de base. La figure 3.12 montre un schéma d'une UAL de 1 bit; a_i et b_i représentent les entrées, c_i la retenue entrante et c_o la retenue sortante, F_i le résultat de l'opération et enfin *com* les commandes de sélection.

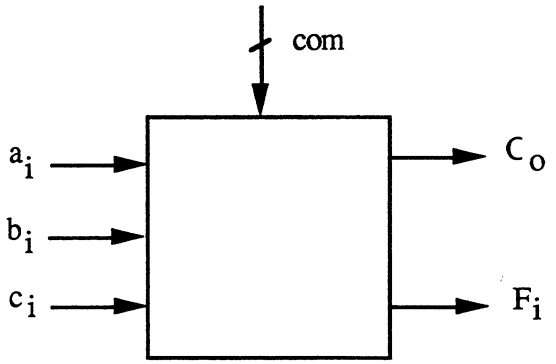


Figure 3.12 Schéma bloc d'une UAL à un bit

Il y a au total 16 opérations logiques différentes de deux variables. Pour les opérations arithmétiques, l'addition, la soustraction, l'incrémentation et la décrémentation constituent l'ensemble de base considéré ici.

La Table 3.2 donne la liste des opérations arithmétiques et logiques qu'une UAL doit effectuer dans notre approche.

OPERATION	EQUATION LOGIQUE
ADD (a_i PLUS b_i PLUS c_i)	$F_i = a_i \oplus b_i \oplus c_i$ $c_o = (a_i b_i) + (a_i + b_i)c_i$
SUB (a_i MOINS b_i MOINS c_i)	$F_i = a_i \oplus \overline{b_i} \oplus c_i$ $c_o = (a_i \overline{b_i}) + (a_i + \overline{b_i})c_i$
INC (a_i PLUS 1)	$F_i = a_i \oplus c_i$ $c_o = a_i c_i$
DEC (a_i MOINS 1)	$F_i = \overline{a_i} \oplus c_i$ $c_o = \overline{a_i} c_i$
NOT	$F_i = \overline{a_i}$
AND	$F_i = a_i b_i$
OR	$F_i = a_i + b_i$
NAND	$F_i = \overline{a_i b_i}$
NOR	$F_i = \overline{a_i + b_i}$
XOR	$F_i = a_i \oplus b_i$
XNOR	$F_i = \overline{a_i \oplus b_i}$

Table 3.2

III.3.4 Elément d'inhibition

Cet élément sert à bloquer le PE en cas de besoin. Le blocage se traduit par l'inhibition du chargement de la mémoire et de la partie communication. Dans ce cas, le PE conserve son état précédent.

III. 4 SYNTHESE AUTOMATISÉE

III.4.1 Génération du graphe de dépendance des données

Cette phase est divisée en deux étapes : une étape de décomposition et une étape de compilation.

III.4.1.1 étape de décomposition

Cette étape, résumée sur la figure 3.13, décompose le fichier *source* en deux fichiers différents l'un contenant uniquement le code écrit en C "*source.c*" et l'autre les déclarations et les instructions images "*source.image*".

Il est à noter que dans le fichier "*source.c*" les déclarations image ont été remplacées par des blancs tandis que les instructions image ont été remplacées par une instruction C.

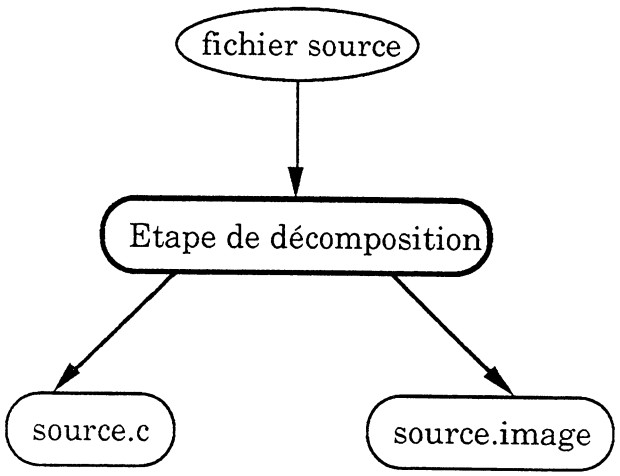


Figure 3. 13 : Etape de décomposition

III.4.1.2 étape de compilation

C'est l'étape de compilation proprement dite. On commence d'abord par compiler le fichier "*source.image* " avec un analyseur syntaxique et sémantique. Si aucune erreur n'est détectée on passe à la compilation du fichier "*source.c* ".

La première phase consiste à vérifier la syntaxe des déclarations et instructions images en appliquant les règles de grammaire qui sont données en

annexe. La méthode d'analyse utilisée est la descente récursive [Aho86]. Une table des symboles est créée lors de cette phase et contient toutes les informations disponibles à ce niveau sur les variables images déclarées. Un arbre syntaxique est généré tout au long de cette phase.

L'analyseur sémantique prend en entrée l'arbre syntaxique et la table des symboles fournis par la phase précédente, et vérifie si les instructions images sont sémantiquement correctes.

Enfin, le fichier "*source.c* " est complètement pris en charge par le compilateur C.

III.4.1.4. Résultat de la compilation

Une fois la phase de compilation du fichier source achevée, on obtient la séquence d'opérations que le PE devra exécuter. Cette description servira à extraire le graphe de dépendance de données.

Après les deux étapes de compilation le système passe à une autre étape : la génération du graphe de dépendance de données correspond à la séquence des instructions image. Chaque nœud du GDD représente une opération image (un nœud est une macroinstruction). Ce graphe va ensuite subir une expansion qui consiste à remplacer chaque nœud par le GDD correspondant. En effet, chaque macroinstruction est représentée par un GDD traduisant la séquence des instructions élémentaires la constituant. C'est ce graphe qui va être passé à l'étape d'allocation de registres. Cette dernière étape sera détaillée un peu plus loin.

III.4.1.2 Création de l'UAL

Notre but ici est de générer une UAL optimisée requise par l'application ; l'UAL doit être minimisée en surface et avoir de très bonnes performances temporelles. Dans ce but, elle sera dédiée et exécute uniquement les opérations requises par l'application.

Il y a deux approches possibles pour synthétiser une unité arithmétique et logique. Dans la première, une UAL générique est utilisée et si besoin est, on procède à son optimisation. La deuxième part d'équations booléennes ou d'une table de vérité et utilise un outil de synthèse de logique combinatoire pour une réalisation en cellules standards par exemple.

C'est la première approche que nous avons retenue et ceci pour deux raisons. Premièrement, pour le traitement d'image de bas niveau toutes les opérations arithmétiques et logiques ne sont pas en général nécessaires. Deuxièmement, les signaux de contrôle qui spécifient l'opération à exécuter

doivent être routés sur une grille, et donc atteindre chaque PE dans le réseau. De telles connexions occuperaient trop de surface, c'est pour ceci que nous allons définir une architecture d'UALs contrôlée par les données. En fixant une ou plusieurs des entrées de données disponibles, une opération est sélectionnée.

L'ensemble restreint d'opérations qui est utilisé dans le traitement d'image de bas niveau est résumé dans la Table 3.3. Cet ensemble est implémenté sur l'architecture de la figure 3.14. Le contrôle par les données est explicitement montré dans la colonne 'condition', de la Table 3.3, indiquant l'entrée forcée pour exécuter une opération.

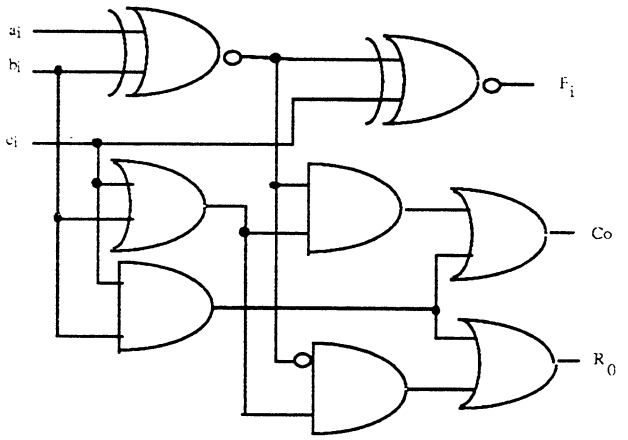


Figure 3.14: Schéma logique de l'UAL générale.

Par exemple, pour obtenir le OR des contenus des registres NS et EW, il faut positionner c_i à 1 et récupérer le résultat sur C_O.

Opération	Description	Condition
NOT	$C_O = \overline{a_i}$	$b_i = 0, c_i = 0$
AND	$C_O = a_i b_i$	$c_i = 0$
OR	$C_O = a_i + b_i$	$c_i = 1$
XOR	$F_i = a_i \oplus b_i$	$c_i = 0$
XNOR	$F_i = \overline{a_i \oplus b_i}$	$c_i = 1$
ADDC	$F_i = a_i$ PLUS b_i PLUS c_i	$c_i = C_O$
SUBC	$F_i = a_i$ MINUS b_i MINUS c_i	$c_i = R_O$

Table 3.3

Exemple de génération d'une UAL dédiée

Nous avons identifié trois sous-catégories de l'UAL générale décrite selon les opérations mises en jeux. La première sous-catégorie est un additionneur complet. Si, nous devons implémenter uniquement le NOT, XOR, et XNOR, le circuit que montre la figure 3.15(a) est utilisé. Si les opérations sont uniquement des OR et AND, le circuit montré en figure 3.15(b) est employé.

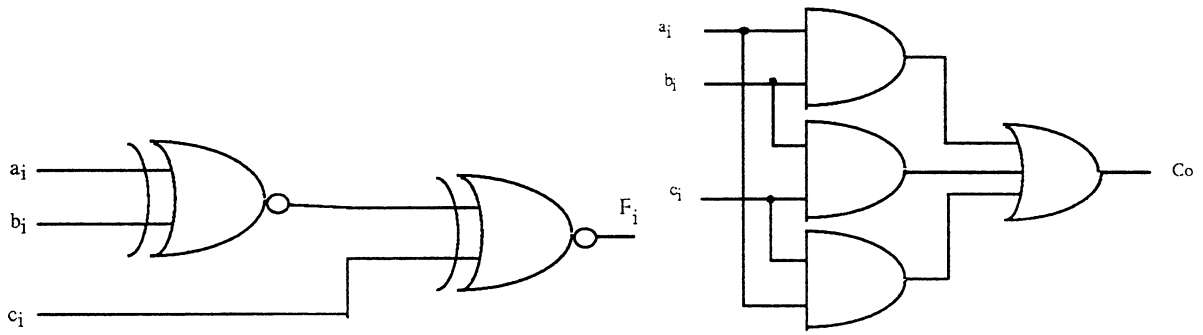
L'outil de synthèse analyse les spécifications de la manière suivante :

1) Si un OR et AND est rencontré, l'UAL III est utilisée,

2) Si un NOT, XOR, et XNOR est rencontré alors l'UAL II est utilisée,

3) Si les opérations précédentes plus l'addition sont rencontrées, l'UAL I est utilisée,

4) Si les toutes les opérations sont utilisées alors l'UAL générale est générée.



(a) (b)

Figure 3.15 Deux UAL dédiées (a) l'UAL II, (b) l'UAL III

La Table 3.4 donne les surfaces normalisées des différentes UALs. Si, par, exemple, une soustraction n'est pas nécessaire, le gain par rapport à l'UAL générale est de 18%, tandis qu'il sera de 48% si l'UAL III est implémentée.

UAL	Surface
UAL générale	1
UAL I	0.82
UAL II	0.78
UAL III	0.52

Table 3.4

III.4.2 Allocation de ressources

L'allocation de ressources permet d'associer les objets manipulés par la description à des ressources matérielles. De ce fait chaque élément de mémorisation est lié à une unité de mémorisation matérielle (registre, bloc mémoire, ...). Chaque opération est liée à une unité capable de l'exécuter. Chaque transfert de données est liée à un chemin de communication (bus ou multiplexeur). Ces éléments peuvent être déclarés explicitement dans la description d'entrée et/ou rajoutés par d'autres étapes du processus de compilation .

L'allocation vise à minimiser certains objectifs sous différentes contraintes telles que la surface totale du circuit, le délai des entrées / sorties ou le temps de calcul. Ces objectifs peuvent être les suivants :

- Minimiser l'ensemble des interconnexions.

- Minimiser le nombre de registres, de bus et de multiplexeurs.

McFarland *et al.* [McPC88] recensent deux classes d'algorithmes d'allocation : les algorithmes itératifs/constructifs, et les algorithmes globaux .

Dans la première classe, on cherche à améliorer itérativement une solution de départ. Ces améliorations sont réalisées via des transformations dont l'objectif est de réduire la surface et/ou augmenter la vitesse du circuit. L'efficacité de ce type de méthode dépend énormément de la solution initiale.

Pour les algorithmes constructifs, il s'agit de choisir une ressource, d'effectuer l'assignation et de réitérer ces deux phases tant qu'il existe des éléments à assigner. Le résultat de ces algorithmes dépend de l'ordre dans lequel les éléments sont traités.

La deuxième classe utilise des techniques de théorie de graphes et programmation mathématique. Une méthode classique [TsSi86] commence par la construction d'un graphe dont les nœuds reliés par un arc représentent les éléments de la description qui peuvent partager la même ressource. L'obtention d'une solution optimale revient à trouver une couverture comportant le nombre minimum de cliques disjointes.

Un ordonnancement préalable des opérations ayant été réalisé, la phase d'allocation se résume en une phase d'assignation de ressources et une phase de génération de l'UAL.

III.4.2.2. Allocation de registres

Ce problème a fait l'objet de nombreuses publications. Citons deux algorithmes, les plus représentatifs et les plus cités, celui de FACET [TsSi88] et celui de REAL [KuPa87]. Ces algorithmes donnent des résultats qui sont le plus proche possible de l'optimum en termes de nombre de registres.

Tseng [TsSi86] est parti de l'idée suivante : deux variables sont assignées à la même entité physique si et seulement si elles ne sont jamais utilisées en même temps. De là, il construit un graphe où les variables sont des nœuds et il existe une arête entre deux nœuds si et seulement si les deux nœuds vérifient la condition précédente. Ce graphe sera ensuite partitionné en cliques maximales et les nœuds de chacune de ces cliques seront affectés au même registre. Bien que le problème de partitionnement d'un graphe en cliques soit NP complet, Tseng a proposé un algorithme polynomial en temps qui donne d'assez bon résultats. Cependant, les essais de réalisations par cette méthode [Jerr89] ont montrés que cette technique est très coûteuse en temps de calcul.

Il est à noter que Tseng traite avec la même technique le problème d'allocation des opérateurs et celui de l'interconnexion des unités d'un circuit [TsSi86].

Kurdahi [KuPa87], dans son approche qui est plus simple et qui donne de très bon résultats, traite le problème différemment. Partant d'un graphe de dépendance de données dont les opérations ont été ordonnancées préalablement, et une table contenant les durées de vie de chacune des variables du graphe, il cherche une affectation, optimum, des registres aux variables. Le principe est basée sur un problème classique d'affectation de chemins dans le routage de canaux [HaSt71]. Cet algorithme, bien que glouton, donne de très bon résultats. En effet, ses résultats sont optimaux dans le cas de graphes sans branchement et assez bons pour ceux avec branchements [KuPa87].

Dans notre cas, nous avons à résoudre un problème d'allocation de registres beaucoup plus simple avec des contraintes spécifiques liées à l'architecture des PEs. Nous avons donc choisi d'appliquer un algorithme décrit dans [MiSa91] à l'architecture cible spécifique des PEs. Cette algorithme a l'avantage d'être basée sur la définition de règles spécifiques à l'architecture cible. Ces règles lient les éléments de stockage virtuelles et les registres. L'allocation est faite en résolvant un problème de couplage de poids maximum sue chaque pas de contrôle du GDD.

L'allocation de registres vise à minimiser d'une part le nombre de plans mémoires et donc une utilisation prioritaire des registres NS, EW (règle 1) puis les registres D0 (règle 2)... et d'autres part le nombre d'opérations de copie et donc l'utilisation du même registre avant et après une copie (règle 3 et 4).

Nous décrivons ces règles par un triplet comprenant la situation où s'applique la dite règle, une liste des candidats possible et un poids pour

• **Règle 1**

Situation : S_i est une entrée/sortie de l'UAL

Candidat : EW ou NS

Poids : 3

• **Règle 2**

Situation : S_i est une entrée (resp. une sortie) de l'UAL

Candidat : registre différent de EW ou NS et déjà assigné en entrée (resp. en sortie) de l'UAL

Poids : 2

• **Règle 3**

Situation : $S_i\ (c_i) := S_i\ (c_{i-1})$

Candidat : registre assigné à $S_i\ (c_{i-1})$ s'il existe

Poids : 3

• **Règle 6**

Situation : $S_i\ (c_{i+1}) := S_i\ (c_i)$

Candidat : registre assigné à $S_i\ (c_{i-1})$ s'il existe

Poids : 3

L'algorithme général d'affectation de registres est le suivant.

Début

Pour chaque pas de contrôle;

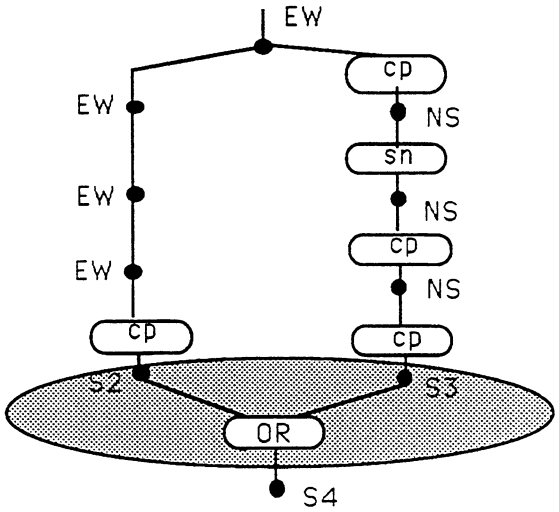
Appliquer les Régles ;

Couplage de poids maximum sur graphe biparti;

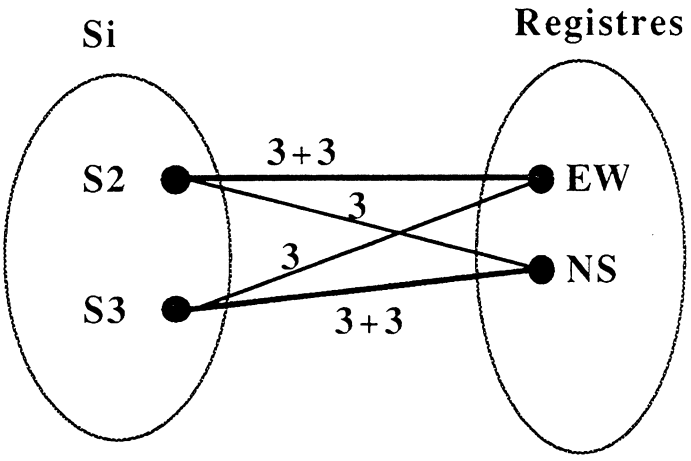
Finpour

Fin

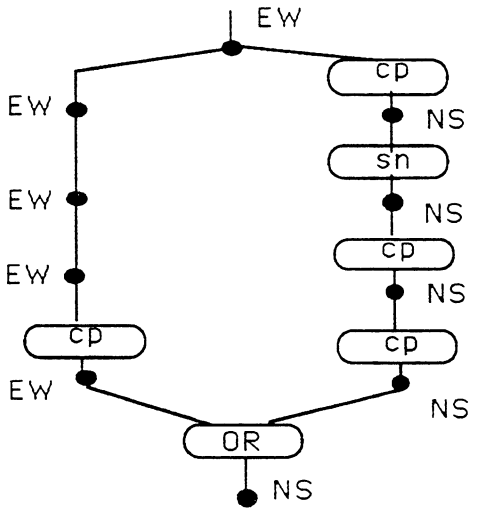
L'application de cet algorithme à un sous-graphe du GDD de la dilatation est illustrée par la figure 3.16. Le résultat de l'application de ces règles au GDD de la dilatation au pas de contrôle encerclé est donné par la figure 3.16.(a).et (b). Le résultat du couplage maximum implique l'assignation donné en figure 3.16 (c). Nous nous intéressons plus spécifiquement aux deux entrées du "OR" S2 et S3. L'ensemble Si qui contient {S2, S3} et l'ensemble qui contient les registres {NS, EW} constituent les deux sous-ensembles du graphe biparti et les arêtes (pondérés) correspondent aux règles. Il en résulte que S2 est attirée par EW et S3 par NS.



(a)



(b)



(c)

Figure 3.16 Résultat de l'assignation de registres. (a) sous graphe, (b) le graphe biparti correspondant, (c) le résultat du couplage maximum

III.4.3. Optimisation de l'ordonnancement

Après l'étape d'allocation de ressources il peut résulter des pas de contrôle contenant des opérations qui consistent en la copie d'un registre dans lui même. Ces "pas de contrôle" sont donc à éliminer en vue d'optimiser l'ordonnancement des opérations. Cette optimisation consiste en l'analyse de chaque chemin du contrôle du graphe. Chaque chemin de contrôle est parcouru du début jusqu'à la fin si deux opérations de stockage sont réalisées sur le même registre à la suite

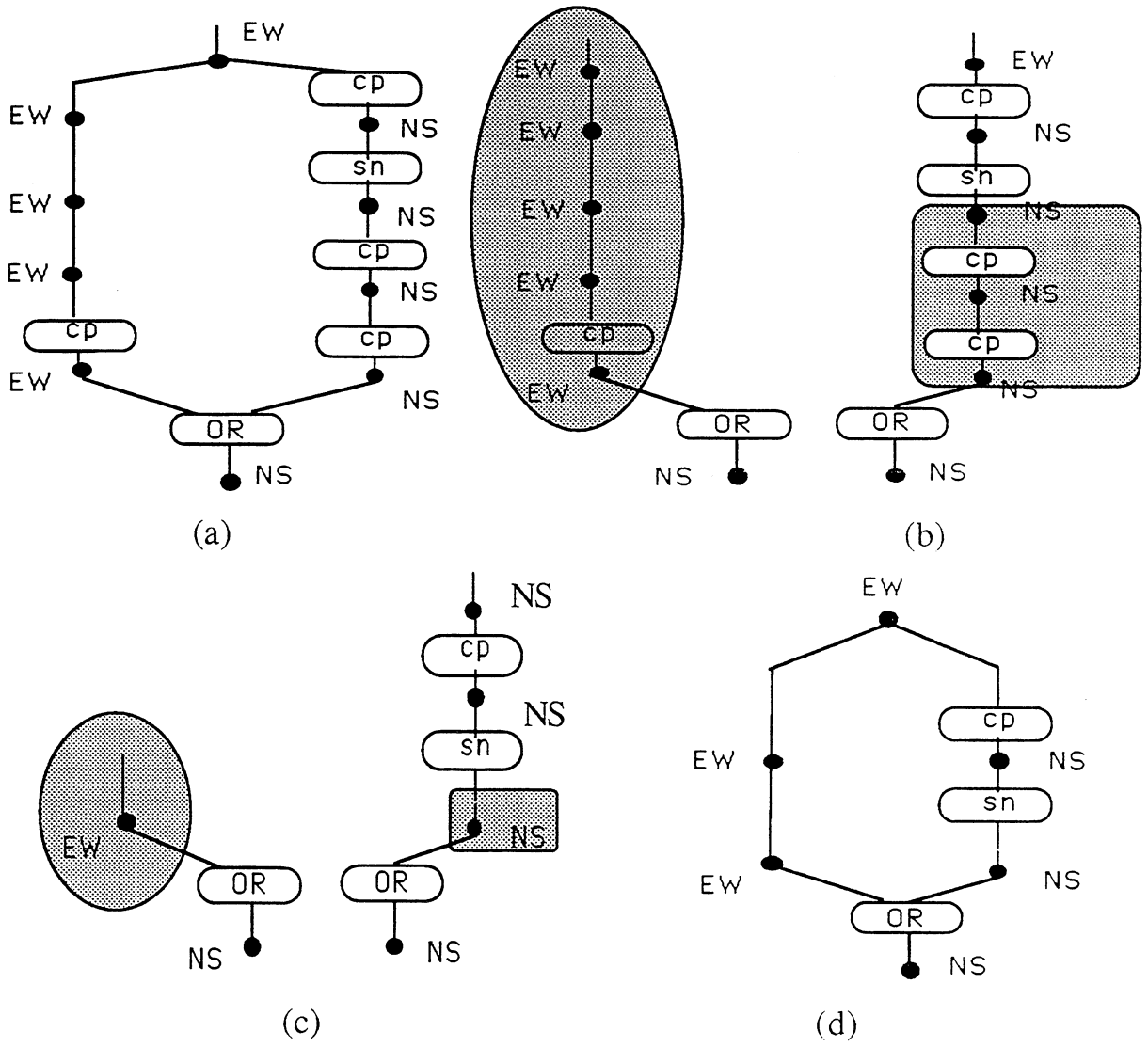


Figure 3.17 Etapes d'optimisation de l'ordonnancement. (a) original, (b) extraction des chemins, (c) compaction et (d) équilibrage.

l'une de l'autre ou séparés par une instruction de copy ("cp") ces deux opérations sont compactées pour n'en faire qu'une seule ce qui conduit à la compaction de nœuds de stockage identiques. Cette étape d'optimisation est illustrée par la figure 3.17 (a), (b) et (c). Puis une étape de décomposition permet d'harmoniser

la longueur des différents chemins mutuellement exclusifs, tel que l'illustre la figure 3.17 (d)

III.4.4. Génération de Multiplexeurs

L'outil de synthèse, tout en allouant les ressources, associe à chaque ressource une variable qui compte le nombre d'entrées de cette ressource spécifique. La valeur de cette variable-compteur est utilisée pour déterminer la taille du multiplexeur associé. Les lignes de sélection d'un multiplexeur sont calculées d'une manière très simple. Si le nombre d'entrées est n alors le nombre de lignes de sélection est $\lceil \ln_2 (n) \rceil$, où $\lceil\ \rceil$ représente le plus petit entier supérieur. Dans l'exemple précédent NS a été sollicité trois fois et donc le nombre de lignes de sélection du multiplexeur associé à NS est de 2.

III.4.5.Génération de Signaux de Contrôle

Une fois la partie opérative entièrement définie, les signaux de contrôle sont obtenus de manière très aisée. Le code généré est stocké dans une mémoire morte. Un simple compteur permet de le dérouler.

III.4.6 Génération du Circuit

Une fois toutes les étapes de synthèse terminé, à savoir l'assignation de registres, la détermination des tailles de multiplexeurs ainsi que le type de l'UAL, l'architecture du PE est terminée. Pour le cas de la fermeture, le PE synthétisé contient deux registres NS et EW. Du fait que seulement deux opérations, à savoir le ET et le OU, l'UAL III est utilisée. Les sorties de NS, EW sont connectées à NS et EW tandis que C reçoit en entrée un 0 ou un 1 logique. Le schéma logique du PE de la fermeture est donné en figure 3.18.

La taille de la matrice à générer est donnée dans la description au niveau algorithmique. En général, cette taille ne peut pas être intégrée sur un seul circuit, sauf si on fait appel aux techniques de l'intégration tranche entière. Il faut donc partitionner la matrice en plusieurs circuits. Le critère de choix le plus déterminant est la taille maximale de la puce ("die"). Il suffit pour une première approximation de diviser la surface de la puce par la surface du PE pour obtenir le nombre de PE par chip. Ce nombre est pris comme borne supérieure. Le nombre qui sera réellement implémenté sera légèrement inférieur.

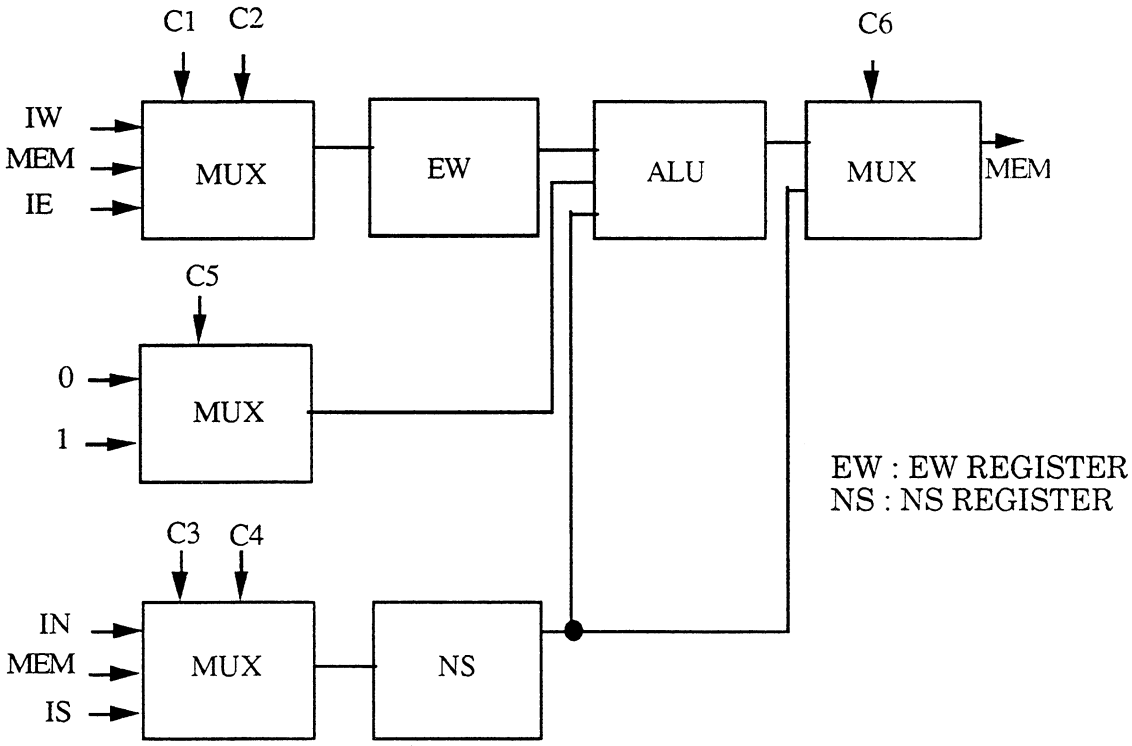


Figure 3.18 Schématique d'un PE (fermeture)

III.4.7 La Chaîne de CAO Cible

Nous avons pris comme première cible la chaîne de CAO de CADENCE avec une bibliothèque de cellules standards fourni par ES2. Le choix de cette chaîne a été motivé par deux raisons. La première est sa disponibilité sur pratiquement tous les sites universitaires. La deuxième raison est son ouverture. En effet, ce système que nous proposons peut être greffé aisément dans cette chaîne. CADENCE donne un moyen (un langage de programmation) qui sert a mettre ceci en œuvre. Toutefois, la bibliothèque de cellules standards fourni par ES2 ne contient pas d'UAL à 1 bit, une conception spécifique a été menée à bien.

III.4.8 Résultats Expérimentaux

Pour valider notre approche, nous avons conçu plusieurs circuits, à savoir un circuit pour la fermeture ou l'ouverture, un pour le filtrage de Sobel, et un pour la détection de contour. Les algorithmes de ces différentes applications sont données en Annexe 4. Nous avons maintenu, à des fin de comparaison, la taille de tous ces circuits à 6x12 PEs. En effet, le circuit qui nous sert de référence est le GAPP qui en contient le même nombre. Le circuit de la fermeture est dix fois moins petit que le GAPP en surface et il est deux fois et demie plus rapide. Le reste des résultats est donné dans la table 3.

PE	Surface	Vitesse
Fermeture	0.11	2.5
Detec_Contour	0.16	2.5
Sobel	0.32	2.5
GAPP	1	1

Table 3.

Les layouts de ces circuits sont données en Annexe 4.

CONCLUSION

Cette thèse a exposé les méthodes de conception d'architectures massivement parallèles pour le traitement d'image de bas niveau. Dans la premier partie, l'attention a porté sur la faisabilité de circuits de très grande dimension. L'architecture répétitive du réseau 2D de processeurs monobits facilite la construction de réseau cible en évitant les processeurs et connexions défaillants. Si les techniques consistant à introduire une colonne redondante au niveau des sous-matrices peut être considérée comme relativement banale, la conception du réseau de commutateurs lui-même tolérant aux défauts est originale. En effet, dans la littérature de la dernière décennie, les études de techniques de reconfiguration et configuration foisonnent mais les réseaux de commutateurs permettant d'implanter des techniques sont restés abstraits ou peu réalistes. Aucune réalisation consistante n'a été proposée. Le réseau présenté ici a l'avantage d'être parfaitement indépendant de l'architecture des PEs et sa programmation à partir de plots d'entrée/sortie le rend facilement utilisable en particulier en ligne. L'originalité de cette thèse a donc consisté à donner une réalité physique à des études papiers et cela a permis de montrer que le surcroît en surface de silicium est réduit. Des études actuelles concernent l'utilisation de ce réseau de commutateurs en tant que réseau de connexion. Il permet de réaliser de façon programmable, tous les types de réseaux de connexions (crossbar, oméga, ...). Des applications dans le domaine des télécommunications et des architectures multiprocesseurs semblent être prometteuses. Pour ce qui concerne la faisabilité industrielle de circuits intégrés tranche entière, elle est loin d'être démontrée. Les coûts de conception et de réalisation rendent une réalité industrielle imminente peu probable. Il s'agit plutôt d'une démonstration à grande échelle des techniques de tolérance aux défauts.

Cette difficulté jointe à la lourdeur et au coût d'un environnement programmable ont conduit à la deuxième partie de cette thèse. A partir de l'expérience acquise en conception de circuits SIMD réalisés par une matrice de processeurs monobits, l'élaboration d'un outil de synthèse automatisé de réseaux dédiés à partir d'une spécification de haut niveau a semblé intéressante. Les circuits sont cette fois dédiés, sur un contrôleur figé, et leur surface et leur performance optimisés. Ces circuits qui sont faciles à concevoir et à réaliser, couvrent le domaine de traitement d'image de bas niveau. Des primitives de déplacement et traitement d'image rendent l'usage de l'outil de synthèse très convivial. Sa réalisation est à l'état de prototype. Le point résistant de cette étude et qui n'a pas été abordé dans cette thèse est la bonne gestion des entrées et sorties

car il est bien connue que rien ne sert de proposer un dispositif de traitement rapide si l'alimentation de données est le point dur.

Néanmoins, cette thèse a permis de bien cerner les possibilités et les limites des architectures programmables ou dédiés du SIMD adaptés au traitement d'image de bas niveau.

BIBLIOGRAPHIE

[Aho86] A. Aho, R. Sethi and J. Ullman, Compilers : Principles, Techniques and Tools. Addison-Wesly, 1986.

[BaAl85] T. Blackman *et al.*, "The SILC Silicon Compiler : Language and Features," *Proceedings of the 22nd Design Automation Conference*, pp. 232-237, Jun. 1985

[Bars77] H. Barshun, "Functionnal Wafer — a New Step in LSI," *European Solid State Conf.*, Ulm, Germany, pp. 79-80, 1977

[Batc68] K. Batcher, "Sorting Networks and their Appllications," in *Proc. AFIPS-Spring Joint Comp. Conf.*, Vol. 32, pp. 307-314, 1968

[Batc74] K. Batcher, "STARAN Parallel Processor System Hardware," *AFIPS 1974 National Comp. Conf.*, pp.405-410, May. 1974

[Batc80] K. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. on Computers*, Vol.C-29, N° 9, pp. 836-840, May. 1980

[Batc82] K. Batcher, "Bit-Serial Parallel Processing Systems," *IEEE Trans. on Computers*, Vol. C-31, N° 5, pp. 377-384, May. 1982

[BBKK68] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick and R. Stokes, "The ILLIAC IV computer," *IEEE Trans. on Computers*, C-17(8), pp. 746-757, Aug. 1968

[BeDH88] D. Belvins, E. Davis, R. Heaton, and J. Reif, "Blitzen : A Highly Integrated Massively Parallel Machine," *Proc. 2nd Symp on the Frontiers of Massively Parallel Computation*, Fairfax (VA), pp. 373-381, Oct. 1988

[Benn85] E. Bennet, " Inova Makes its Move in Risky Wafer Scale Work," *Electron Des.*, pp. 57-58, Dec. 1985

[BoRa91] R. Boppana and C. Raghavendra, "Generalized Schemes for Access and Alignment of Data in Parallel Processors with Self-Routing Interconnection Networks," *Journal of Parallel and Distributed Computing* 11, pp. 97-111, 1991

[Bouk72] N. Bouknight *et al.*, "The ILLIAC IV System," *Proc. of the IEEE*, Vol. 60 , pp. 369-388, Apr. 1972

[Calh69] D. Calhoun, "The Pad Relocation Technique for Interconnecting LSI Arrays of Imperfect Yield," *AFIPS-Fall Joint Computer Conference*, pp. 99-109, 1969

[ChCh90] Y. Chen, W. Chen, G. Chen and J. Shew, "Designing Efficient Parallel Algorithms on MCC with Multiple Broadcasting," *Journal of Parallel and Distributed Computing* 11, pp. 241-246, 1991

[Clou88] E. Cloud, "The Geometric Arithmetic Parallel Processor," *The 2nd Symp on the Frontier of Massively Parallel Computation*, Fairfax (VA), pp. 373-381, Oct. 1988

[Cohe84] C. Cohen "Full-wafer Memory for Color Displays Has 1.5 Mbit Capacity," *Electronics*, pp. 77-78, Jan. 1984

[DaTh84] R. Davis and D. Thomas, "Systolic Array Chip Matches the Pace of High-Speed Processing," *Electronic Design*, pp. 42-50, Oct. 1984

[DeRe85] P. Denyer and D. Renshaw, VLSI Signal Processing : A Bit-Serial Appraoch. Reading, MA : Addison- Wesley, 1985

[Duff73] M. Duff, "A Cellular Logic Array for Image Processing," *Pattern Recognition,* Vol. 5, pp.229-247, 1973

[Dunc90] R. Duncan, "A Survey of Parallel Computer Architectures," *Computer,* pp. 5-16, Feb. 1990

[FiLo77] C. Finnila and H. Love, "The Associative Linear Array Processor," *IEEE Trans. on Computers*, C-26, N° 2, pp 112-115, Feb. 1977

[Flan82] P. Flanders, "Data Mouvement on an Array Processor," *IEEE Trans. on Computers*, C-31(9), pp. 808-819, Sep. 1982

[Flyn66] M. Flynn, "Very High Speed Computing Systems," *Proc. IEEE*, Vol 54, pp. 1901-1909, 1966

[Fox83] J. Fox, "The MacPitts Silicon Compiler : A View from the Telecommunication Industry," *VLSI Design*, Vol. IV(3), pp. 30-37, 1983

[Gand87] S. Gandemer, *Modélisation de l'impact des Défauts de Fabrication sur le Rendement des Microcircuits Intégrés Fabriqués en Technologie Silicium.* Thèse présentée a l'Ecole Nationale Supérieure des Télécommunications, Sept. 1987

[GiKn84] E. Girczyc and J. Knight, "AN ADA to Standard Cells Hardware Compiler Based on Graph Grammars and Scheduling," *Proc. of ICCD*, pp. 726-731, Oct. 1984

[Harb89] J.R Harbridge, "ELSA" *Proc. IFIP Workshop on Parallel Architectures on Silicon,* Grenoble, Dec 89

[HaSt71] A. Hashimo and J. Stevens, "Wire Routing by optimizing Channel Assignment Within Large Apertures," *Proc. 8th DA workshop*, pp. 155-169, 1971

[Heat89] R. Heaton and D. Blevins, "The Architecture and Design of the Blitzen Parallel Processor," *Proc. IFIP Workshop on Parallel Architectures on Silicon,* Grenoble, Dec. 89

[HiTh83] C. Hitchcok and D. Thomas, " A Method of Automatic Data Path Synthesis," *Proceedings of the 24th Design Automation Conference*, pp. 484-489, 1983

[HoHe63] S. Hofstein and F. Heiman, "The Silicon Insulated-Gate Field Effect Transistor," *Proc. IEEE*, Vol. 51, pp. 1190-1202, Sep. 1963.

[Hunt76] J. Hunter, "Database Retreival Using Superchip," *Symposium Advanced Memory Concepts*, S. Miller and U. Gagliardi, Eds, SRI Menlo Park, CA, pp. 450-470, Jun.1976.

[JaWL89] I. Jalowiecki, K. Warren and R. Lea, "WASP : A WSI Associative String Processor," *Proc. of the Wafer Scale Integration*, C. Jesshope and W. Moore, Eds, Adam Hilger, pp. 83-93, 1985

[Jerr89] A. Jerraya, *Contribution à la compilation de silicium et au compilateur SYCO*. Thèse présentée à l'université Joseph Fourier (INP) de Grenoble 1989.

[Kita80] Y. Kitano, S. Kohda, H. Kikuchi, and S. Sakai, "A 4 Mbit Full-wafer ROM," *IEEE Trans. Electron Devices*, Vol. ED-27, pp. 1621-1628, Aug. 1980

[KuPa87] F. Kurdahi and A. Parker "REAL : A Program for Register Allocation," *Proceedings of the 24th Design Automation Conference*, pp. 210-215, Jun.1987

[Lass90] S. Lassere, *Redondance et Reconfiguration Pour L'augmentation du Rendement à la Fabrication: Influence de la Variété Architecturale.* Thèse présentée à l'Université de Paris sud, centre d'Orsay, Jan. 1990.

[LaVo82] D. Lawrie and C. Vora, "The Prime Memory System for Array Access," *IEEE Trans. on Computers*, C-31(5), pp. , May. 1982

[Lea91] R. Lea, "Comparing Monolitic and Hybrid WSI Massively Parallel Processing Modules for Cost-effectiveness Real-time Signal and Data Processing," *Inter. Conf on WSI*, pp. 199-206, Jan. 1991

[LeMN82] S. Levialdi, A.Maggiolo-Schettini, M. Napoli, G. Tortora and G. Uccella, "On the Design and Implementation of PIXAL, a Language for Image Processing," in M. Duff and S. Levialdi, edit., Languages and Arcitectures for Image Processing, Academic Press, New York, NY., pp. 88-98, 1982

[Lesk88] M. Lesk and E. Schmidt, "Lex - A Lexical Analyser Generator," Sun Release 4.1 , pp. LEX-1- LEX- 13,.1988

[LiBC89] J. Little, G. Blelloch and T. Cass, "Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 11, N° 3, pp. 244-257, Mar. 1989

[Magg83] A. Maggiolo-Schettini "Comparison of some High-level Languages for Image Processing," in M.G. Duff and S. Levialdi, Eds., Languages and Arcitectures for Image Processing, Academic Press, New York, NY., pp. 157-164, 1982

[McCo63] B. McCormick, "The Illinois Pattern Recognition Computer—ILLIAC III," IEEE Trans. on Electronic Computers, Vol. 12, pp. 791-813, 1963

[McPC88] M. McFarland, A. Parker and R. Camposano, "Tutorial on High-level Synthesis," *Proc of the 25th Design Automation Conference*, pp. 330-336, 1988

[MiCr91] A. Mignotte and M. Crastes, "Resource assignment with different target architecture", *EURO ASIC 91*, Paris, France, pp. 172-177, May 1991

[Moor85] W. Moore, "A Review of Fault-Tolerant Techniques for the Enhancement of Intgrated Circuit Yield," *Proceedings of the IEEE*, Vol. 74, N°5, pp 684-698, May. 1986

[MoWa87] D. Walker, *Yield Simulation for Integrated Circuits*. Boston: Kluwer Academic Publishers, 1987

[Murp64] B. Murphy, "Cost-size Optima of Monolithic Integrated Circuits," *Proc. IEEE*, Vol. 52, pp. 1537-1545, Dec. 1964.

[NaSa81] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD Computers," *IEEE Trans. on Comp.*, vol. C-30, no. 2, pp. 101-107, 1981

[Nasr88] B. Nasreddine, *Conception d'une Mémoire Reconfigurable Intégrée sur Tranche*. Thèse présentée à l' INPG, Grenoble 1988

[PaGa87] B Pangrle and D. Gajski, "Design Tools for Intelligent Silicon Compiler," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems,* Vol CAD-6, N° 3, pp 1098-1112, Nov. 1987

[PaKn86] P. Paulin, J. Knight, and E. Gyrczyc,"HAL," *Proc. 23rd Design Automation Conference*, pp. 263-270, 1986

[PaPM86] A. Parker, J. Pizarro, and M. Mlion, "MAHA," in *Proc. 23rd Design Automation Conference*, pp. 461-466, 1986

[Peltz83] D. Peltzer "Wafer Sale Integration : The Limit of VLSI," *VLSI Design*, pp. 43-47 , Sep. 1983

[Petr67] R. Petriz, "Current Status of Large Scale Integration Technology," *IEEE Journal of Solid-State Circuits*, Vol. SC-2, N° 4, Dec. 1967

[Pfei90] J. Pfeiffer, "HCL : A Language for Low-level Image Analysis," *Jour. of Parallel and Distributed Computing*, Vol. 8, N°., pp. 231-244, 1990

[PoMe89] J. Potter and W. Meillander, "Array Processor Supercomputers," *Proc. of the IEEE* , Vol. 77, N° 12, pp. 1896-1914, Dec. 1989

[PrRe89] V. Prasannakumar and D. Reisis, "Image Computations on Meshes with Mutiple Broadcast," *IEEE Trans. on PAMI*, Vol.11 , N° 11, pp. 1194-1201, Nov. 1989

[PWSE86] B. Perterson, B. White, D. Solomon, and M. Elmasry, "SPIL" *Proc. ICCAD*, pp 500-503, Nov. 1986

[RaBG83] T. Radhakrishnan, R. Barra, A. Guzman and A. Jirich, "Design of a High-Level Language (L) for Image Processing," in M. Duff and S. Levialdi, Eds., Languages and Arcitectures for Image Processing, Academic Press, New York, NY., pp. 88-98, 1982

[Raff85] J.I. Raffel, "The RVLSI Approach to wafer Scale Integration," *Proc. of the Wafer Scale Integration*, C. Jesshope and W. Moore, Eds, Adam Hilger, 1985

[RaPB86] J. Rabey, S. Pope, and R. Brodersen, "An Integrated Automated Layout Generation System for DSP Circuits," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol CAD-4, N° 3, pp 285-296, Jul. 1985

[RaSa90] S. Ranka and S. Sahni, "Convolution on Mesh Connected Multicomputers," *IEEE Trans. on PAMI*, Vol. 12, N° 3, pp. 315-318, Mar. 1990

[RaSa90] S. Ranka and S. Sahni, "Odd Even Shifts in SIMD Hypercubes," *IEEE Trans. of Parallel and Distributed Systems*, Vol. 1, N° 1, Jan. 1990

[Redd73] S. Reddaway, "DAP - A Distributed Array Processor," *Proc. 1st Symp. Comp. Archi.*, Florida, pp. 61-65, Dec. 1973

[Reev82] A. Reeves, "The local median and other window operations on SIMD computers," *Computer Graphics and Image Processing* 19, pp. 165-178, 1982

[Reev84] A. Reeves, "Parallel Computer for IP," *Computer Vision, Graphics and Image Processing*, Vol.25, pp.65-68, 1984

[RMVG88] J. Rabeay, H. De Man, J. Vanhoof, G. Goosens, and F. Catthoor, "CATHEDRAL-II" in *Silicon Compilation*, D. Gajski, Ed. 1988

[Rose83] A. Rosenfeld, "Parallel IP using Cellular Arrays," *Computer*, pp. , 1983

[Rose85] A. Rosenfeld, "Parallel Algorithm for Image Analysis," in VLSI and Modern Signal Processing, S.Y. Kung, H.J. Whitehouse and T. Kailath, Eds, pp. 422-433, 1985

[SaLC64] K. Sack, R. Lyman and G. Chang, "Evolution of the Concept of a Computer on a Slice," *Proc. of the IEEE*, Vol. 52, pp. 1717-1770, 1964

[Sche81] A. Schettini, "Comparing some High-level Languages for Image Processing," in M.G. Duff and S. Levialdi, Eds., Languages and Arcitectures for Image Processing, Academic Press, New York, NY, pp. 157-16, 1981

[Schm88] L. Schmitt, S. Wilson, "The AIS-5000 Parallel Processor," *IEEE Trans. on PAMI*, Vol. 10, N° 3, pp. 320-330, May 1988

[Seig79] H. Seigle, "A Model of SIMD Machines and a Comparison of Various Interconnection Network," *IEEE Trans. on Comp*uters, Vol. C-28, N° 12, pp. 907-917, Dec. 1979

[ShGL86] Schuck, M. Glesner, and M. Lacken, "ALGIC," in *VSLI Signal Processing II*. New York; IEEE press, Nov. 1986

[Skil88] D. Skillitron, "A Taxonomy for Computer Architectures," *Computer*, Vol. 21, N° 11, pp. 46-57, Nov. 1988

[SlBR62] D. Slotnick, W. Borck, R. Reynolds, "The SOLOMON Computer,"*Proc. AFIPS-Fall Joint Computer Conference*, pp. 97-107, 1962

[Slot71] D.L. Slotnick, "The Fastest Computer," *Sci. Amer.*, Vol. 224, pp. 76-87, 1971

[SmDe88] S. Smith, P. Deneyer, *Serial Data Computation*, Kluwer Academic Publishers, Boston, 1988

[Stap73] C. Stapper, "Defect Density Distribution for LSI yield calcultations," *IEEE Trans. Electron Devices*, Vol. ED-20, pp. 655-657, Jul. 1973.

[StRo82] C. Stapper and R. Rosner, "A Simple Method for Modeling VLSI Yields," *Solid-State Electronics*, Vol. 25, pp. 487-489, Jun. 1982.

[TsSi86] C. Tseng and D. Siewiorek "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems,* CAD-5, pp 379-395,.Jul. 1986

[Tuck88] R. Tuck, "An Optimally Portable SIMD Programming Language," *2nd Sym. on the Frontier of Massively parallel Computation*, Faifax, VI., pp. 617-624, Oct. 1988

[Unge58] S. Unger, "A Computer oriented towards Spacial Problems," *Proc. IRE.*, Vol. 46, pp. 1744-1750, 1958

[West87] R. Westmore, "WSI : An Introduction, Overview, and Perspective," *Electronic Design Automation Conf.*, pp 267-272, 1987

[WyRa89] P. Wyatt and J. Raffel, "Restructurable VLSI—A Demonstated Wafer Scale Technology," *Proc. of Int. Conf. on Wafer Scale Integration*, San Franscico, CA, pp. 13-20, 1989

[Yama89] K. Yamashita *et al.*, "A Wafer-Scale 170 000-Gate FFT Processor with Built-in Test Circuits," *IEEE Journal of Solid State Circuits*, Vol 23, N°2, pp. 336-341, Apr. 1988

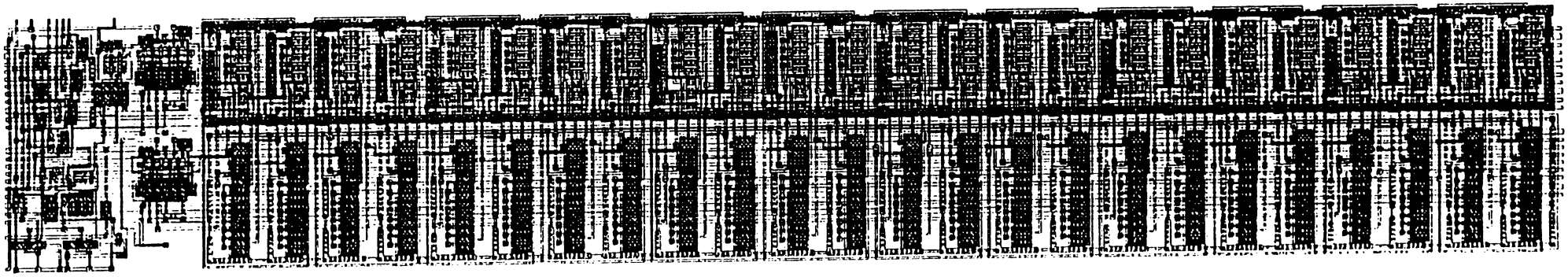
[Zakh84] V. Zakharov, "Parallelism and Array Processing," *IEEE Trans. on Computers*, C-33(1), pp. 45-77, Jan. 1984

ANNEXE I

VLSIplot p40 of [ly]buffer12 on "21-Feb-92" "0:25" by jluc at a scale of .26 mm/micron (261X), strip 1, sheet 1

Amplificateur bidirectionnel à 12 bits

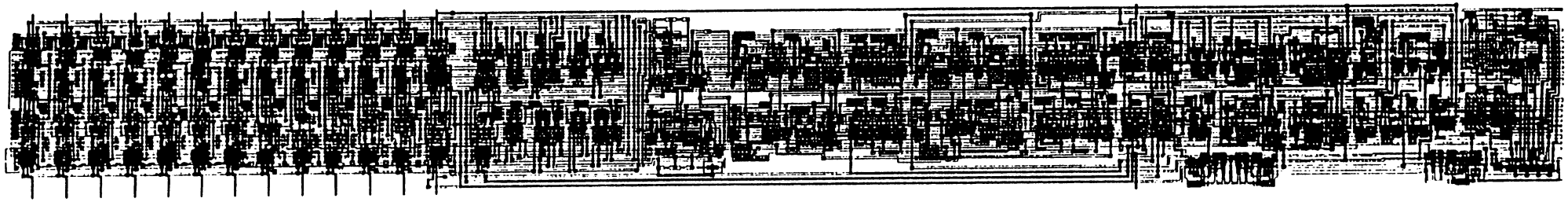
Surface = 1003 x 158 μ^2



VLSIplot p4(0) of [ly]new_switch4b on "21-Feb-92" "6:03" by jluc at a scale of .18 mm/micron (180X), strip 1, sheet 1

commutateur à quatre bits

Surface = 1453 x 168 μ^2

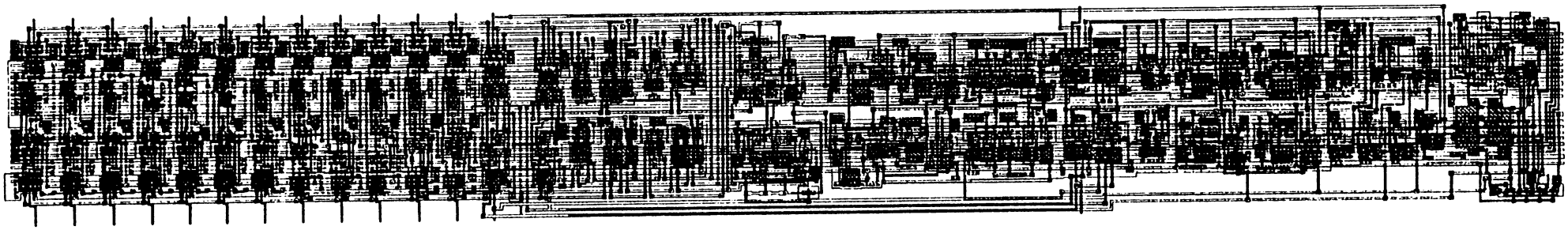


Partie commutation

Partie contrôle

commutateur à deux bits

Surface = 1279 x 165 μ^2



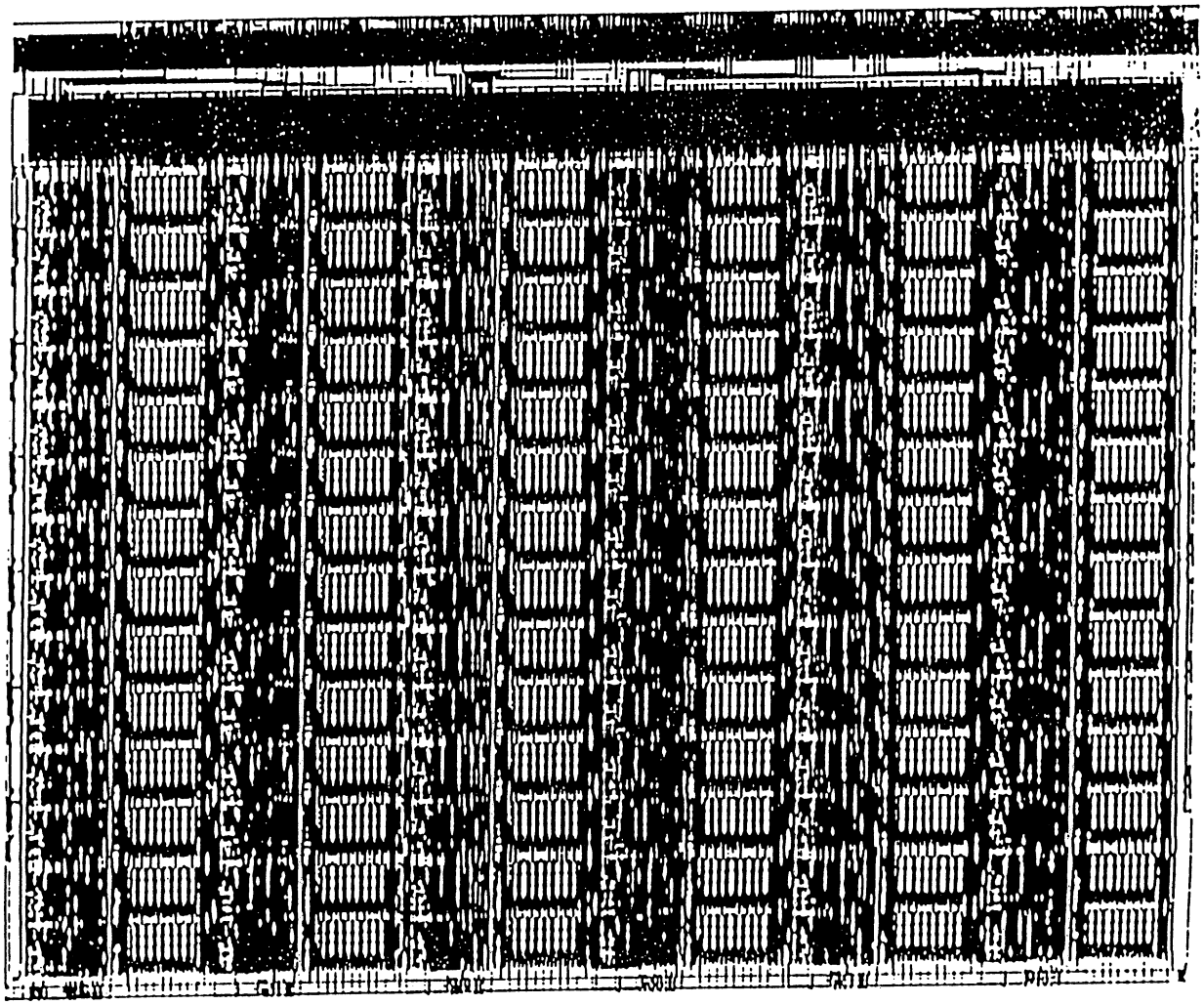
Partie commutation

Partie contrôle

ANNEXE II

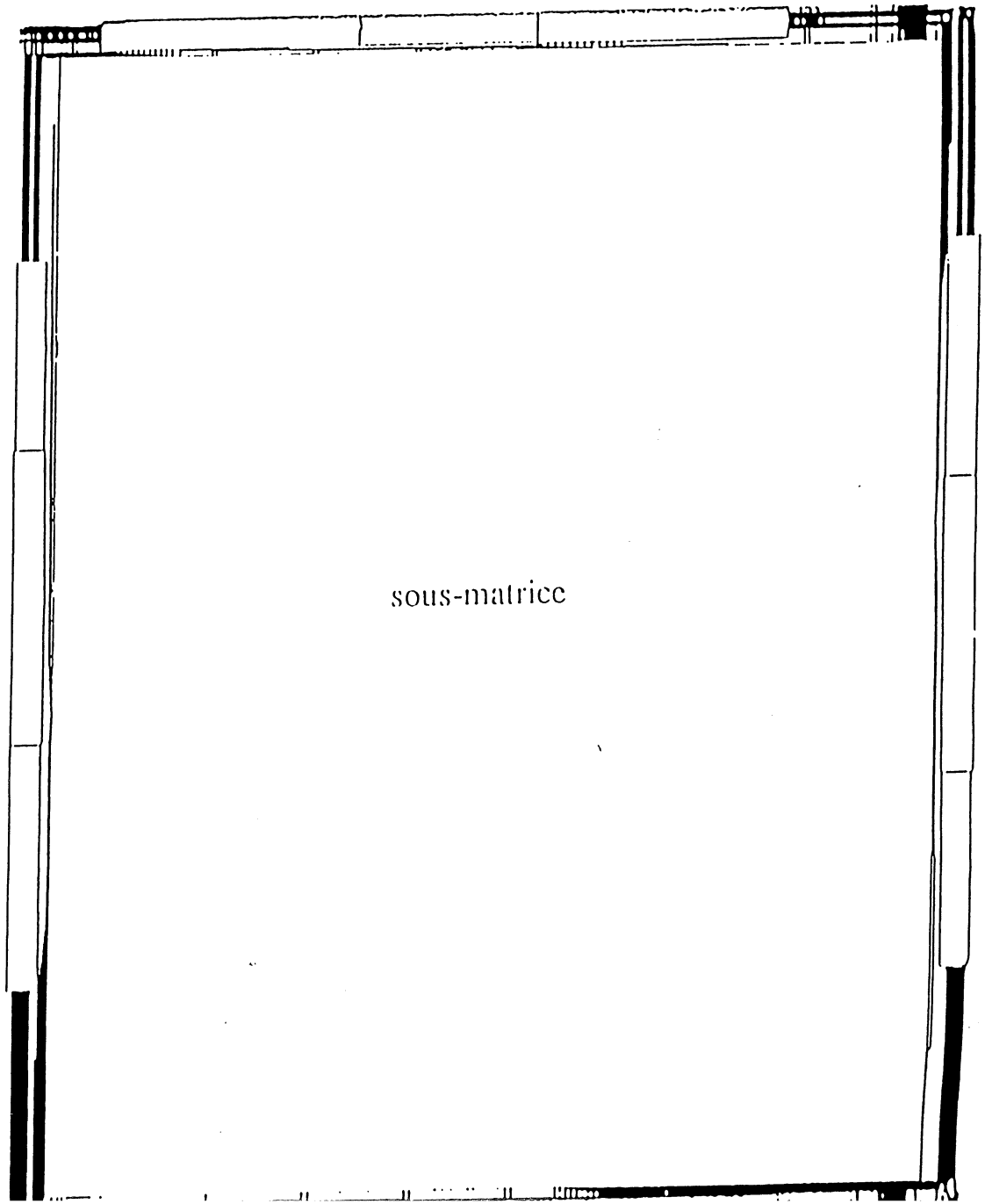
Sous-matrice de 7x12 PEs

surface = 4945,8 x 6102,2 μ^2



commutateur amplificateur commutateur

1/4 RETICULE

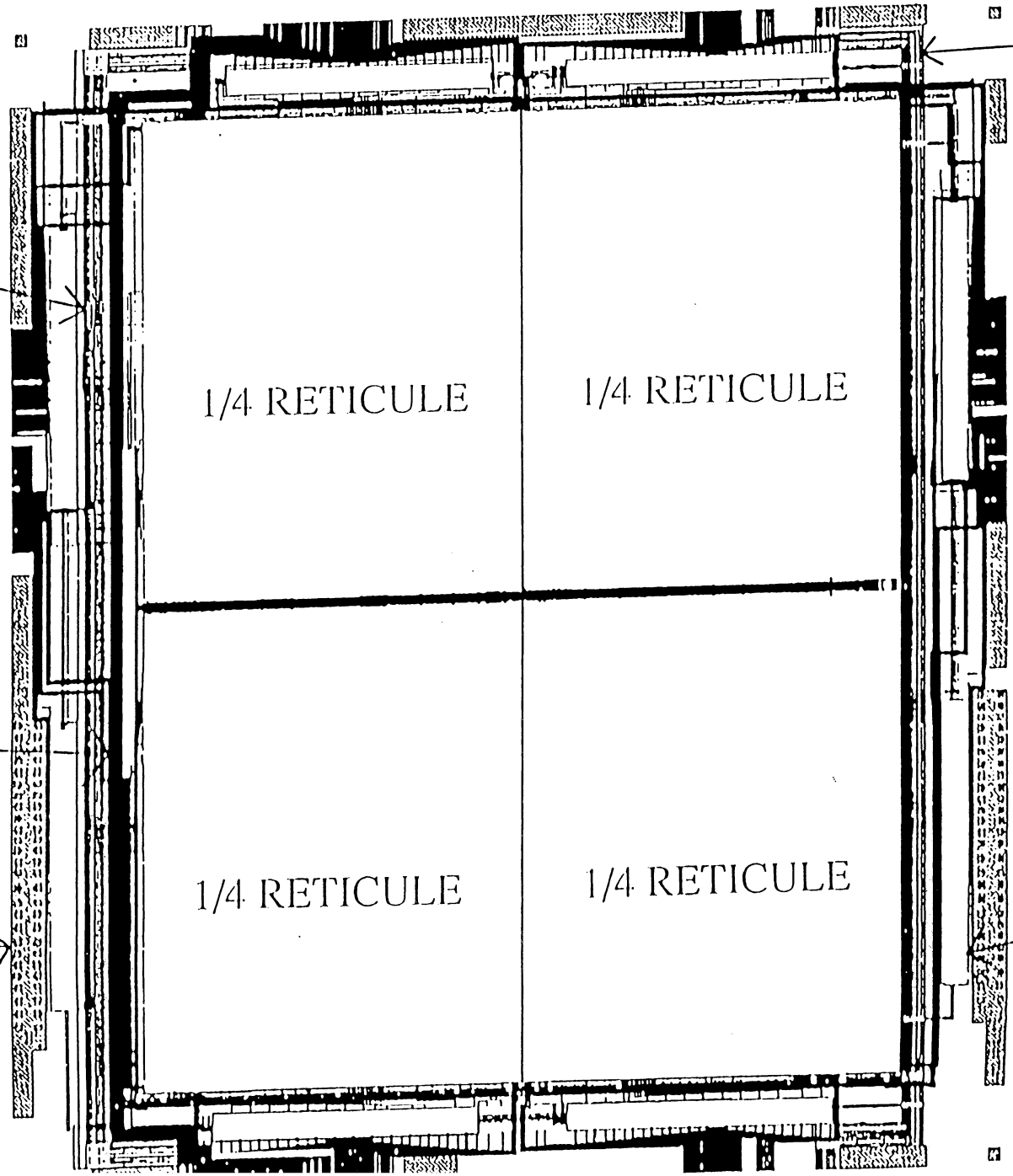


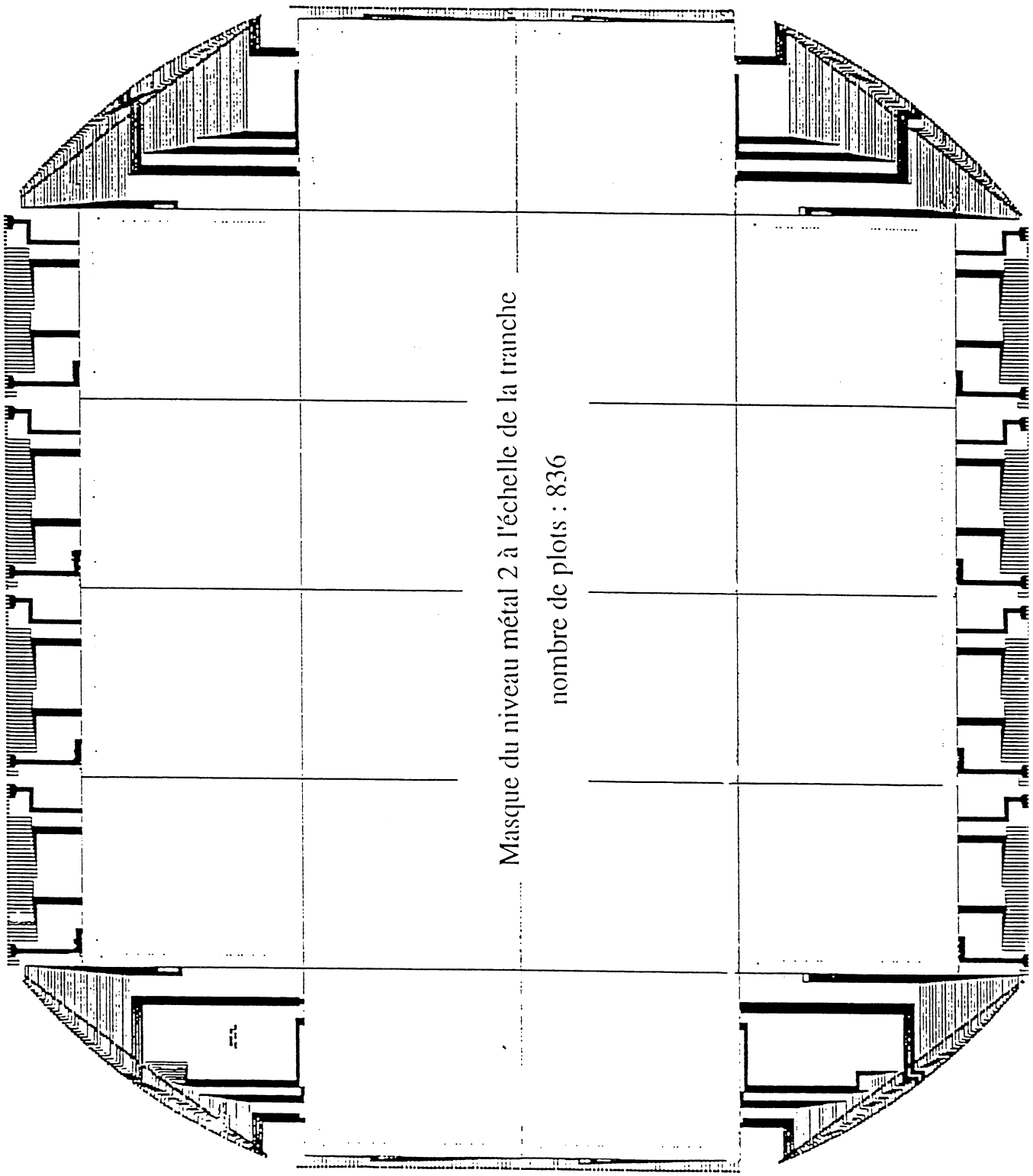
Surface totale = 5398,3 x 6556,7 µ2

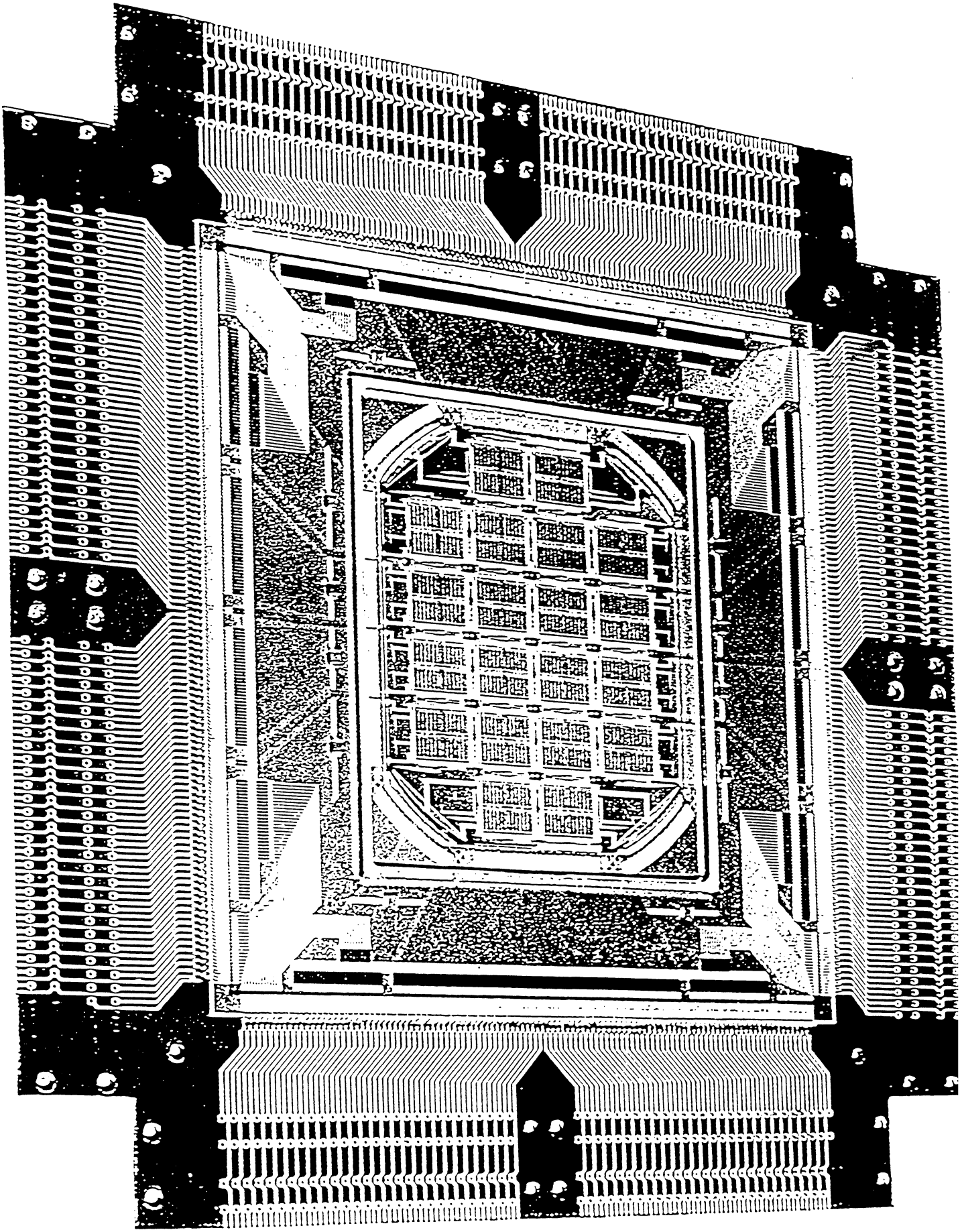
Surface de la sous-matrice = 4945,8 x 6102,2 µ2

Réseau de configuration = 14,7% de la surface totale

RETICULE







ANNEXE III

Annexe III

Procédure de test du PE

4 PROCEDURES

The HITEST simulation uses a number of procedures for performing various tasks, although when converted to the IMS tester these are expanded into the main program it is convenient to describe them here and reference them when they are used.

4.1 COMP

This performs the comparison between the states of the NS and EW registers on which the self–test is based. At the commencement of the self–test the result latch will have been reset (into the

PASS state) by a signal GRESET, and during the test the result latched by a signal SET. These signals are both produced from the inputs CE and MRESET. Before calling this procedure the input CE must be set in the main program to "1", the procedure then makes MRESET="0", to perform the result sampling and reconfiguration, and then returns it to "1".

4.2 ARAMLOAD

This produces an increment of the RAM A address combined with a toggling of the global input CE (BIT), this is used to help create patterns of alternating "1"s and "0"s. The pattern is repeated 15 times, after the initial setting before the procedure is called, to enable one row of a RAM to be loaded with data.

4.3 BRAMLOAD

This is similar to ARAMLOAD except that the RAM B address is incremented.

5 INITIALISATION

All "0"s are applied to the W, S and CMS inputs. Although these inputs are not used for the test they should not be left floating. In the initial packaged devices these inputs will be held by built–in pull down resistors.

Other input conditions are set are to control the multiplexors, RAM addresses and write controls. These are:

cm=R1	fg=1	alu=add	flg=R2
ns=BIT	ew=BIT	c=BIT	ram=sum (sm)
wa=0	wb=0	A=000000	B=000000
cset=0	mreset=0		

CE (BIT) is made "0" to produce GRESET and reset the self–test result register.
mreset is then returned to 1 to remove GRESET.

Following the initialisation the test sequence is run in the following order.

6 RAM LOADING

The following conditions are next set to allow all the RAM locations to be loaded with data.
c11=1 so that R1=RA R2=RB
(ALU function is still set to ADD and the inputs NS, EW and C to the input BIT and the RAM input selector to SM).
A=000000 B=000000 CE(BIT)=1
RAM A write is then activated,
The procedure ARAMLOAD is then run which causes a 1010.... pattern to be written into the 1st RAM A row.
The row address is then set to 01, CE(BIT) ="0", and ARAMLOAD run again to put the pattern

0101... into the 2nd row. This procedure is repeated with the 3rd and 4th rows to write a checker-board pattern into RAM A.

RAM A write is then deactivated, RAM B write activated and a similar checkerboard pattern written into RAM B, except that the patterns are the inverse of those written into the RAM A. ie. the 1st row is 0101.... , 2nd row 1010.... etc. RAM B write is then deactivated.

7 RAM COMPARE

The next stage is to compare the data in the A and B RAM's.

ew=R1 ns=R2 A=000000 B=000000

CE (BIT) ="1" to enable SET to be produced if at any time ew=ns.

The procedure COMP is then run to compare ew and ns, A and B incremented, COMP run again and this repeated until all the RAM locations have been compared. If at any address the data from the two RAM's is the same the PE is regarded as having failed the self–test.

8 SECOND RAM LOAD and COMPARE

Stage 6 is now repeated to load both the RAM's again but with the inverse data patterns to those used previously. Stage 7 is then repeated to again compare the data stored in the two RAM's. At this stage all the RAM locations have now been checked for storing both "1" and "0".

9 LATCH MULTIPLEXORS

Some checks are made on the other latch input multiplexors:

ns=ew ew=ns

This sets each of these latches to the state of the other, COMP is then run to check that this has happened (both latches should have changed state).

The following sequence is then run.

ns=BIT ew=BIT
ram=cy A=000001 CE(BIT)=1

RAM A write is then activated to write the carry ("1") into RAM A.

cm=R1 Fetches the carry from RAM A to the cm register.
ram=cm B=111100

RAM B write is then activated to put the data from the cm register into RAM B.

ns=R2 Data from RAM B to ns register.
CE(BIT)=0 puts a "0" into ew register
ns=ns ew=ew retains the data ("1" and "0" respectively) in the registers.
CE(BIT)=1 to enable SET to be produced during compare.

COMP is then run to check that the data in the two latches is different.

10 CONTROL of FG

Input c2 and then RAM R2 are used in turn to set the internal control signal FG so that the registers are not updated.

ew=BIT ns=BIT c=BIT
ram=cy B=001111 CE(BIT)=1
RAM B write is then activated to put the carry ("1") into RAM B.
CE(BIT)=0 B=110000
RAM B write is then activated to put the carry ("0") into RAM B.
B=001111 FLG=R2 puts FLG under the control of R2 ("1")
c2=0 puts FG under the control of FLG=R2 ("1")
ns=ns retains the "0" from BIT in ns.
CE(BIT)=1 ew=BIT puts a "1" in ew
ew=ew retains the "1" from BIT in ew
B=110000 makes FG="0" from data in RAM B
The latches should now be inhibited from being updated regardless of the multiplexor controls.
ns=BIT ew=BIT
If updating was inhibited the latches will retain their previous data ("1" and "0"), if not they will take the same state as BIT.
COMP is then run to check that the latches were not updated.
c2=1 restore FG control to C2

11 ALU FUNCTIONS

Up to now all tests have used the ALU function of ADD, the other functions are now checked.

11.1 NS-EW-C

First some data is written to the RAM's:
ns=BIT ew=BIT c=BIT
A=000000 B=000000 CE(BIT)=0
RAM A and B write are now activated.
this stores "0" in B=000000 A=000000.
A and B are both incremented,
this stores "0" in A=000001 (and B=000001)
CE(BIT)=1 RAM A write deactivated
this stores "1" in B=000001 (overwriting the previous "0")
A and B are both incremented. RAM A write activated, RAM B write deactivated
this stores "1" in A=000010
CE(BIT)=0 RAM B write activated, RAM A write deactivated
this stores "0" in B=000010
A and B are both incremented. CE(BIT)=1, RAM A write activated
this stores "1" in A=000011 B=000011
Patterns have now been written into the first 4 locations of each RAM.

CE(BIT)=0 ns=R2 ew=R1 alu=NS-EW-C
ram=carry (cy) A=000000 B=000000
RAM A write is then activated to put the carry ("0") into A=000000

A and B are then incremented to put the carry ("1") into A=000001
A and B are then incremented to put the carry ("0") into A=000010
A and B are then incremented to put the carry ("0") into A=000011

11.2 EW–NS–C

The above RAM loading sequence is repeated using RAM addresses starting from 100000, then with the alu=EW–NS–C selection the carry (cy) result is written into the RAM B locations (from 100000 onwards).

CE(BIT)=1 ns=R2 ew=R1 alu=NS–EW–C
ram=carry (cy) A=000000 B=000000
RAM B write is then activated to put the carry ("1") into B=100000
A and B are then incremented to put the carry ("0") into B=100001
A and B are then incremented to put the carry ("1") into B=100010
A and B are then incremented to put the carry ("1") into B=100011

At this stage the ram contents should be:

address	RAM A	address	RAM B
000000	0	100000	1
000001	1	100001	0
000010	0	100010	1
000011	0	100011	1

Now the contents of the two RAM's are compared:
A=000000 B=100000 ew=R1 ns=R2
CE(BIT)=1 to allow SET to be produced

COMP is then run, followed by increments of A and B and COMP until the data from all 4 sets of addresses have been used. In each case the data from the ew and ns registers should be different and no error recorded.

12 RAM CROSSOVER

The final check is of the RAM output multiplexor (RAM crossover)
ns=BIT ew=BIT alu=add A=000000
B=000001 CE(BIT)=1 RAM A and B write activated
this writes "1" into the RAM locations

CE(BIT)=0 A=000001 B=000000
this writes "0" into the RAM locations

RAM A and B write deactivated

c11=0 sets the RAM crossover so that R1=RB R2=RA
A=000000 B=000000 CE(BIT)=0
ns=R2 (=RAM A) ew=ew (="0")
CE(BIT)=1 to allow SET to be produced
COMP is then run to compare ew="0" with A=000000="1"
if the crossover failed ew would be compared with B=000000="0" and would have found an error.

ew=BIT A=000001 B=000001 CE(BIT)=1
COMP is then run to compare ew="1" with A=000001="0"
if the crossover failed ew would be compared with B=000001="1" and would have found an error.

ns=BIT (=1) ew=R1 (=RAM B) B=000000
COMP is then run to compare ns="1" with B=000000="0"
if the crossover failed ns would be compared with A=000000="1" and would have found an error.

CE(BIT)=0 B=000001 A=000001 this puts "0" into ns
ns=ns (="0")
COMP is then run to compare ns="0" with B=000001="1"
if the crossover failed ns would be compared with A=000001="0" and would have found an error.

Annexe III

Procédure de test de la sous-matrice

4 PROCEDURES

The HITEST simulation uses two procedures for performing various tasks, although when converted to the IMS tester these are expanded into the main program it is convenient to describe them here and reference them when they are used.

4.1 ARAMLOAD

This produces an increment of the RAM A address combined with a toggling of the global input CE (BIT), this is used to help create patterns of alternating "1"s and "0"s. The pattern is repeated 15 times, after the initial setting before the procedure is called, to enable one row of a RAM to be loaded with data.

4.2 BRAMLOAD

This is similar to ARAMLOAD except that the RAM B address is incremented.

5 INITIALISATION

Input conditions are set are to control the multiplexors, RAM addresses and write controls. These are:

cm=R1	fg=1	alu=add	flg=R2
ns=BIT	ew=BIT	c=BIT	ram=sum (sm)
wa=0	wb=0	A=000000	B=000000
cset=0	mreset=0	c11=1 (RAM normal)	

CE (BIT) is made "0" to produce GRESET and reset the self–test result register.
mreset is then returned to 1 to remove GRESET.

Following the initialisation the test sequence is run in the following order.

6 RAM LOADING

The following conditions are next set to allow all the RAM locations to be loaded with data.
c11=1 so that R1=RA R2=RB
(ALU function is still set to ADD and the inputs NS, EW and C to the input BIT and the RAM input selector to SM).
A=000000 B=000000 CE(BIT)=1
RAM A write is then activated,
The procedure ARAMLOAD is then run which causes a 1010.... pattern to be written into the 1st RAM A row.
The row address is then set to 01, CE(BIT) ="0", and ARAMLOAD run again to put the pattern 0101... into the 2nd row. This procedure is repeated with the 3rd and 4th rows to write a checker-board pattern into RAM A.

RAM A write is then de–activated, RAM B write activated and a similar checkerboard pattern written into RAM B, except that the patterns are the inverse of those written into RAM A. ie. the 1st row is 0101.... , 2nd row 1010.... etc. RAM B write is then de=activated.

7 RAM COMPARE

The next stage is to check the data in the A and B RAM's. This is achieved by loading the data from a RAM location into the CM registers, comparing the top–row CM registers then shifting and comparing until the data from each row has been compared. The RAM address is then incremented and the process repeated.
First RAM A is checked:

A=000000 B=000000

CSET=1 (to compare)
cm=cms (to shift data) repeated 12 times

CSET=0

cm=RAM1
Increment A then the shift & compare loop is run and the process repeated a total of 64 times to read each RAM A location.

Next c11=0 to set, the RAM crossover, so that the RAM data to CM is taken from RAM B

The above process is then repeated only this time incrementing the RAM B address.

8 SECOND RAM LOAD and COMPARE

Stage 6 is repeated but this time the data stored in each of the RAM locations is the opposite state to that stored previously. Stage 7 is then repeated to again check all the RAM locations. First RAM A and then, using the crossover, RAM B.
At this stage all the RAM locations have now been checked for storing both "1" and "0".

9 ARRAY LOAD

The following conditions are set to load all the NS and EW registers to a "1" state from BIT.
ce(BIT)=1

The registers are then set to transfer data from the S and W edges and so load all latches with "0".
ns=s ew=w
These states are held to 12 cycles to enable the "0"'s to propagate across the chip and so set all the NS and EW latches.

All latches should now be set to "0", to check this an ADD of ns, ew and c=bit=1 is performed. Carry (cy) should be "0" but if either or both latches are still at "1" then cy will be "1".
ram=cy ns=ns ew=ew B=111000

RAM B write is activated (c11 is still=0 !) to store cy in RAM B

RAM B write is de–activated

The cy result of the addition is now stored in RAM B within each PE, they are now output:
cm=R1=RAM B Puts cy in RAM to cm

cm=cms CSET=1 Shift and compare of cm data, repeated a total of 12 times.

CSET=0

10 CONTROL of FG

Input c2 and then RAM R2 are used in turn to set the internal control signal FG so that the registers are not updated.

ew=BIT	ns=BIT	c=BIT	c11=1
ram=cy	A=001111	B=001111	CE(BIT)=1

RAM's A & B write are then activated to put the carry ("1") into the RAM's.

CE(BIT)=0 B=110000
RAM A write is de–activated. The carry ("0") is written into the B RAM.

RAM B write is de–activated
ce(BIT)=1

B=001111	FLG=R2	puts FLG under the control of R2 ("1")
c2=0		puts FG under the control of FLG=R2 ("1")

ce(BIT)=0
B=110000 FG=FLG=R2("0") so latches should be inhibited from receiving new BIT data.
RAM A write is activated to put cy into A(=001111)
If the latches were inhibited from receiving new data then cy will be "1", calculated from BIT=1. If however the latches were still updated with BIT=0 then cy will be "0". The result is written into RAM A

cm=R1	To load cy from RAM	
CSET=1	To compare top row of CM registers	
cm=cms	Shift data in cm registers, repeat total of 12 times	
CSET=0	c2=1	Terminate comparison & return FG control to c2

11 ALU FUNCTIONS

Up to now all tests have used the ALU function of ADD, the other functions are now checked. This is performed by first loading data into RAM locations 000000 to 000011 by using A=B=C=BIT, then performing A–B–"0" and putting the CY result into RAM A 000000 to 000011. Data is then loaded into four more locations 100000 to 100011 then performing B–A–"1" and putting the CY result into RAM B 100000 to 100011. These two sets of carry results should be the inverse of each other and so when added should always produce a sum of "1"

11.1 NS–EW–C

First some data is written to the RAM's:

ns=BIT	ew=BIT	c=BIT
A=000000	B=000000	CE(BIT)=0

RAM A and B write are now activated.
this stores "0" in B=000000 A=000000.
A and B are both incremented,
this stores "0" in A=000001 (and B=000001)
CE(BIT)=1 RAM A write deactivated
this stores "1" in B=000001 (overwriting the previous "0")

A and B are both incremented. RAM A write activated, RAM B write deactivated
this stores "1" in A=000010
CE(BIT)=0 RAM B write activated, RAM A write deactivated
this stores "0" in B=000010
A and B are both incremented. CE(BIT)=1, RAM A write activated
this stores "1" in A=000011 B=000011
Patterns have now been written into the first 4 locations of each RAM.

CE(BIT)=0 ns=R2 ew=R1 alu=NS–EW–C
ram=carry (cy) A=000000 B=000000
RAM A write is then activated to put the carry ("0") into A=000000
A and B are then incremented to put the carry ("1") into A=000001
A and B are then incremented to put the carry ("0") into A=000010
A and B are then incremented to put the carry ("0") into A=000011

11.2 EW–NS–C

The above RAM loading sequence is repeated using RAM addresses starting from 100000, then with the alu=EW–NS–C selection the carry (cy) result is written into the RAM B locations (from 100000 onwards).
CE(BIT)=1 ns=R2 ew=R1 alu=NS–EW–C
ram=carry (cy) A=000000 B=000000
RAM B write is then activated to put the carry ("1") into B=100000
A and B are then incremented to put the carry ("0") into B=100001
A and B are then incremented to put the carry ("1") into B=100010
and B are then incremented to put the carry ("1") into B=100011

At this stage the ram contents should be:

address	RAM A	address	RAM B
000000	0	100000	1
000001	1	100001	0
000010	0	100010	1
000011	0	100011	1

Now the data is read from the two RAM's

A=000000	B=100000	ew=R1	ns=R2
ce(BIT)=0	alu=add	Rin=sm	

RAM A write is activated to write the sum back into RAM A

RAM addresses A and B are then incremented, whilst maintaining write until all four addresses have been used.

RAM A write is de-activated

The stored sum results are now output one at a time through cm and compared.

A=000000 cm=R1

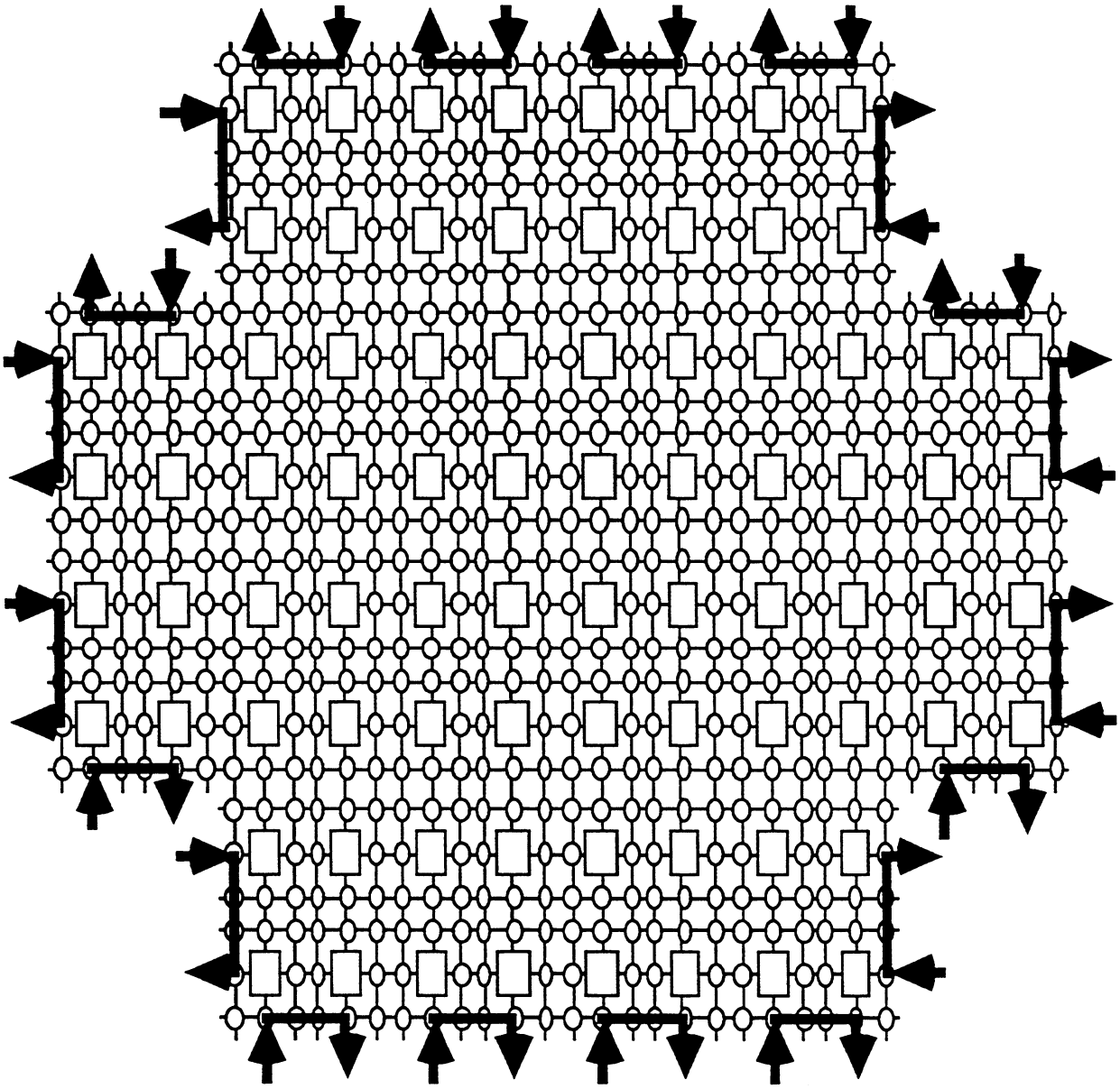
cm=cms increment A CSET=1

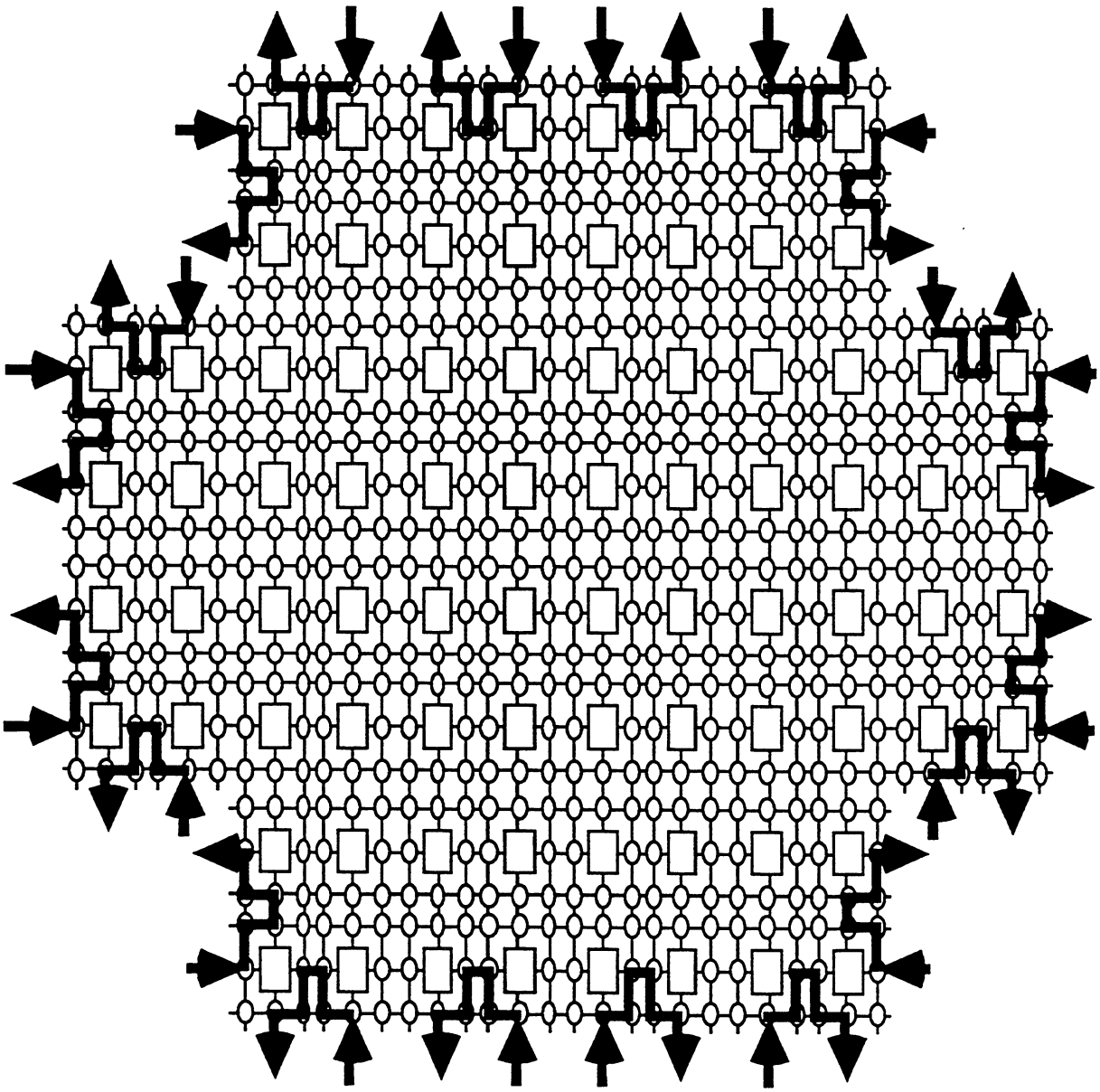
This is run for 12 cycles to output and compare the first set of sum data from all the rows in the array.

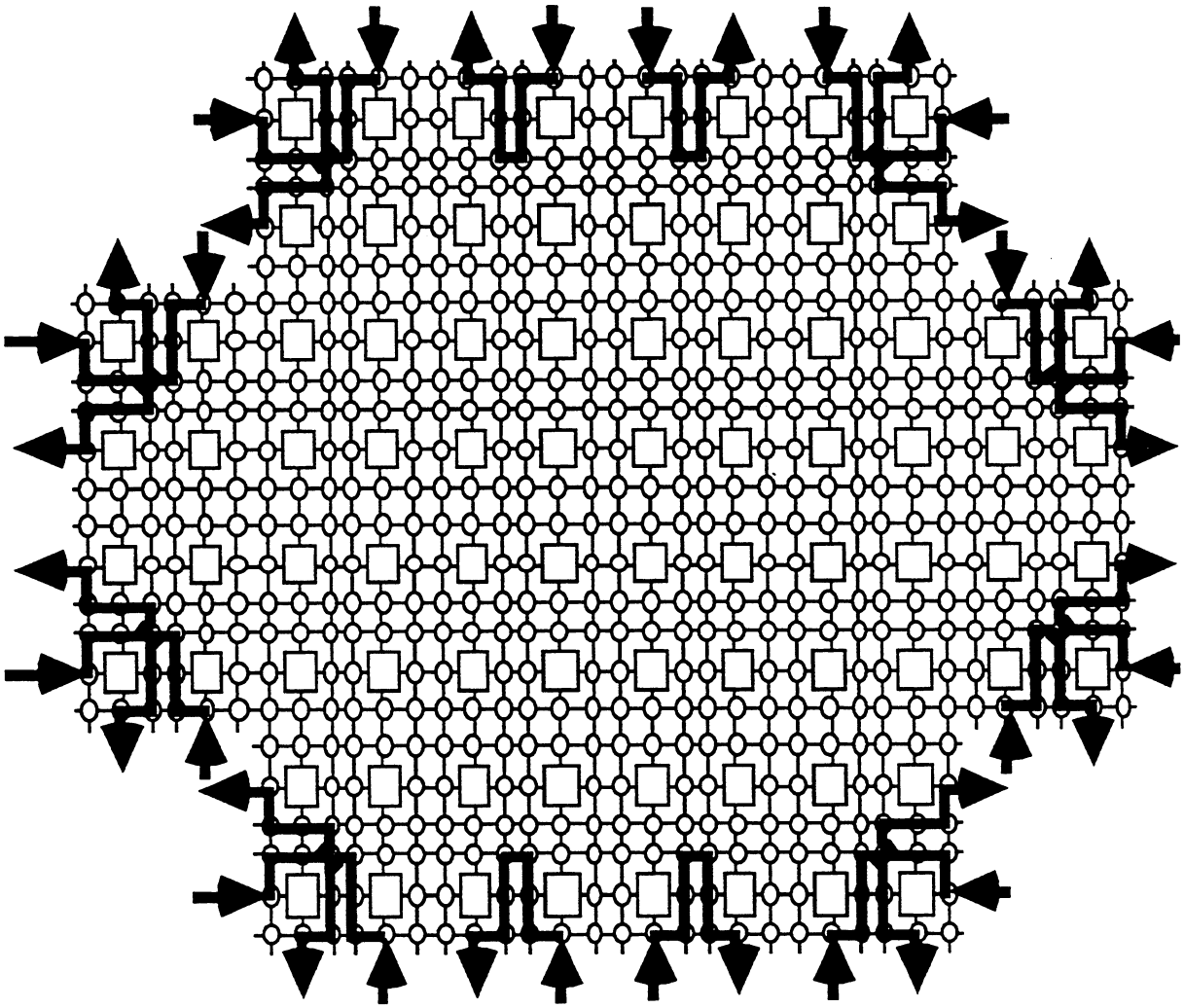
cm=R1 Increment A To fetch next set of sum data from RAM

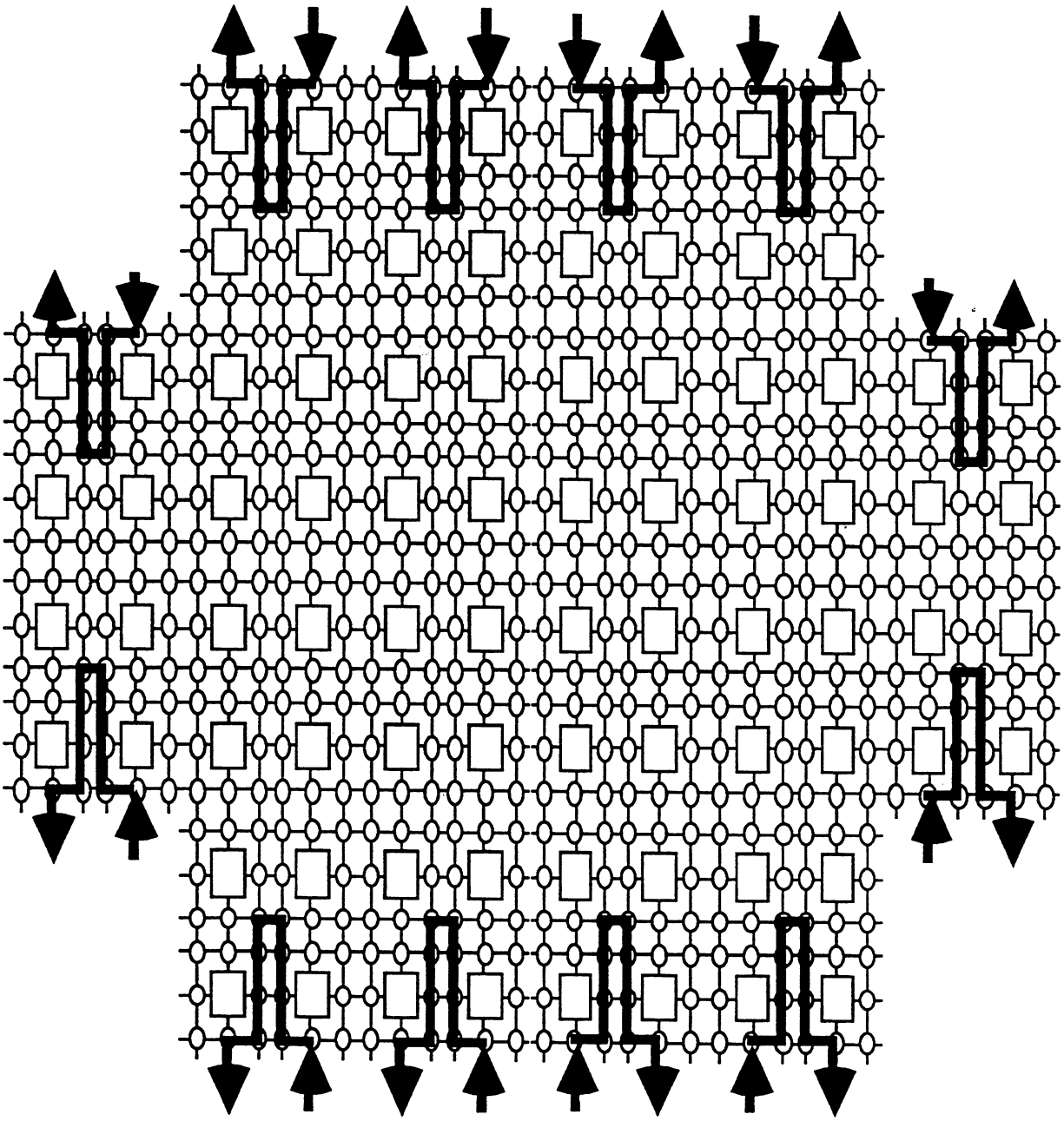
This data is then output and compared and the procedure repeated for the remaining two RAM locations.

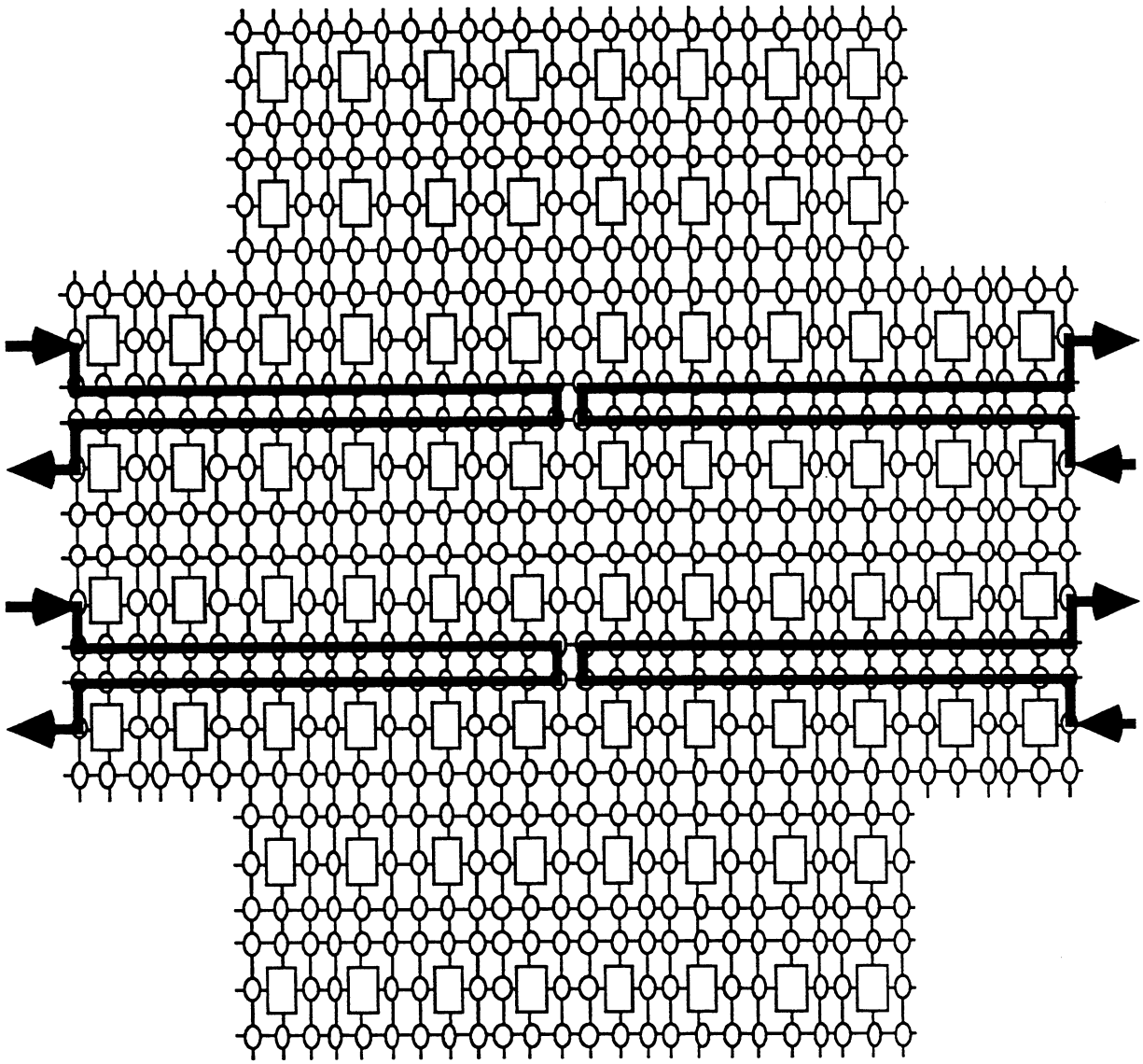
Test du réseau de commutateur : Différentes étapes de la phase I.

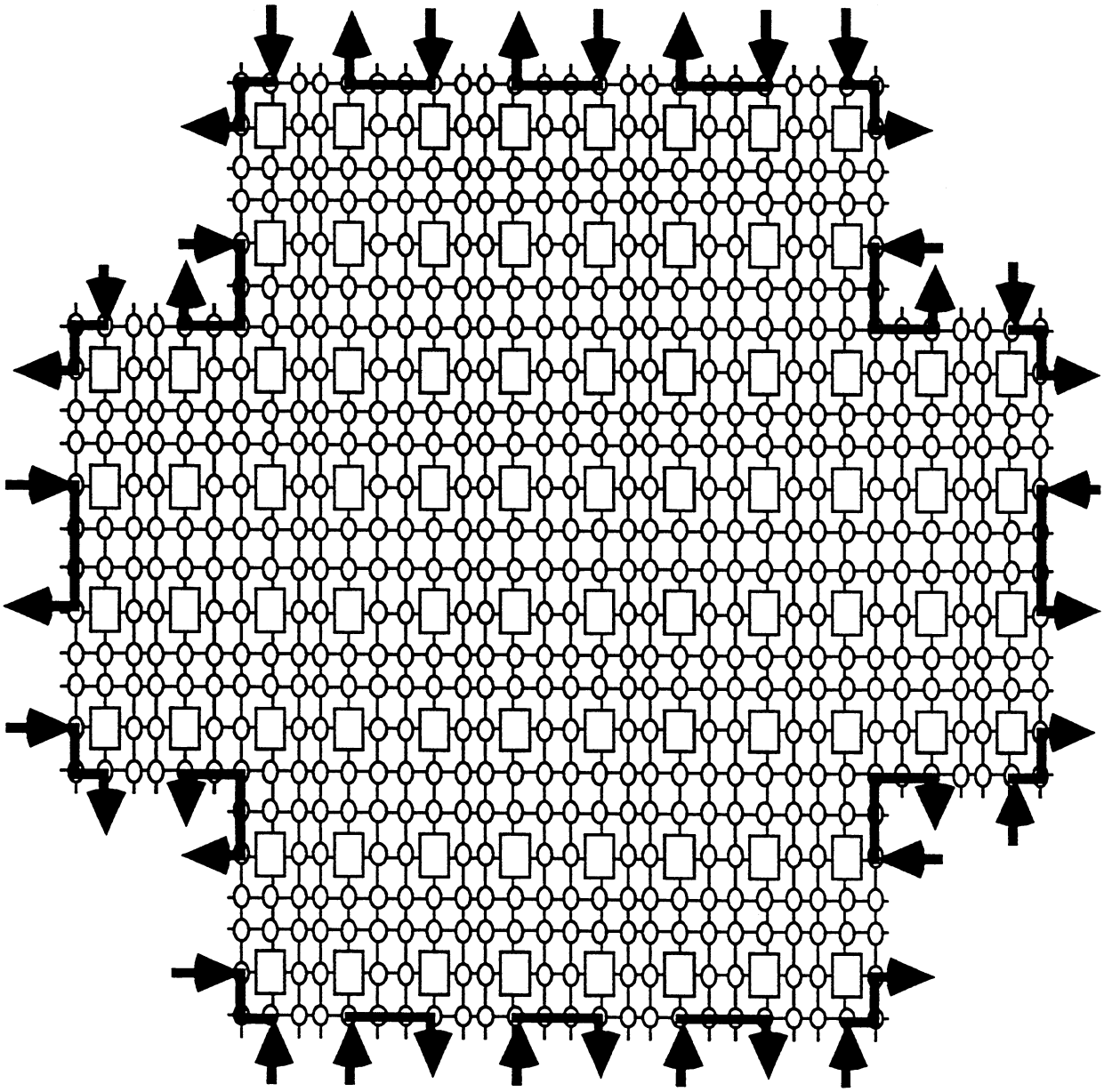


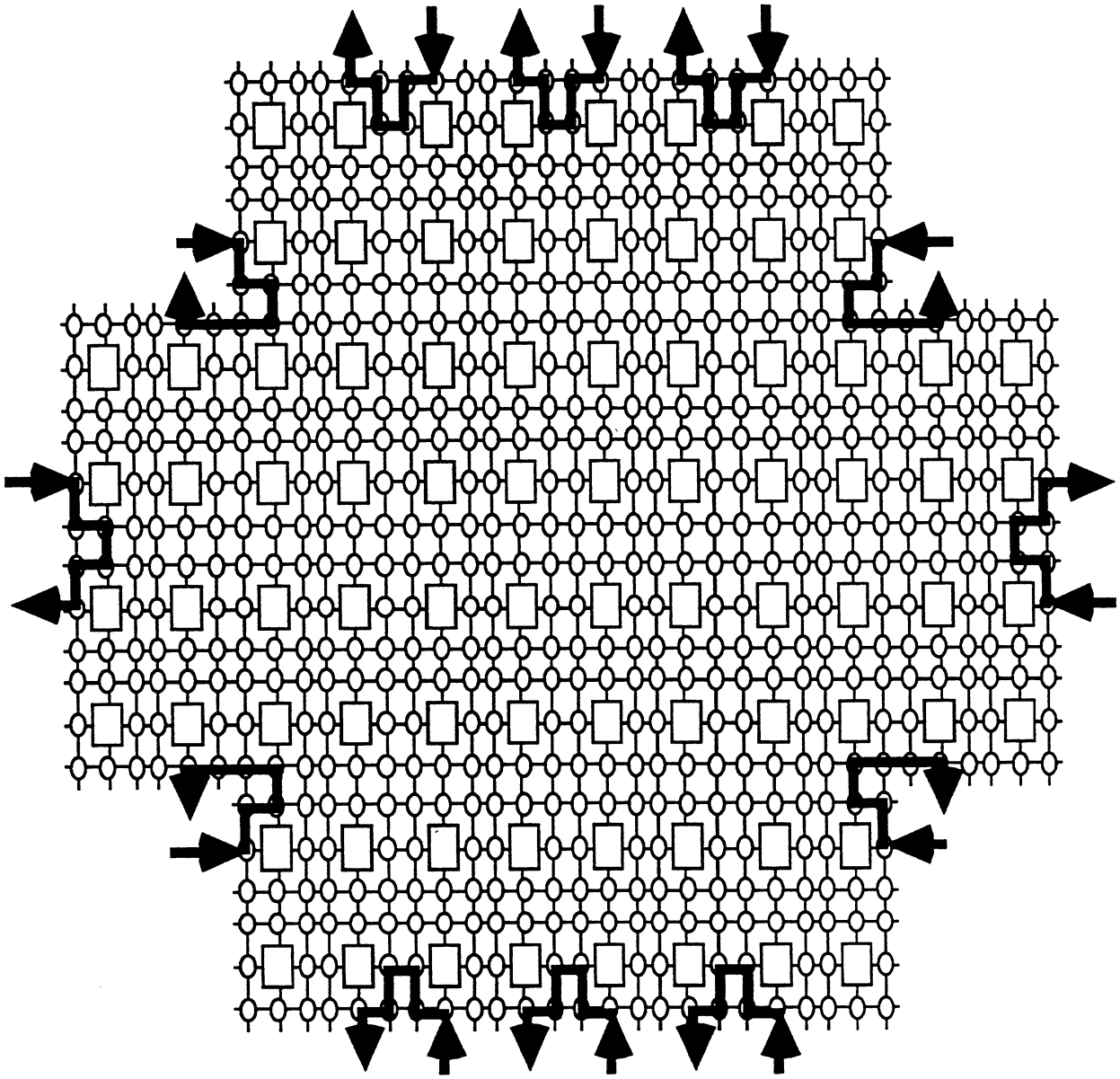












Annexe III

Procédure de test de la tranche configurée

4 INITIALISATION

Given the proposed strategy for assigning tester channels all data transfers through the array must be uni–directional ie. West to East and South to North.
Input conditions are set are to control the multiplexors, RAM addresses and write controls. These are:

cm=cms	fg=1	alu=add	flg=R2
ns=s	ew=w	c=BIT	ram=sum (sm)
wa=0	wb=0	A=000000	B=000000
cset=0	mreset=0	c11=1 (RAM normal)	

CE (BIT) is made "0" to produce GRESET which is used to reset the self-test result register. mreset is then returned to 1 to remove GRESET.

5 TEST SEQUENCE

In order to verify the interconnection of the chips and reticles data patterns are passed through the array from cms to cm, s to n and w to e. In each case the data input is 16 bits wide. It is presented in 4 bit hex format.

cms=5A5A	s=A5A5	w=A5A5
cms=A5A5	s=5A5A	w=5A5A
cms=0000	s=FFFF	w=FFFF
cms=FFFF	s=0000	w=0000
cms=CCCC	s=3333	w=3333
cms=3333	s=3333	w=CCCC
cms=3333	s=3333	w=CCCC
cms=CCCC	s=CCCC	w=3C3C
cms=FFFF	s=0000	w=0000

From these latest values s and w are incremented, and cms decremented, once on each cycle for 48 cycles. Then:

cms=0000	s=FFFF	w=0000

Then s is decremented and cms incremented once per cycle for 48 cycles.

Finally there are 96 cycles to clear out all the data from the registers.

ANNEXE IV

Les mots clés

Outre les mots resérvés du langage C, notre langage contient les mots clés suivants

in

shift

stair-shift

and

nand

or

nor

xor

xnor

not

add

sub

mult

Les déclarations et les instructions images sont définies par les règles syntaxiques suivantes[1] :

List_déclarations ::= déclaration; liste déclarations

Déclaration ::=

image "("expression, expression, expression")""," déclaration

Expression :: "identificateur_image" | "constante_entière"

Liste_instructions ::= Instruction_image ";" Liste_instructions

Instruction_image ::=

Instrction_shift ("identificateur_image", Direction, Pas)

lopérateur_logique ("identificateur_image","identificateur_image")

lopérateur_arithmétique ("identificateur_image","identificateur_image")

lopérateur_unair ("identificateur_image","identificateur_image")

Instrction_shift ::= **shift** | **stair_shift**

opérateur_logique ::= **and** | **nand** | **or**| **nor** | **xor** | **xnor**

opérateur_arithmétique::= **add** | **sub** | **mult**

opérateur_unair ::= **not**

direction ::= **north** | **south** | **east** | **west**

pas ::= "constante_entière"

---

[1] Une unité syntaxique terminale est écrite soit en gras, soit entre côtes.

ANNEXE V

Nous présentons dans cet annexe quelque exemples de traitement d'image de bas niveau.

1. Introduction

Le prétraitement englobe l'amélioration de l'image et la suppression de bruit. L'amélioration de l'image consiste en un ensemble de méthodes destinées à améliorer l'aspect visuel de l'image. Ceci peut se faire soit par des méthodes fondées sur la modification ou transformation de l'histogramme, soit par des méthodes de filtrage. Le filtrage, qui sert à éliminer le bruit ou améliorer la fidélité de l'image, peut se faire par des filtres linéaires et non linéaires. Le filtrage linéaire consiste a convoluer l'image avec un masque tel que la moyenne locale ("local averaging"). Bien que le filtrage linéaire élimine le bruit, il introduit une apparence confuse ("blur") sur les contours. Pour cette raison, des filtres non linéaires tels que les filtres morphologiques ou "rangs" sont habituellement utilisées.

Les techniques de filtrage d'image peuvent être divisées en deux catégories: globales et locales. Les techniques globales agissent sur la totalité de l'image ou une très grande partie de celle-ci. Parmi ces filtres, nous avons le filtrage Wiener ou moindre carré et de Kalman. Le filtrage wiener utilise des modèles statistiques de l'image et du bruit. Ces modèles sont dans la plupart des cas inconnus ou très complexes à décrire par un simple processus aléatoire. Les résultats sont souvent pas satisfaisants et très couteux. Les techniques locales sont plus efficaces et leur grand avantage et de pouvoir traiter plusieurs fenêtres en parallèle.

2. Filtrage Morphologique

Le terme *morphologie* vient de l'*étude des formes* des plantes et animaux. Dans notre contexte, il veut dire l'étude de la topologie ou la structure des objets à partir de leurs images. Le traitement morphologique consiste en certaines opérations où un objet est *balayé* par un *élément structurant* et donc réduit à une forme plus révélatrice.

L'élément structurant est un masque de forme quelconque dont les éléments forment un motif. La figure 1 montre quelques exemples d'éléments structurants.

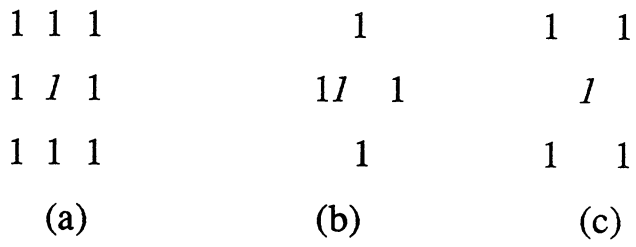


Figure 1. exemples d'éléments structurants. Le caractère italique représente l'origine de l'élément structurant. (a) carré 3x3, (b) croix vertical, (c) croix oblique

La plupart des opérations morphologiques peuvent être définies à l'aide de deux opérations de base la dilatation et l'érosion, appelé aussi opérateurs de Minkowski.

2.1. La dilatation [Serr82]

Supposons un objet X et un élément structurant T représentés comme deux sous-ensembles de $N \times N$. Soit T' la réflexion de l'ensemble T par rapport à son origine, i.e. l'ensemble de points tel que $-t \in T$. Soit X_b le translaté de X par le vecteur b, i.e., $\{x : x-b \in X\}$ La dilatation de X par T est définie comme l'ensemble de tout les points x tel que T_x intersecte X. Ceci est exprimé par

$$X \oplus T' = \{x : T_x \cap X \neq \varnothing\} = \cup_{b \in T'} X_b$$

Ceci revient à prendre l'objet initiale et le décaler d'une position vers le bas ensuite faire le OU logique de l'objet initial et son décalé. Ensuite décaler l'objet vers le haut et OU logique avec l'objet décalée.

Si on choisi de dilater par l'élément structurant carré 3x3, on aura l'algorithme suivant.

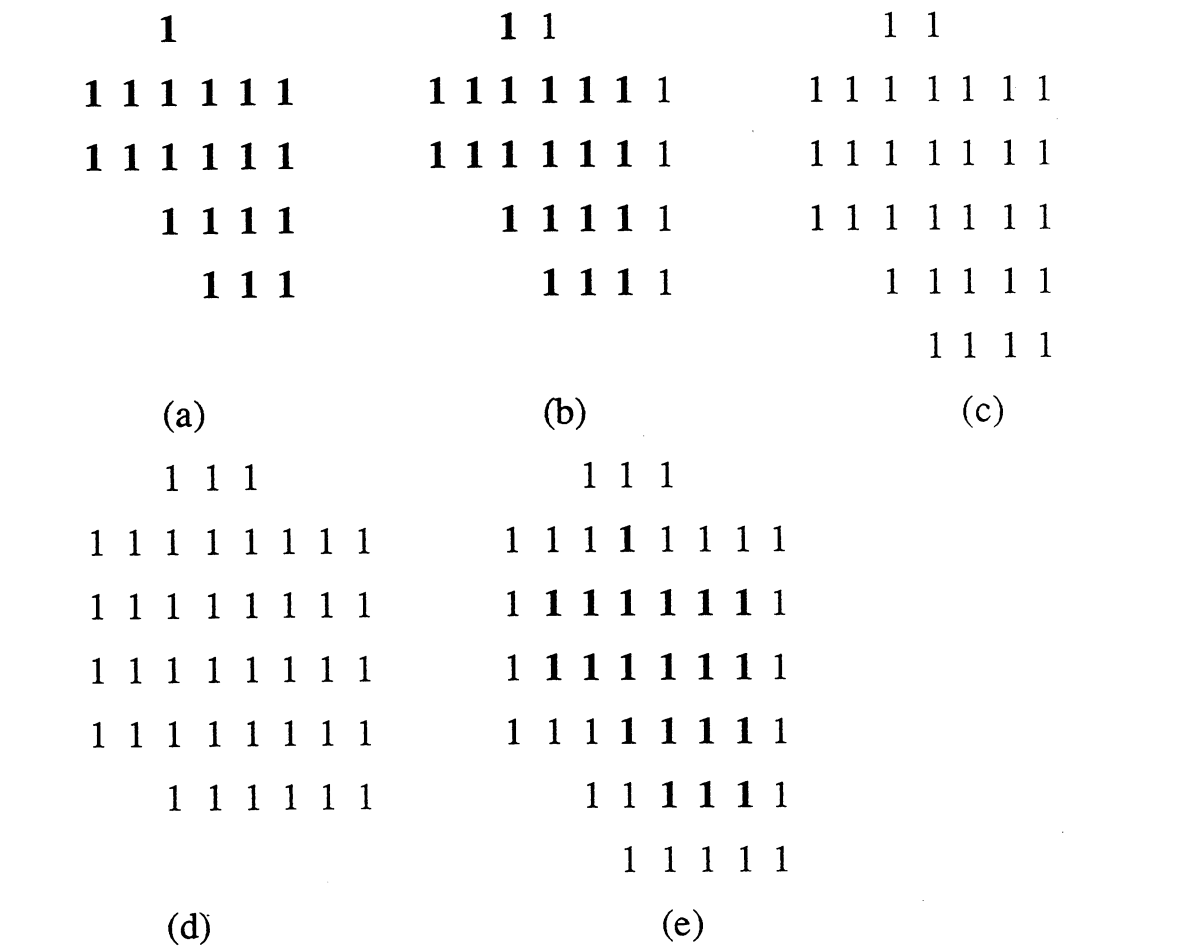
1) *Décalage de l'image initiale vers l'est et fonction OU entre image et image décalée*

2) *Décalage de l'image résultat vers le sud et fonction OU entre image résultat et image résultat décalée.*

3) *Décalage de l'image résultat vers le ouest et fonction OU entre image résultat et image résultat décalée.*

4) *Décalage de l'image résultat vers le nord et fonction OU entre image résultat et image résultat décalée.*

Ceci est montré dans la figure 2.



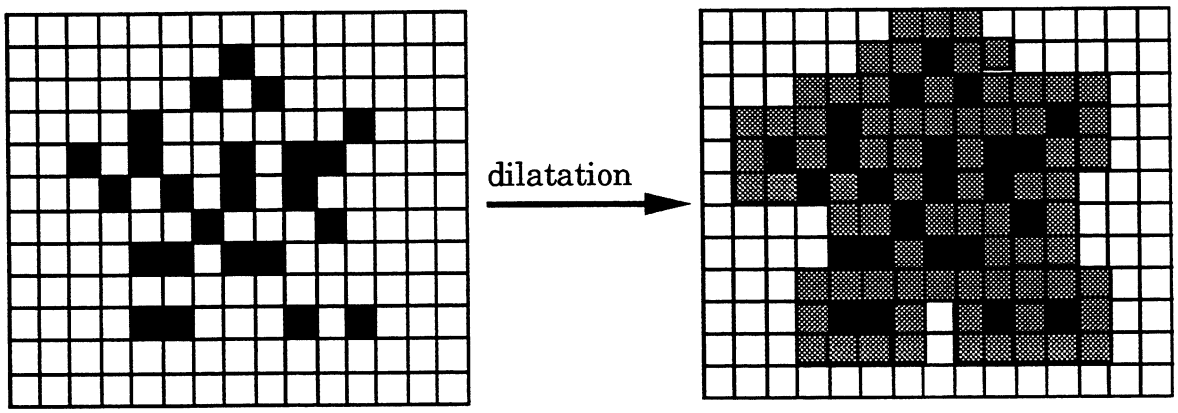


Figure 2 Exemple de dilatation (a) figure originale (b) après la 1ère étape(c) après la 2ème étape (d) après la 3ème étape (e) le résultat (original en gras) (f) le dilaté de la carte de France

Le programme de la dilatation s'écrit de la manière suivante.

```
main()   /* Dilation program */
{
image MAP (12, 15, 1),IMAGE (12, 15, 1),
        IMAGEsh (12, 15, 1),RESULT  (12, 15, 1);

IMAGE = in (MAP, 12);
IMAGEsh = shift (IMAGE, north, 1);
RESULT = or (IMAGE, IMAGEsh);
IMAGEsh = shift (RESULT, west, 1);
RESULT= or (RESULT, IMAGEsh);
IMAGEsh = shift (RESULT, south, 1);
RESULT = or (RESULT, IMAGEsh);
IMAGEsh = shift (RESULT, east, 1);
RESULT= or (RESULT, IMAGEsh);}
/* End of dilation */
```

Le graphe de dépendance de données (GDD) du programme de la dilatation est donné par la figure 3. La figure 3.(a) donne le GDD avant l'expansion. La figure 3 (b) donne le GDD de la dilatation après l'expansion. Rappelons que l'opération d'expansion remplace chaque macro-instruction (qui est un nœud dans le premier GDD) par son sous-GDD (cf. Chapitre 3)

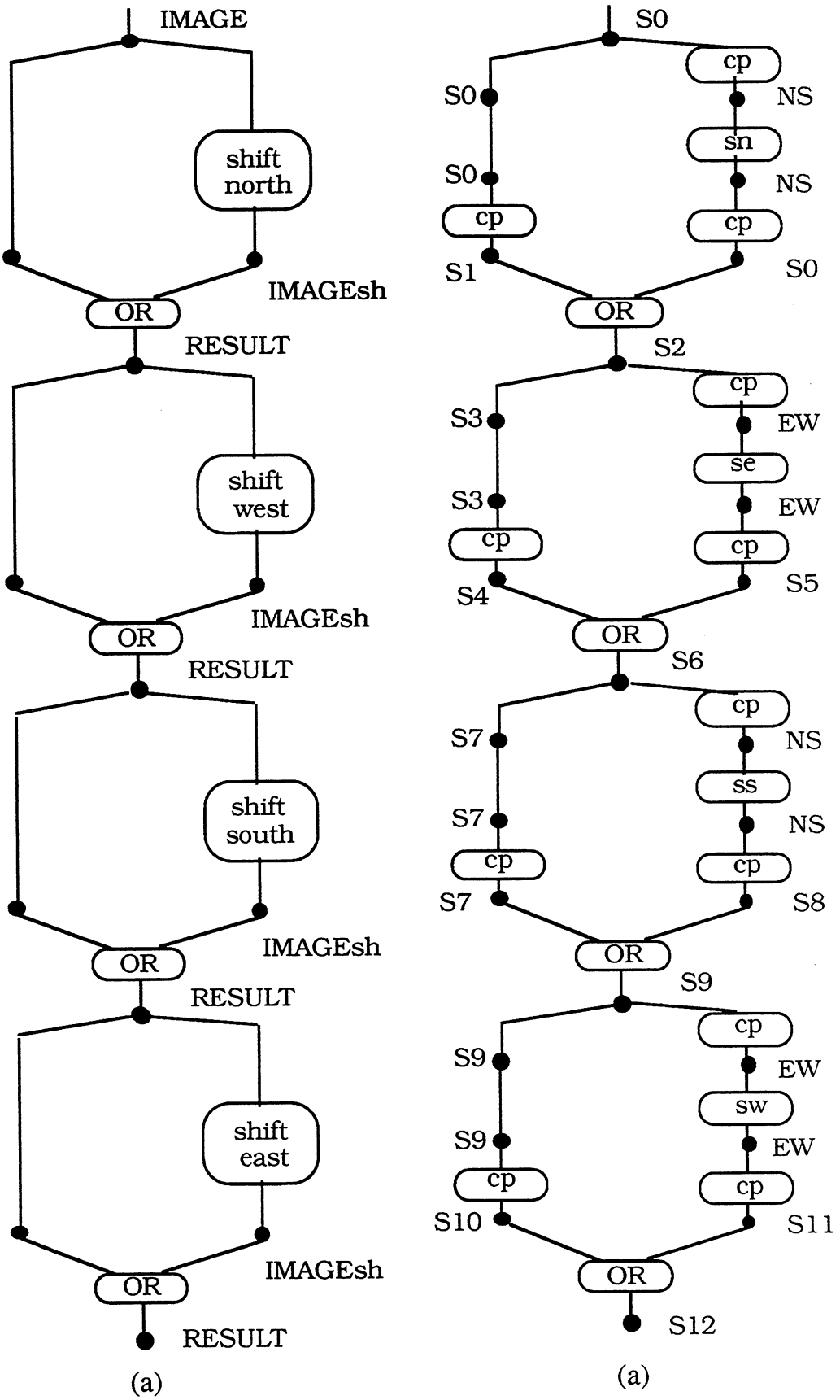


Figure 3. Graphe de dépendance de données de la dilatation. (a) avant l'expansion (b) après l'expansion.

2.2. L'érosion

L'érosion est l'opération duale de la dilatation. L'érodée de l'objet X par l'élément structurant T est définie par l'ensemble de tout les points x tel que T_X est inclus dans X, c'est-à-dire

$$\text{Erosion} = X \ominus T' = \{x : T_X \subset X\} = \bigcup_{b \in T'} X_b$$

Si on choisi de dilater par l'élément structurant (carré 3x3), nous obtiendrons l'algorithme suivant.

1) *Décalage de l'image vers l'EST et fonction 'ET' entre image d'origine et image décalée*

2) *Décalage de l'image résultat vers le SUD et fonction 'ET' entre l'image résultat et l'image résultat décalée.*

3) *Décalage de l'image résultat vers l'OUEST et fonction 'ET' entre l'image résultat et l'image résultat décalée.*

3) *Décalage de l'image résultat vers le NORD et fonction 'ET' entre l'image résultat et l'image résultat décalée.*

2.3. La fermeture

La fermeture est l'érosion d'un objet dilaté. On a,

$$\text{Fermeture } (X) = (X \oplus T) \ominus T$$

La fermeture est souvent utilisée pour lisser des images. Elle lisse les contours, joint les cassures étroites, remplit les anses et élimine les petits trous. On reconnait la carte de la France plus facilement sur l'image "fermée" que sur l'image d'origine (Figure 4).

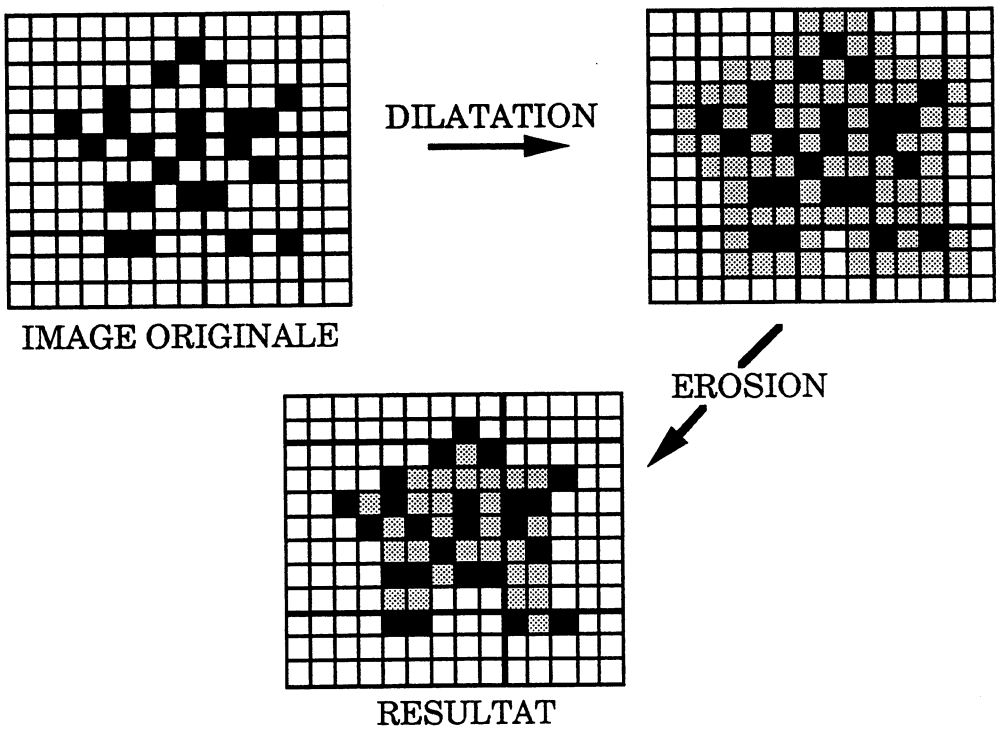


Figure 4 Exemple de lissage d'une image avec l'opération fermeture

L'algorithme de la fermeture est très simple

- acquérir l'image
- dilater cette image
- éroder l'image dilatée

L'exemple en haut peut être décrit par le programme suivant

```
main()   /* Dilation program */
{
image map (12, 15, 1), image (12, 15, 1), imagesh (12, 15, 1),
image result  (12, 15, 1);

  image = in (map, 12);    /*get image in */
  imagesh = shift (image, north, 1); /*dilate image */
  result = or (image, imagesh);
  imagesh = shift (result, west, 1);
  result= or (result, imagesh);
  imagesh = shift (result, south, 1);
```

```
result = or (result, imagesh);
imagesh = shift (result, east, 1);
result= or (result, imagesh);
/* End of dilation */
imagesh = shift (image, north, 1); /*erode image */
result = or (image, imagesh);
imagesh = shift (result, west, 1);
result= or (result, imagesh);
imagesh = shift (result, south, 1);
result = or (result, imagesh);
imagesh = shift (result, east, 1);
result= or (result, imagesh);}/* End of dilation */
/* End of fermeture */
```

2.4. L'ouverture

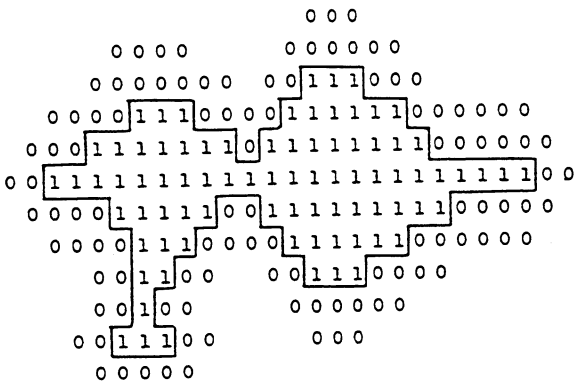
L'ouverture est la dilatation d'un objet érodé. Elle sert à lisser les contours de l'objet, à éliminer les îlots inférieurs à T, à supprimer les points isolés. On a,

$$\text{Ouverture } (X) = (X \ominus T) \oplus T$$

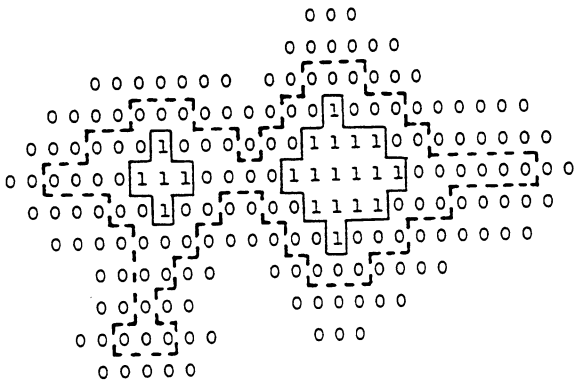
Nous donnons la description d'un exemple d'application qui utilise l'opération d'ouverture pour séparer deux taches connectées (figure 5).

```
/* This program separates two connected gel spots and clean them */
main()
image  gel(27,12,1), gspot(27,12,1), gspotn(27,12,1);
image  gspots(27,12,1), gspote(27,12,1), gspotw(27,12,1);
image  gspotns(27,12,1); gspotnse(27,12,1);
image  gspotnse(27,12,1), gspotnsew(27,12,1);
image  gspotnsew3(27,12,1), gspotnsew4(27,12,1);
{
```

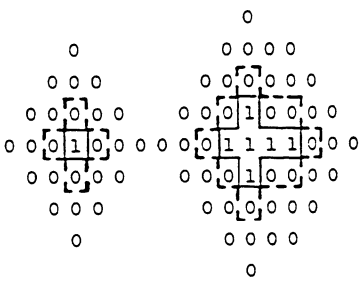
```
gspot = in (gel, 12); /* get in silhouette of gel spots */
gspotn = shift (gspot, north, 1); /* start first erosion */
gspotn = and (gspot, gspotn);
gspots = shift (gspotn, south, 1);
gspotns = and (gspots, gspotn);
gspote = shift (gspotns, east, 1);
gspotnse = and (gspote, gspotns);
gspotw = shift (gspotns, west, 1);
gspotnsew = and (gspotw, gspotnse); /* result of first erosion */
/* erode using second structuring element */
gspotn = shift (gspotnsew, north, 1); /* start second erosion */
gspotn = and (gspotnsew, gspotn);
gspots = shift (gspotnsew, south, 1);
gspotns = and (gspotnsew, gspots);
gspote = shift (gspotnsew, east, 1);
gspotnse = and (gspotnsew, gspote);
gspotw = shift (gspotnsew, west, 1);
gspotnsew2 = and (gspotnsew, gspotw); /* result of second erosion */
/* Dilation by a cross */
gspotn = shift (gspotnsew2, north, 1);
gspotn = or (gspotnsew2, gspotn);
gspots = shift (gspotnsew2, south, 1);
gspotns = or (gspotnsew2, gspots);
gspote = shift (gspotnsew2, east, 1);
gspotnse = or (gspotnsew2, gspote);
gspotw = shift (gspotnsew2, west, 1);
gspotnsew3 = or (gspotnsew2, gspotw); /* result of first dilation */
/* Dilation by a square */
gspotn = shift (gspotnsew3, north, 1);
```



(a) image originale (I)

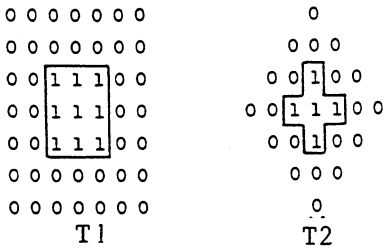


(b) Après 1ère érosion I - T1

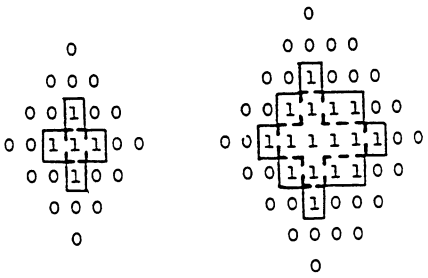


(c) Après 2ème érosion

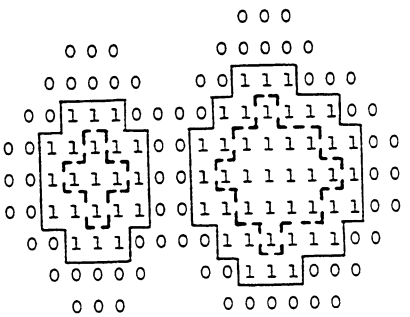
R = (I - T1) - T2



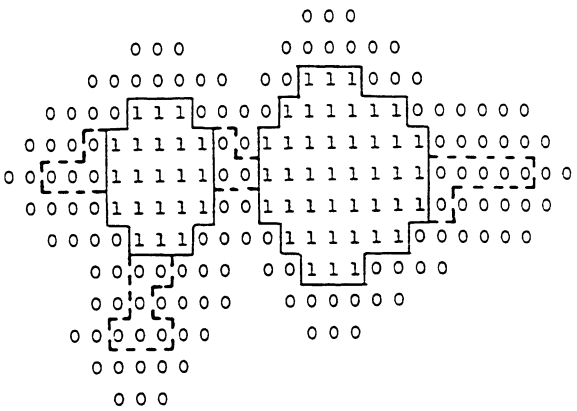
(d) éléments structurants T1, T2.



(e) 1ère dilatation R + T2



(f) 2ème dilatation (R + T2) + T1



(g) ouverture de I.

Figure 5 Séparation de taches par ouverture.

```
Res = add (A, Ash);
Ash = shift (Res, south, 1);
Res = add (Res, Ash);
B = shift (Res, east, 1);
Bsh = shift (Res, west, 1);
B = sub (Bsh, B)
}/* end of Sobel*/
```

3. Détection de Contour

La détection de contour est une opération très importante qui permet de réduire une image à ses contours seulement. Ces contours servent à la phase ultérieure de reconnaissance. Beaucoup d'algorithmes ont été proposés pour la détection de contour dans la littérature. Nous présenterons ici un détecteur très simple pour images binaires. Notons que le filtre Sobel décrit dans le paragraphe précédent est aussi utilisé comme un des opérateurs de différentiation utilisés pour la détection de contour.

La figure 6 illustre bien ce processus.

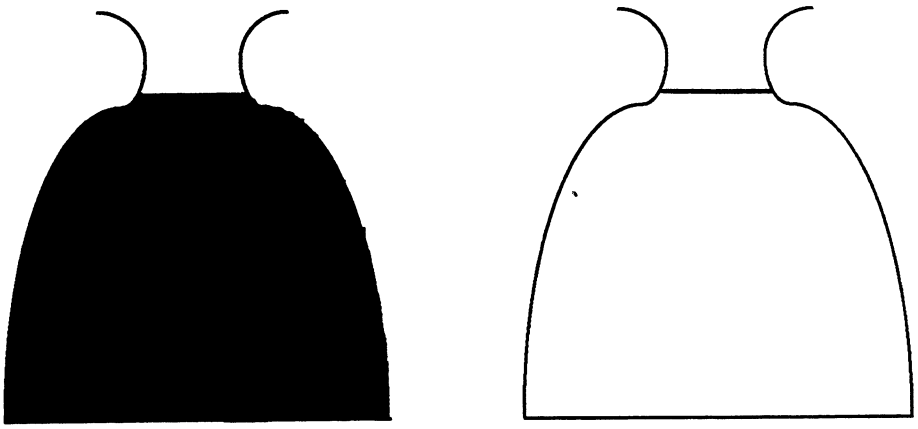


image originale — les contours détectés

Figure 6. Exemple de détection de contour

```
gspotn = or (gspotnsew3, gspotn);
gspots = shift (gspotn, south, 1);
gspotns = or (gspotn, gspots);
gspote = shift (gspotns, east, 1);
gspotnse = or (gspotns, gspote);
gspotw = shift (gspotnse, west, 1);
gspotnsew4 = or (gspotnse, gspotw); /* result of second dilation */
}/* end of opening */
```

3.2. Filtrage Sobel

Le filtrage de Sobel rentre dans la catégorie de technique de masquage qui est une technique particulière de filtrage. Le produit de convolution d'une image par un filtre de fonction impulsionnelle donnée est une image obtenue par le balayage de l'image par ce filtre.

Il s'agit en fait de calculer le gradient de l'image autour d'un pixel dans la direction est-ouest. En d'autres termes, il faut calculer pour chaque pixel de l'image une quantité notée X obtenue à partir des données stockées par ses 8 voisins, selon la formule suivante:

$$X = (A+2B+C) - (D+2E+F)$$

où A, B, C, D, E, F représentent les données stockées par les 8 voisins.

A	I	G
B	H	F
C	D	E

```
/* This program calculates the gradient of an image about a pixel in the
east-west direction*/
main()
image M(128,128,4), A(128,128,4), Ash(128,128,4);
image B(128,128,4), Bsh(128,128,4), Res(128,128,4),
{
A = in (M, p);
Ash = shift (A, north, 1);
```

La description dans notre langage est donné par le programme suivant.

```
/* This program uses a simple algorithm to find edges in a binary image
*/

main()

image gel(27,12,1), gspot(27,12,1), gspotN(27,12,1), gspotNS(27,12,1);
image gspotS(27,12,1), gspotE(27,12,1), gspotW(27,12,1), ;
image gspotNN(27,12,1);gspotES(27,12,1), gspotEN(27,12,1);
image gspotNE(27,12,1), gspotNW(27,12,1), gspotEE(27,12,1);
{
/* get in silhouette of gel spots */
gspot = in (gel, p);
gspotW = shift (gspot, west, 1);
gspotNW = not (gspotW);
gspotEE= and (gspot, gspotNW); /* East edge */
gspotN = shift (gspot, north, 1);
gspotNN = not (gspotN);
gspotES = and (gspot, gspotNN); /* South edge */
gspotS = shift (gspot, south, 1);
gspotNS = not (gspotS);
gspotEW = and (gspot, gspotNS); /* West edge */
gspotE = shift (gspot, east, 1);
gspotNE = not (gspotE);*/
gspotEN = and (gspot, gspotns); /* North edge */
gspotEdges=gspotEE+gspotES+gspotEW+gspotEN
}/* end of edge detection */
```

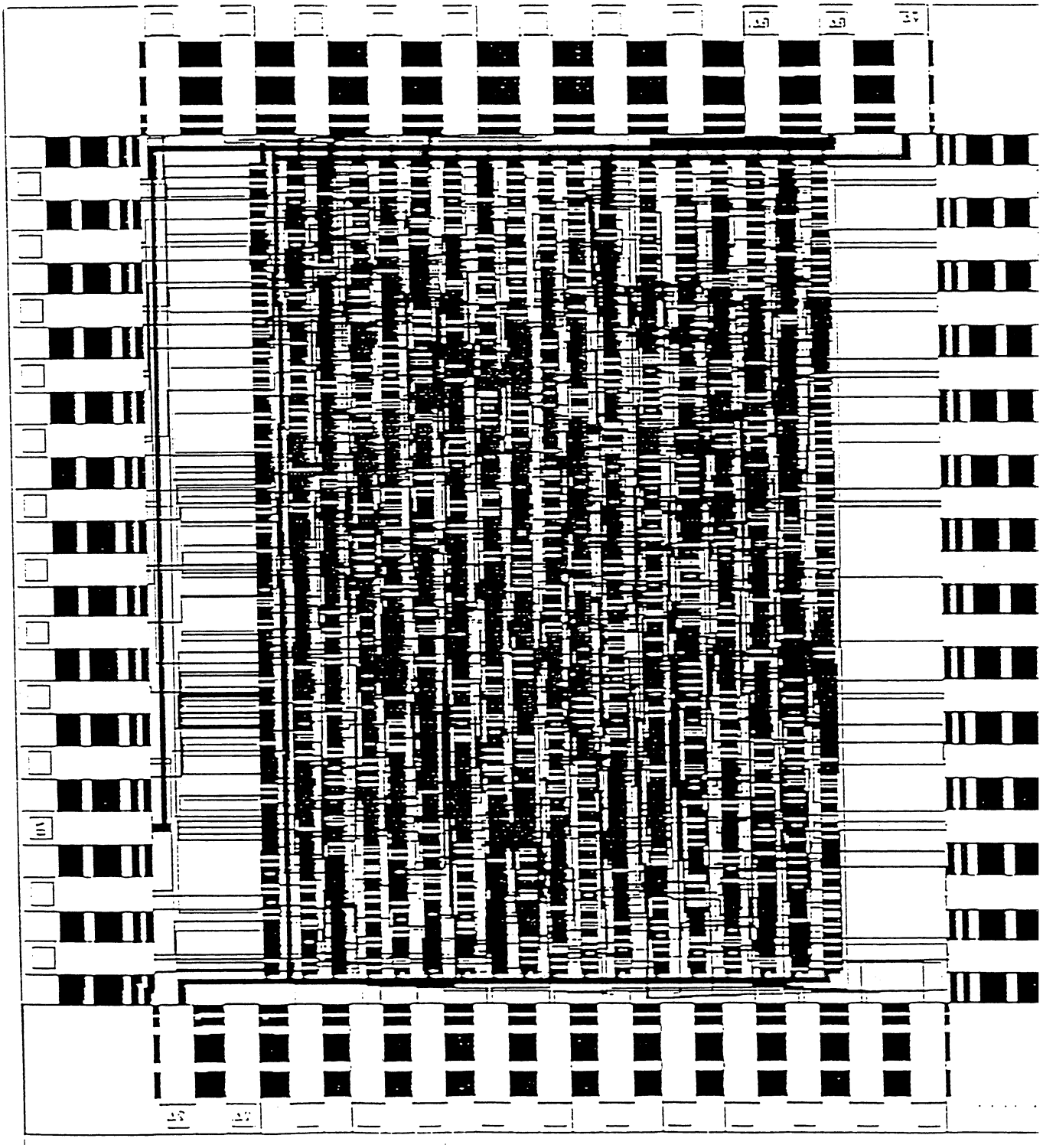


Figure 7 Layout du ciruit de la fermeture (72) PEs.

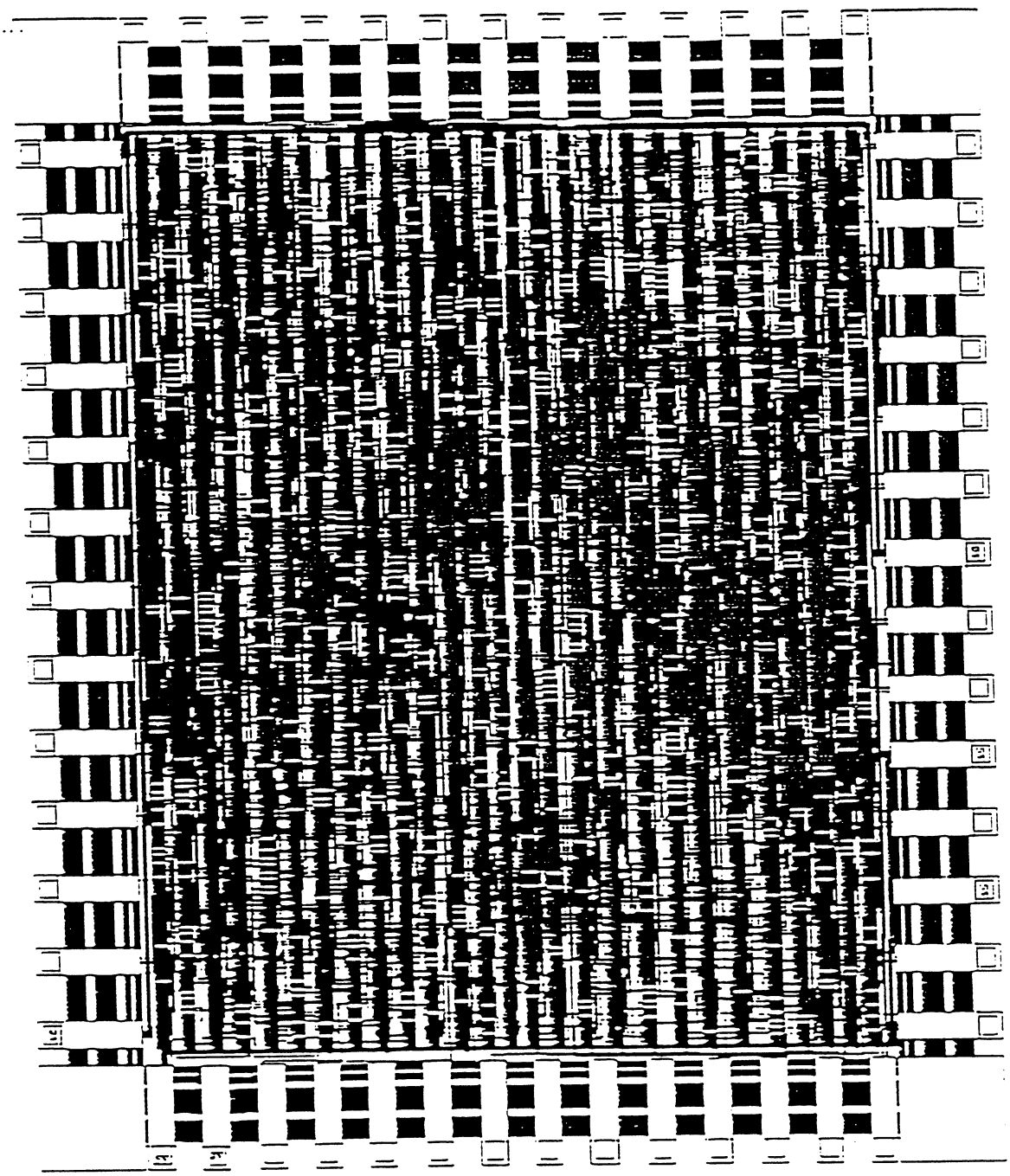


Figure 8 Layout du ciruit de la détection de contour (72 PEs).

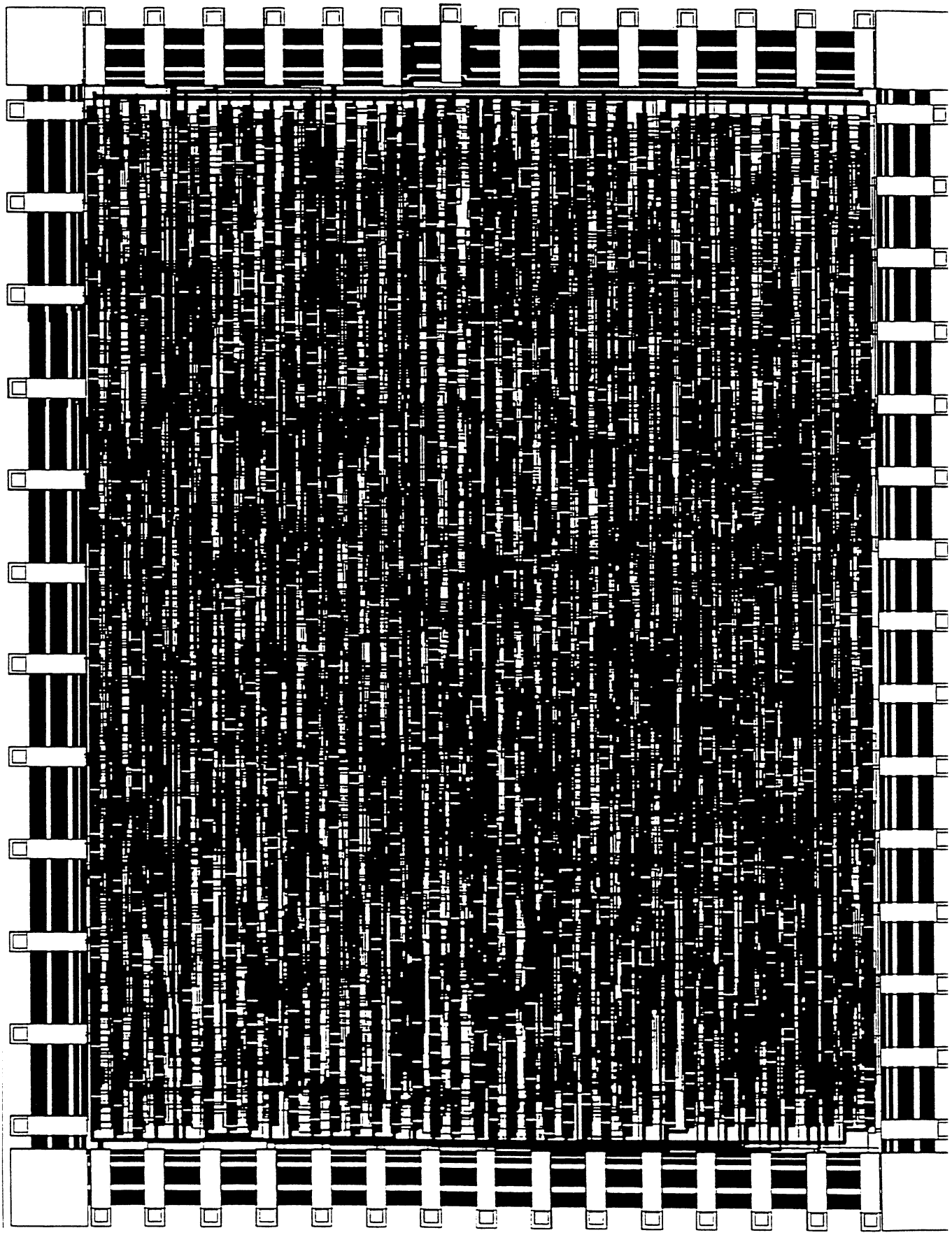


Figure 9 Layout du ciruit de filtre Sobel (72 PEs).

PUBLICATIONS

Les travaux décrits dans cette thèse ont donné lieu aux publications suivantes

A. Boubekeur, J. Patry, and G. Saucier, "Practical Experiences in the Design of a Wafer Scale 2-D Array," *International Workshop on Defect and Fault Tolerance in VLSI Systems*, Tampa, Fl, Oct. 1989.

A. Boubekeur, J. Patry, and G. Saucier, "Practical Experiences in the Design of a Wafer Scale 2-D Array," *International Workshop on Defect and Fault-Tolerance in VLSI Systems*, Vol 2, editors C.H. Stapper, V.K. Jain and G. Saucier, Oct. 1989.

A. Boubekeur, J. Patry, and G. Saucier, "Reconfiguration in ELSA," *IFIP Parallel Architectures on Silicon*, Grenoble, France, Dec. 1989.

A. Boubekeur, Y. Wang, and G. Saucier, "ELSA in a Real-Time Image Processing Environment," *IFIP Parallel Architectures on Silicon*, Grenoble, France, Dec. 1989.

A. Boubekeur and G. Saucier, "A Silicon Compiler for Massively Parallel Image Processing ASICs", *International Workshop on Algorithms and Parallel VLSI Architectures*, Pont-à-Mousson, France, Jun. 1990.

A. Boubekeur and G. Saucier, "A Silicon Compiler for Massively Parallel Image Processing ASICs", *3rd Symposium on the Frontiers of Massively Parallel Computing*, College Park, MA, Oct. 1990

A. Boubekeur and G. Saucier, "A Silicon Compiler for Massively Parallel Image Processing ASICs," *1990 IEEE Workshop on VLSI Signal Processing*, San Diego, CA, Nov. 1990.

A. Boubekeur, G. Saucier, and J. Trilhe " A Reconfigurable WSI SRAM," *IEEE Journal of Solid State Circuits*, Vol 26, N° 10, Oct. 1991

A. Boubekeur, J. Patry, and G. Saucier, "Switch Programming in a 2-D Array," *1990 International Workshop on Defect and Fault-Tolerance*, Grenoble, France, Nov. 1990.

A. Boubekeur, J. Patry, and J. Trilhe, " Universal Switching Network : Application to a WSI SIMD Array," *1991 IEEE International WSI Conference*, San Fransisco, CA, Jan. 1991.

A. Boubekeur and G. Saucier, "A High-level Synthesis tool for Massively Parallel Image Processing ASICS," *Fifth International High-level Synthesis Workshop*, Buhlerhohe, Black Forest, Germany, Mar. 1991.

A. Boubekeur and G. Saucier, "Automatic Generation of Massively Parallel Processing ASICs," *IEEE CompEuro 91 Conference*, Bologna, Italy, May 1991.

A. Boubekeur and G. Saucier, "Automatic Generation of Massively Low-level Parallel Image Processing ASICs," in *Algorithms and Parallel VLSI Architectures*, edit E.F. Deprette, Elsevier, North Holland, 1991.

A. Boubekeur, J. Patry and G. Saucier, "State of the Art of the Wafer Scale ELSA project," *International Workshop on Defect and Fault-Tolerance in VLSI Systems*, Hydden Valley, PA, Nov. 1991.

A. Boubekeur et A. Mignotte, "Un Compilateur pour les Applications Spécifiques au Traitement d'Images Bas Niveau," AFCET/GCIS Workshop sur la Synthèse et Compilation d'Architectures : Méthodes de Spécification et de Conception, Grasse, France, Jan. 1992

A. Boubekeur, J. Patry and G. Saucier, "Configuration of a wafer scale 2D array of Single-bit processor," To appear in *IEEE Computer*, April 1992.

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de

. Monsieur GUYOT Alain , Maître de Conférence
. Monsieur GLESNER Manfred , Professeur

Monsieur BOUBEKEUR Ahmed

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité " Microélectronique "

Fait à Grenoble, le 20 février 1992

/ BV.

Pour le Président de l'INPG
et par délégation
le Directeur de l'Ecole Doctorale
J.-L. LACOUME

INPG

RESUME

Cette thèse propose des méthodes et outils de synthèse d'architectures intégrées massivement parallèles destinées au traitement d'image de bas niveau.

Dans une première partie, l'implantation sur tranche entière d'un réseau 2D de 6720 processeurs (PEs) monobits, dans le cadre d'un projet Européen ESPRIT (WSI 824), est présentée. L'accent a été mis sur les techniques de tolérance aux défauts de fin de fabrication, obligatoires pour un circuit de grande échelle. Une approche hiérarchique a été adoptée. Au niveau sous-système, une colonne de PEs de réserve destinée à remplacer les PEs défaillants a été implantée. Au niveau de la tranche, un réseau de commutateurs original, contrôlé à partir des plots d'entrée/sortie permet de contourner les sous-systèmes défaillants et de construire la cible définitive.

Dans une deuxième partie un système de synthèse de haut niveau permet de concevoir automatiquement des réseaux de PEs dédiés à une application données à partir d'une spécification comportementale de haut niveau. Les ressources de réseau en termes de registres et connexions (multiplexeurs) sont alors minimisées.

Mots-clés : WSI, reconfiguration, rendement, tolérance aux défauts, réseau d'interconnexion, 2D array, SIMD, traitement d'image, synthèse de haut niveau.

ABSTRACT

This thesis presents methods and tools for the design of massively parallel low level image processing integrated systems.

In a first part, the design of a programmable 2D array of 6720 single bit processors (PEs) implemented on a full wafer within an ESPRIT (824) project is addressed. Two-level defect tolerance techniques have been used. At a subsystem level, a spare column is provided to replace defective PEs. At the wafer level, an original switching network externally controlled is used to bypass defective subarrays.

In a second part, a high level synthesis tool is used to synthesize automatically from a behavioral specification a dedicated array from a specific application. Resources in terms of registers, connections (multiplexers) and memory points are minimized.

Keywords : WSI, reconfiguration, yield, defect-tolerance, switching network, 2D array, SIMD, image processing, high-level synthesis.