Intégration sur tranche d'une architecture massivement parallèle tolérant les défauts de fin de fabrication

Jean-Luc Patry

THESE

Présentée par

PATRY Jean-Luc

pour obtenir le grade de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 23 novembre 1988)

Spécialité : **Micro-électronique**

Intégration sur tranche d'une architecture massivement parallèle tolérant les défauts de fin de fabrication

---

Date de soutenance : 4 mars 1992

Composition du jury :

Madame	Gabrièle SAUCIER	
Messieurs	Jacques MOSSIERE	Président
	Alain GUYOT	Rapporteur
	Peter IVEY	Rapporteur
	Jacques TRILHE	

Thèse préparée au sein du Laboratoire Conception de Systèmes Intégrés

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. L'aboutissement de ce travail a été grandement facilité 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,

Monsieur Peter IVEY, Professeur à l'université de Sheffield, d'avoir accepté d'être rapporteur de cette thèse,

Monsieur Jacques TRILHE, Docteur Ingénieur à SGS-Thomson, d'avoir accepté de faire partie de mon jury. Je le remercie aussi de m'avoir aidé et conseillé durant le projet ELSA.

Je remercie plus particulièrement Ahmed BOUBEKEUR et Edmond KOUKA pour leur collaboration à ce travail. Enfin, je remercie également tous mes collègues du CSI pour l'agréable ambiance de travail.

Résumé

Cette thèse présente des méthodes et outils de conception de systèmes intégrés sur tranche entière (Wafer Scale Integration). L'application traitée (dans le cadre d'un projet européen ESPRIT) est une architecture, constituée d'un réseau 2D de 6720 processeurs (PE) monobits, destinée au traitement d'image de bas niveau. Pour tolérer les défauts de fin de fabrication, une approche hiérarchisée a été implantée. Au niveau sous-système, une technique de redondance figée a consisté à implanter une colonne de PEs de réserve, destinés à remplacer les PEs défaillants. Au niveau tranche entière, une technique de construction d'une cible maximale n'utilisant que des sous-systèmes s'appuie sur l'implantation d'un réseau de commutateurs permettant d'éviter les sous-systèmes défaillants. Une architecture originale des réseaux de commutateurs, contrôlée à partir des plots externes et des algorithmes efficaces de définition et construction du réseau opérationnel constituent les points forts de cette thèse.

Mots-clés : WSI, rendement, tolérance aux défauts, reconfiguration, réseau de commutateur, SIMD.

Abstract

This thesis presents methods and tools for the design of Wafer Scale Integration systems. The demonstrator implemented within an european ESPRIT project is an SIMD 2D array of monobit processors (PE) dedicated to low level image processing. To cope with end of manufacturing defects, two levels of defect tolerance have been implemented. At a subsystems level, spare columns of PEs have been added to replace defective elements. At the wafer level, a switching network allow to bypass defective subsystems. Configuration algorithms aims at defining the largest array that may be constructed which is then physically established on the wafer. The originality of the switching array and the efficiency of the configuration algorithms are the strong points of this thesis.

Keywords : WSI, yield, defect tolerance, reconfiguration, switching network, SIMD.

TABLE DES MATIERES

INTRODUCTION

Cette thèse à comme objectif l'étude de la limite supérieur de la taille d'un circuit intégré CMOS à savoir son intégration sur une tranche entière de silicium. Un circuit de cette dimension contiendra bien entendu des défauts de fin de fabrication et l'étude portera sur les techniques de tolérance à ces défauts. Pour ce faire, un démonstrateur a été choisi, il s'agit d'un système très répétitif à granularité fine ce qui est indispensable pour mettre en place des techniques de redondance par élément de réserve ou des techniques de construction de cibles ne contenant que des éléments bons. L'architecture choisie est une architecture programmable massivement parallèle spécialement adaptée au traitement d'image de bas niveau; elle est constituée d'un matrice 2D de processeurs monobits.

La tolérance aux défauts à été implantée de façon hiérarchique, ceci après des études de rendement qui sont toujours très délicates par suite de la difficulté à obtenir des données industrielles significatives. Sur la tranche entière les processeurs monobits ont été implantés. Ceux-ci sont regroupés à un premier niveau en sous-réseaux. Dans ces sous-réseaux la redondance aux défauts a été implantée en définissant une cible de 6 lignes et 12 colonnes et en y ajoutant une colonne redondante. Quand un élément est défaillant dans la matrice cible, il est remplacé par un élément de la même ligne dans la colonne redondante. Il s'agit donc d'une redondance à réserve relativement classique dans le domaine des architectures tolérant les pannes. Au niveau du réseau global, une technique de configuration plus originale a été utilisée à savoir une technique de configuration du plus gros réseau 2D que l'on peut construire sur la tranche à partir des sous-réseaux opérationnels. Cette technique requiert la résolution de problèmes matériels et logiciels. Au point de vue matériel un réseau double rail de commutateurs a été proposé, conçu et réalisé. Ce réseau permet de contourner les processeurs défaillants pour construire la cible. Sa conception est délicate. Il est configuré à partir de plots externes car un contrôle par des processeurs sur la tranche aurait permis la contamination du réseau par des processeurs défaillants. Une indépendance de ce réseau par rapport aux processeurs a donc été recherchée; ce réseau est une structure universelle et peut s'accommoder de n'importe quelle architecture environnante. Son implémentation a été faite "full custom" pour garantir une grande compacité et maintenir un fonctionnement rapide de l'ensemble de l'architecture, une bufferisation à été nécessaire. Ce réseau de commutateurs est un point clé de la tolérance aux défauts car dans la littérature abondante sur ce problème de la tolérance aux défauts, on ne voit

guère de réalisation de réseaux dûment prouvée et compétitive. Quant aux techniques logicielles nécessaire pour réussir un tel projet, elles ont concerné d'une part le logiciel de configuration définissant un réseau 2D de taille maximale à partir d'un réseau 2D incluant des processeurs défaillants et des connexions défaillantes, d'autre part le logiciel de programmation optimisée à partir des plots E/S de la structure cible définie par le logiciel précédent.

Le premier logiciel est fondé sur des techniques de placement. Il s'agit de définir un réseau 2D opérationnel bien centré sur la tranche et minimisant le maximum de la longueur des connexions entre éléments voisins ou, pour la périphérie, entre un élément périphérique et le plot d'E/S auquel il est connecté. en effet cette distance détermine la fréquence de fonctionnement de la cible. Les méthodes proposées dans ce domaine se différencient totalement de la littérature abondante dans ce domaine en général fondée sur l'hypothèse optimiste que les connexions ne sont pas défaillantes.

Le deuxième logiciel, consistant à programmer les commutateurs dans la position correspondant à la cible, est fondée sur la définition d'arbre permettant d'atteindre tous les commutateurs de la cible par des chemins, en minimisant le maximum de la longueur de ces chemins. Le critère d'optimisation tend, en fait, à minimiser le temps de programmation de la cible.

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.

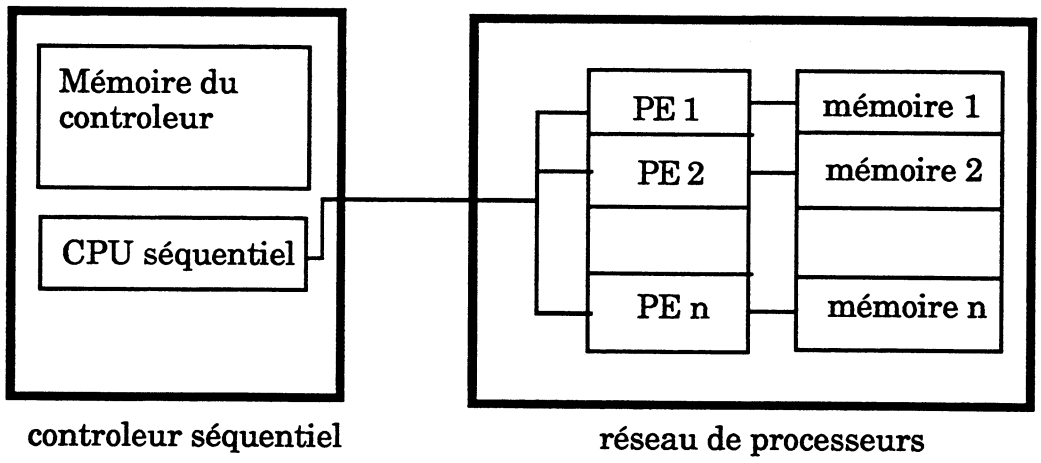


Figure 1.1 Modèle général d'une machine SIMD

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.

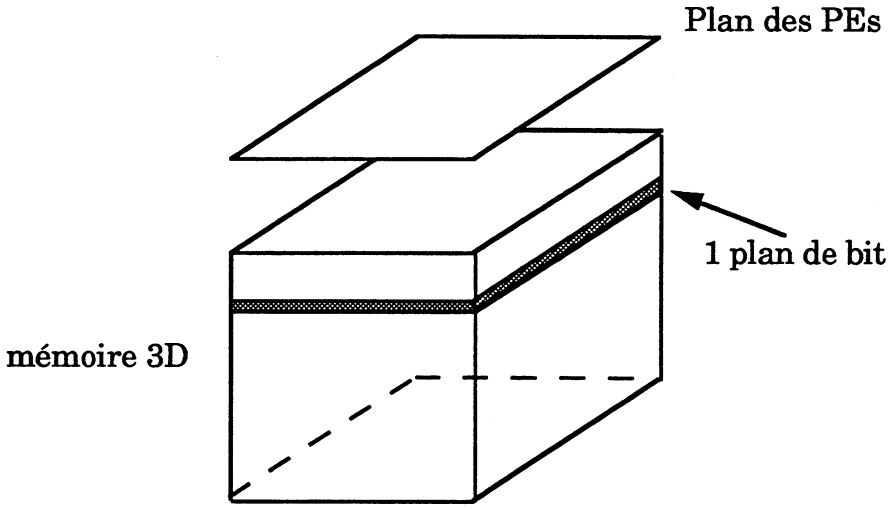


Figure 1.2 Le concept d'une mémoire 3D

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

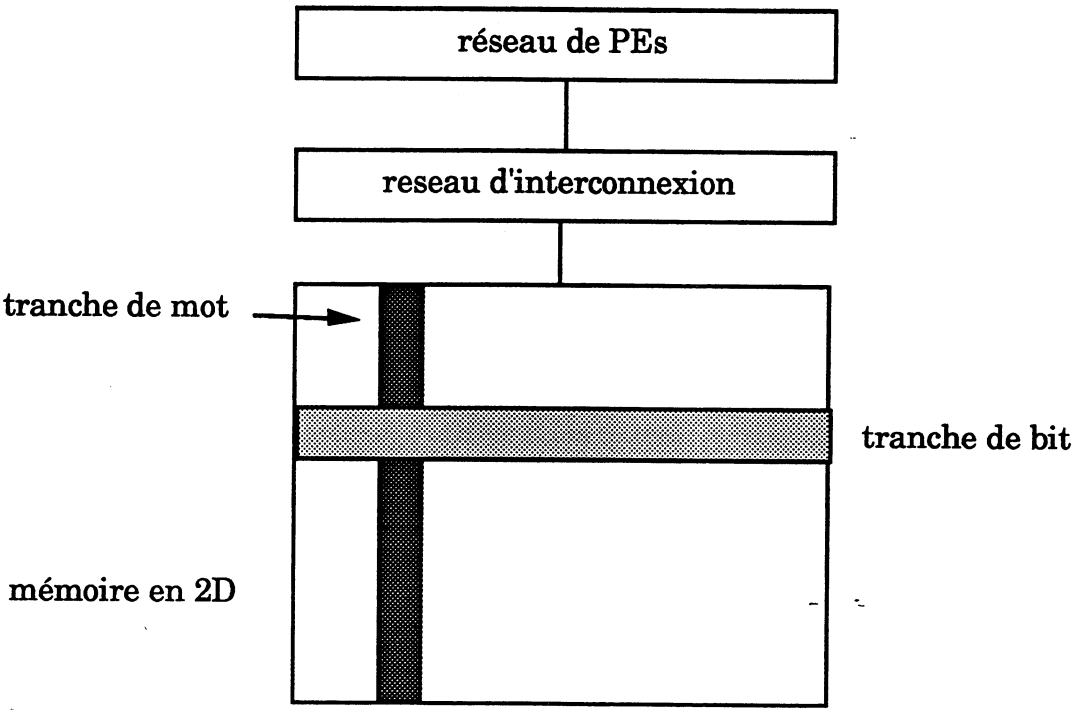


Figure 1.3 Le concept d'une mémoire 2D

Machine	ILLIAC IV	MPP	DAP	GAPP	CM
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.

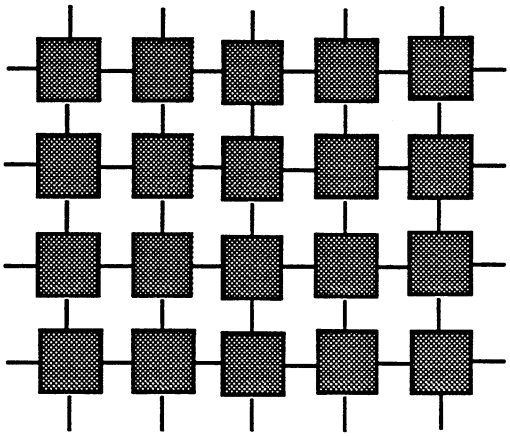


Figure 1.4 Structure de grille

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).

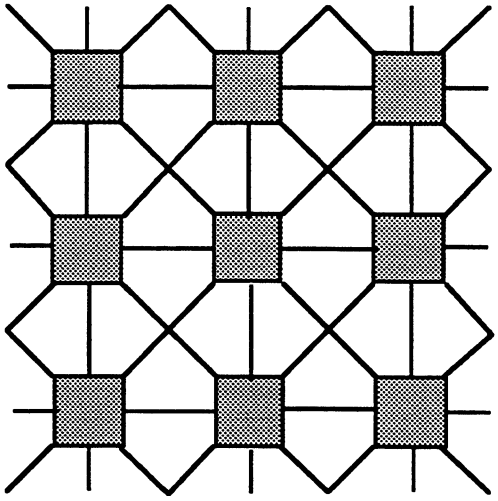


Figure 1.5 Interconnexion en X

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.

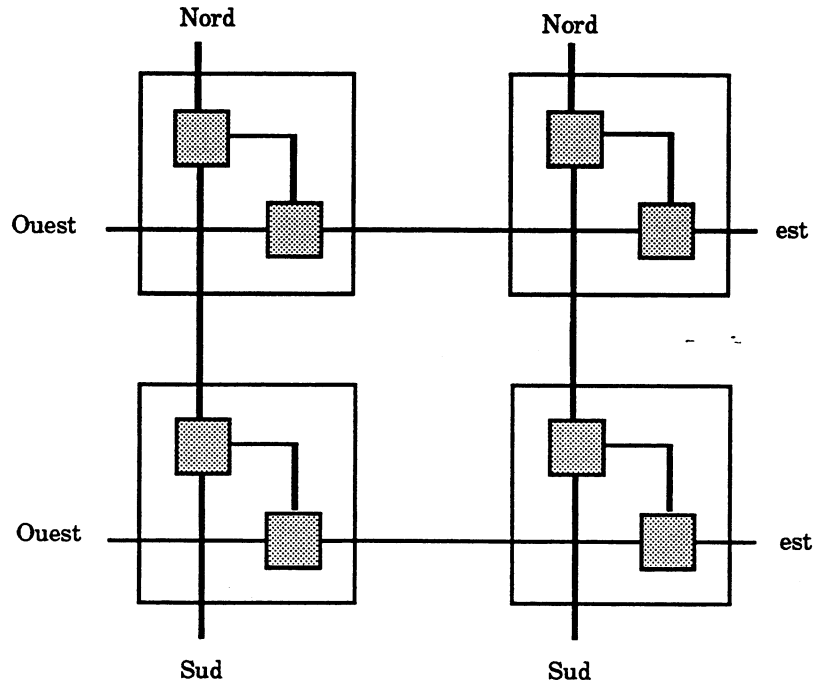


Figure 1.6 Interconnexion ligne-colonne

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.

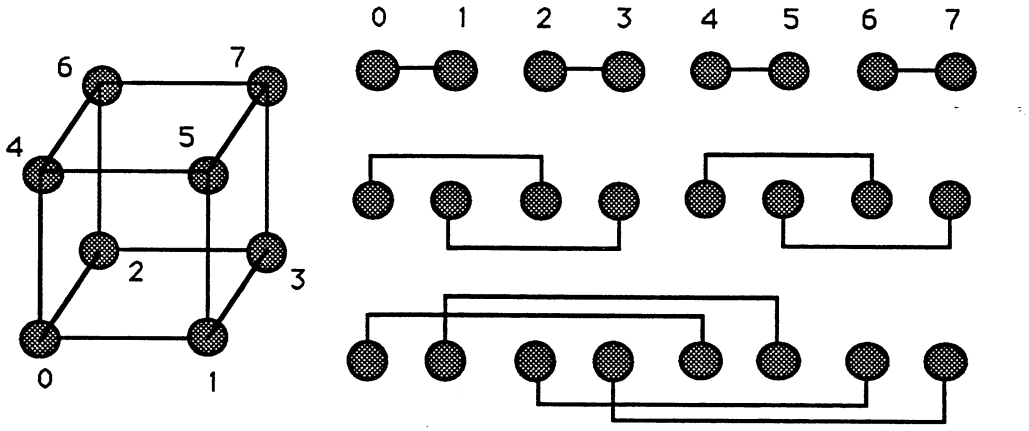


Figure 1.7 Exemple de 3-cube et de ses interconnexions

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 [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 Micro-electronics 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 (NS, EW, 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.

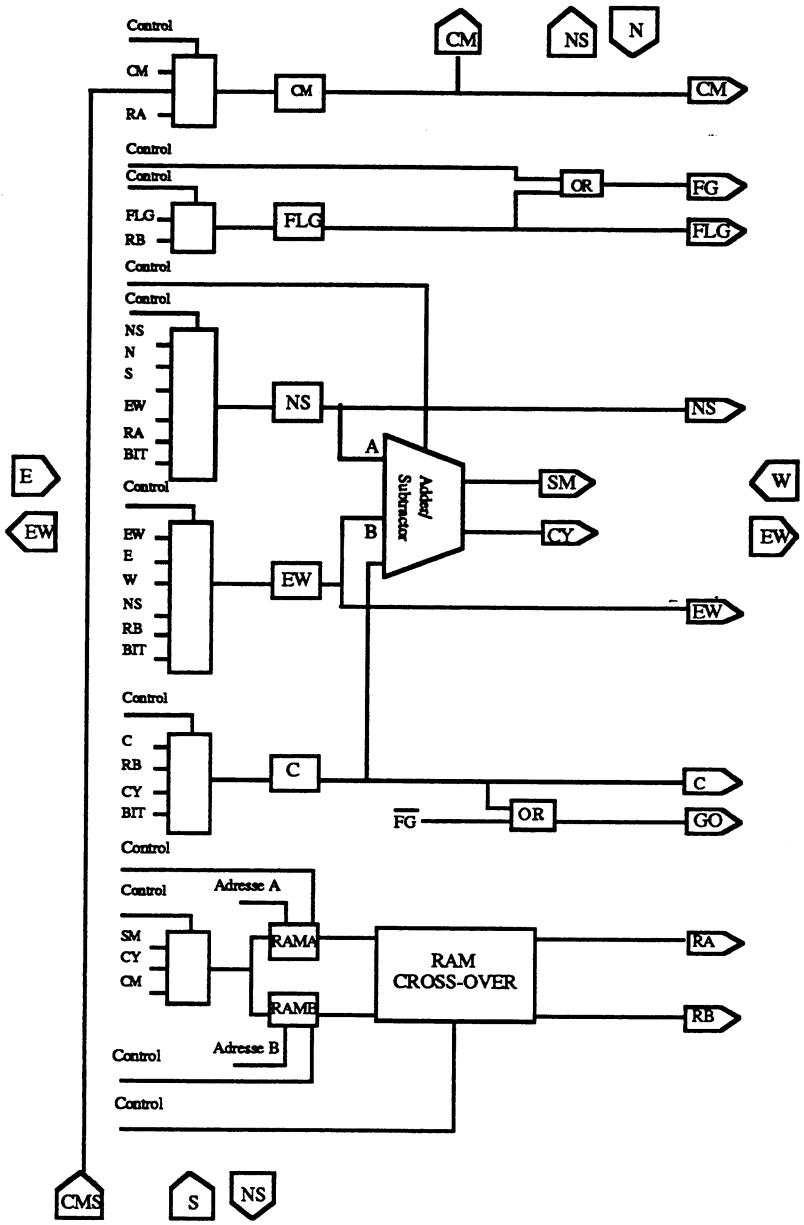


Figure 2.1 Le processeur élémentaire d'ELSA.

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 pendant PHI2.

Celle-ci peut se faire soit dans une RAM ou dans les deux à l'endroit lu auparavant. Pour pallier l'inconvénient dû aux relations figées entre les deux RAMS et les multiplexeurs, un sélecteur à été ajouté. Celui-ci permet d'échanger les sorties des RAMS.

L'architecture du PE est proche de celle du GAPP NCR [Clou88]. Mais, l'introduction d'un registre FLAG rend ce processeur SIMD plus performant que le processeur GAPP; le PE peut exécuter ou non une tâche. D'autre part, la séparation de la mémoire RAM en deux RAMS de 64 bits entraîne un gain de cycle pour l'UAL ainsi qu'une souplesse d'accès et une gestion de la mémoire.

II.1.2 La structure de la matrice

Les PEs forment un réseau 2D (figure 2.2). Chaque PE communique avec ses quatres voisins. Le réseau permet plusieurs possibilités de communication. Les registres CM sont considérés comme une matrice 2D appelé plan CM, utilisé pour les entrées/sorties. De même on parlera des plans NS et EW. Un plan de bits dans le plan NS (ou EW) peut être décalé en un seul coup d'horloge simultanément dans les deux sens.

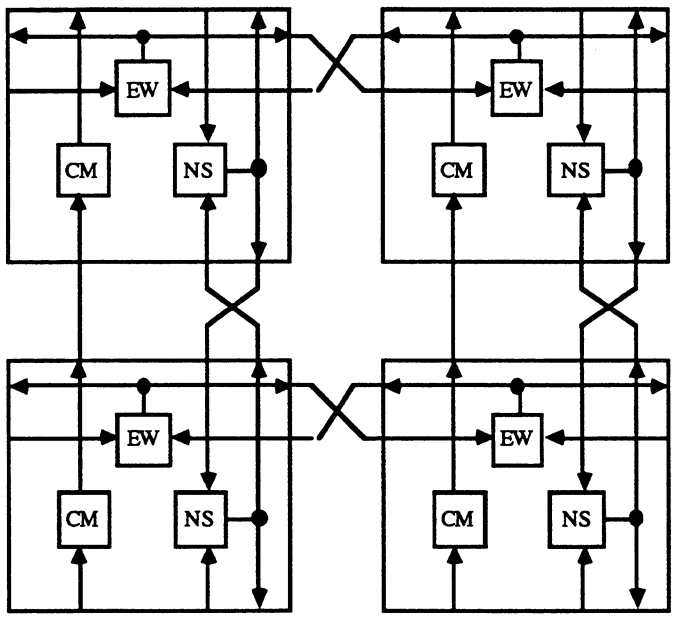


Figure 2.2 Structure de la matrice

L'objet de ce projet est d'intégrer un nombre maximal de PEs sur une tranche et de configurer le plus grand réseau opérationnel.

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.

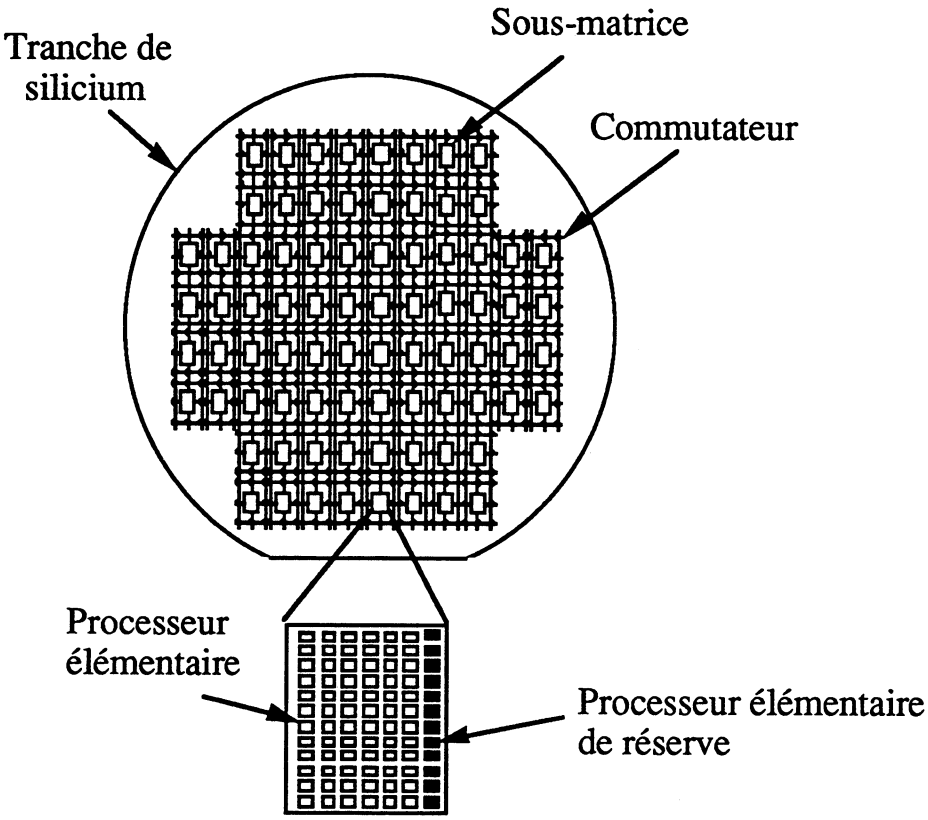


Figure 2.3 L'architecture d'ELSA

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 D_m.

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

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 D_m, puis linéairement décroissante jusqu'à $2D_m$ 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.

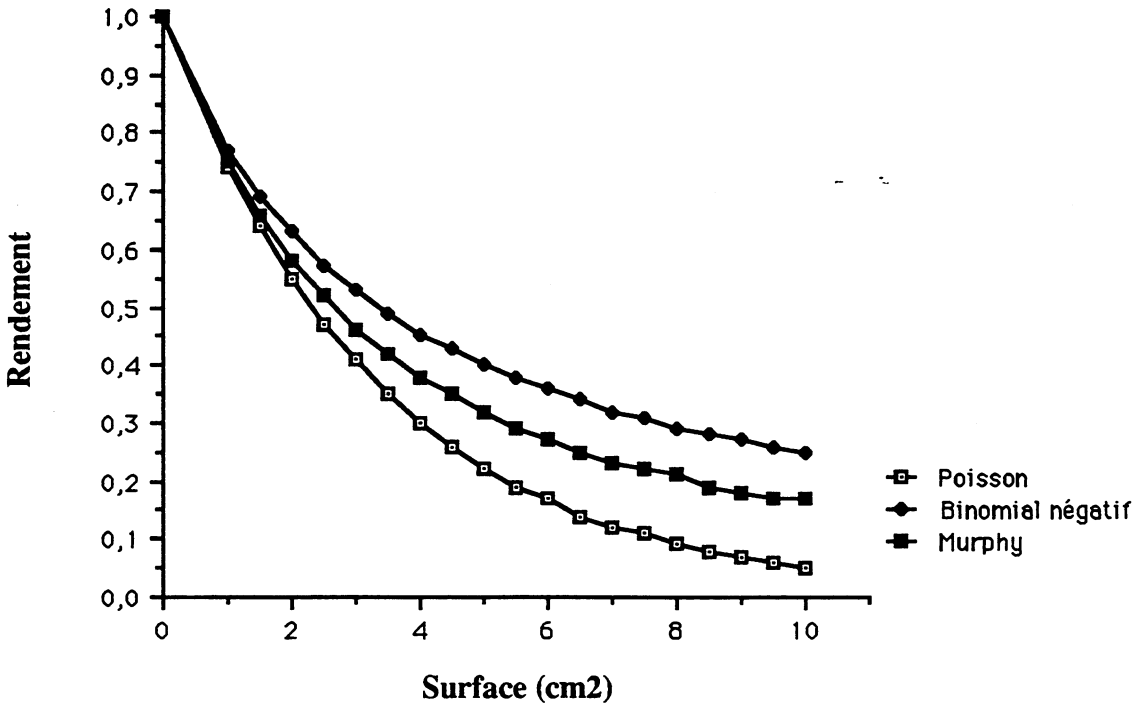


Figure 2.4. Rendement des différents modèles.

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.

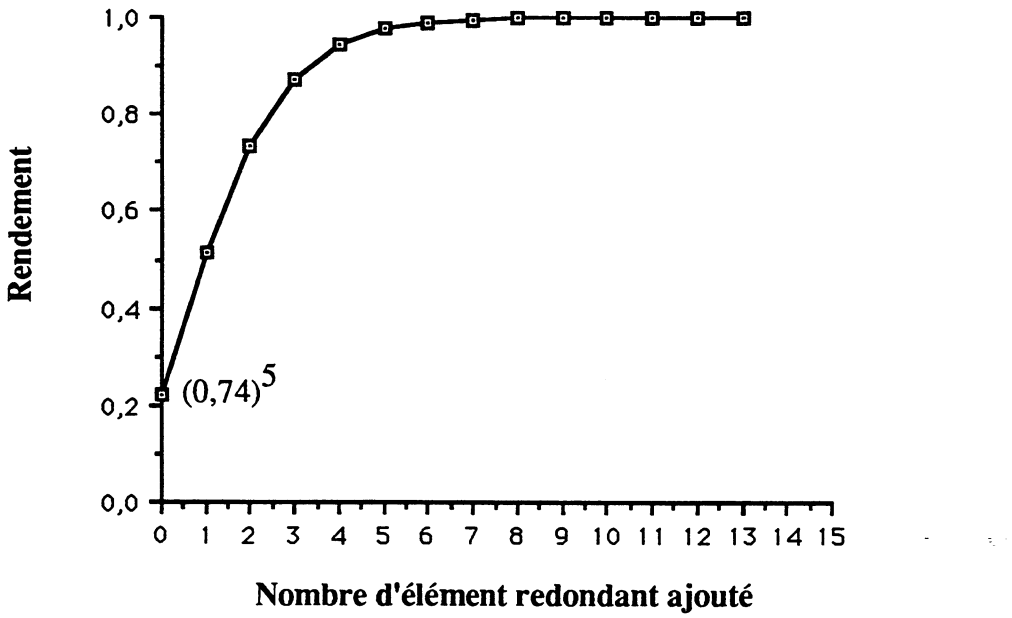


Figure 2.5. Rendement en fonction du nombre d'éléments redondants ajoutés.

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.

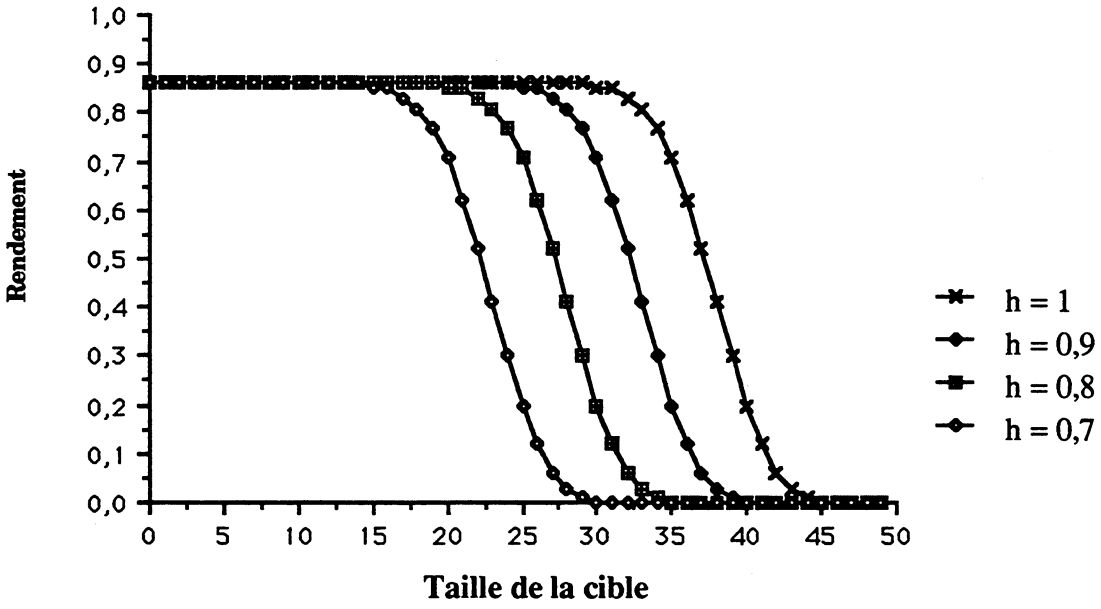


Figure 2.6. Le rendement en fonction de la taille de la cible.

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

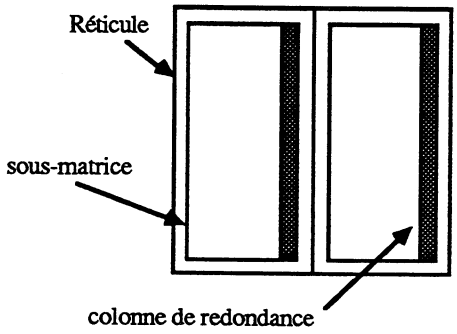
Tableau 2.2.

II.2.3 Rendement d'ELSA

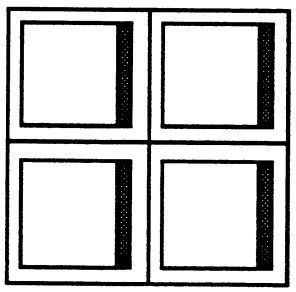
Nous allons utiliser le modèle pessimiste du paragraphe précédent afin d'évaluer le rendement d'ELSA.

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.

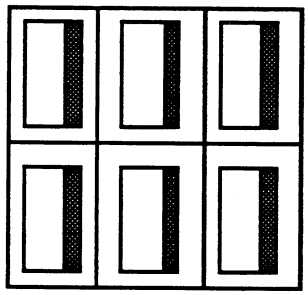
A partir des surfaces du PE (0,361 mm^2), du commutateur (0,256 mm^2) et du réticule (14 x 16 mm^2), nous pouvons évaluer les différentes partitions; c'est-à-dire le nombre de PEs dans une sous-matrice, le nombre de lignes redondantes dans une sous-matrice et le nombre de sous-matrices sur la tranche.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

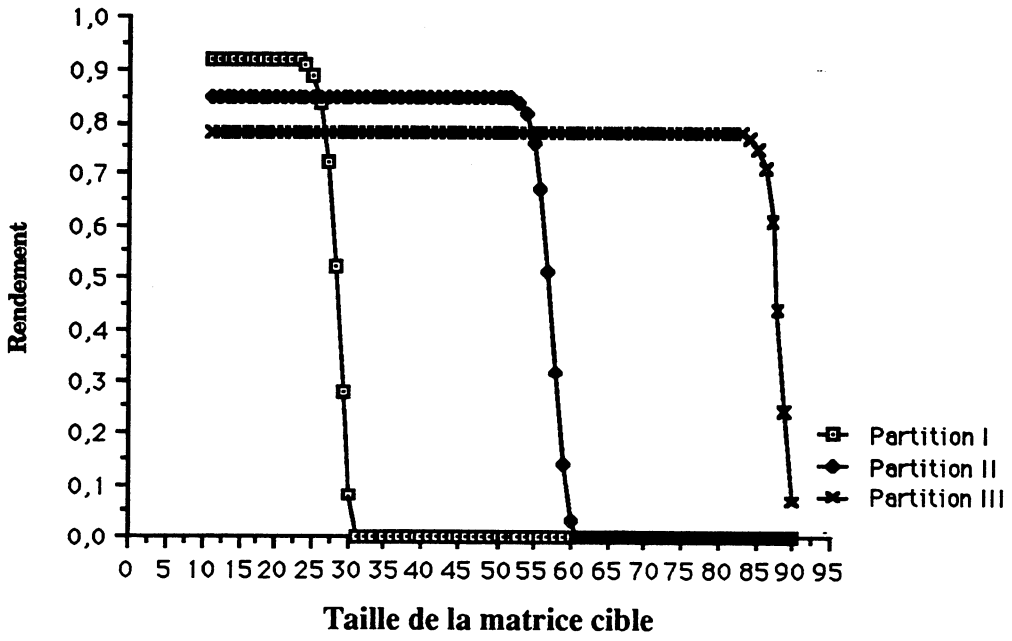


Figure 2.8. Rendement d'ELSA en fonction de la taille de la matrice cible.

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 suffissait 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 autodiagnostic 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'autodiagnostic 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.

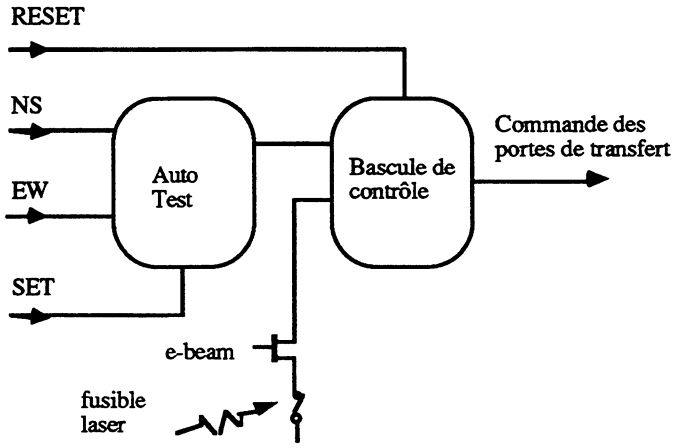


Figure 2.9. Dispositif de reconfiguration au niveau de la sous-matrice.

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

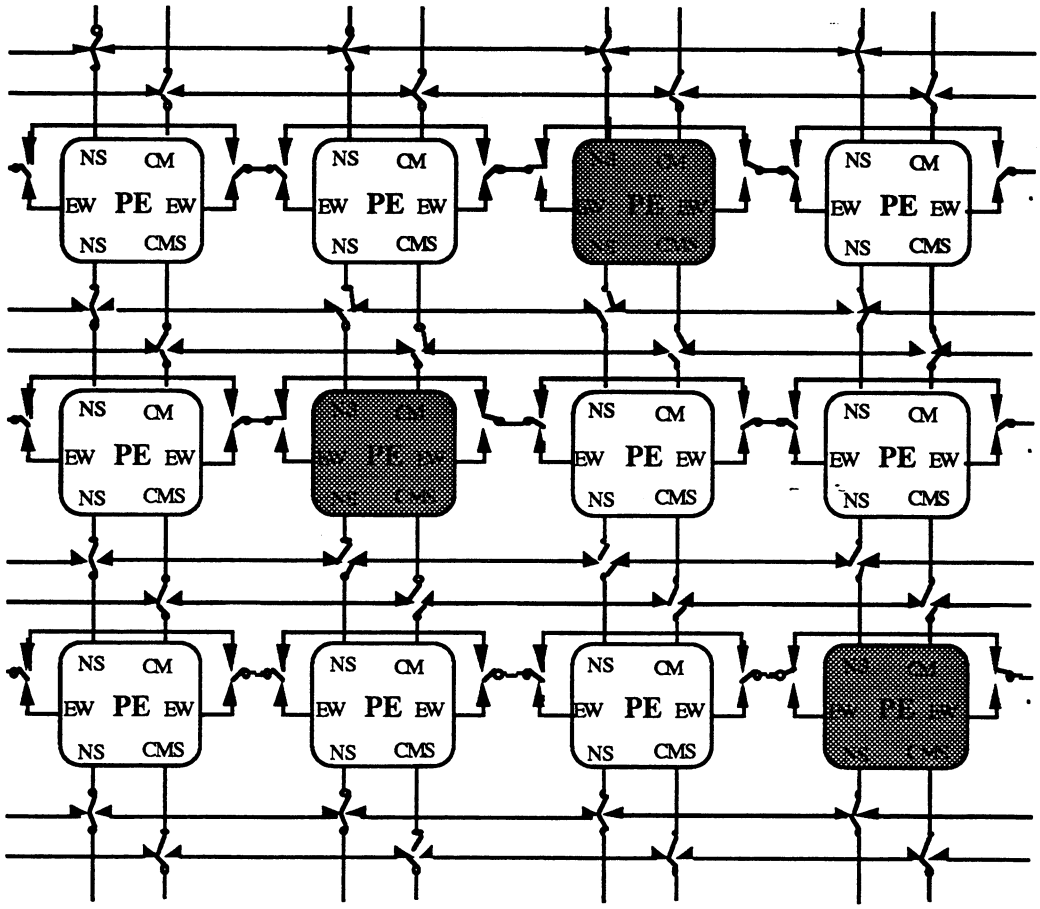


Figure 2.10. Schéma de la sous-matrice.

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.

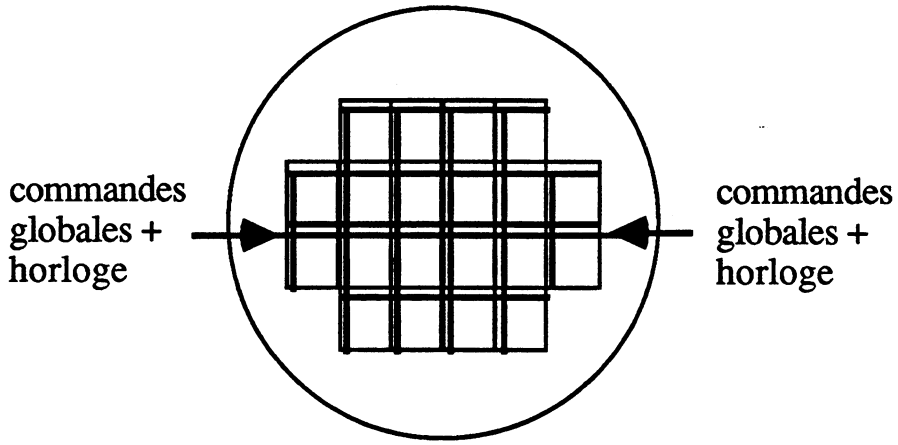


Figure 2.11. Le réseau de commandes globales.

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.

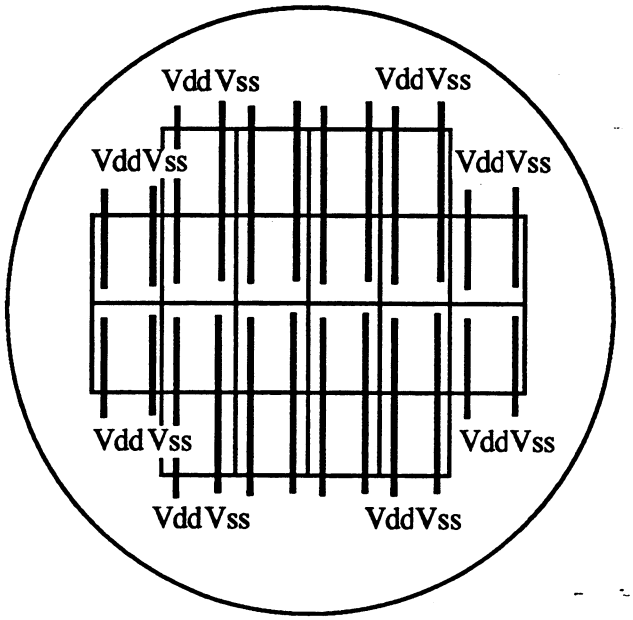


Figure 2.12. La distribution de l'alimentation dans ELSA.

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.

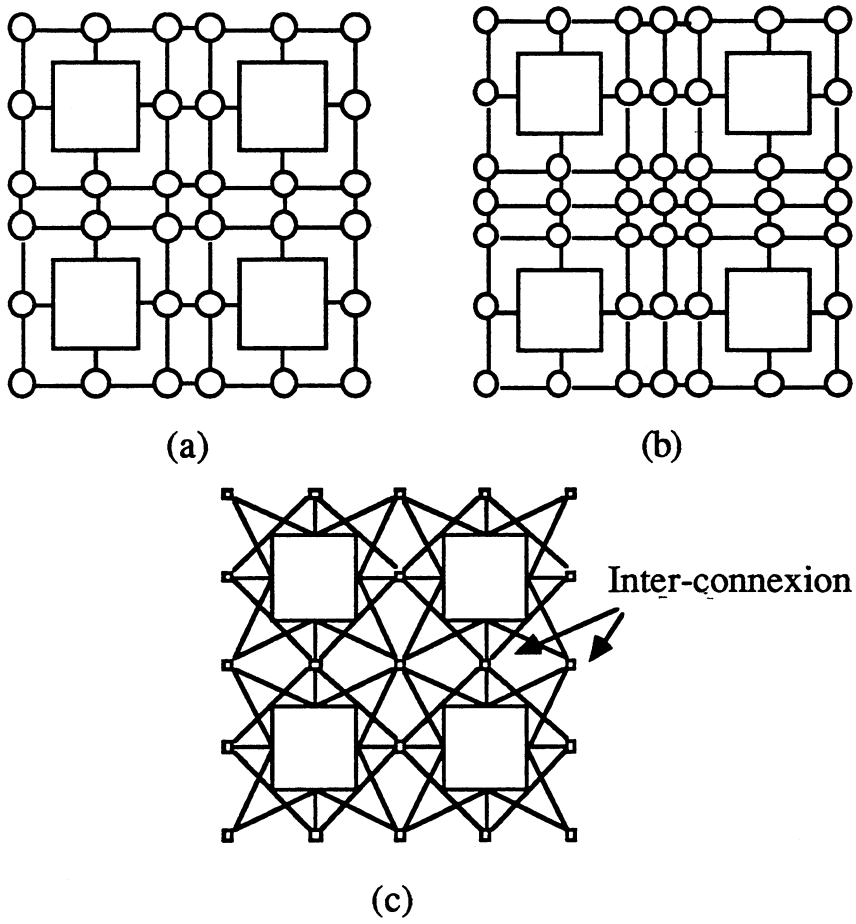


Figure 2.13. (a) Le réseau double-rail. (b) Le réseau triple-rail. (c) Le réseau en étoile.

Critère de tolérance à des cartographie de défauts :

Les cartographies de défaut étudiés 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 complétement 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.

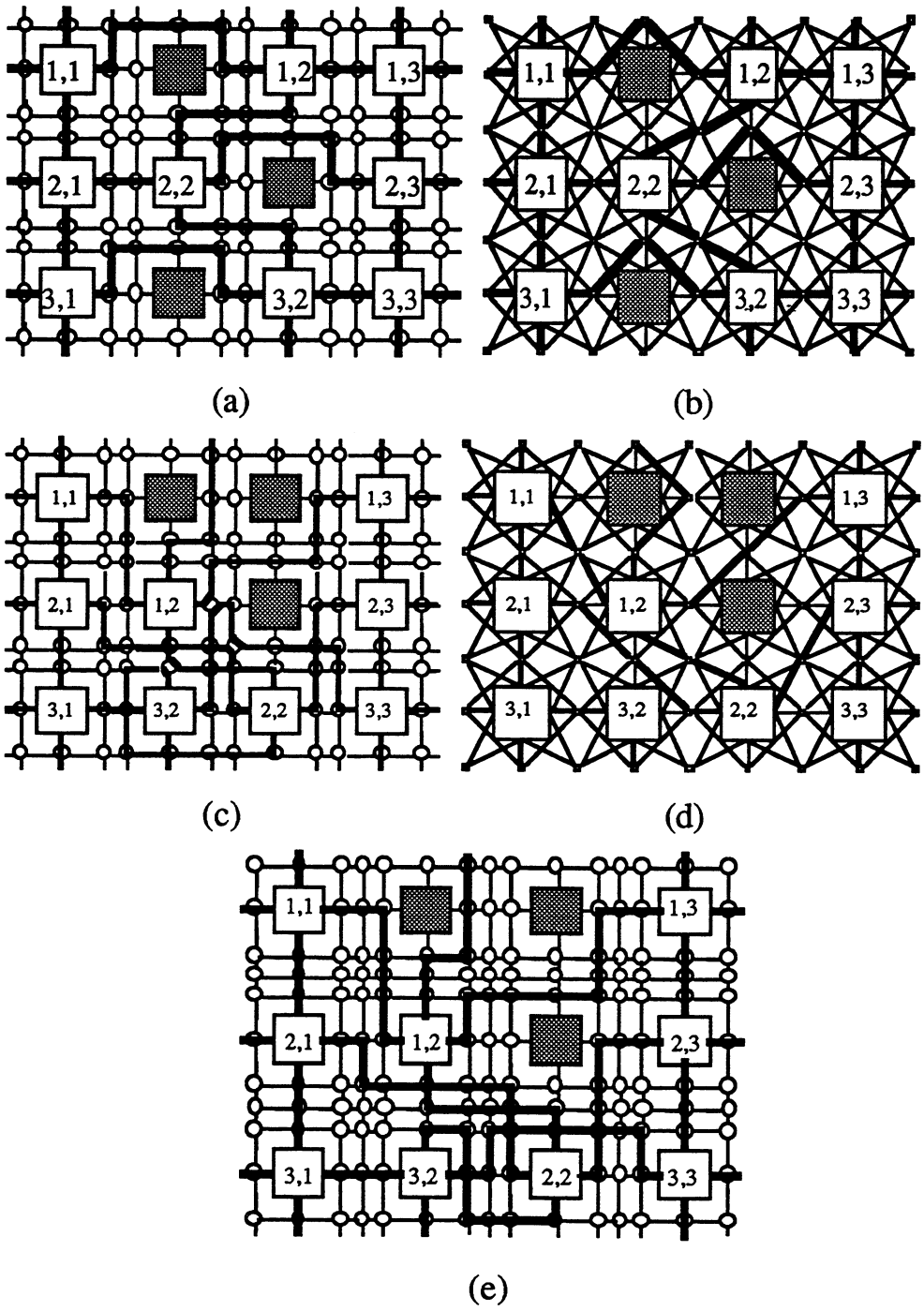


Figure 2.14. (a) et (b) configuration avec le réseau double-rail et le réseau en étoile pour une distribution uniforme de défaut. (c), (d) et (e) configuration

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.

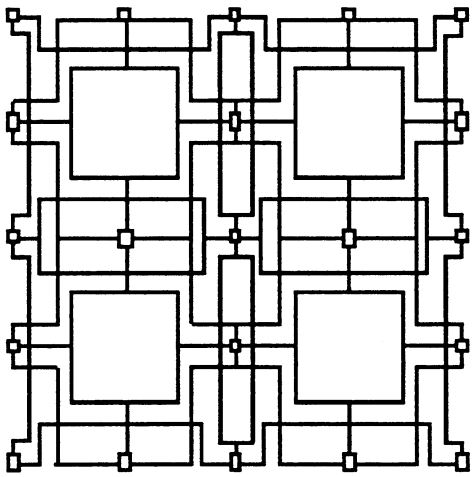


Figure 2.15. Plan de masse du réseau en étoile.

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).

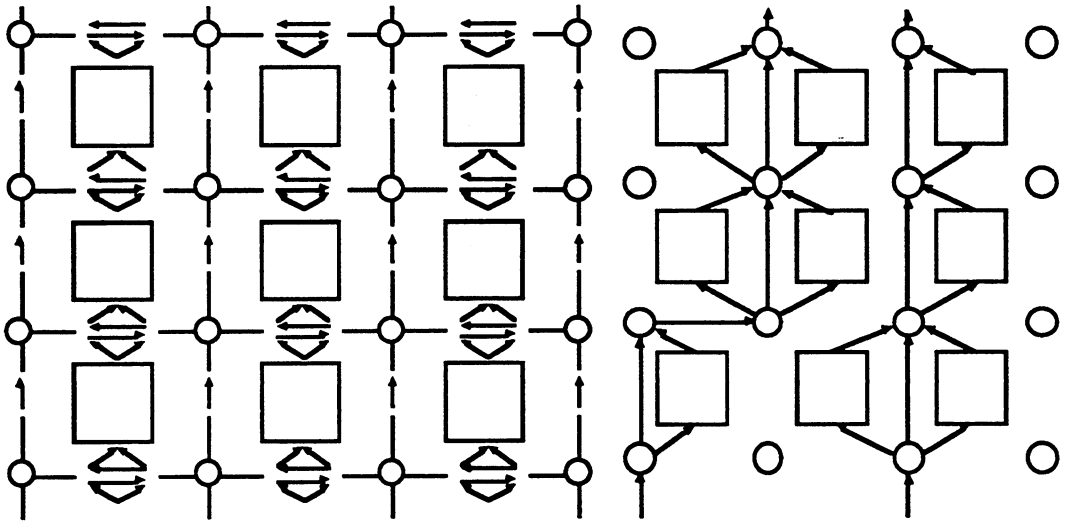
II.3.3 Les commutateurs programmables

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 [KiRe 89], 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.

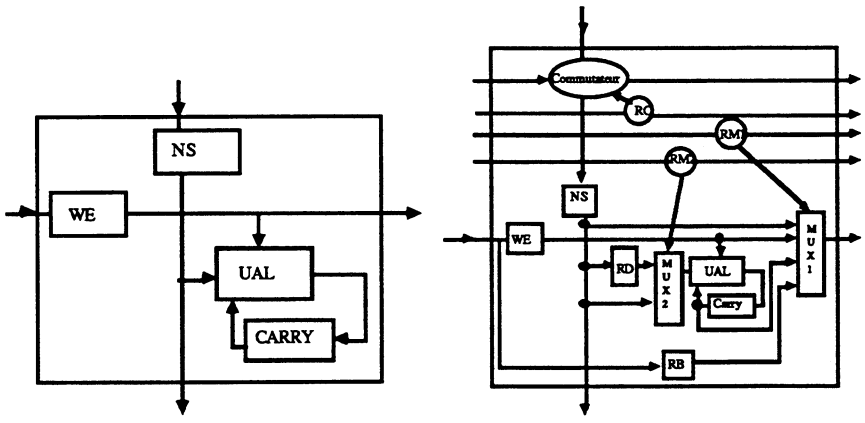


Figure 2.17. (a) L'architecture générale d'un processeur élémentaire. (b) L'architecture du processeur permettant la configuration.

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.

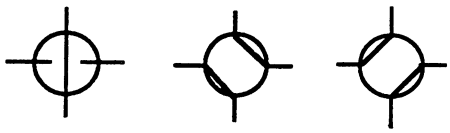


Figure 2.18. Les trois configurations du commutateur.

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.

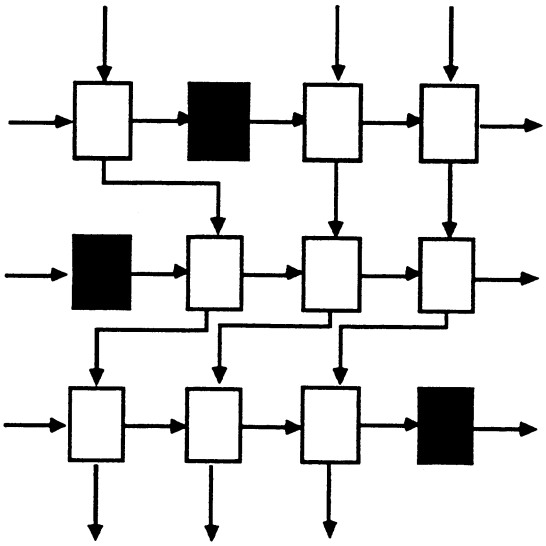


Figure 2.19. Reconfiguration d'une matrice logique (3 x3).

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 tolèrant 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.

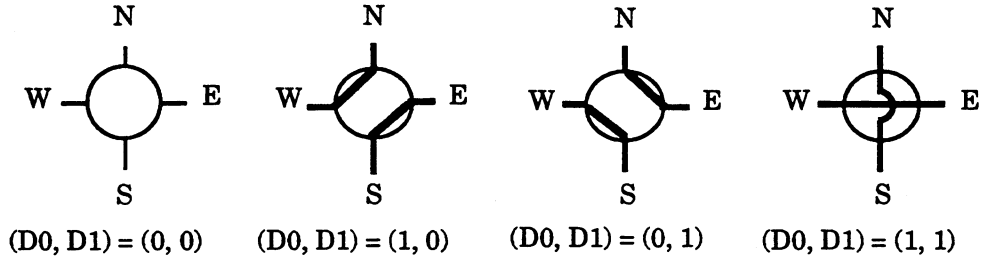


Figure. 2.20. Les quatre configurations du commutateur.

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).

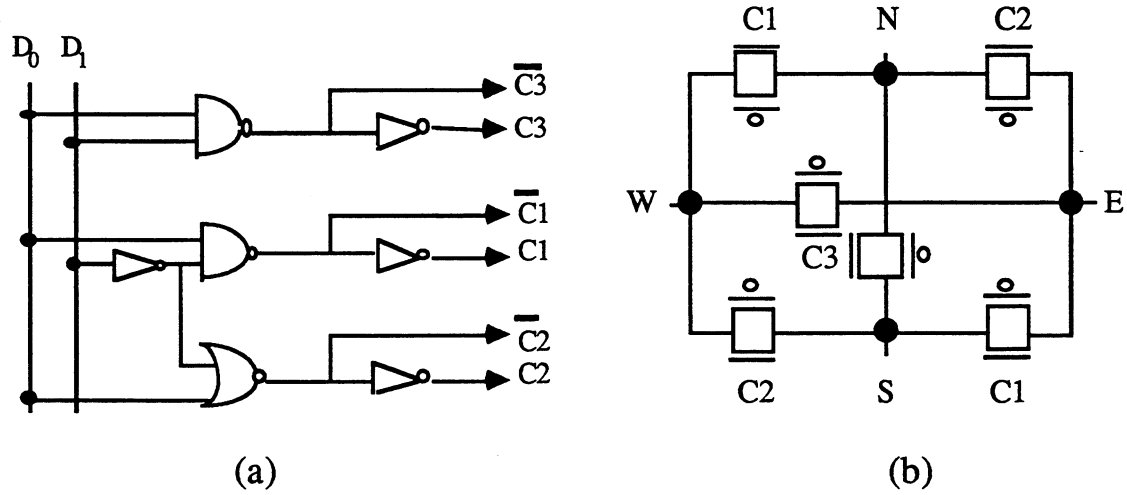


Figure 2.21.(a) Le contrôle de la partie TRANSFERT.(b) La partie TRANSFERT.

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).

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.

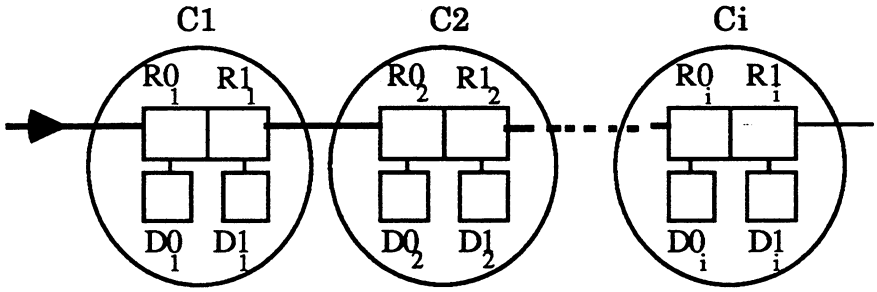


Figure 2.22. Chemin pour atteindre le commutateur i.

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.

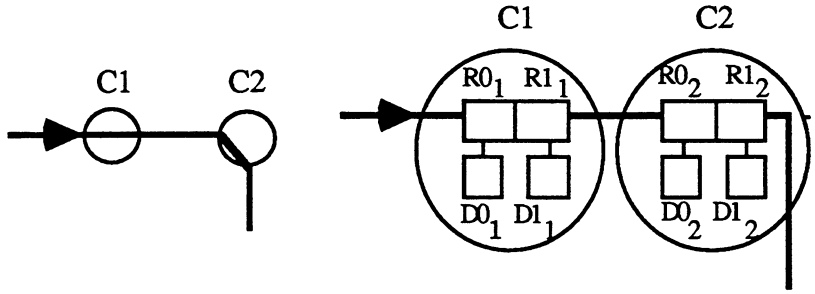


Figure 2.23 (a) Chemin a créer. (b) les registres et les bascules associés.

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.

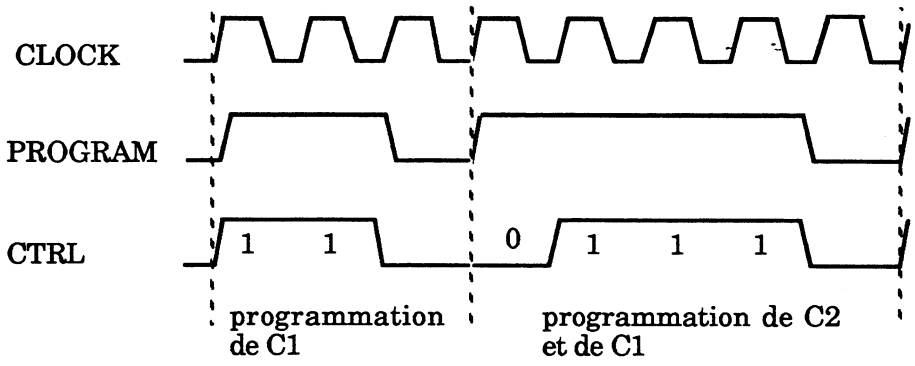


Figure 2.24. Chronogramme de la programmation de C1 et C2.

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.

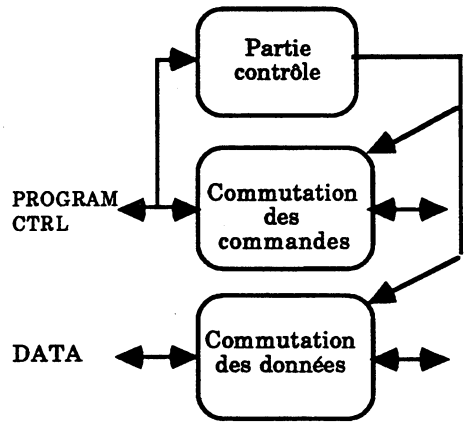


Figure 2.25. L'Architecture du commutateur.

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).

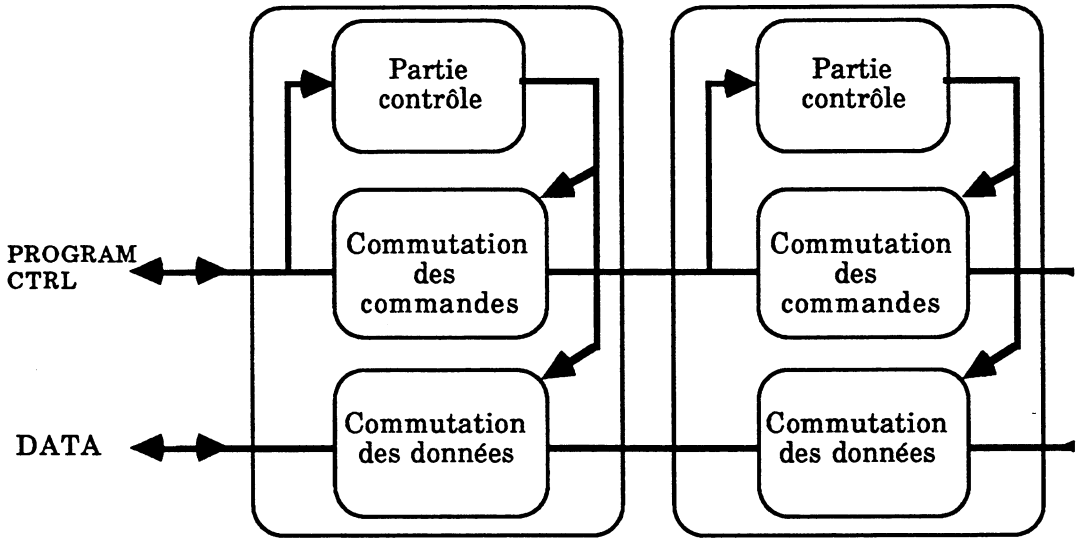


Figure 2.26. Les signaux de programmation sont propagés par la partie commutation.

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.

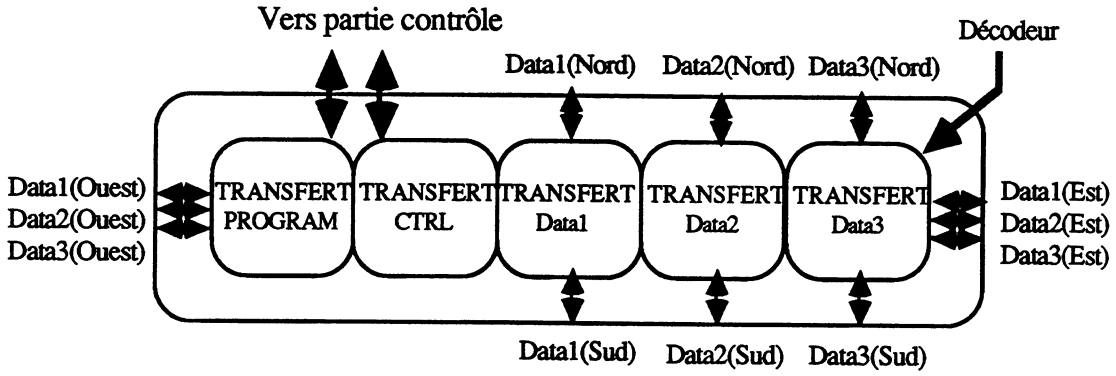


Figure 2.27. La partie commutation.

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).

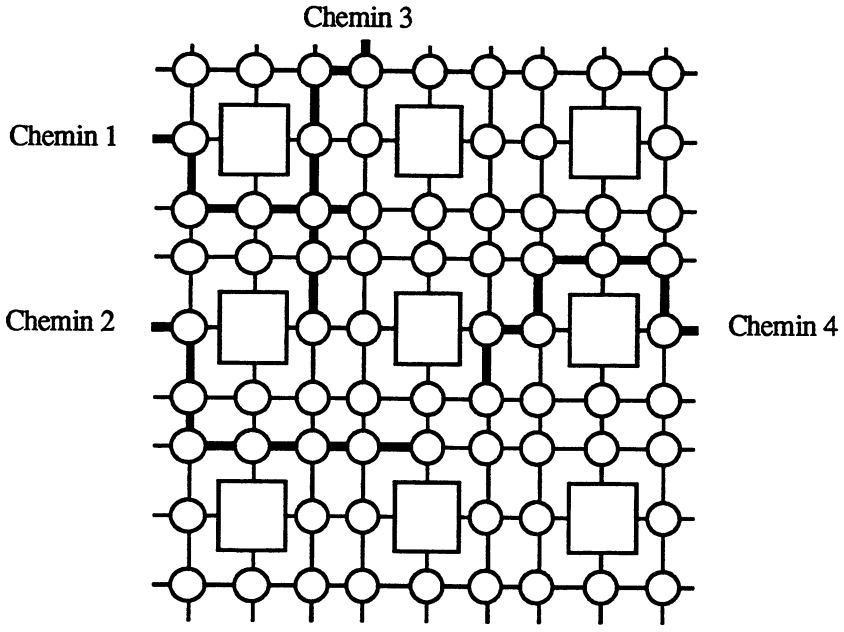


Figure 2.28. Programmation de plusieurs chemins.

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).

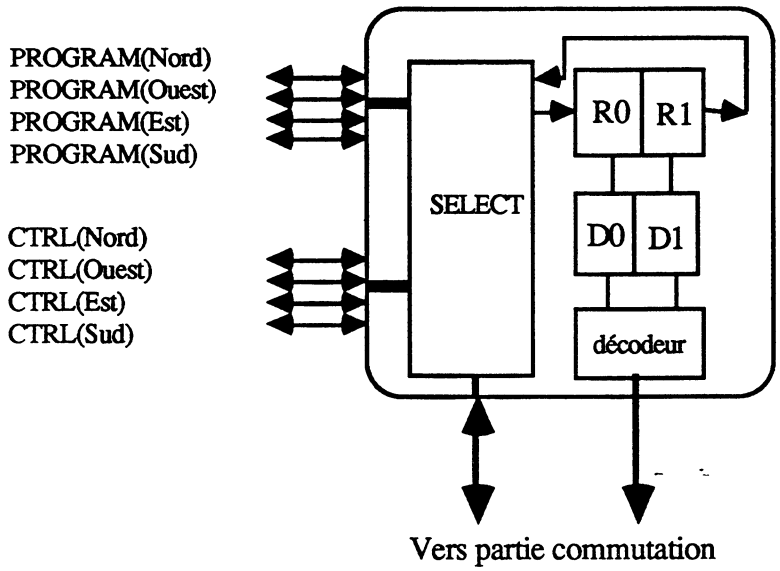


Figure 2.29. L'architecture de la partie contrôle.

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 signa 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)

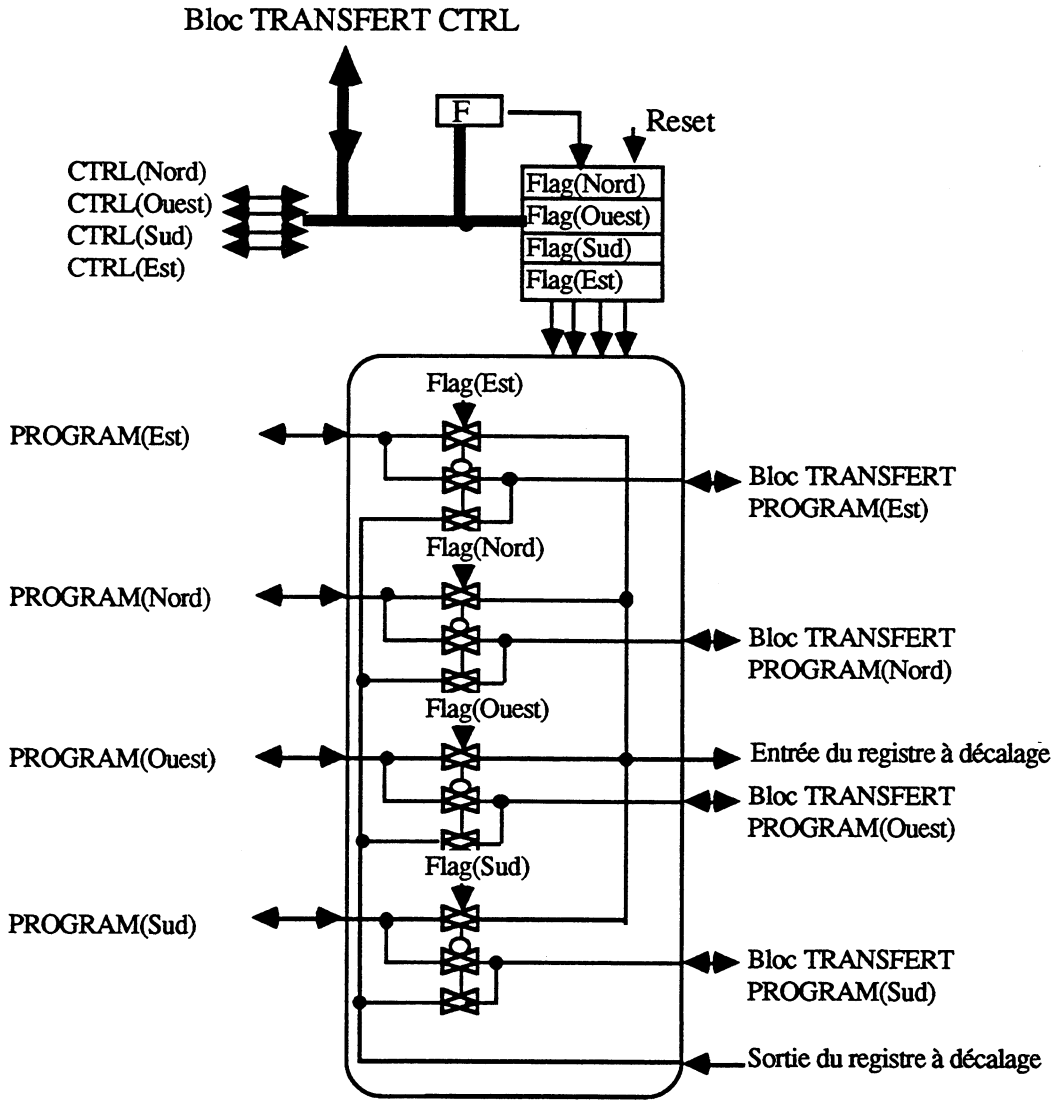


Figure 2.30. L' architecture du bloc SELECT

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.

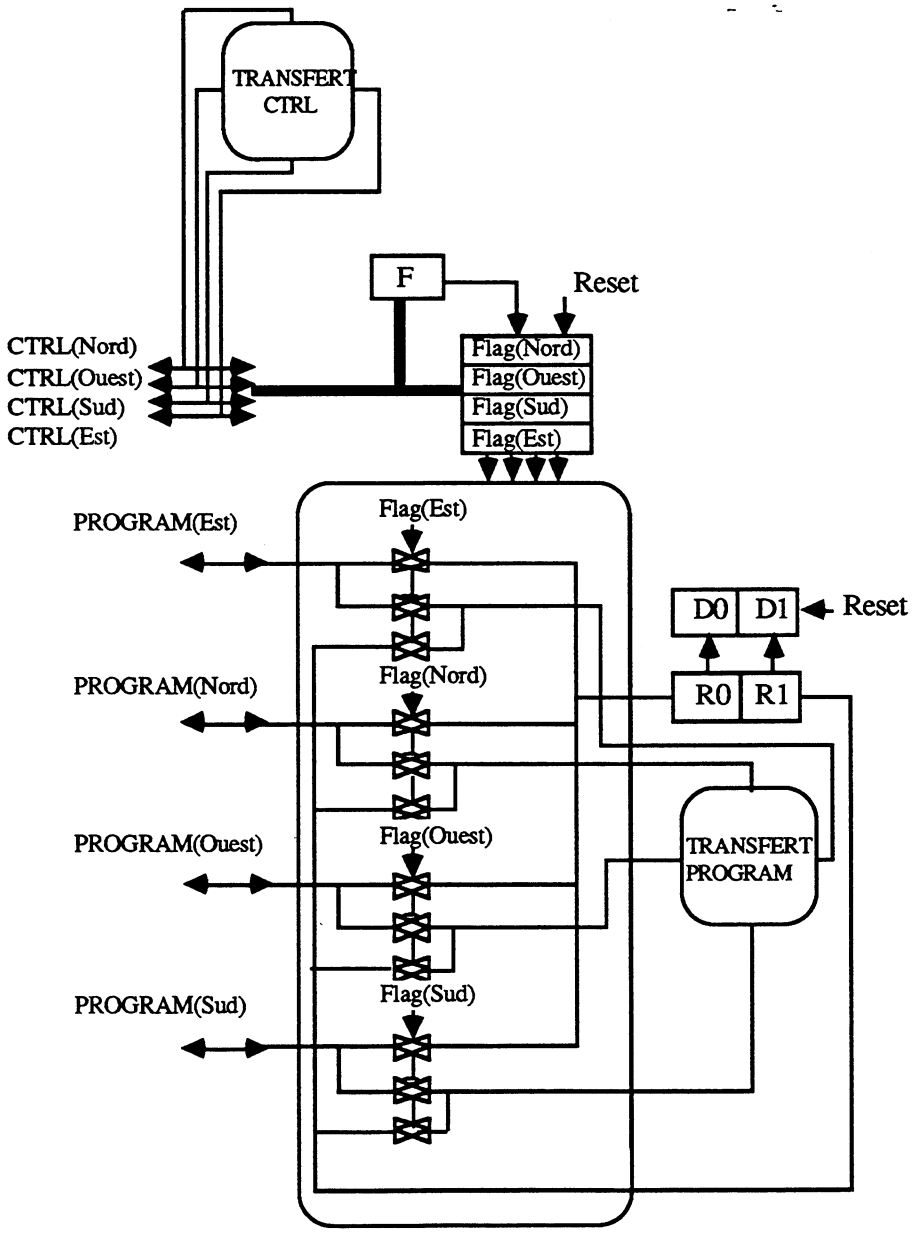


Figure 2.31. La relation entre le bloc SELECT et les autres éléments du commutateur.

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.

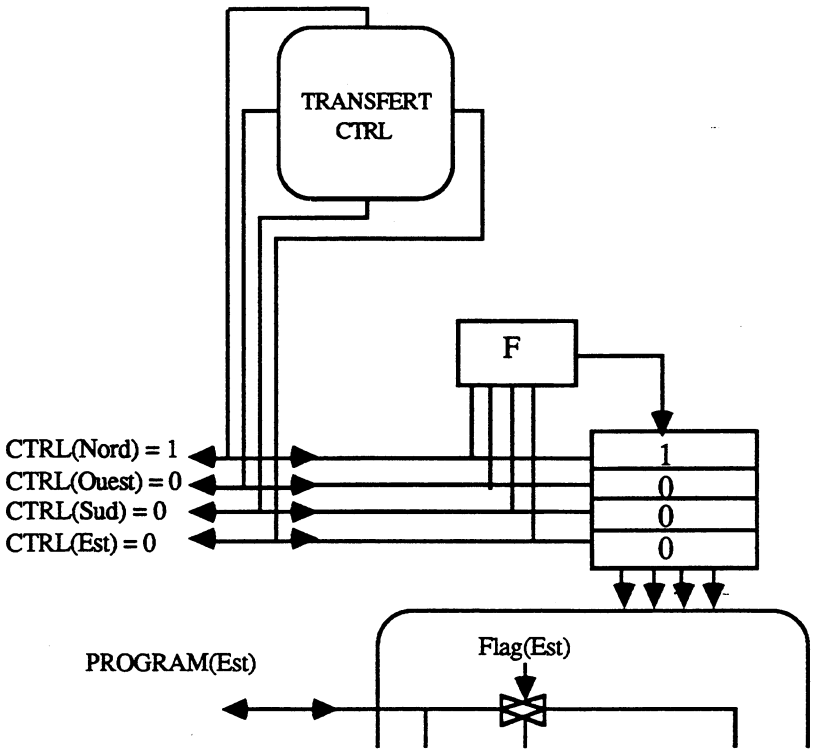


Figure 2.32. Le chargement du “Flag(Nord)”.

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).

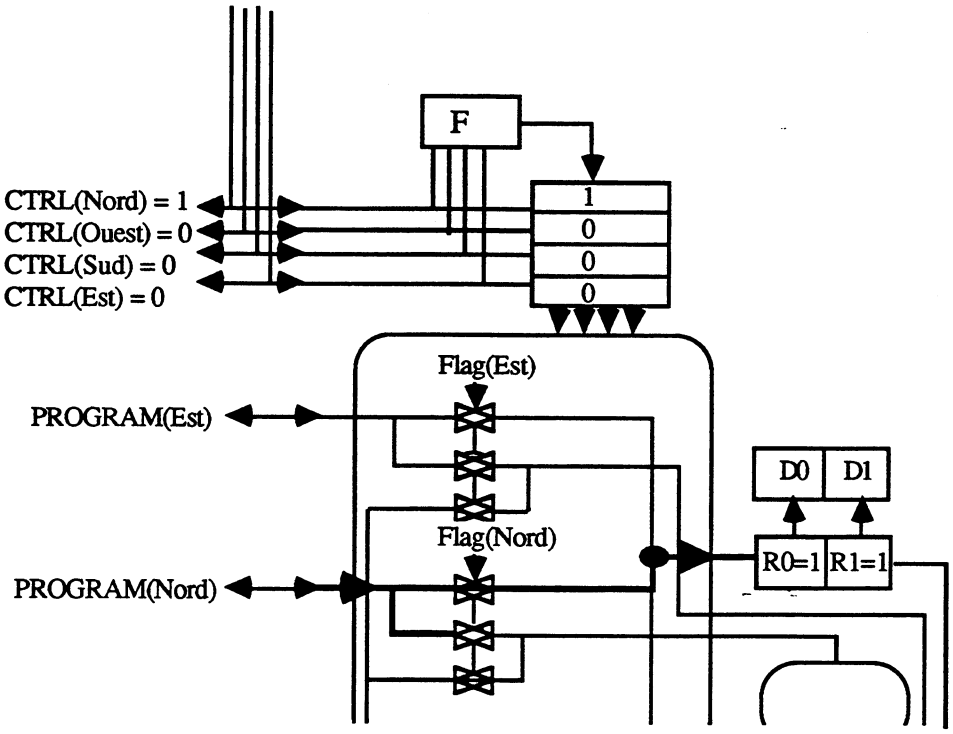


Figure 2.33. La circulation des données pour atteindre le registre (R0, R1).

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.

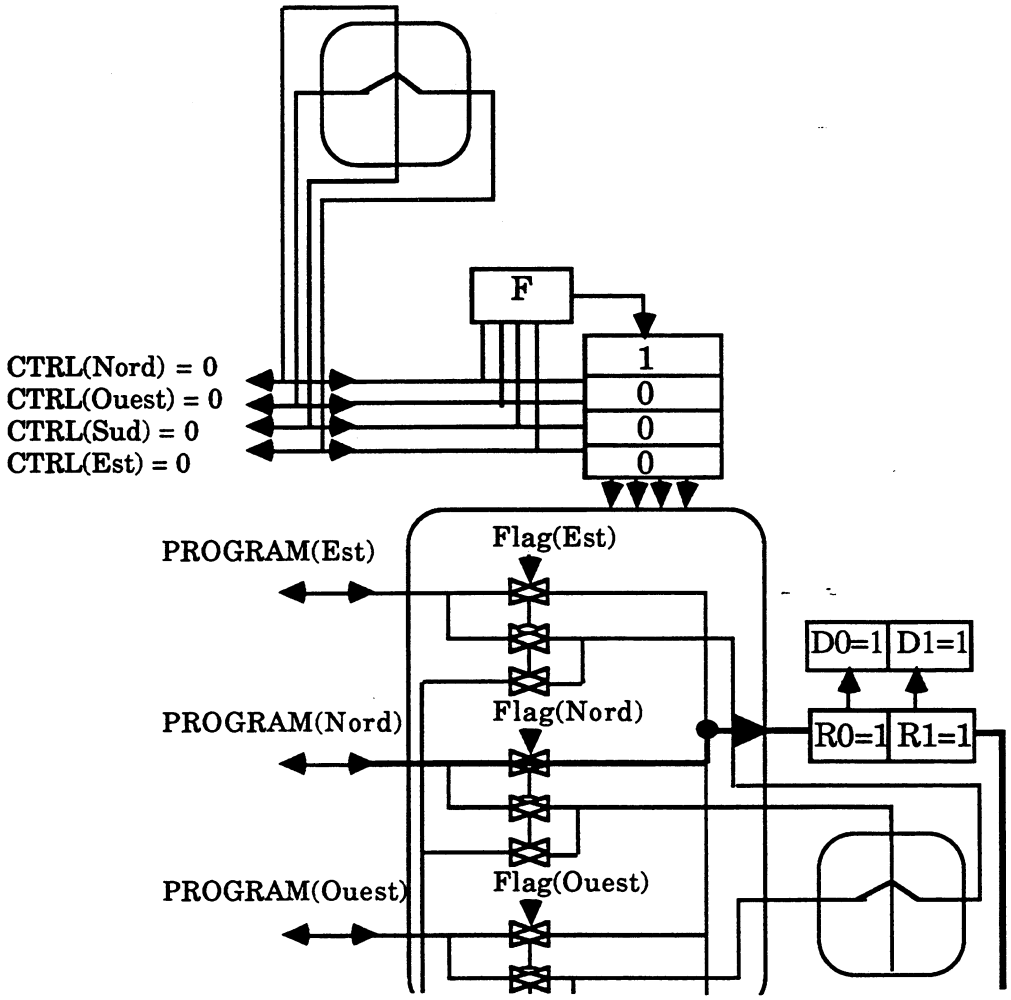


Figure 2.34. La configuration des blocs de TRANSFERT.

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.

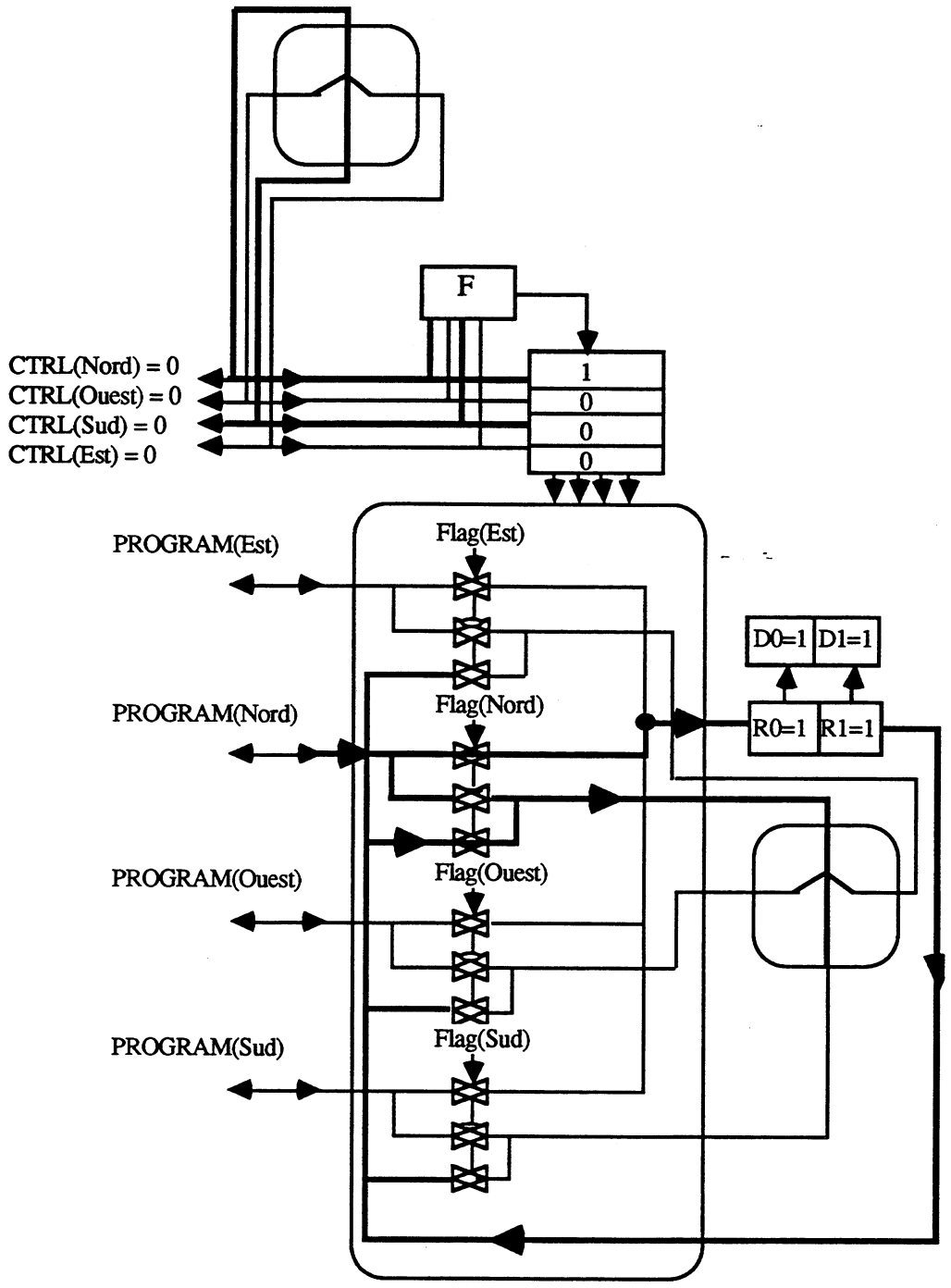


Figure 2.35. Le contrôle de la direction des données PROG(Nord).

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.

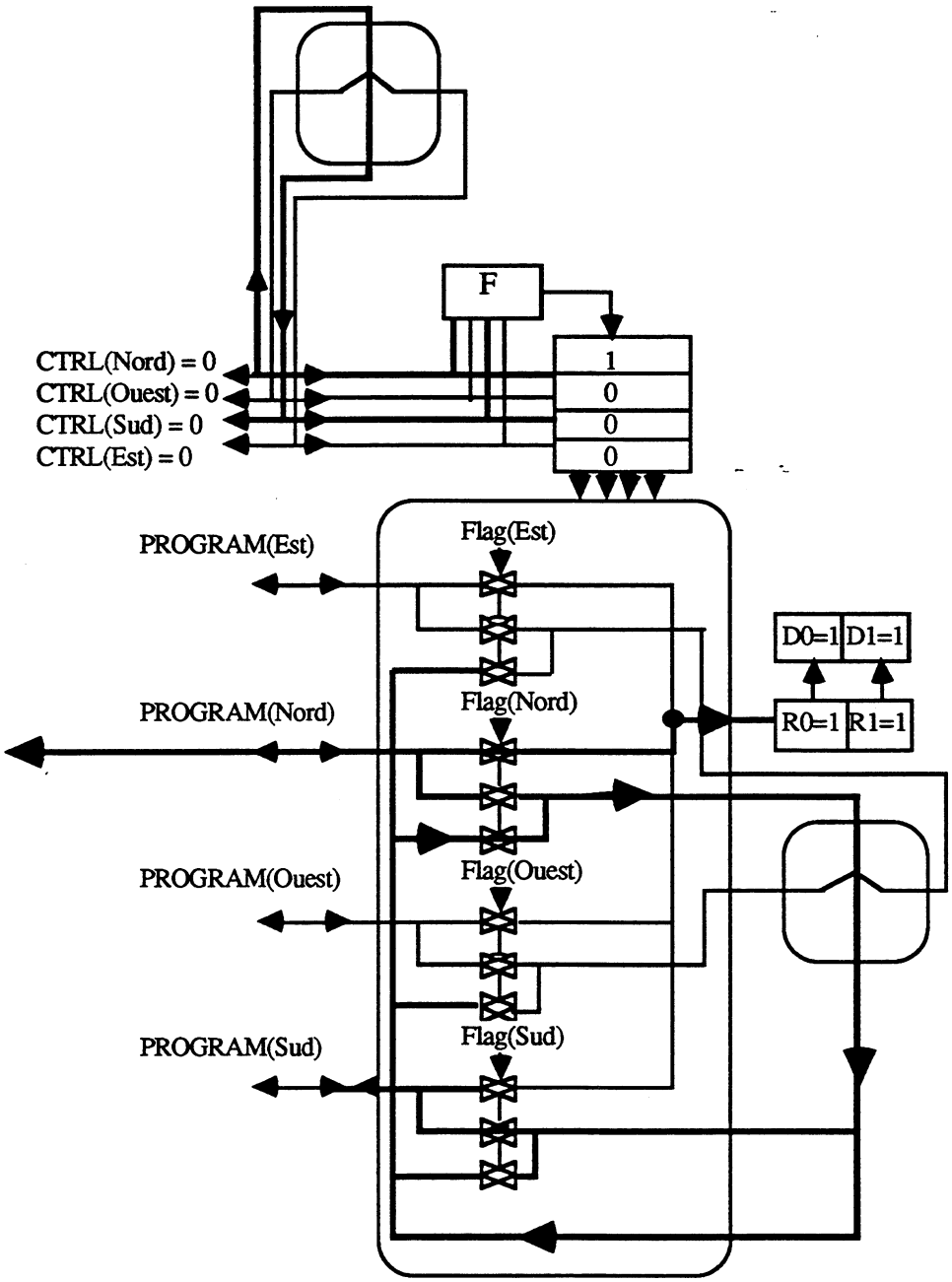


Figure 2.36. La liaison entre la sortie du bloc TRANSFERT PROGRAM et le connecteur de sortie PROGRAM(Sud).

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.

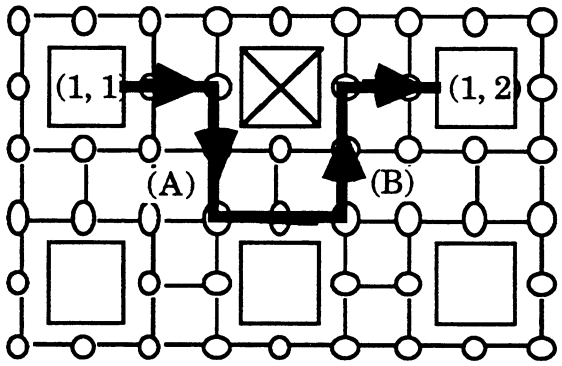


Figure 2.37. Un exemple de connexion logique.

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,

$\overline{\text{Couest}} \rightarrow$ 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.

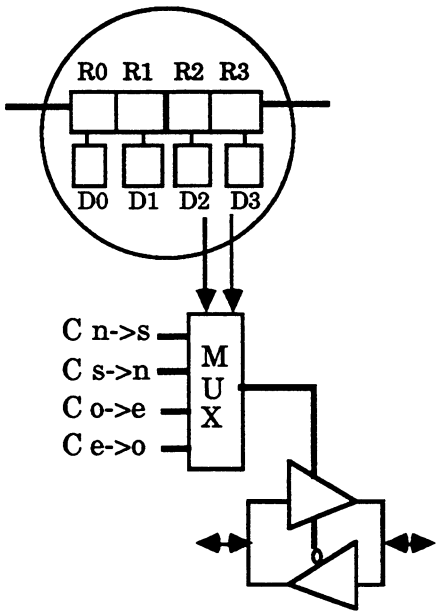


Figure 2.38. Le contrôle de l'amplificateur à partir du commutateur.

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.

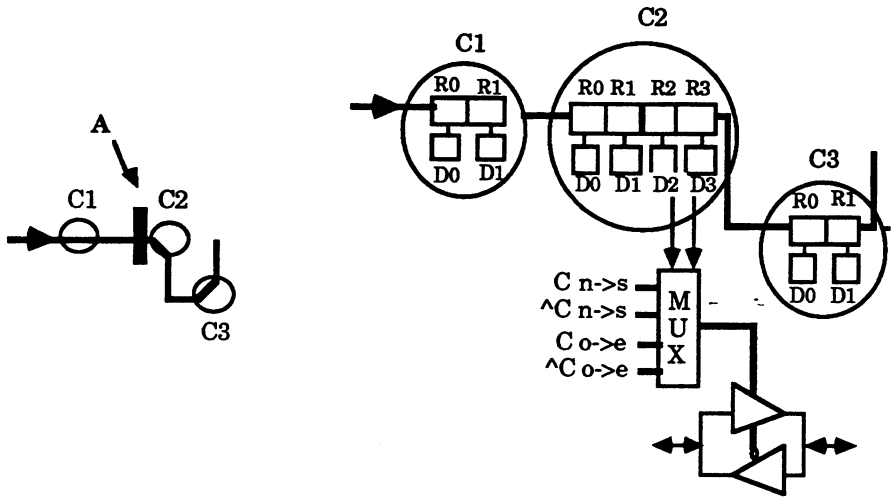


Figure 2.39. (a) Le chemin à programmer.(b) Le modèle associé.

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

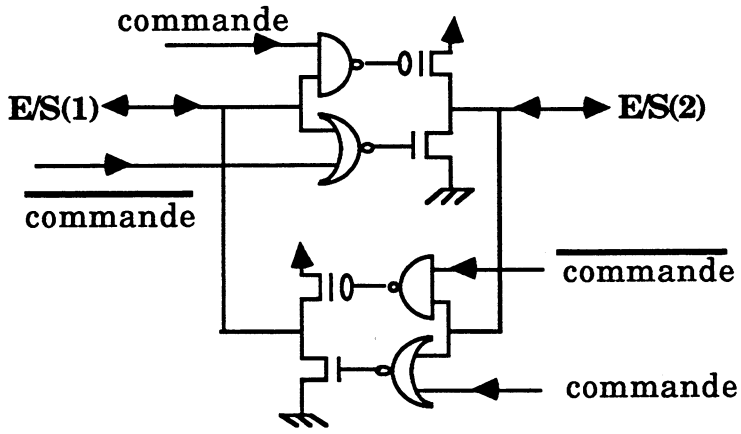


Figure 2.40. Schéma logique de l'amplificateur.

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 10ns et un temps de décharge de 10ns.

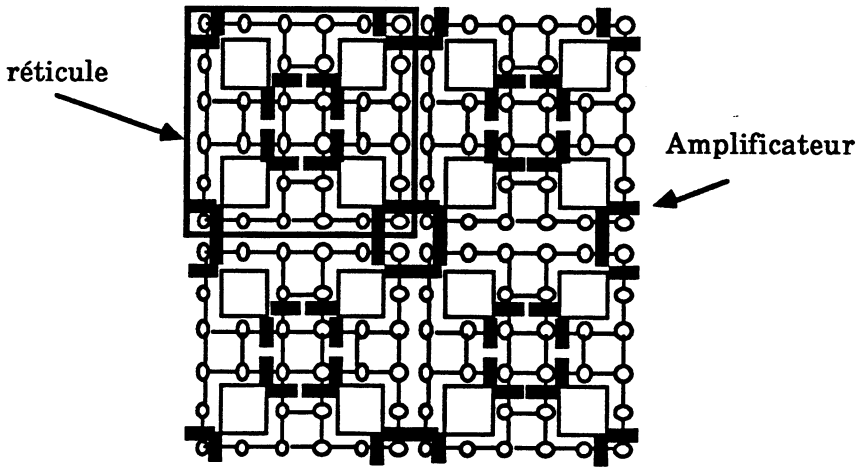


Figure 2.41. La disposition des amplificateurs.

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.

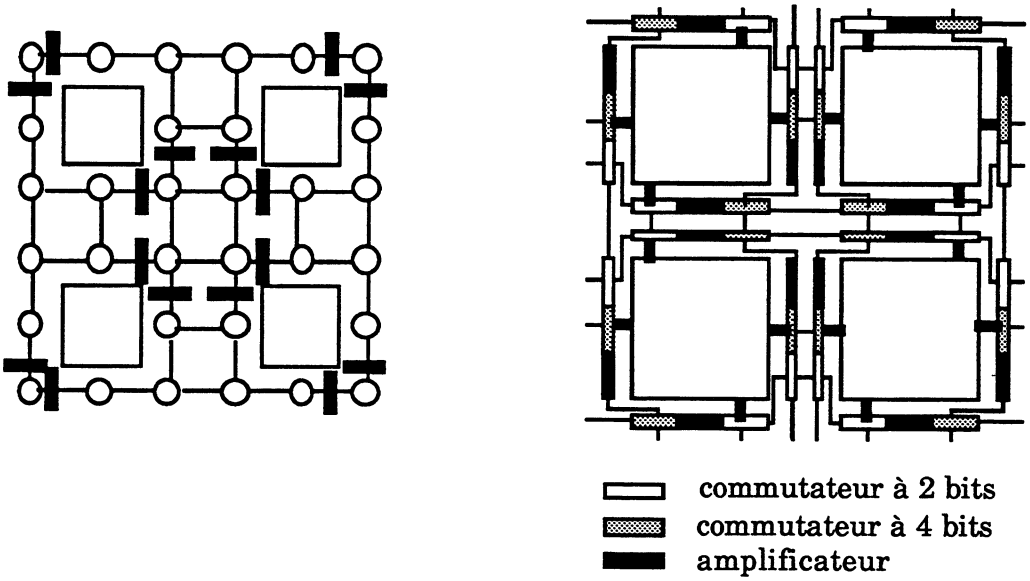


Figure 2.42. Implantation physique du réseau de commutateurs.

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

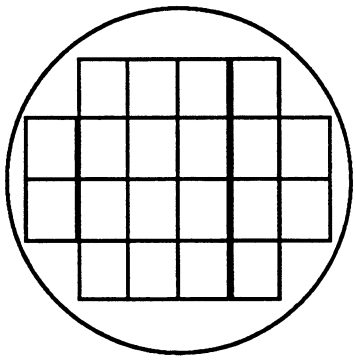


Figure 2.43. La disposition des réticules sur la tranche.

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).

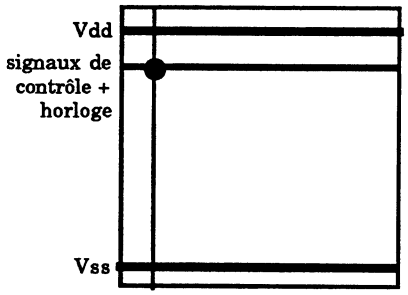


Figure 2.44. La distribution résultante dans le réticule.

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:

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 ...

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.

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.

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.

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 procésus est répété jusqu'à la quarantième étape. La quarantième étape correspond au chemin de taille maximum.

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.

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 :

ALGORITHME DE CONFIGURATION

Nous nous intéressons dans ce chapitre à l'algorithme construisant une cible maximale sur la tranche à partir de sous-matrices et de commutateurs valides. Rappelons que ce problème est vu comme un problème de configuration sans identification, à priori, d'éléments faisant partie d'une cible initiale et d'éléments de réserve. La littérature dans le domaine WSI s'est plutôt orientée vers des techniques de reconfiguration dans lequel les éléments de réserve sont identifiés. Vu l'importance de cette littérature, elle sera rappelée ici.

III.1 Les différentes méthodes de configuration

Rappelons que les méthodes de reconfiguration sont définies comme suit : Soit P une structure physique comportant des éléments défectueux et des éléments de réserves et soit L une structure logique. Pour reconfigurer la structure logique L (dite structure cible) sur la structure physique P (dite structure hôte), il faut d'une part affecter à chaque élément logique de la structure L un élément de la structure P, et d'autre part établir les connections physiques correspondant aux connections logiques. Lors d'une reconfiguration, on cherche à optimiser la longueur des connexions.

La structure logique L peut être une chaîne, un arbre, un tableau, un réseau en treillis, une matrice, un hypercube [ItSu89]... Dans le cadre de cette thèse, nous cherchons à immerger un réseau 2D cible sur un réseau hôte 2D.

Pour résoudre ce problème, il n'est pas envisageable d'utiliser une méthode exacte pour des raisons de complexité. Seules sont envisageables des heuristiques approchant la solution optimale dans des temps raisonnables.Les heuristiques de reconfiguration existantes peuvent être classées suivant deux critères.

Le premier critère se rapporte au caractère global de la méthode. Une méthode de reconfiguration est dite globale lorsque dans ses paramètres elle tient compte de la répartition globale des éléments défectueux. Une méthode qui n'est pas globale est dite locale.

Le second critère se rapporte à la décomposition ou non du placement et du routage des connexions. Certaines méthodes reconfigurent en deux temps. En premier elles placent les éléments logiques sur la matrice physique, puis dans un second temps elles établissent les connexions. D'autres méthodes effectuent en même temps le placement et le routage des éléments. En conséquence, nous avons quatre classes d'heuristiques. Le tableau ci-dessous donne la répartition des heuristiques en fonction de ces critères.

	Locale	Globale
Placement puis routage.	"Fault Stealing" [SaSt86a] [SaSt86b] [LNSS87] "Reconfiguration with faulty interconnections" [DiSa89] "Row & Column elimination" [Lafo89]	"Divide&Conquer" [LeiLei85] "Orthogonal mapping" [JeLo88] "Spanning Tree method" [LoSc87] "Compensation Path" [JeKu89]
Placement et routage en même temps.	"Diogene approache" [ChLe83]	"Design of fault tolerant arrays"[KiRe89] "Configuration and reconfiguration algorithms"[BoPa89]

III.1.1 Méthode locale effectuant un placement puis un routage

Dans cette classe, les méthodes dites "Fault stealing", développées par M.G. Sami et R. Stefanelli [SaSt86a] [SaSt86b], sont les plus connues. Elles proposent un placement des éléments logiques avec un algorithme glouton remplaçant progressivement les éléments défectueux par des éléments de réserve. Puis dans un second temps, les auteurs supposent que les interconnexions sont toujours réalisables à l'aide d'élément non défectueux. Pour cela, ils proposent un réseau dupliqué ou tripliqué.

La redondance utilisée avec ces méthodes consiste à implanter une colonne redondante à l'Est ou(et) une ligne redondante au Sud de la matrice. La matrice est scrutée ligne par ligne et lorsqu'un élément défectueux est rencontré, il est remplacé par l'élément redondant de la ligne courante. Les autres éléments défectueux de la ligne courante sont remplacés par des éléments valides de la ligne du-dessous en les "volant" ("Fault Stealing"). Ces éléments "volés" ne sont plus utilisables et sont alors déclarés virtuellement défectueux. Différentes méthodes de "Fault Stealing" ont été développées pour reconfigurer une matrice (N x N) sur une matrice physique défectueuse ((N+1) x (N+1)).

a) "Fixed-Stealing"

La première méthode est appelé "Fixed-Stealing". L'algorithme est le suivant: Supposons que dans une ligne i, il y ait s éléments défectueux ou "volés" notés (i, k1), ..., (i, ks) avec k1< ... <ks. Si les éléments (i+1, k1), ..., (i+1, ks-1) sont non défectueux alors il y a reconfiguration sinon il y a échec. L'élément (i, ks) est remplacé par l'élément redondant de la ligne (i, N+1) et les éléments (i, k1), ..., (i, ks-1) "volent" les positions des éléments (i+1, k1), ..., (i+1, ks-1). Ces éléments "volés" sont déclarés défectueux. La figure 3.1 montre un exemple de reconfiguration avec cette méthode. En premier les éléments (1, 2) et (1, 3) "volent" les positions des éléments (2, 2) et (2, 3), et l'élément (1, 4) est reconfiguré le long de la ligne. Puis, l'élément (2, 2) "vole" à son tour la position de l'élément (3, 2), et l'élément (2, 3) est reconfiguré le long de la ligne. Enfin l'élément (2, 2) est reconfiguré le long de la ligne.

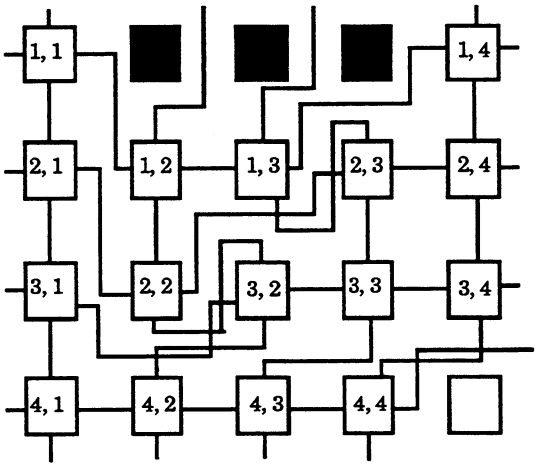


Figure 3.1. Reconfiguration d'une matrice (4 x 4) sur une matrice (4 x 5) avec la méthode dite "Fixed-Stealing".

Cette méthode "Fixed Stealing"" fait un choix arbitraire à chaque étape. Pour palier les nombreux cas d'échec, ce choix est élargi avec une seconde méthode dite "Variable Stealing ".

b) "Variable Stealing"

L'algorithme de cette méthode est le suivant : Chaque ligne est parcourue de la gauche vers la droite. Supposons que dans la ligne i, il y ait s éléments défectueux ou "volés" notés (i, k1), ..., (i, ks) avec k1< ... <ks. L'élément défectueux (i, ks) est reconfiguré par l'élément redondant en bout de ligne. L'élément courant (i, k) (k1≤ k<ks) défectueux ou "volé" de la ligne est remplacé par (i+1, h). Si (i+1, h) est défectueux alors (i, h) "vole" temporairement

l'élément (i, h+1) même s'il est défectueux. A la prochaine itération, cet élément "volé" défectueux doit être remplacé, sinon il y a échec. La figure 3.2 montre un exemple de reconfiguration avec cette méthode. A la première ligne, l'élément (1, 2) "vole" la position de (2, 2), (1, 3) "vole" la position de l'élément défectueux (1, 4) car (2, 3) est défectueux, puis (1, 4) "vole" la position de (2, 4). En bout de ligne, l'élément (1, 4) est remplacé par l'élément redondant. A la deuxième ligne, (2, 2) "vole" la position de (3, 2), (2, 3) "vole" (3, 3) et (2, 4) est remplacé par l'élément redondant. A la troisième ligne, (3, 2) "vole" (4, 2) et (3, 3) est reconfiguré le long de la ligne. Enfin à la quatrième ligne, (4, 2) est reconfiguré le long de la ligne.

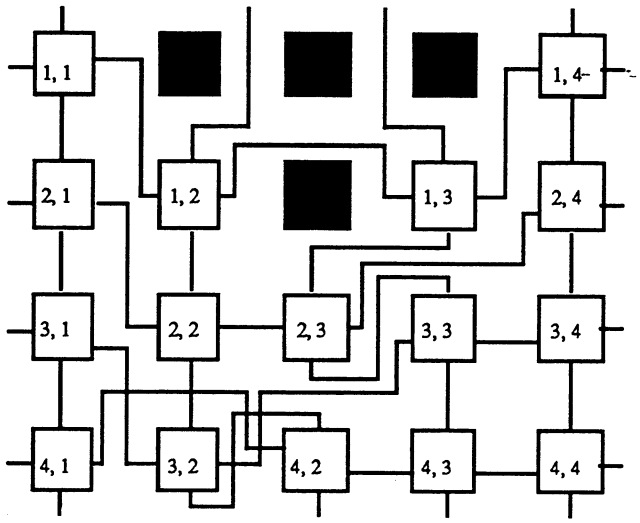


Figure 3.2. Reconfiguration d'une matrice (4 x 4) sur une matrice (5 x 4) avec la méthode dite "Variable Stealing".

c)"Complex Fault Stealing"

Une autre méthode dite "Complex Fault Stealing" élargit de manière plus importante le choix de reconfiguration. L'algorithme est le suivant: Chaque ligne est parcourue de la gauche vers la droite. Supposons que dans la ligne i, il y a s éléments défectueux ou "volés" notés (i, k1), ..., (i, ks) avec k1< ... <ks. L'élément défectueux (i, ks) est reconfiguré par l'élément redondant en bout de ligne. L'élément courant (i, k) (k1≤ k<ks) (défectueux ou "volé" de la ligne est remplacé soit par (i+1, h) ou alors par (i+1, h+1). Si ce n'est pas possible alors (i, h) "vole" temporairement l'élément (i, h+1) même si celui-ci est défectueux. A la prochaine itération, cet élément "volé" défectueux doit être remplacé, sinon il y a échec. La figure 3.3 montre la reconfiguration d'une matrice avec cette méthode. A la première ligne, l'élément (1, 2) "vole" la position de l'élément défectueux (1, 3), cet élément (1, 3) "vole" l'élément valide (2, 4) (les deux

méthodes précédentes échouent à cet itération). Puis, (1, 3) "vole" (1, 4) et (1, 5) est remplacé par l'élément redondant. A la deuxième ligne, (2, 2) "vole" (3, 2), (2, 3) "vole" l'élément défectueux (2, 4) et cet élément (1, 4) "vole" (2, 4). (2, 5) est remplacé par l'élément redondant. A la troisième ligne, (3, 2) "vole" (4, 2), (3, 3) "vole" (4, 3), (3, 4) "vole" (4, 4), (3, 5) "vole" (4, 5) et (4, 5) est remplacé par l'élément redondant. A la quatrième ligne, (4, 2) "vole" (5, 2), (4, 3) "vole" (5, 3), (4, 4) "vole" (5, 4) et (4, 5) est remplacé par l'élément redondant. A la cinquième ligne, (5, 2) "vole" (6, 2) et (5, 3) "vole" (6, 3).

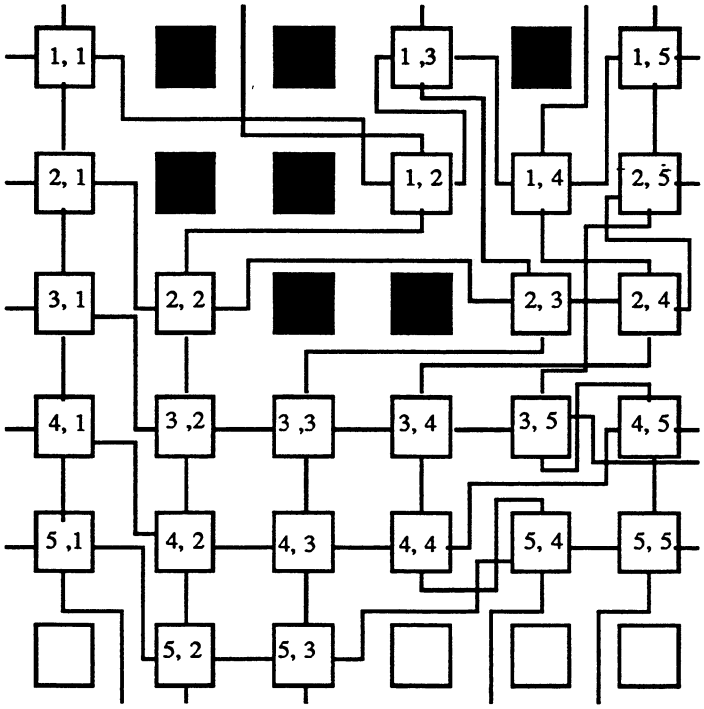


Figure 3.3. Reconfiguration d'une matrice (5 x 5) sur une matrice physique (6 x 6) défectueuse.

d) Méthode dite "MORA" (Modified Optimal Reconfiguration Algorithm) .

Les méthodes de "Fault Stealing" font un choix à chaque étape de l'algorithme pour remplacer les éléments défectueux, ce choix peut occasionner l'échec de la reconfiguration. L'idée de la méthode MORA est de conserver pour un élément défectueux tous les candidats possibles pour les conditions de "Fault Stealing". La reconfiguration est alors modélisée par un graphe biparti, les sommets étant d'une part les éléments défectueux et d'autre part les éléments redondants valides. La méthode se ramène à chercher un algorithme de couplage parfait dans un graphe biparti.

III.1.2 Méthode globale effectuant un placement puis un routage

Dans cette classe, nous citons une méthode à base de "chemin de compensation" [JeKu89]. Avec cette méthode, la matrice est entourée par deux lignes et deux colonnes redondantes, il s'agit en faite de la reconfiguration d'une matrice (N x N) sur une matrice ((N+2) x (N+2)). Lorsqu'un élément est défectueux il n'est pas substitué par un élément redondant, il y a simplement décalage des indices logiques entre l'élément défectueux et l'élément redondant. L'auteur introduit le terme de chemin de "compensation" pour ce décalage.

La méthode proposée est en deux étapes. La première étape établit les chemins de compensation "directs" pour le plus grand nombre d'élément défectueux, ces chemins de compensation sont horizontaux ou verticaux. Il y a alors quatres chemins de compensation possibles (Nord, Sud, Est, Ouest) pour un élément défectueux. Puis dans une seconde étape, un couplage maximun est établi entre les éléments défectueux restants et les éléments redondants.

L'établissement des chemins de compensation "directs" de la première étape se fait successivement pour chaque côté Nord, Est, Sud, Ouest de la matrice. La procédure appliqué est équivalente pour les quatres côtés. Cette procédure scrute la matrice ligne par ligne en commençant par la première. Pour la ligne courante, elle établit les chemins de compensation des éléments défectueux si aucun des éléments redondants sélectionnés est défectueux. Cette procédure s'applique successivement tantqu'il y a évolution. La figure 3.4 (a) représente la matrice défectueuse entourée d'éléments redondants dont certains sont défectueux défectueux. La figure 3.4 (b) montre le résultat de la procédure sur le coté Nord et la figure 3.4 (c) donne le résultat des procédures successives Nord, Est, Sud, Ouest puis de nouveau Nord.

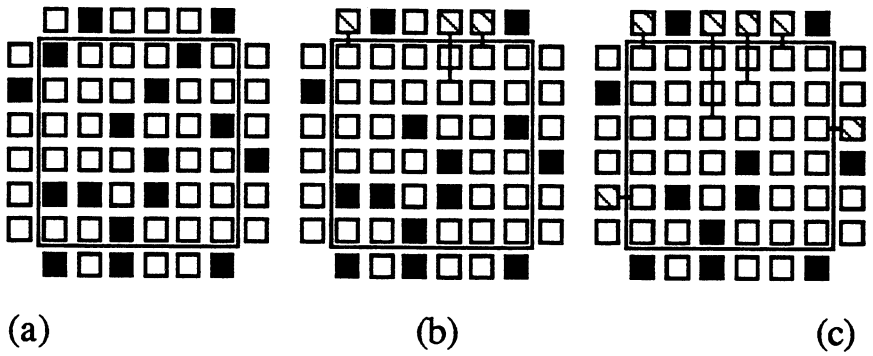


Figure 3.4.(a) La matrice défectueuse.(b) Résultat de la procédure Nord. (c) Résultat de la première étape.

La seconde étape utilise un graphe G(V, E), nommé graphe des incompatibilités, dont l'ensemble V des sommets représente les chemins de "compensation" des éléments défectueux restants et l'ensemble des arêtes E représente les relations d'incompatibilité entre ces chemins. Pour l'auteur deux chemins sont incompatibles s'ils s'intersectent ou se recouvrent. A partir de ce graphe des incompatibilités un algorithme de recherche d'une clique maximale permet d'affecter à chaque élément défectueux un chemin de compensation. La figure 3.5 (a) montre les quatres éléments défectueux restant de la première étape. La figure 3.5 (b) donne le graphe des incompatibilités associés et la figure 3.5 (c) montre ce graphe simplifié. A partir de ce graphe, on obtient pour les éléments 1, 2, 3 et 4 les chemins de compensation respectifs Ouest, Sud, Est et Est. La figure 3.5 (d) montre la reconfiguration totale de la matrice.

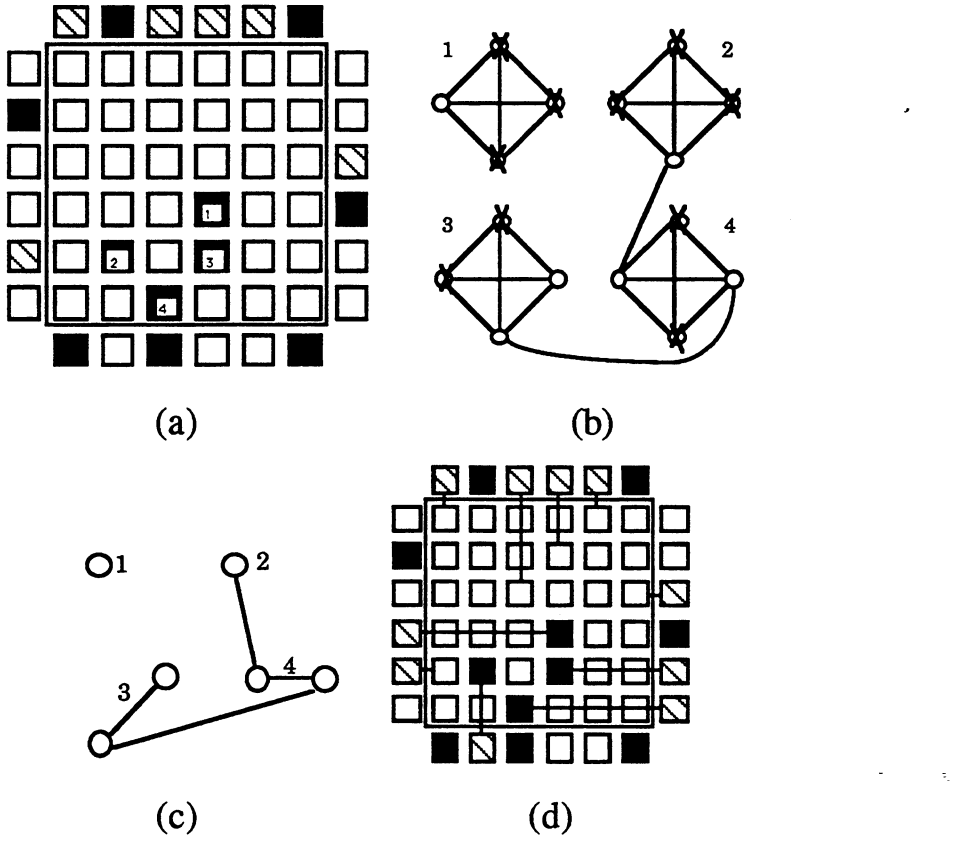


Figure 3.6. (a) La matrice a reconfigurer. (b) Le graphe des incompatibilités. (c) Le graphe des incompatibilités simplifié. (d) Reconfiguration de la matrice.

Dans cette méthode les défauts affectant le réseau de reconfiguration sont pris en compte. Ils interviennent dans l'établissement des chemins de compensations et dans l'établissement des arêtes dans le graphe des incompatibilités en engendrant des restrictions.

III.2 Algorithme de construction d'une matrice cible

Comme dit en introduction, les méthodes de configuration ou de création de cible ont été beaucoup moins explorées. Citons néanmoins les travaux de Leighton et Leiserson [LeiLei85] qui ont énoncé un certain nombre de résultats de faisabilité théorique. Cette approche utilise une méthode du type "Divide & Conquer" en construisant des sous-matrices puis la matrice définitive. Il se sont en particulier intéressé au problème de la réduction de la longueur du chemin maximal qui est primordial car elle détermine la fréquence d'horloge de la cible finale. Rappelons que dans ELSA, la méthode est mixte puisqu'au niveau de la sous-matrice une technique de reconfiguration a été adoptée, et une méthode constructive a seulement été utilisée au niveau de la tranche. Le problème consiste donc a créer une matrice 2D ne comportant que des éléments valides (PE, commutateur et connexion) en minimisant la plus longue distance (en terme de commutateurs traversés) entre 2 sous-matrices voisines. Ce dernier point est primordial car la performance de la cible en dépend.

III.2.1 Premier algorithme implanté

III.2.1.1 Algorithme général

Le premier algorithme implanté appelé CRAWL1 (Configuration and Reconfiguration Algorithms at Wafer Level) est fondée sur une construction progressive de la matrice logique sur la matrice physique défectueuse. La matrice physique (ou matrice hôte) contient le réseau double rail de commutateurs présenté au chapitre II. Les éléments défectueux peuvent être des sous-matrices de PEs, des commutateurs et (ou) des connexions.

A partir de la cartographie des éléments défectueux, la construction de la matrice logique (appelée matrice cible) est réalisée couche par couche. Une couche étant soit une ligne, une colonne ou secteur de la matrice logique. Un secteur est définit par une sous-matrice.

L'algorithme de base, CREER_COUCHE, est une version améliorée de l'algorithme de LEE telle que l'a proposée HIGHTOWER [Higt74]. Cet algorithme cherche le plus proche voisin d'un élément de la couche en propageant une onde de recherche à partir de celui-ci. Le champs d'exploration de l'onde est limité par la longueur maximale admissible des connexions physiques. Cette longueur est mesurée en nombre de commutateurs et elle vaut dans notre cas au maximum 15.

La propagation de l'onde s'arrête dés qu'elle a atteint un élément valide et libre. L'algorithme établit la connexion entre les deux éléments par retour en arrière en minimisant le nombre de bifurcations. Le placement et le routage d'un élément logique sont effectués simultanément en minimisant localement la longueur des connexions.

Les données d'entrée de l'algorithme sont le type de la connexion à établir (Sud vers Nord, Est vers Ouest) et la taille de la matrice cible a construire. Lors de la construction de la couche, chaque nouveau élément de la matrice courante doit avoir deux connexions logiques. Par exemple pour une immersion par colonne, la première connexion établie est celle reliant l'élément courant (i, j) à son nouveau voisin logique de la colonne (i, j+1), la seconde connexion est celle reliant ce dernier à l'élément (i-1, j+1) (figure 1). Une onde de recherche à partir de l'élément (i, j) permet l'établissement de la première connexion, et pour établir la seconde connexion une autre onde de recherche est lancée à partir de l'élément (i-1, j+1) avec comme test d'arrêt la rencontre de l'élément (i, j+1) et la longueur de la connexion. L'établissement de ces connexions se fait par retour en arrière en minimisant le nombre de bifurcations. La figure 3.7 donne l'organigramme de cet algorithme, la variable Onde représente la distance parcourue par l'onde et D représente la distance maximum.

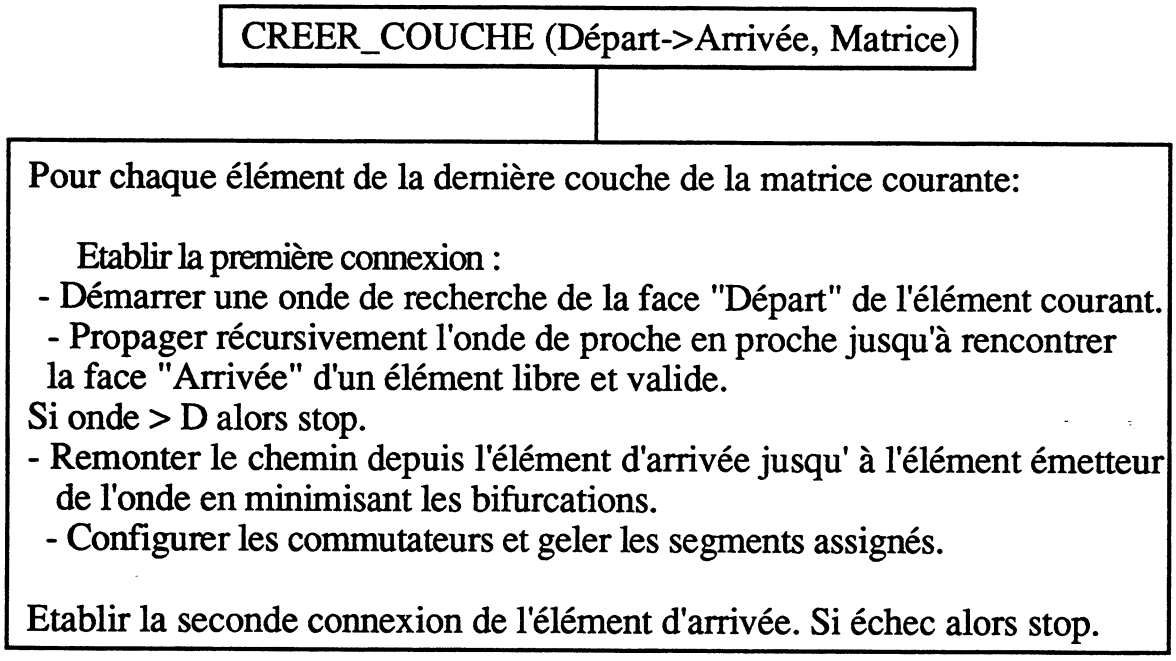


Figure 3.7. Organigramme de l'algorithme CREER_COUCHE.

Après cette description générale de l'algorithme, les différentes immersions possibles seront présentées. Pour une immersion par colonne, une

couche représente une colonne de la cible. L'immersion est conduite de la partie gauche vers la partie droite de la matrice hôte. Au début, l'algorithme sélectionne l'élément physique sur lequel sera immergé le premier élément de la matrice cible, soit (1, 1). Ce point initial est choisi de préférence dans la partie supérieure gauche. On fait alors appel à une procédure de routage pour établir les connexions Nord et Ouest de cet élément, ces connexions aboutissent à des plots périphériques respectivement situés au Nord et à l'Ouest de la matrice hôte.

Pour une matrice cible de L lignes et M colonnes, alors L-1 appels successifs à la procédure CREER_COUCHE(Sud->Nord, Lc x 1) suffiront à former la première colonne (Lc = 1, 2, ..., L-1). Ensuite, M-1 appels successifs à la procédure CREER_COUCHE(Est->Ouest, L x Mc) réaliseront l'immersion complète des colonnes (Mc = 1, 2, ..., M-1). Le tracé des connexions des éléments périphériques aux plots est effectué en conséquence.

L'immersion par ligne est une approche symétrique de la précédente, une couche représente une ligne. L'immersion est conduite de haut en bas de la matrice hôte. L'immersion du premier élément (1,1) est réalisée selon le même principe évoqué ci-dessus. A partir de là, M-1 appels successifs à la procédure CREER_COUCHE(EST->Ouest, 1 x Mc) suffiront à la création de la première ligne (Mc = 1, 2, ..., M-1). Ensuite, L-1 appels successifs à la procédure CREER_COUCHE(Sud->Nord, Lc x M) conduiront à l'immersion complète des L lignes de la cible (Lc=1,2..L-1).

Pour une immersion par secteur, l 'algorithme sélectionne l'élément physique sur lequel sera immergé le premier élément logique de la matrice cible. Cet élément représente une sous matrice de taille 1 x 1. On fait alors appel à la procédure de routage pour établir les connexions périphériques Nord et Ouest de cet élément. Le reste de l'immersion est réalisée de telle sorte qu'à chaque itération, la sous matrice immergée soit toujours carrée. On complète ensuite cette matrice par autant de lignes (ou de colonnes) qu'il est nécessaire pour réaliser la cible lorsque celle-ci est rectangulaire.

Le point (1,1) étant placé sur la structure hôte, on immerge ce secteur de façon à obtenir une sous matrice 2 x 2. Pour cela, on fait appel successivement aux procédures CREER_COUCHE(Est->Ouest, l x 1) et CREER_COUCHE(Sud->Nord, l x 2). De façon générale, pour une matrice cible de L lignes et M colonnes avec (L < M). L-1 appels successifs aux procédures

CREER_COUCHE(Est->Ouest, Lc x Lc) et CREER_COUCHE(Sud->Nord, Lc x Lc+1) réaliseront l'immersion complète de la sous-matrice L x L (Lc = 1, 2, ..., L-1). Ensuite, M-1-L appels successifs à la procédure CREER_COUCHE(Est->Ouest, L x Mc) réaliseront l'immersion complète des colonnes résiduelles(Mc = L,.. M-1).

Le tracé des connexions des éléments périphériques aux plots est effectué en conséquence.

III.2 1.2 Défauts de CRAWLI

Cet algorithme est très simple et il a de graves défauts. Le plus important est la déformation de la matrice logique dues aux éléments défectueux. Si une ligne est déformée par un élément défectueux la déformation se propage sur toute la ligne. La figure 3.8 montre un exemple de cette déformation. Les éléments valides et libres de cette ligne ne sont plus utilisés, on obtient alors une mauvaise moisson (appelé aussi taux de "harvesting" = (nombre d' élément utilisé / nombre d' élément valide)).

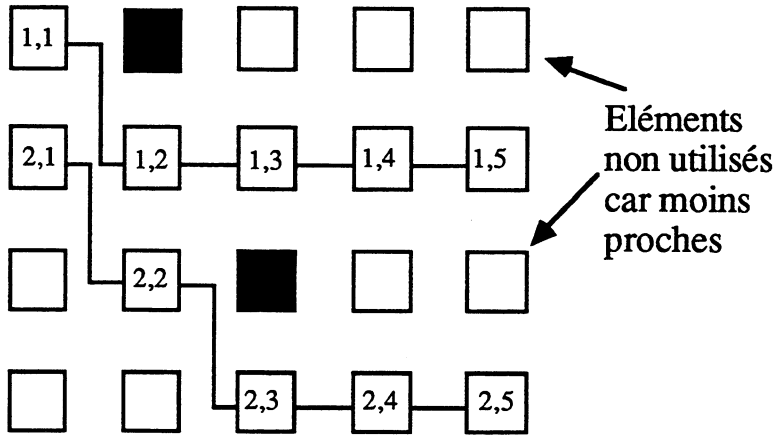
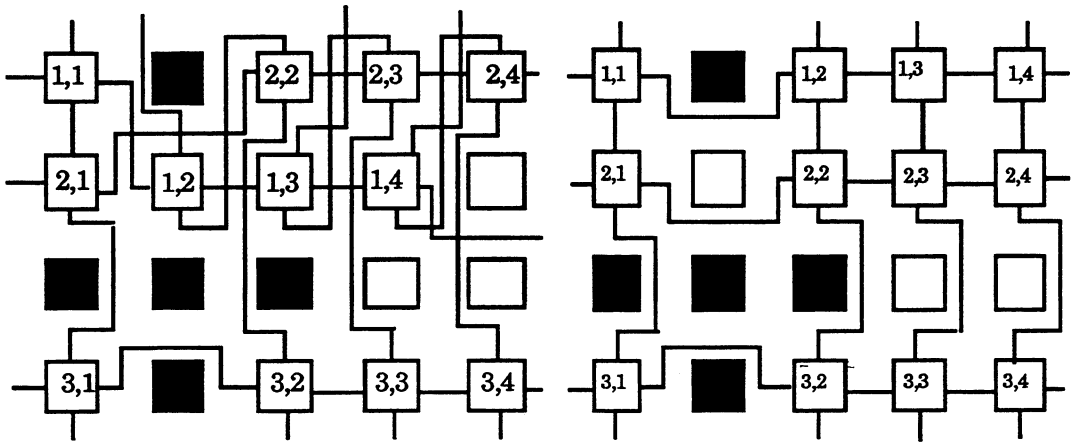


Figure 3.8. Déformation d'une ligne.

La figure 3.9 (a) illustre un autre défaut de CRAWL I. Celle-ci montre l'immersion d'une matrice 4 x 3 sur un hôte défectueux 4 x 4. Le mauvais placement des éléments logiques du au choix du plus proche voisin donne un mauvais routage. En effet, on obtient le croisement des lignes logiques 1 et 2. Dans le cas d'un réseau défectueux, ce routage devient difficilement réalisable voir impossible. Dans cet exemple et dans le cas général d'agglomérat de défauts, les défauts doivent être contournés de manière intelligente pour ne pas mettre en échec le processus de routage. Pour cela l'immersion doit avoir une vue globale

pour tenir compte de la répartition des défauts. L'inconvénient majeur de cet algorithme réside dans sa vision purement locale. A titre d'exemple, la figure 3.9 (b) montre une immersion avec une vue globale, celle-ci donne un routage plus réalisable même avec un réseau défectueux.



(a) (b)

Figure 3.9.(a) Immersion par CRAWL I. (b) Immersion plus réalisable.

III.2.2 CRAWLII

Dans la seconde version de CRAWL, un pré-placement effectué avant l'immersion donne une vue globale de la répartition des défauts. Au-cours de l'immersion CRAWL II essaye de minimiser l'écart entre le placement courant des éléments de la matrice et le pré-placement initial. Ce dernier sert de guide à l'immersion, il n'est pas nécessaire qu'il soit optimal. Nous avons choisi deux types de pré-placement. Le premier ne tient pas compte des défauts c'est un placement idéal, le second tient compte des défauts il est basé sur la méthode de bipartition. Il semble à priori étrange d'utiliser un placement idéal ne tenant pas compte des défauts, alors que nous venons de voir la nécessité d'une vue globale des défauts. Ce placement est en faite utile lorsque la matrice hôte contient peux de défauts répartis de manière uniforme.

III.2.2.1 Pré-placement

Le pré-placement idéal donne une immersion idéale sur une matrice hôte non défectueuse. Supposons que l'on veuille immerger une matrice logique (N x P) sur une matrice physique (N° x P°). Ce pré-placement réparti de manière uniforme la matrice logique. Pour cela la distance entre les lignes et les colonnes de la matrice logique sera respectivement (N°-1)/(N-1) et (M°-1)/(M-1). Dans le cas ou ces distances ne sont pas des entiers, elles sont arrondis à l'entier le plus près. La figure 3.10 montre un pré-placement idéal d'une matrice logique (5 x 4) sur une matrice hôte physique (7 x 7). Dans cet exemple la distance entre les lignes est soit 0 ou 1 et la distance entre les colonnes est toujours 1.

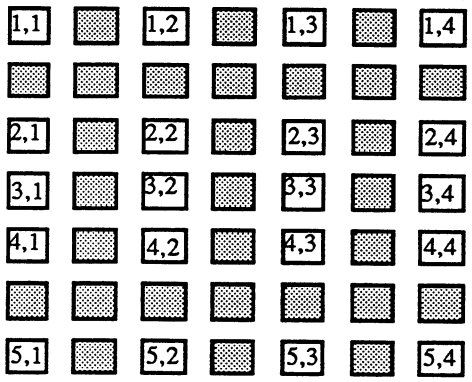


Figure 3.10. Le pré-placement idéal d'une matrice (5 x 4) sur une matrice (7x7).

L'intérêt de ce pré-placement est de rendre la matrice immergé aérée et régulière et il permet aussi de limiter le nombre de croisements entre les connexions.

Le second pré-placement implanté dans CRAWL II tient compte de la distribution des éléments défectueux de la matrice hôte et il suppose le réseau double rails de commutateur non défectueux. Nous avons choisi la méthode de bipartition car elle se rapproche du placement optimal et elle donne toujours une solution. Ce pré-placement s'effectue en trois phases:

La première est une phase de descente récursive. Supposons que nous voulons immerger une matrice logique *(N x M)* sur une matrice hôte contenant *P* éléments valides *($P \geq N \times M$)*. La matrice logique et la matrice hôte sont divisées simultanément en deux parties, suivant l'étape courante de l'algorithme la partition est horizontale ou verticale

Partition horizontale: si N est pair, la matrice logique est divisée en deux sous-matrices (A, B) de taille $\frac{NxM}{2}$ contenant chacune $P/2$ éléments. Si N est impair, la matrice logique est divisée en deux sous-matrices (A, B). La taille de A est $\frac{(N-1)xM}{2}$ et celle de B est $\frac{(N+1)xM}{2}$ Dans ce cas, A contient $\frac{(N-1)xP}{2N}$ éléments et B contient $\frac{(N+1)xP}{2N}$ éléments

Partition verticale: si$M$ est pair, la matrice logique est divisée en deux sous-matrices (A, B) de taille $\frac{NxM}{2}$ contenant chacune $P/2$ éléments. Si M est impair, la matrice logique est divisée en deux sous-matrices (A, B). La taille de A est $\frac{Nx(M-1)}{2}$ et celle de B est $\frac{Nx(M+1)}{2}$ .Dans ce cas, A contient $\frac{(M-1)xP}{2M}$ éléments et B contient $\frac{(M+1)xP}{2M}$ éléments.

La seconde est une phase d'immersion. Cette phase commence lorsque la sous-matrice logique courante contient au plus deux éléments. Cette sous-matrice correspond à une portion de ligne ou de colonne. Elle est immergée sur les éléments valides de la partie de la matrice hôte correspondante. Si cette matrice est obtenue par une partition horizontale (resp. verticale), l'immersion se fait de la gauche vers la droite (resp. du nord vers le sud). Si dans la partie correspondante, il y a plus d'éléments valides que d'éléments de la sous-matrice logique, alors les premiers rencontrés sont choisis.

Les figures 3.11 illustrent ce pré-placement. Elles correspondent à l'immersion d'une matrice logique (6 x 6) sur une matrice physique hôte (7 x 7) contenant 39 éléments valides. La figure 3.11 (a) montre le résultat de la phase de descente sur la matrice physique et la figure 3.11 (b) montre la phase de descente sur la matrice logique correspondante. La figure 3.11 (c) montre le résultat de la phase d'immersion.

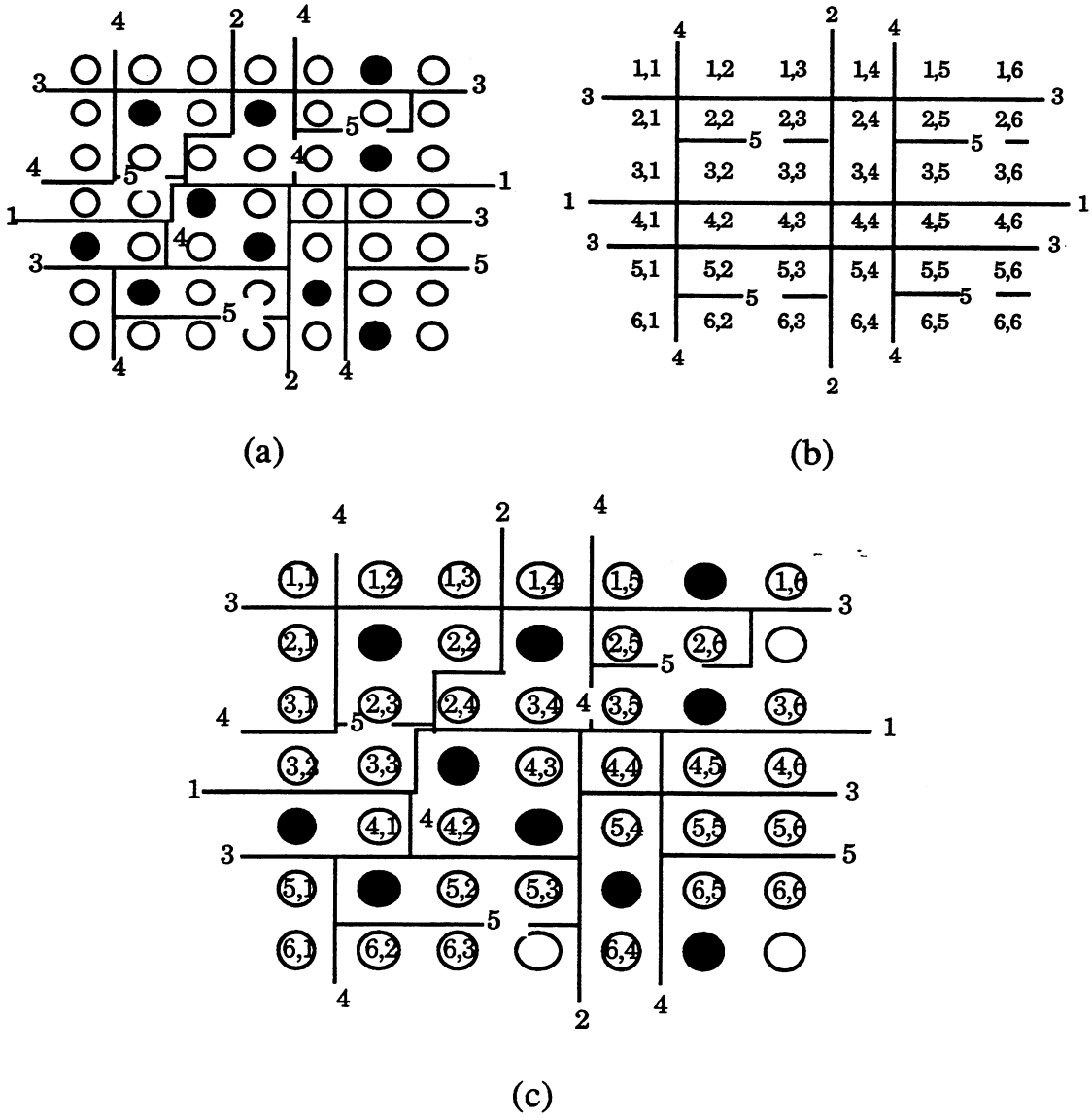


Figure 3.11. (a) Phase de descente sur la matrice physique. (b) Phase de descente sur la matrice logique. (c) Résultat de la phase d'immersion.

La troisième est une phase d'harmonisation. Pendant cette phase, il se peut que des portions de lignes ou de colonnes de la matrice logique se croisent. De tels croisements donnerait un mauvais pré-placement. Pour cela pendant la phase d'immersion, les éléments frontières de deux lignes (ou colonnes) partielles immergées sont examinés, s'il y a un croisement alors on échange les indices logiques des zones immergées.

III.2.2.2 Algorithme d'immersion.

Comme dans la première version de CRAWL, l'immersion de la matrice se fait par couche. Une couche étant soit une ligne, une colonne ou un secteur de la matrice logique.

Pour immerger le voisin logique de l'élément courant de la couche on propage une onde à partir de celui-ci, l'onde se propage tant que l'on a pas atteint la longueur maximale admissible des connexions. A chaque élément rencontré par l'onde est associé un coût par une fonction d'évaluation et c'est l'élément dont le coût est le plus faible qui est choisi comme voisin logique de l'élément courant.

Pour évaluer le coût de chaque élément, nous avons choisit une fonction d'évaluation *(F)* linéaire dépendant de trois facteurs *(f, ß, μ)*. Son expression est: $F = af + b\beta + c\mu$.

Le premier facteur *(f)* représente la longueur de la connexion courante. Ce facteur est une fonction continue et linéaire par morceau de la longueur de la connexion *(L)*. L'expression de $f(L)$ est donnée ci-dessous, elle est obtenue de manière empirique à la suite de nombreux test.

$$f(L) = L \qquad si\ L < 8$$
$$f(L) = ((3xL) / 2) - 4 \qquad si\ 8 \leq L \leq 12$$
$$f(L) = (3xL) - 22 \qquad si\ L > 12$$

Le second facteur *(ß)* représente la distance de Manhattan de l'élément par rapport au pré-placement. Supposons que l'on veuille immerger l'élément d'indices logiques *(i, j)*. Soit *(I, J)* les indices physiques d'un candidat et *(I°, J°)* les indices physiques de l'élément *(i, j)* du pré-placement, l'expression pour *ß* est alors $\beta = |I - I°| + |J - J°|$.

Le troisième facteur *(μ)* représente le routage. Son rôle est de pénaliser la déformation des connexions (le croisement des lignes et des colonnes). Pour cela, ce facteur dépend de l'élément courant à immerger et de ses éléments voisins déjà immergés. Pour ne pas avoir de déformation des connexions nord-sud et ouest-est, l'élément *(i, j)* doit être à la droite de *(i, j-1)* et au sud de *(i-1, j)*. La figure 3.12 (a) montre la déformation de la connexion ouest-est pour une immersion de *(i, j)* à "gauche" de *(i, j-1)* et la figure 3.12 (b) montre la déformation de la connexion nord-sud pour une autre immersion de *(i, j)* au "nord" de *(i-1, j)*.

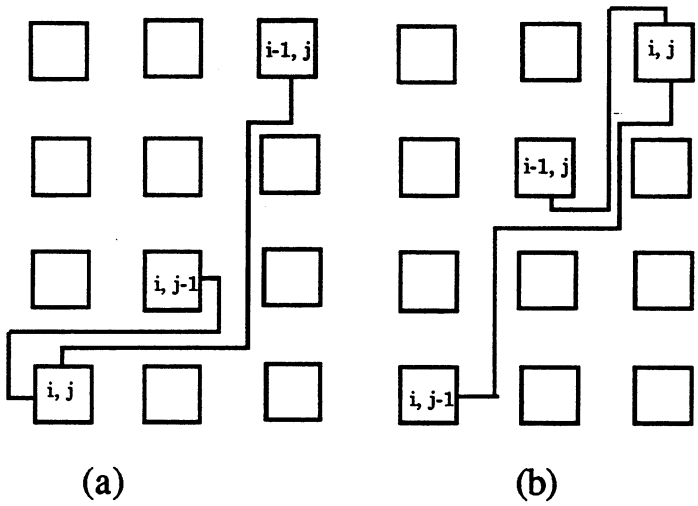


Figure 3.12. (a) Déformation ouest-est. (b) Déformation nord-sud.

Soit *(I, J)* les indices physiques de l'élément logique *(i, j)*, *(I', J')* les indices physiques de l'élément logique *(i, j-1) et (I", J")* les indices physiques de l'élément logique *(i-1, j)*. L'immersion de *(i, j)* devrait satisfaire les conditions suivantes: $I - I' - 1 \geq 0$ et $J - J'' - 1 \geq 0$. En conséquence, l'expression de μ est :

$$\mu = /min\{(I - I' - 1), 0\}/ + /min\{(J - J'' - 1), 0\}/$$

La figure 3.13 donne les valeurs de μ en fonction de la position du candidat pour l'immersion de *(i, j)*. On remarque le coût de l'immersion augmente lorsque l'on s'éloigne du rectangle qui définit les meilleurs candidats.

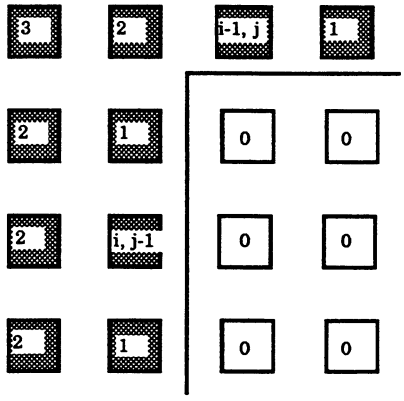


Figure 3.13. Valeur de μ en fonction de la position du candidat de *(i, j)*.

Les trois facteurs de la fonction d'évaluation permettent de minimiser la longueur des connexions *(f)*, d'avoir une vue d'ensemble de la répartition des défauts *(β)* et de tenir compte de la connexion de l'immersion courante *(μ)*. Les

valeurs *a*, *b* et *c* de cette fonction dépendent de la répartition des défauts et elles sont évaluées après des tests qualitatifs.

La figure 3.14 rappel l'organisation des étapes principales de CRAWL II.

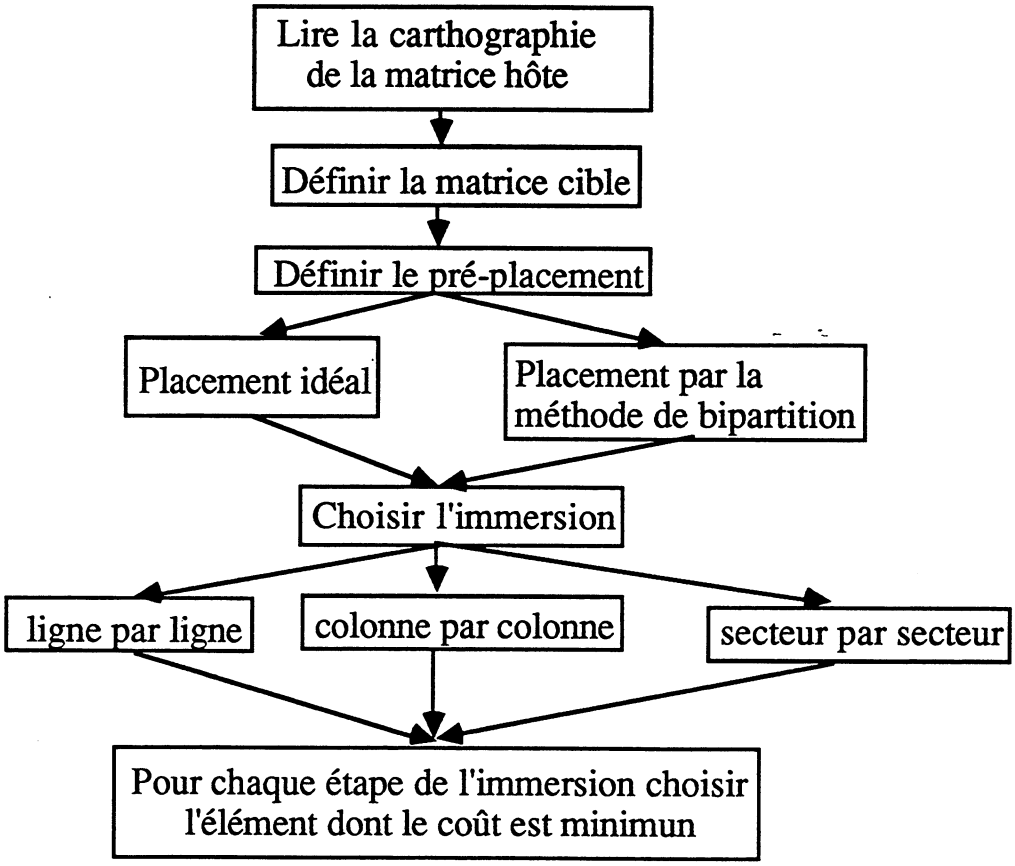


Figure 3.14. Organisation des principales étapes de CRAWL II.

III.2.3. Evaluation et Résultats.

De nombreux tests qualitatifs et quantitatifs ont permis d'évaluer cette algorithme. Le critère d'évaluation est la moisson normalisée, elle correspond au rapport entre la cible réalisé et la plus grande cible réalisable. Les tests qualitatifs ont portés sur l'évaluation de la configuration en présence d'amas de défaut. Ces derniers pouvant être transparents ou opaques vis-à-vis des connexions. La présence d'amas de défaut correspond à des situations délicates pour la configuration, voir critiques dans le cas d'amas opaques.

La figure 3.15 représente la configuration d'une matrice (5x5) sur une matrice hôte comportant un amas défectueux transparent de 4 sous-matrices. (La matrice hôte correspond à la matrice d'ELSA simplifiée pour des raisons de

clarté du dessin). L'algorithme réussit à configurer la matrice avec une moisson normalisée de 59%.

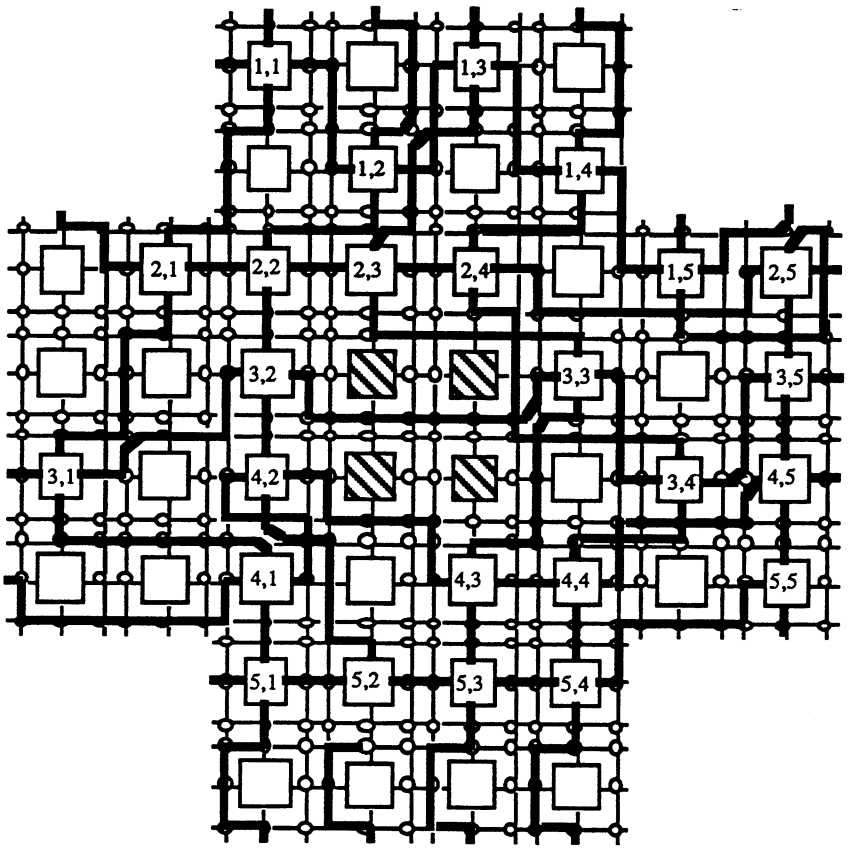


Figure 3.15. Configuration d'une matrice (5x5) avec un amas défectueux transparent de 4 sous-matrices.

La figure 3.16 représente la configuration d'une matrice (5x5) sur une matrice hôte comportant un amas défectueux opaque de 4 sous-matrices. L'algorithme configure une matrice (5x5) correspondant à une moisson normalisée de 59%. On peut dire que la configuration en présence d'amas est satisfaisante, dans les deux cas la longueur maximum des connexions correspond à la traversé de 10 commutateurs. Cette longueur est convenable puisqu'elle est largement inférieur à la limite maximum de 15 commutateurs.

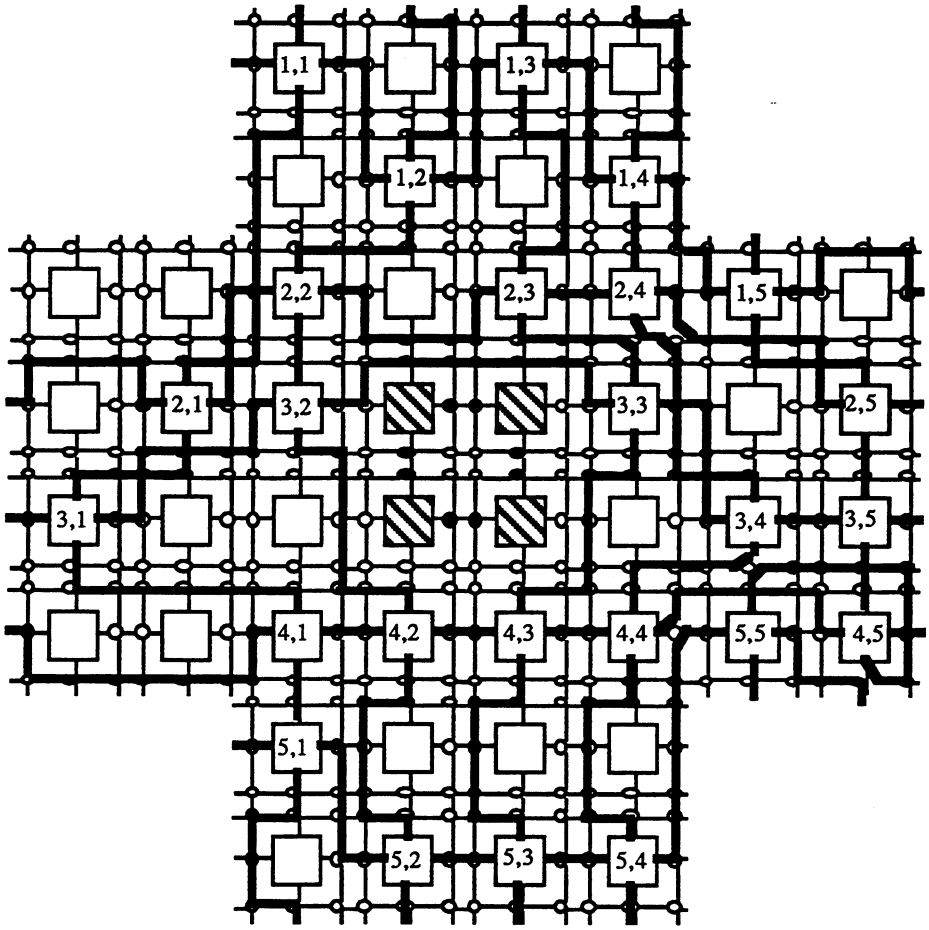


Figure 3.16. Configuration d'une matrice (5x5) avec un amas défectueux opaque de 4 sous-matrices.

Les tests quantitatifs ont porté d'une part sur la comparaison entre les différentes méthodes de configuration et d'autre part sur les pourcentages d'obtention des cibles données.

Comparaison entre les méthodes

La configuration peut être obtenue en construisant la matrice cible colonne par colonne, ligne par ligne ou secteur par secteur. 20 matrices hôtes comportant 10% de sous-matrices défectueuses et 1% de commutateurs défectueux ont été configurées suivant les trois méthodes. Le tableau 3.1 donne les moyennes des moissons normalisées obtenues, on remarque qu'il n'y a pas de différence entre les trois méthodes de configuration.

Méthode	par colonne	par ligne	par secteur
Moisson normalisée	88.7	85	85.6

Tableau 3.1

Probabilité d'obtenir une cible donnée

La réalisation de cible donnée à été testée sur plusieurs taux de défaut, dans tous les cas la matrice hôte défectueuse est une matrice (7x7). Les taux de défaut correspondent à 10% de sous-matrices défectueuses et 1% de commutateurs défectueux (I), 15% de sous-matrices défectueuses et 1.5% de commutateurs défectueux (II), 20% de sous-matrices défectueuses et 2% de commutateurs défectueux (III), 10% de sous-matrices défectueuses et 5% de commutateurs défectueux (IV). Le taux I est le plus prés des estimations de rendement et le taux IV avec 5% de commutateurs défectueux correspond au pire cas en terme de sous-matrice valides, cela provient du fait qu'une sous-matrice est déclarée défectueuse si un de ses commutateurs adjacents est défectueux.

Les résultats sont résumés dans le tableau 3.2, pour une cible donnée les nombres indiquent d'une part le pourcentage de configuration réussit en fonction du taux de défaut et d'autre part la moisson normalisée associée à une cible en fonction du taux de défaut. A partir de ces résultats, on peut évaluer la moisson de l'algorithme à 0.7, et on peut la garantir pour le taux I.

	Taux I		Taux II		Taux III		Taux IV	
7x6	20%	1						
6x6	52%	0,85	10%	0,92				
7x5	44%	0,83	12%	0,89	10%	1		
6x5	100%	0,71	50%	0,76	60%	0,85	10%	0,88
7x4	100%	0,66	48%	0,71	55%	0,8	8%	0,82
5x5	100%	0,59	60%	0,64	92%	0,71	30%	0,73
6x4	100%	0,57	65%	0,57	94%	0,68	75%	0,7
5x4	100%	0,47	100%	0,51	100%	0,57	100%	0,58

Tableau 3.2

III.3. Algorithme de programmation des commutateurs

A l'aide de l'algorithme précédent nous avons défini la matrice logique à construire. La prochaine étape est d'établir cette matrice physiquement sur la tranche en programmant le réseau de commutateurs. A titre d'exemple, la figure 3.17 (a) représente une matrice logique (2 x 2) construite sur un réseau physique (3 x 3). Les commutateurs utilisés pour créer la matrice sont dit "utiles". Un commutateurs utiles (Ci) a un état utile défini au cours de la reconfiguration du réseau physique. La figure 3.17 (b) représente les commutateurs utiles.

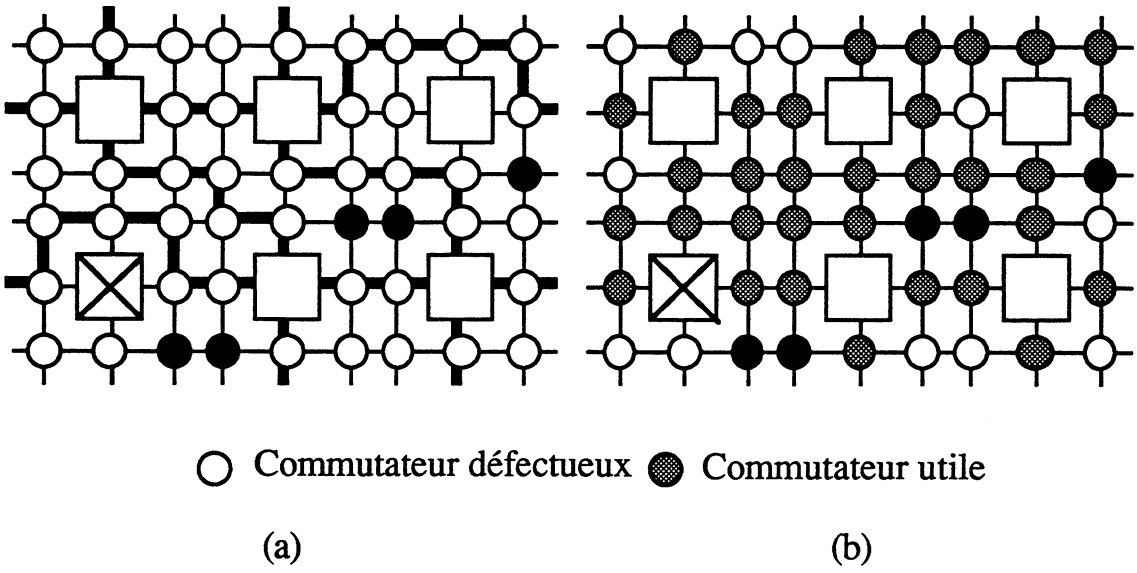


Figure 3.17.(a) Création d'une matrice logique (2 x 2).(b) Les commutateurs utiles.

Pour atteindre tous les commutateurs utiles et les programmer dans un état donné Di, des chemins ou des arbres sont crées sur le réseau afin de les atteindre à partir de la périphérie. Ces chemins ou ces arbres peuvent être programmés en parallèles et ils doivent être disjoints.

La figure 3.18 montre un chemin créé pour atteindre le commutateur C1 et un arbre créé pour atteindre les deux commutateurs C2 et C3.

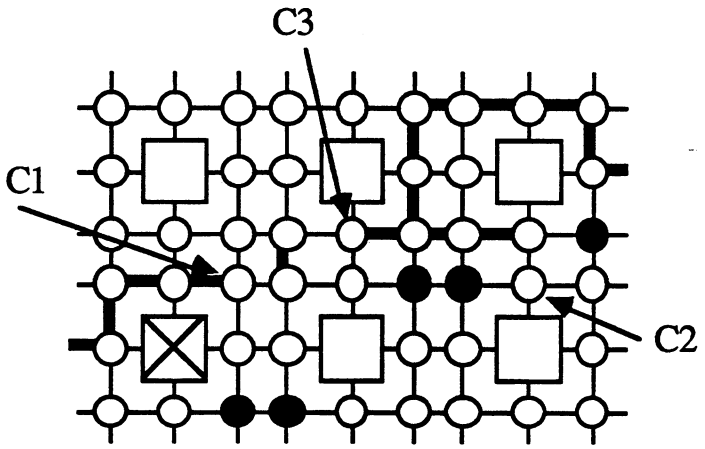


Figure 3.18. Programmation d'un chemin et d'un arbre sur le réseau.

Avant de présenter les différentes méthodes utilisées pour programmer l'ensemble des commutateurs utiles (V_u), nous allons définir un coût de programmation.

III.3.1 Coût de programmation de commutateur

III.3.1.1 Coût de programmation d'un chemin de commutateur

a) Définition.

Nous définissons le coût de programmation d'un chemin comme étant le nombre de bit nécessaire pour le programmer.

De manière général, soit P un chemin de k commutateurs P={C1, C2, ..., Ck). Le coût de P est alors définit par $CP = \sum_{i=1}^{i=k} CPi$, où CPi représente le nombre de bit nécessaire pour programmer le commutateur Ci. $CPi = \sum_{j=1}^{j=i} NBj$ où NBj vaut 2 (resp. 4) si le commutateur Cj est à deux (resp. quatre) bits. Un commutateur à quatre bits contrôle également un amplificateur (cf. § II.5.1).

b)Exemple.

La figure 3.19 représente la reconfiguration d'une matrice logique (2 x 2) sur une matrice hôte (3 x 3). Nous allons programmer le commutateur utile C13 dans la configuration (1, 1).

C13 peut être programmé à partir des commutateurs périphériques C1, C2, ..., C10. Parmi tous les chemins possibles qui peuvent être construits à partir de ces commutateurs périphériques pour atteindre C13, nous allons programmer le chemin

P1. On remarque en construisant le chemin P1 nous en profitons aussi pour programmer deux autres commutateurs utiles C1 et C12.

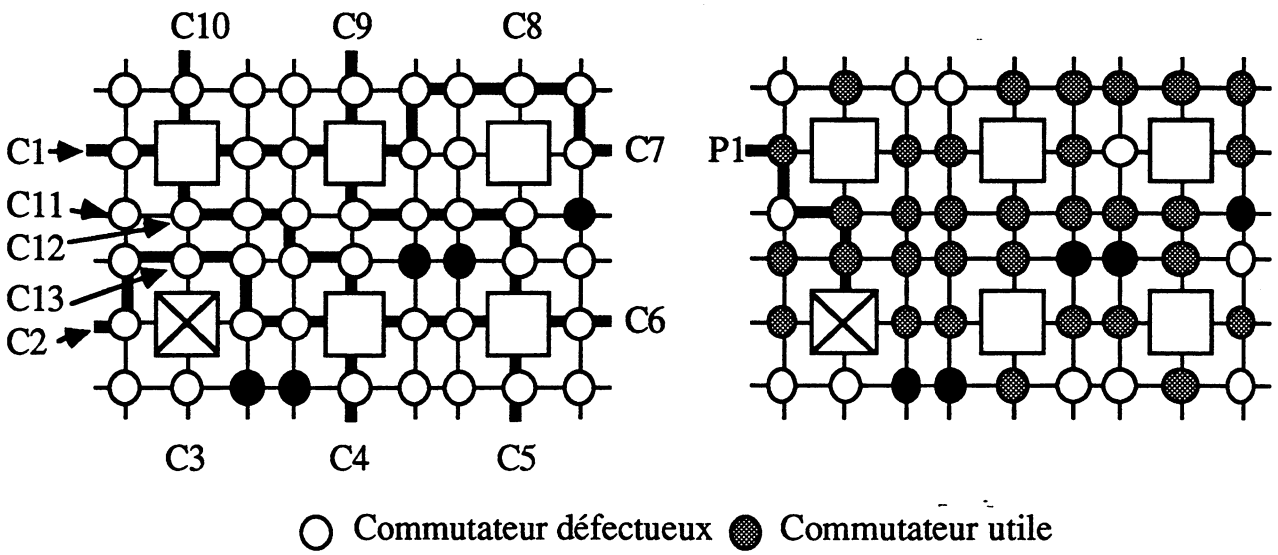


Figure 3.19. Un exemple de chemin pour atteindre le commutateur utile C13.

La figure 3.20 montre la construction progressive du chemin P1. Pour plus de détails, on se référera au § II.4.2. Pendant les trois premières étapes le chemin est construit, puis à la dernière étape les commutateurs utiles C1, C12 et C13 sont programmés respectivement dans les états définifs (1, 1), (0, 1) et (1, 1). Les états des commutateurs utiles pendant la construction du chemin sont différents des états définitifs, ceux-ci sont définis par la reconfiguration.

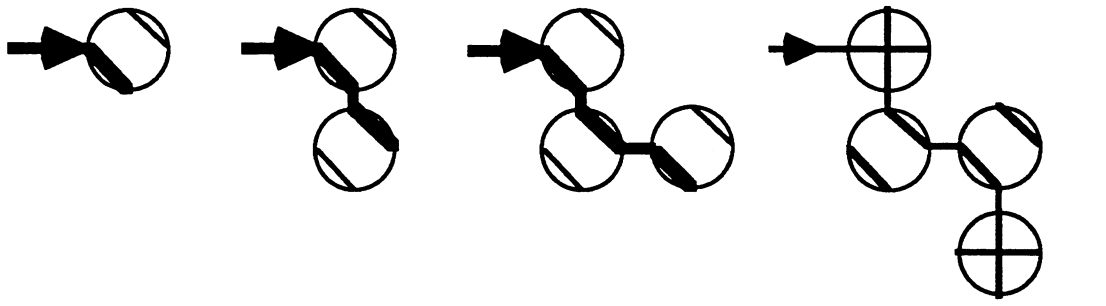


Figure 3.20. La construction du chemin P1.

Rappelons que le commutateur C11 est à deux bits et que les commutateurs Ci (i= {1, 12, 13}) sont à quatre bits (cf II.2.2). Pour ces derniers commutateurs, deux des quatres bits contrôlent la direction des amplificateurs bi-directionnels. La valeur de ces deux bits pour chaque commutateur est donnée dans le tableau 3.2.

Commutateur	Valeur des deux bits	Direction
C'1	xx	Inutilisé
C12	00	Cnord->sud
C13	11	Couest->est

Tableau 3.2

Le tableau 3.3 donne la séquence de données nécessaire pour programmer le chemin P1. Les bits en gras correspondent aux états définitifs de C1, C12 et C13.

Etape 1: C1	xx01
Etape 2: C1+C11	01xx01
Etape 3: C1+C11+C12	000101xx01
Etape 4: C1+C11+C12+C13	**1111000101xx01**

Tableau 3.3.

Le coût de P1 est 4+6+10+14=34.

III.3.1.2 Coût de programmation d'un arbre de commutateur

De la même manière qu'au paragraphe précédent, nous allons définir le coût de programmation d'un arbre et traiter un exemple.

a) Définition.

Nous définissons le coût d'un arbre comme étant le nombre de bits nécessaire à le programmer.

De manière général, soit T un arbre constitué d'un tronc de commutateur {C1, C2, ..., Cp} et d'une branche de commutateur {Cp+1, Cp+2, ..., Ck}. Le coût de P est $CP = \sum_{i=1}^{i=p} CPi + \sum_{i=p+1}^{i=k} CPi$, soit $CP = \sum_{i=1}^{i=k} CPi$. CPi représente le nombre de bit nécessaire pour programmer le commutateur Ci. $CPi = \sum_{j=1}^{j=i} NBj$ où NBj vaut 2 (resp. 4) si le commutateur Cj est à deux (resp. quatre) bits.

b)Exemple

Parmi tous les arbres possibles qui peuvent être construits à partir des commutateurs périphériques pour atteindre C13 et C15, nous allons programmer l'arbre T1 (figure 3.21). On remarque en construisant l'arbre T1 que nous en profitons aussi pour programmer trois autres commutateurs utiles C1, C12, C14. L'arbre T1 est constitué d'un tronc (C1, C11, C12, C13) et d'une branche (C14, C15). Rappelons que les commutateurs C11, C14 et C15 sont à deux bits et que les commutateurs Ci (i= {1, 12, 13}) sont à quatre bits (cf II.2.2).

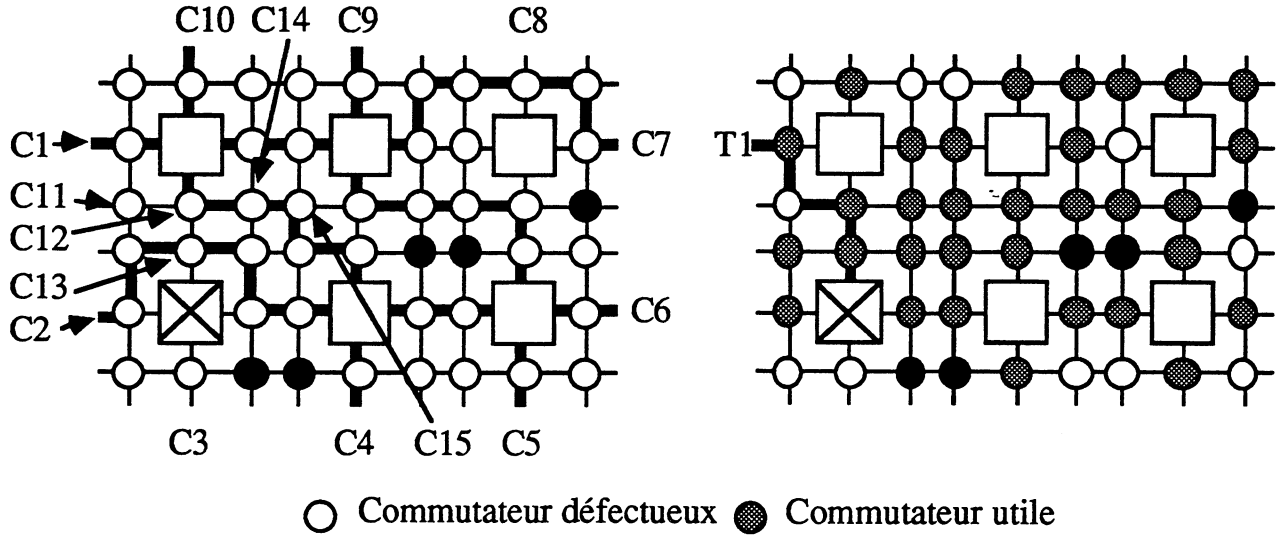


Figure 3.21. L'arbre T1 pour atteindre les commutateurs utiles C13 et C15.

La figure 3.22 montre la construction progressive de l'arbre. Durant les trois premières étapes, nous construisons comme précédemment le chemin du commutateur C1 à C13. Pendant la quatrième étape, C12 est programmé de manière à accéder aux commutateurs C14 et C15. C13 est programmé dans l'état utile D13. Durant les étapes 5 et 6, nous programmons C14 et C15. C'est à la dernière étape que les commutateurs utiles C1, C12, C14 et C15 sont programmés respectivement dans les états (1, 1),(0, 1),(1, 1) et (1, 1) définis par la reconfiguration .

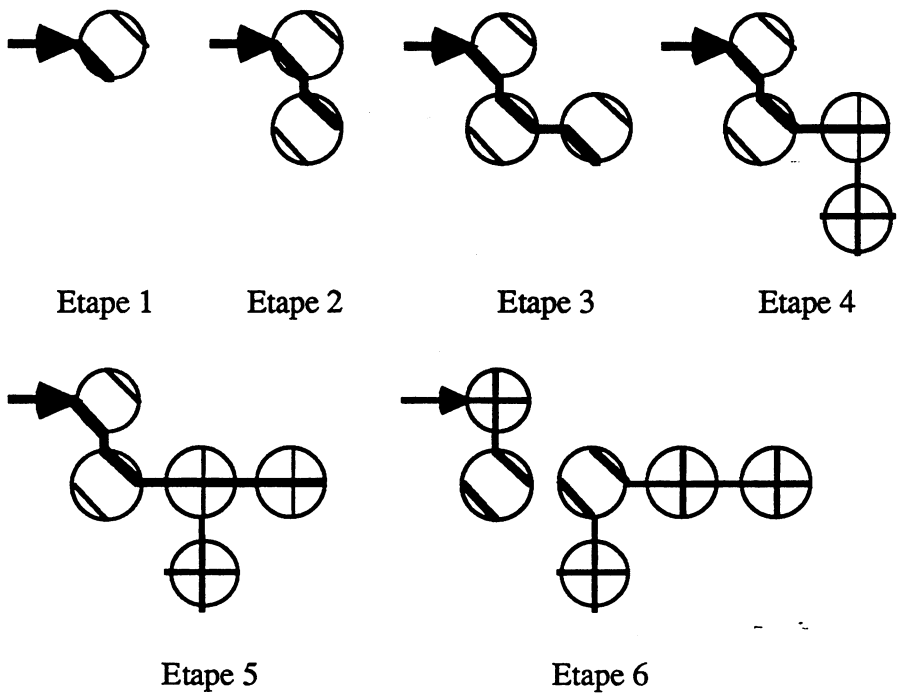


Figure 3.22. La construction de l'arbre T1.

Le tableau 3.4 donne la séquence de données pour programmer le tronc. Les bits en gras correspondent à l'état définitif de C13.

Etape 1: C1	xx01
Etape 2: C1+C11	01xx01
Etape 3: C1+C11+C12	000101xx01
Etape 4: C1+C11+C12+C13	**1111**000101xx01

Tableau. 3.4.

Le tableau 3.5 donne la séquence de données pour programmer la branche. Les bits en gras correspondent aux états définitifs de C1, C12, C14 et C15.

Etape 5: C1+C11+C12+C14	11001101xx01
Etape 6: C1+C11+C12+C14 +C15	**1111000**101**xx11**

Tableau 3.5

Le coût de T1 est égal au coût du tronc plus le coût de la branche, soit CT1= (4+6+10+14)+(16+18)= 68.

III.3.2. Algorithme de programmation

Dans ce paragraphe nous allons présenter deux méthodes programmant l'ensemble des commutateurs utiles (V_u) en minimisant le temps de programmation. Nous allons traiter à titre d'exemple, la configuration ci-dessous (figure 3.23)

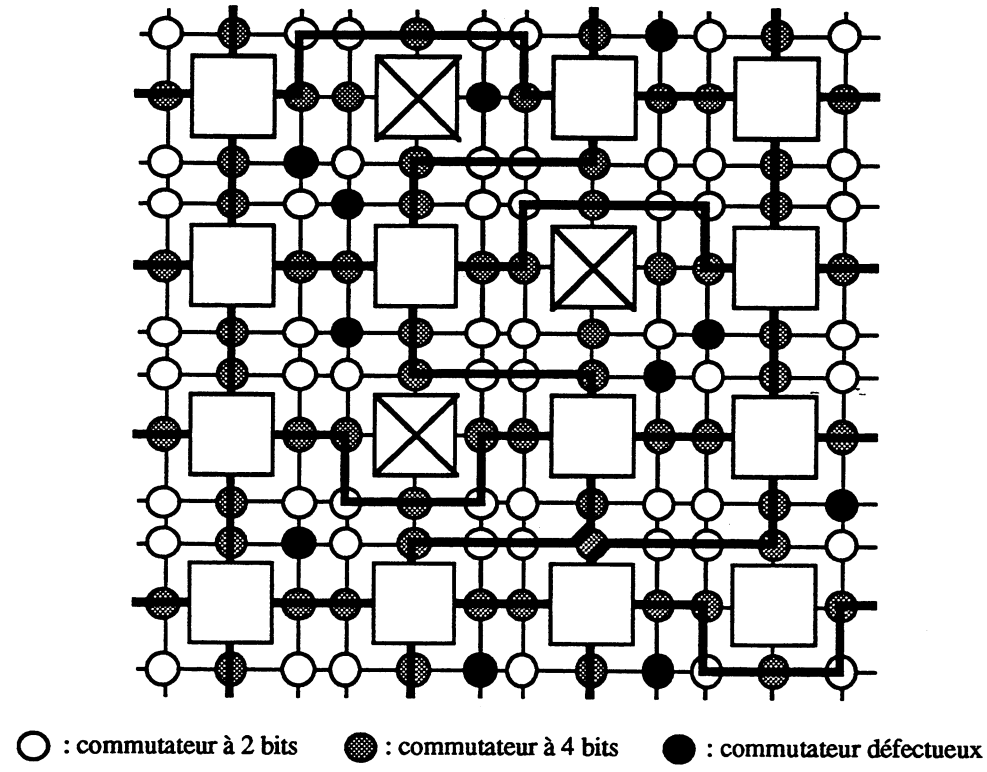


Figure 3.23. Matrice a configurer.

Le temps de programmation d'un arbre (ou d'un chemin) Ti est évalué par le coût (CTi) définit ci-dessus. Il est évident que pour minimiser le temps de programmation, il faut programmer des arbres (et/ou des chemins) en parallèle à partir des commutateurs périphériques non défectueux. En conséquence le coût de programmation de V_u (CV_u) est égale au Max{CTi} des arbres (et/ou des chemins) utilisés pour atteindre tous les commutateurs utiles (V_u).

Soit G = (V, E) le graphe représentant le réseau de reconfiguration. Les sommets de V représentent les commutateurs non défectueux, à chaque sommet est associé un poids de 2 ou de 4 représentant le nombre de bit du commutateur. Les arêtes de E représentent les connexions entre les commutateurs. Ce problème revient à trouver un sous-graphe partiel G'= (V', E') de G tel que :

- V_u est inclus dans V'.
- Les arbres (et/ou les chemins) de G' sont disjoints.
- CV_u est minimisé.

Il n'est pas envisageable d'utiliser une méthode exacte en raison de la complexité du problème, seul une approche heuristique est possible. Nous avons développé pour cela une méthode de couverture par des arbres et une méthode de couverture par des chemins.

III.3.2.1 Couverture par des arbres

Cette méthode s'effectue en trois étapes. En premier, une phase de construction couvre par des arbres tous les commutateurs non défectueux . En second, une phase d'élimination supprime les commutateurs inutiles, puis en dernier une phase d'optimisation diminue l'arbre de coût maximum.

a) Phase de construction.

Cette phase est réalisée par l'algorithme ci-dessous. Celui-ci permet de construire n arbres en parallèles (n étant le nombre de commutateurs périphériques non défectueux) en ajoutant à la feuille de l'arbre courant tous les commutateurs voisins non défectueux.

Faire toujours :

| **Pour chaque** arbre pris dans le sens inverse des aiguilles d'une montre
| | Ajouter aux feuilles de l'arbre courant tous les commutateurs
| | voisins non défectueux et libres.
| **Fin Pour**

Si aucun ajout n'est fait **alors** stop.

Fin Faire

Les figures 3.24 (a),(b) et (c) représentent respectivement la première, seconde et dernière étape de l'algorithme. Sur la figure (c), les chiffres représentent le coût de l'arbre, et on remarque sur cette dernière figure tous les commutateurs non défectueux sont atteints.

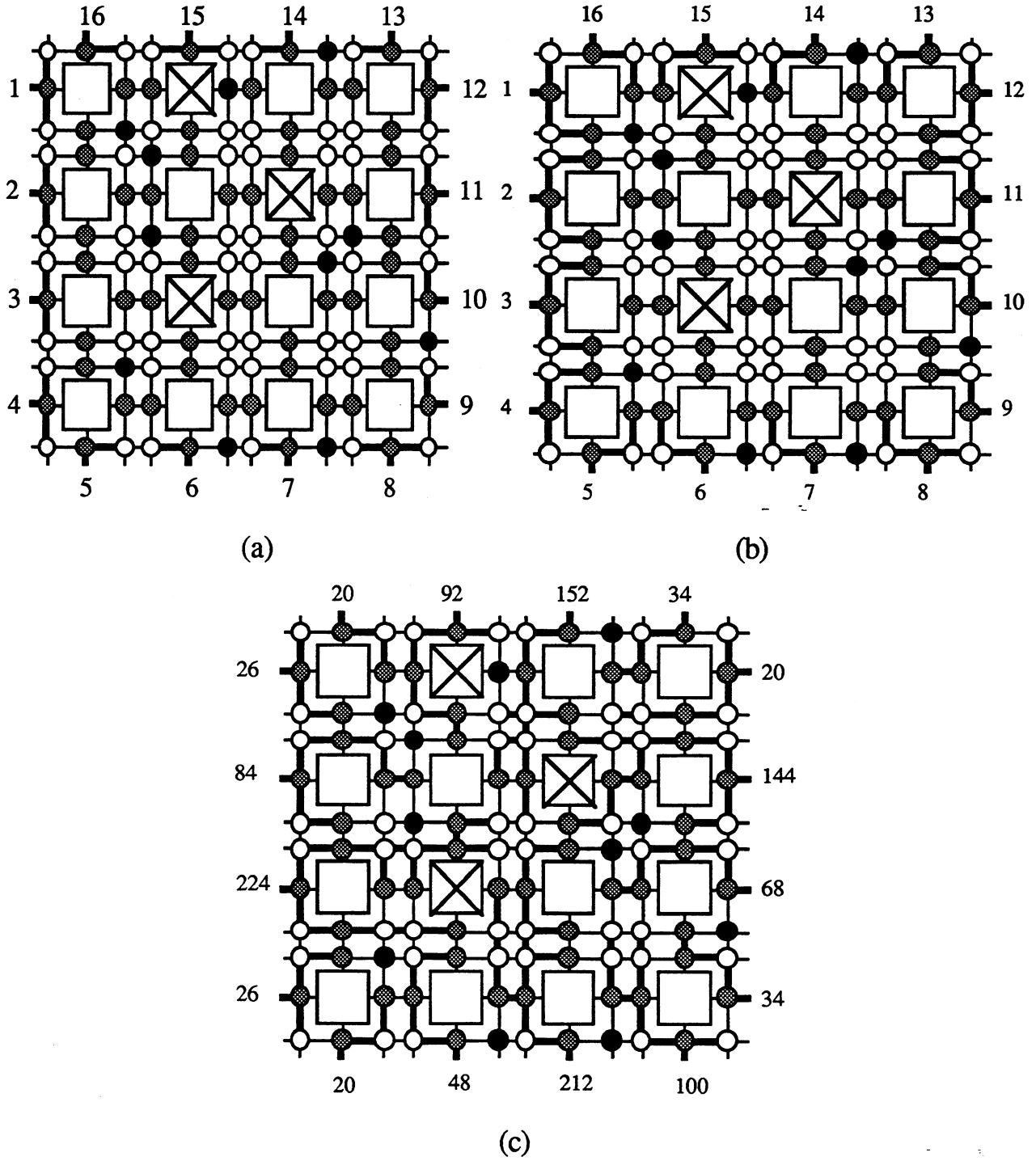


Figure 3.24. (a) Première étape. (b) Seconde étape. (c) Dernière étape.

b) Phase d'élimination.

Les feuilles d'un arbre doivent être des commutateurs utiles, pour cela la phase d'élimination supprime pour chaque arbre de manière récursive les feuilles tant que celles-ci ne sont pas des commutateurs utiles. La figure 3.25 donne le résultat de cette phase.

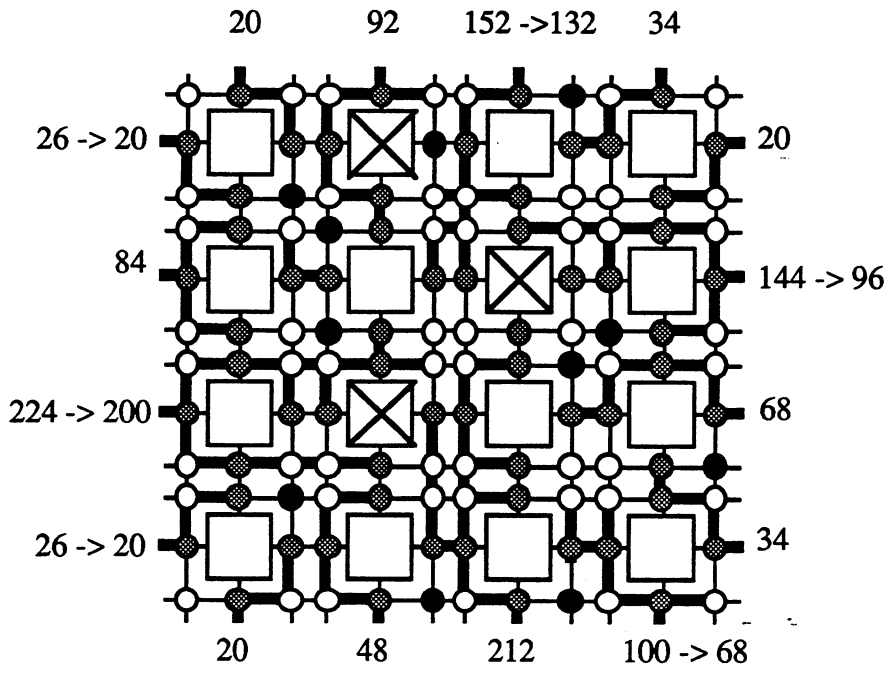


Figure 3.25. Résultat de la phase d'élimination.

c) Phase d'optimisation.

Cette dernière phase permet de diminuer l'arbre de coût maximum. Celui-ci est diminué par échange de commutateurs avec ses arbres voisins. l'algorithme est le suivant :

Faire toujours:

Pour chaque arbre sélectionné par coût décroissant :

Pour chaque feuille de l'arbre courant:

Faire toujours :

Sélectionner les arbres voisins de la feuille courante.

Choisir l'arbre qui aurait le plus petit coût si la feuille courante lui était ajoutée.

Si son nouveau coût est plus petit que le coût de l'arbre courant, alors ajouter la feuille courante à l'arbre sélectionné.

Si aucun ajout n'est fait **alors** stop.

Fin Faire.

Si le coût n'a pas diminué **alors** stop.

Fin Faire.

La figure 3.26 (a) donne le résultat de cette phase d'optimisation. On remarque le coût décroît de 218 à 178, cela représente un gain de 18%.

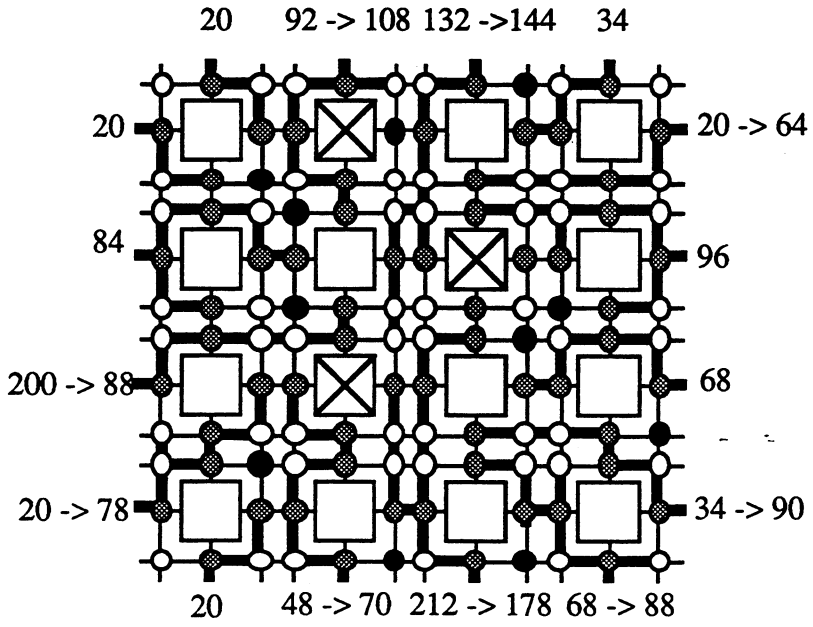


Figure 3.26. Résultat de la phase d'optimisation.

III.3.2 2 Couverture par des chemins

Cette méthode s'effectue aussi en trois étapes. En premier, une phase de construction couvre par des chemins certains commutateurs non défectueux. En second, une phase d'élimination supprime les commutateurs inutiles. Puis en dernier une phase d'addition additionne les commutateurs utiles aux chemins.

a) Phase de construction.

Cette phase est réalisée par l'algorithme ci-dessous. Celui-ci permet de construire n chemins en parallèles convergeant vers le centre.

Faire toujours :

> **Pour chaque** chemin pris dans le sens inverse des aiguilles d'une montre:
>
> > Sélectionner les commutateurs voisins non défectueux et libres de la feuille du chemin courant. Ajouter au chemin courant celui qui est le plus près du centre.
>
> **Fin pour.**

Si aucun ajout **alors** stop.

Fin Faire.

b) Phase d'élimination.

La feuille d'un chemin doit être un commutateur utile, pour cela la phase d'élimination supprime, pour chaque chemin, de manière récursive les feuilles tant que celles-ci ne sont pas des commutateurs utiles.

Les figure 3.27 (a) et (b) montrent respectivement le résultat de la première et seconde phase.

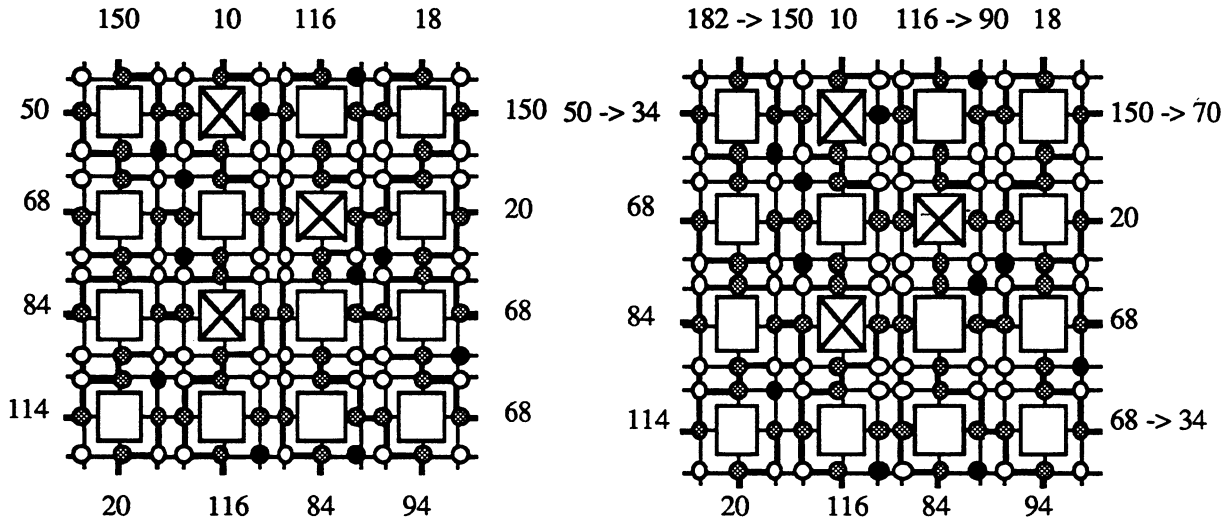


Figure 3.27. (a) Résultat de la première phase. (b) Résultat de la seconde phase.

c) Phase d'addition.

Cette dernière phase additionne les commutateurs utiles non atteints. Ceux-ci sont ajoutés à un de leur chemins voisins, on choisit celui dont le nouveau coût est le minimum. La figure 3.28 montre le résultat de cette phase, le coût maximal est toujours de 150.

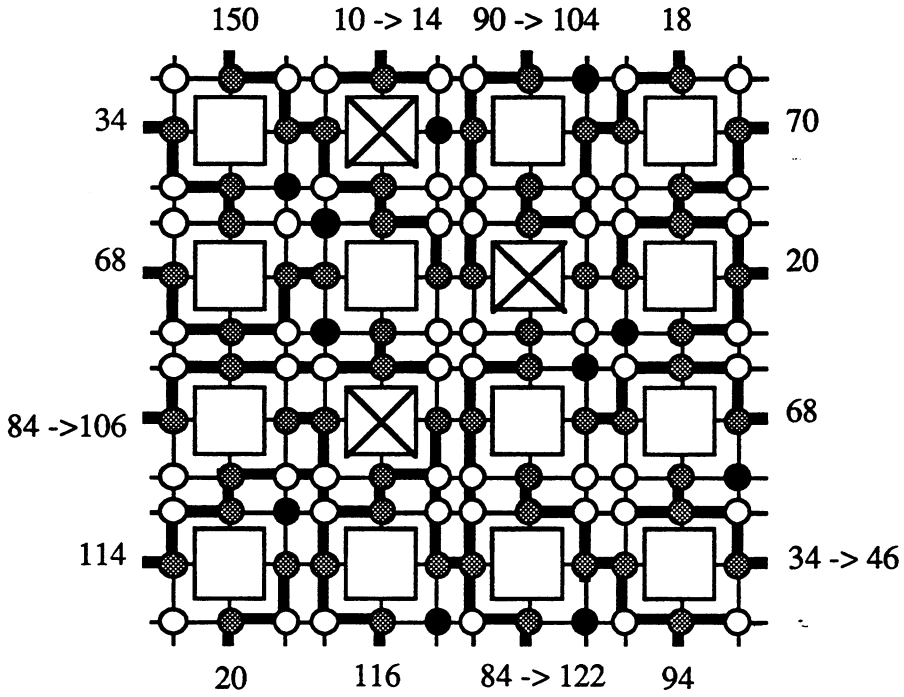


Figure 3.28. Résultat final de la couverture par des chemins.

A la suite de tests, il s'est avéré que la couverture par des chemins est la plus appropriée pour une distribution non uniforme de défaut.

CONCLUSION

Cette thèse propose des méthodes de conception d'architectures massivement parallèles sur tranche entière. La première partie porte sur les techniques de tolérance aux défauts. En effet, la réalisation sur tranche entière ne peut être menée à bien sans l'étude de ces techniques.

Une technique de tolérance aux défauts à deux niveaux a été proposée. Au niveau de la sous-matrice, une technique assez simple d'introduction d'une colonne redondante permet de palier les processeurs défectueux. Au niveau de la tranche, un réseau de commutateurs permet de contourner les sous-matrices non opérationnelles et de connecter celle en état de fonctionnement. Les réseaux de commutateurs proposés dans la littérature sont abstraits et peu réalistes. Le réseau de commutateurs présenté est original car il est lui-même tolérant aux défauts, et il est complètement indépendant de son environnement. En effet, sa programmation est externe et elle ne provient pas des processeurs du réseau. Sa conception topologique donne un surcoût en surface très faible, et sa conception électrique a conduit à une implantation très performante.

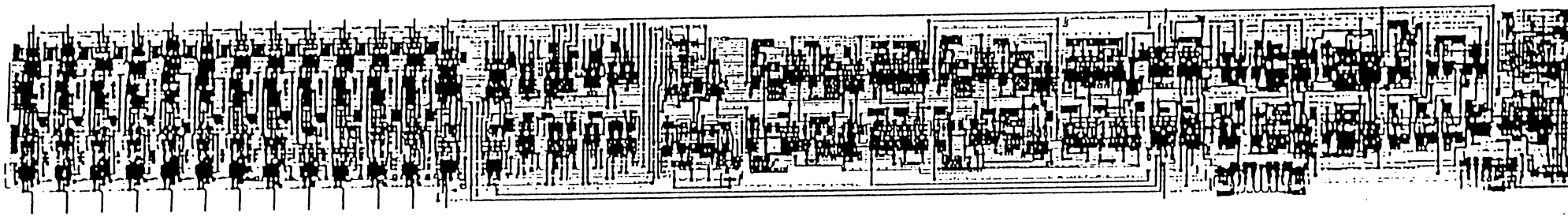
L'étude des méthodes de configuration d'une cible 2D ont conduit à la deuxième partie de cette thèse. Un très grand nombre de méthodes sont présentés dans la littérature, mais aucune ne semble complète et réalisable en ce qui concerne le placement et le routage des éléments. Une méthode ayant une vue globale de la répartition des défauts a été proposée. Elle permet de définir une matrice de taille maximale à partir d'une matrice contenant des processeurs et des connexions défaillantes. Cette méthode a donné lieu à un logiciel opérationnel. Pour la programmaton effective sur la tranche, une méthode originale construit le réseau cible de façon optimisée en temps, en programmant les commutateurs dans les positions requises à partir des plots d'entrée/sortie.

Ces méthodes et outils montrent la possibilité d'avoir à l'échelle de la tranche, une tolérance aux défauts efficace (14,7% de la surface pour le réseau de commutateur). En ce qui concerne la faisabilité industrielle de circuits intégrés tranche entière, elle est loin d'être satisfaisante. En effet, le matériel nécessaire à la programmation et au test est conséquent (laser, microscope électronique). Au point de vue commercial, les coûts de conception et de réalisation rendent une réalité industrielle imminente peu probable.

ANNEXE I

commutateur à quatre bits

Surface = 1453 x 168 μ^2

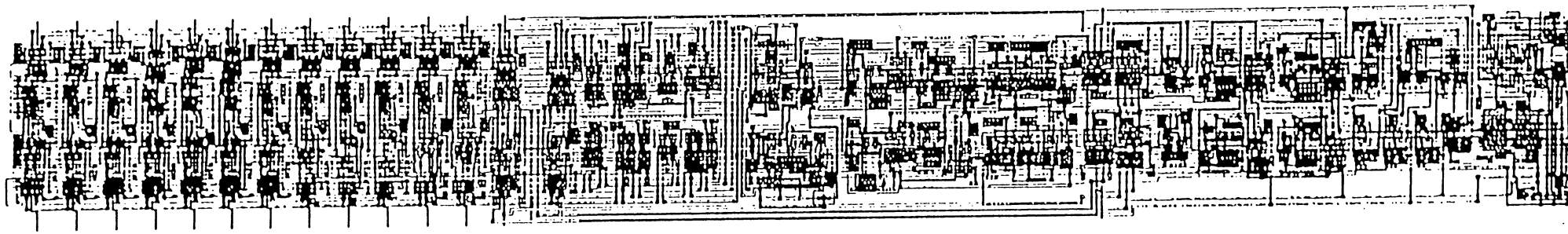


Partie commutation

Partie contrôle

commutateur à deux bits

Surface = 1279 x 165 μ^2

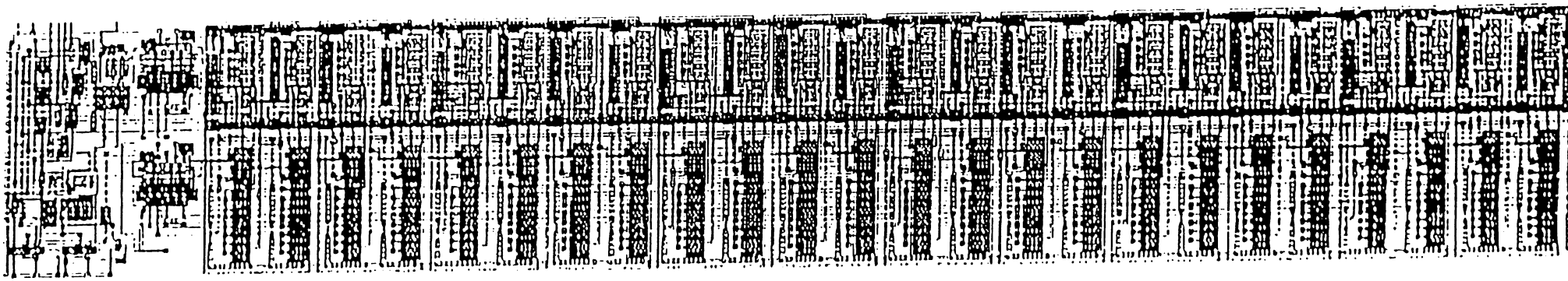


Partie commutation

Partie contrôle

Amplificateur bidirectionnel à 12 bits

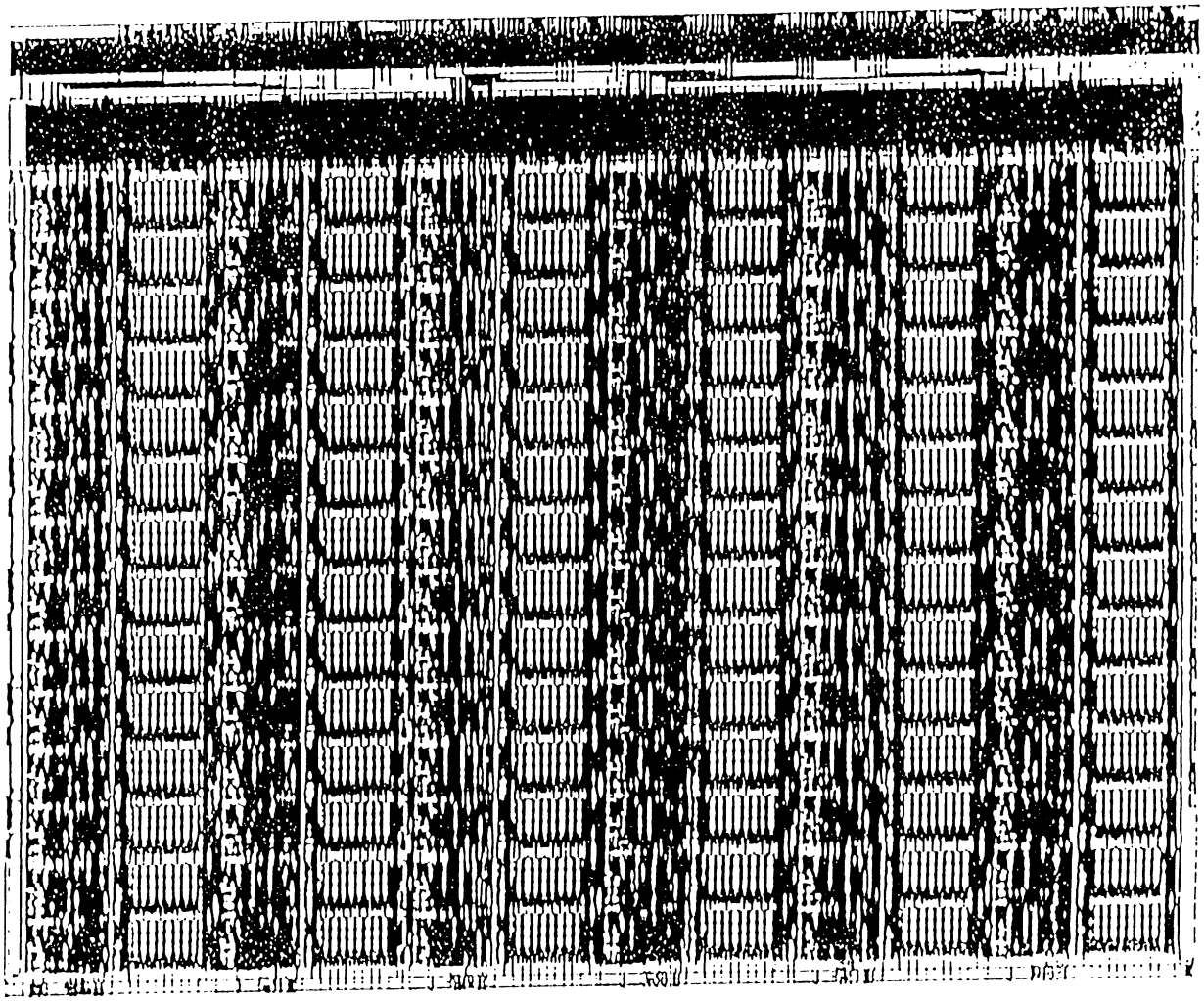
Surface = 1003 x 158 μ^2



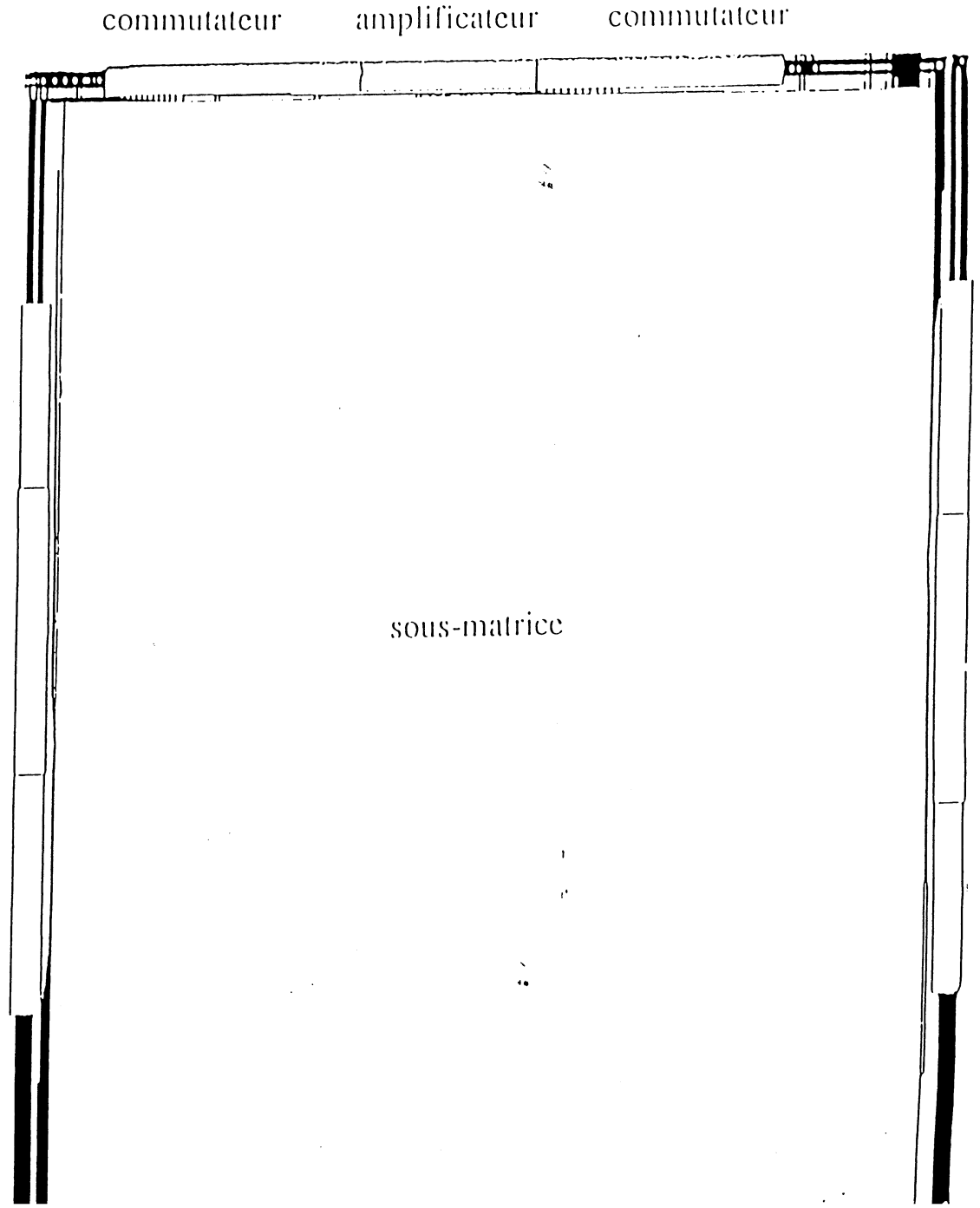
ANNEXE II

Sous-matrice de 7x12 PEs

surface = 4945,8 x 6102,2 μm^2



1/4 RETICULE

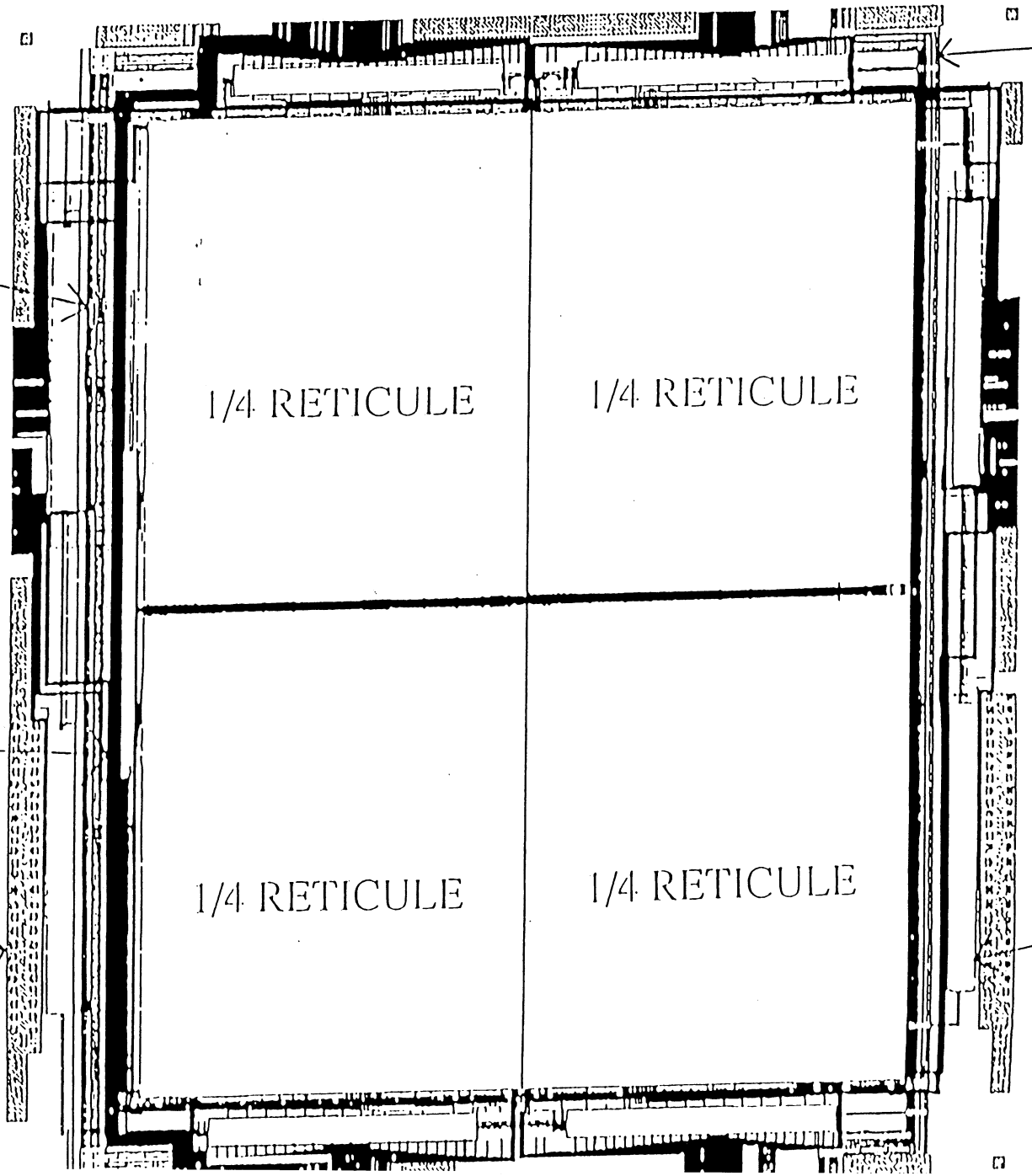


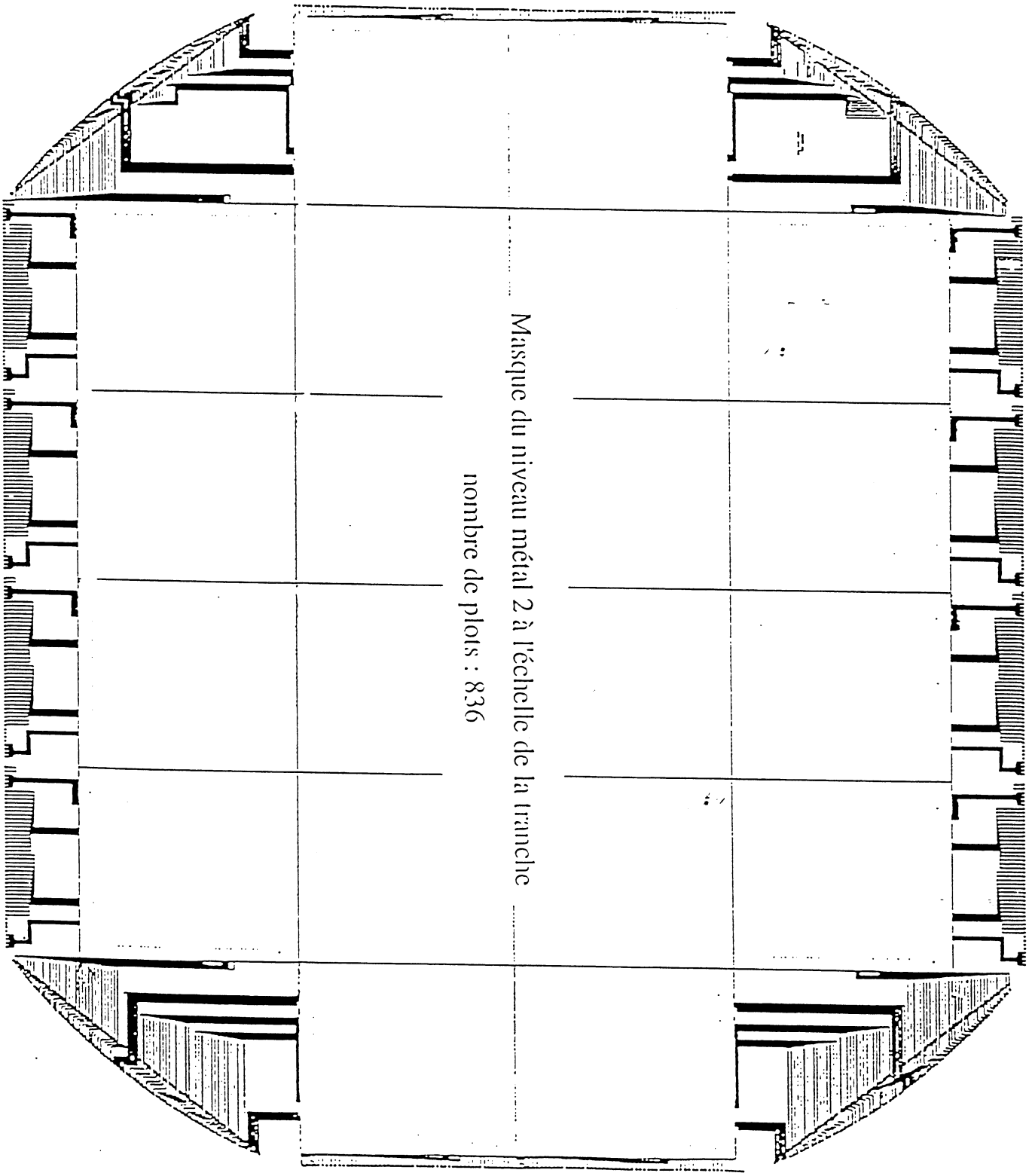
totale = 5398,3 x 6556,7 µ2

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

de configuration = 14,7% de la surface totale

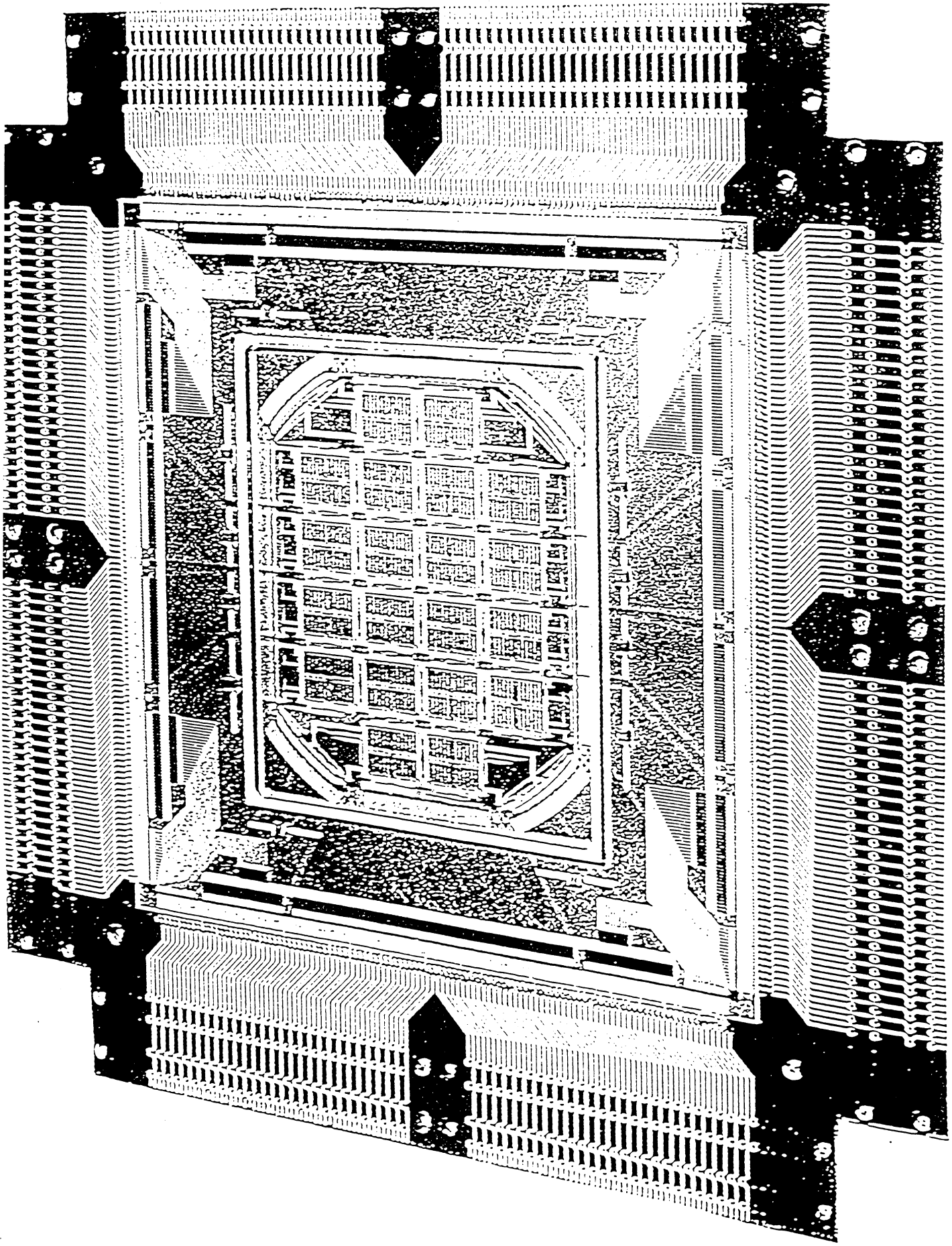
RETICULE





Masque du niveau métal 2 à l'échelle de la tranche

nombre de plots : 836



ANNEXE III

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.

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.

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

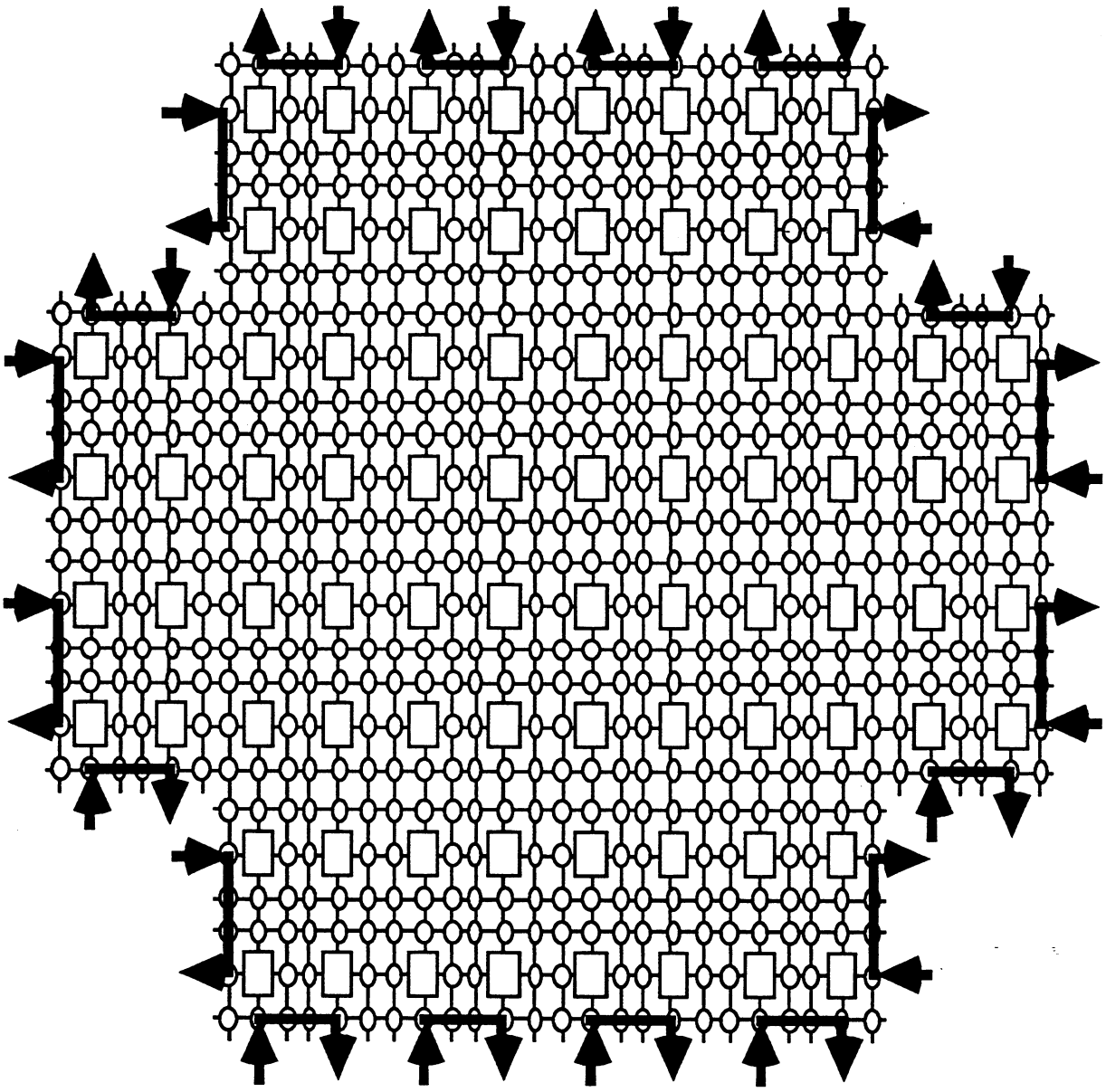
cm=cms increment A CSET=1

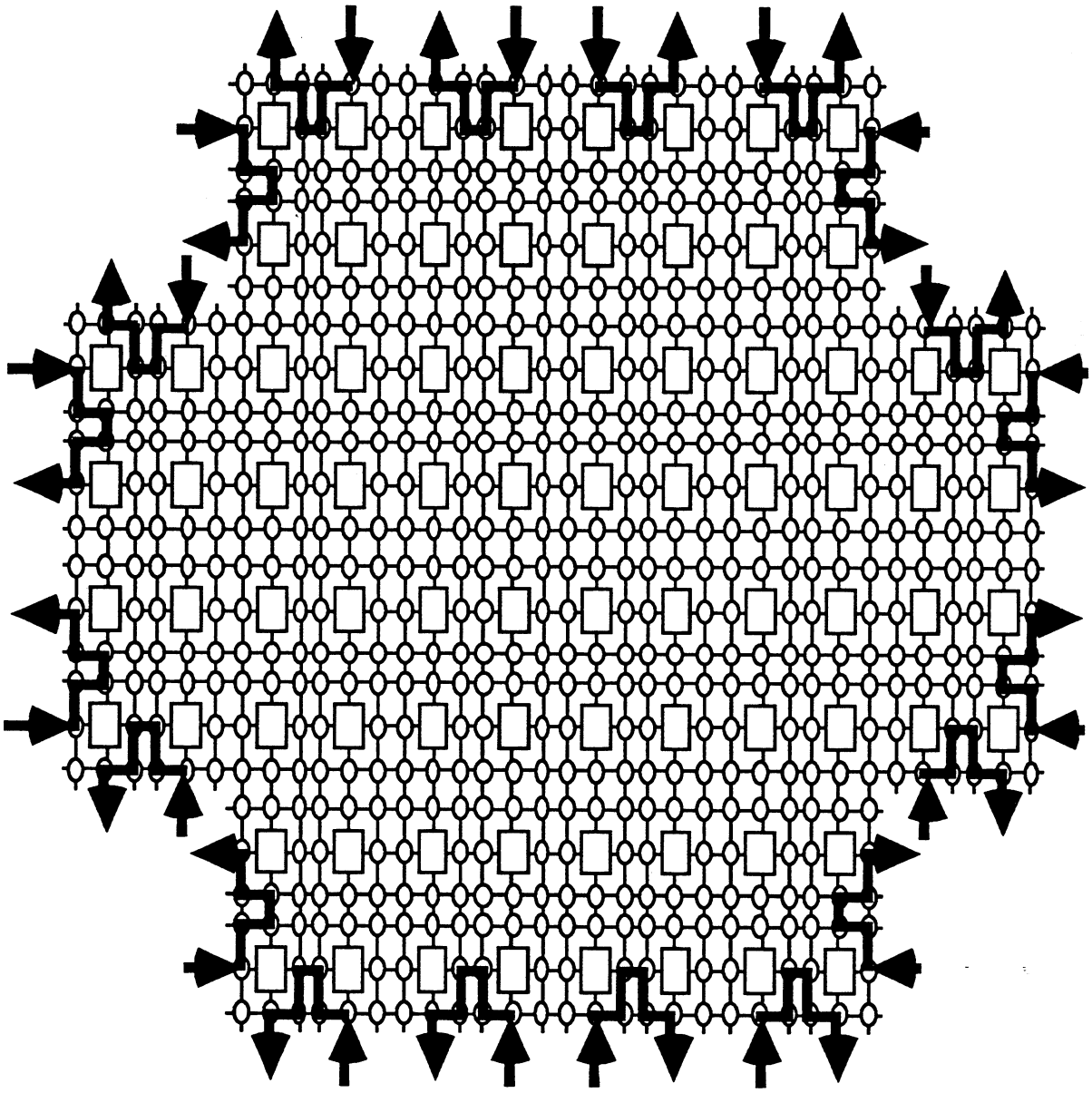
This is run for 12 cycles to output and compare the first set of sum data from all the rows in the array.

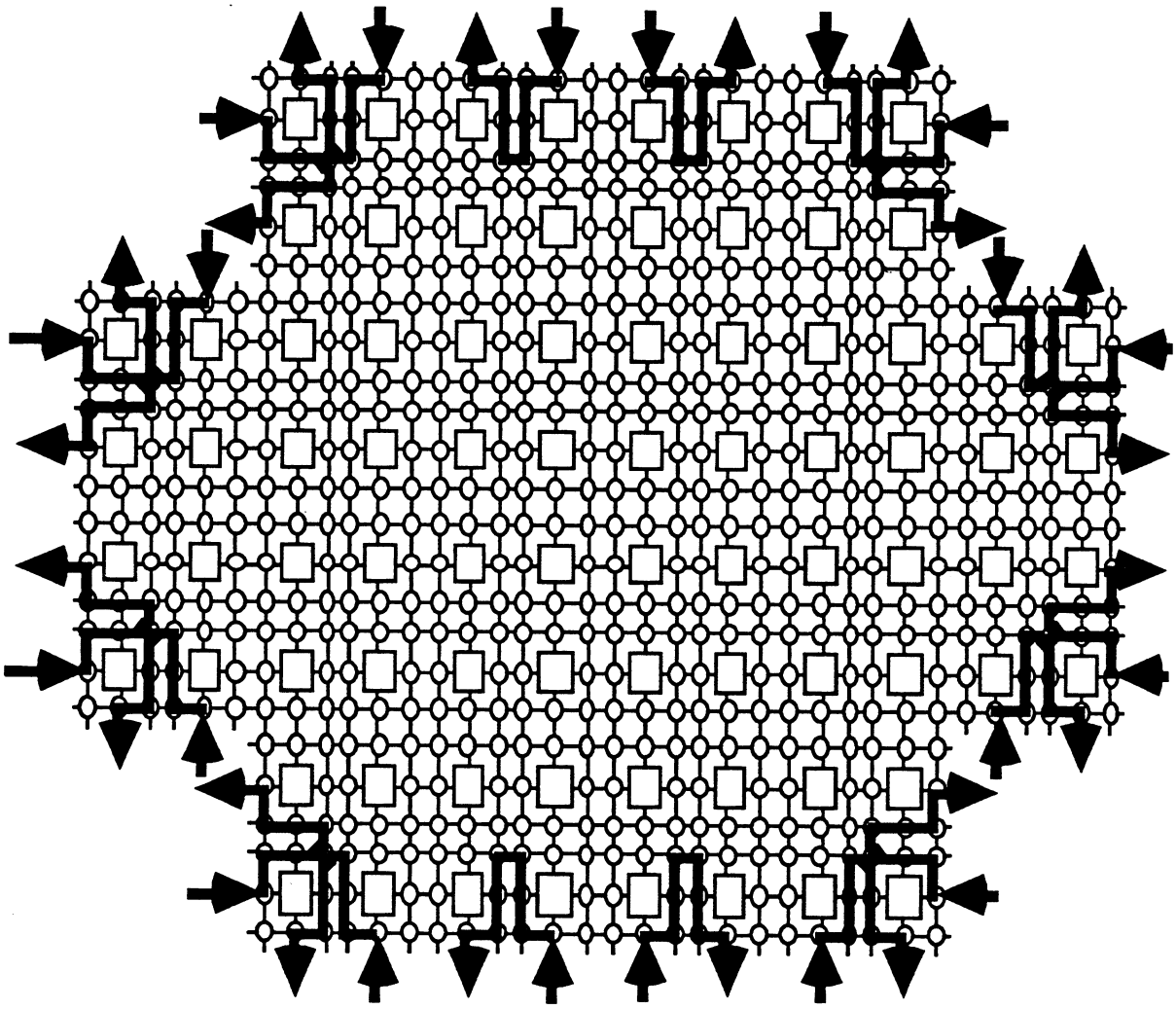
cm=R1 Increment A To fetch next set of sum data from RAM

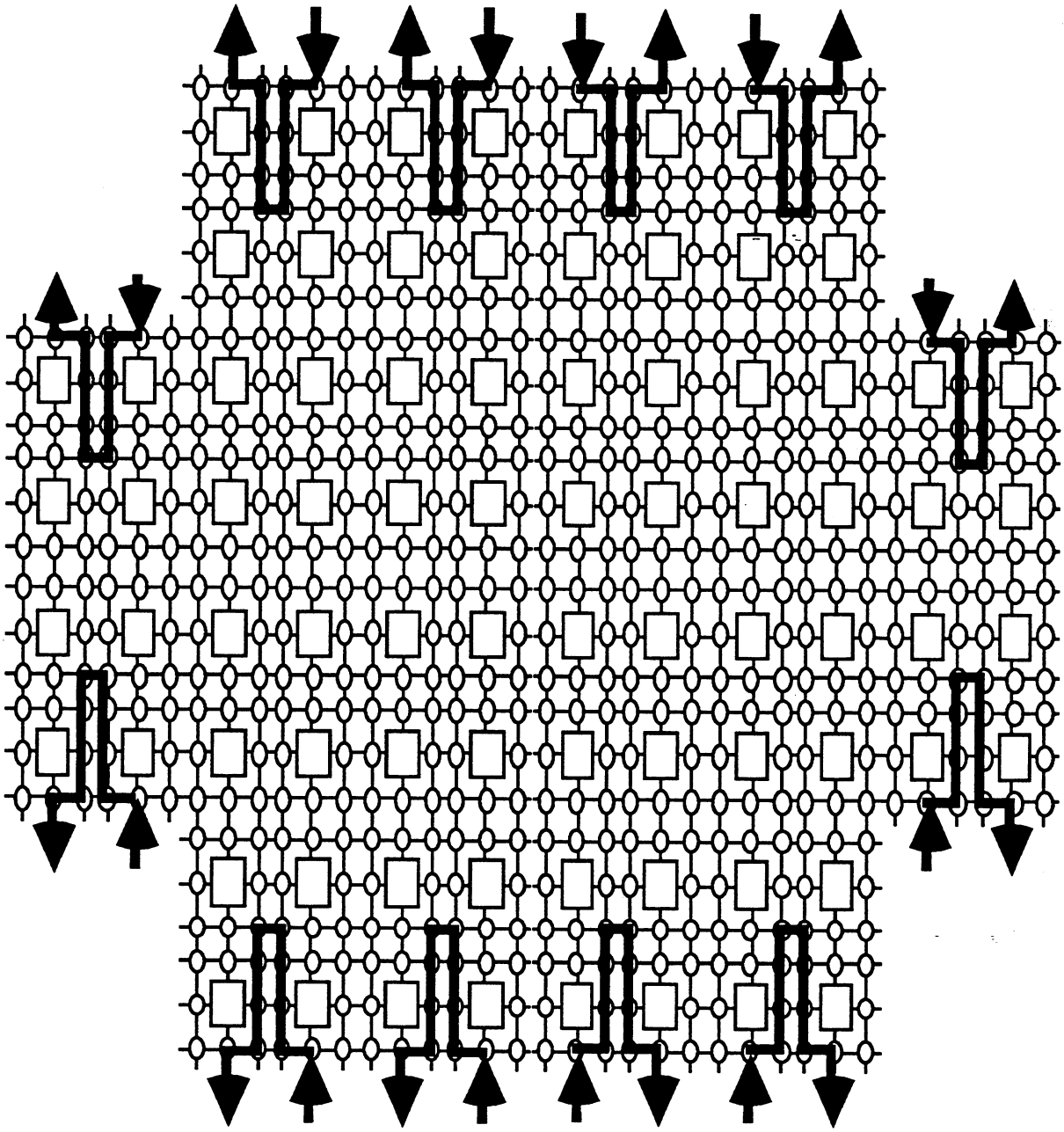
This data is then output and compared and the procedure repeated for the remaining two RAM locations.

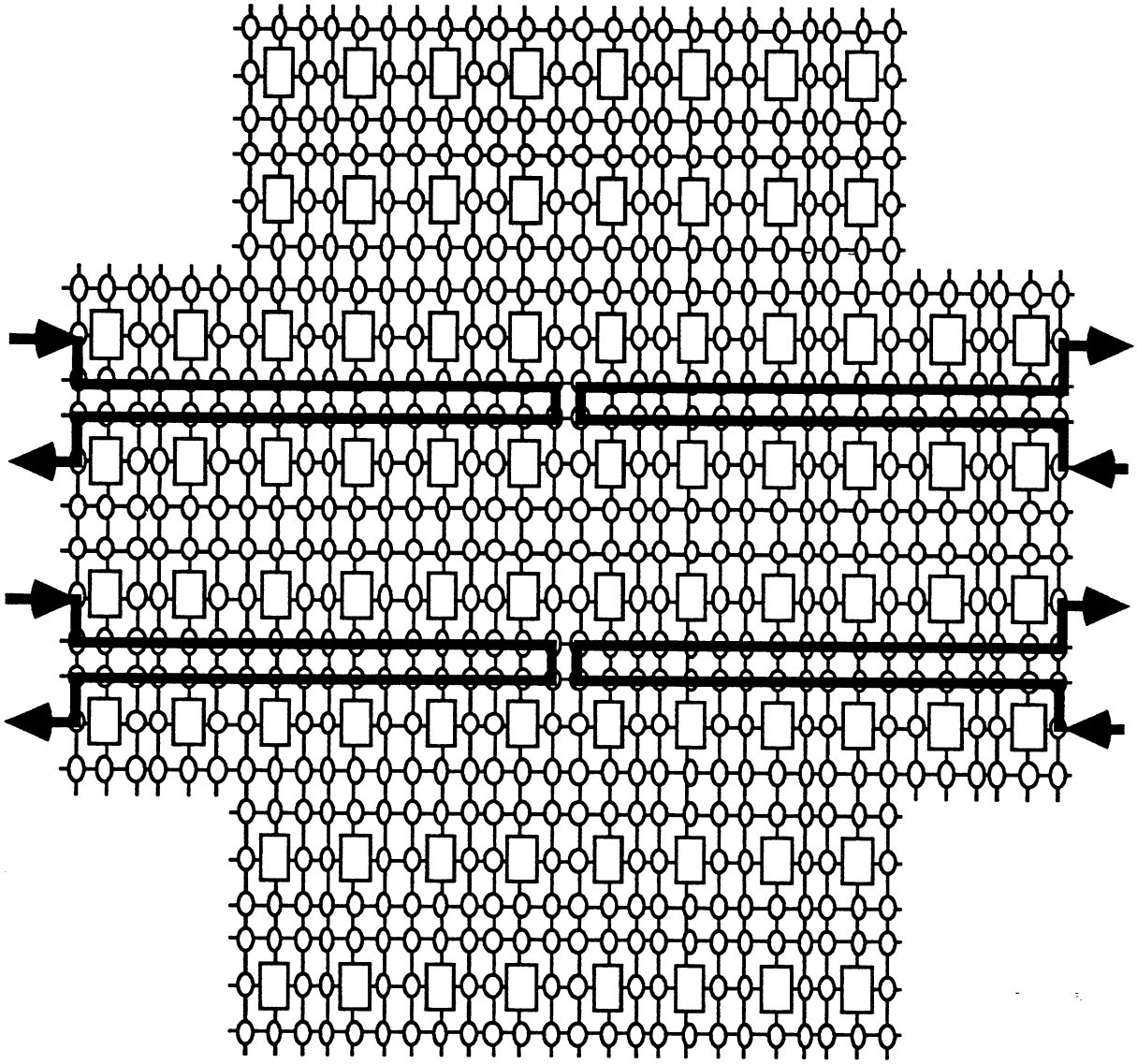
Test du réseau de commutateur : Différentes étapes de la phase I.

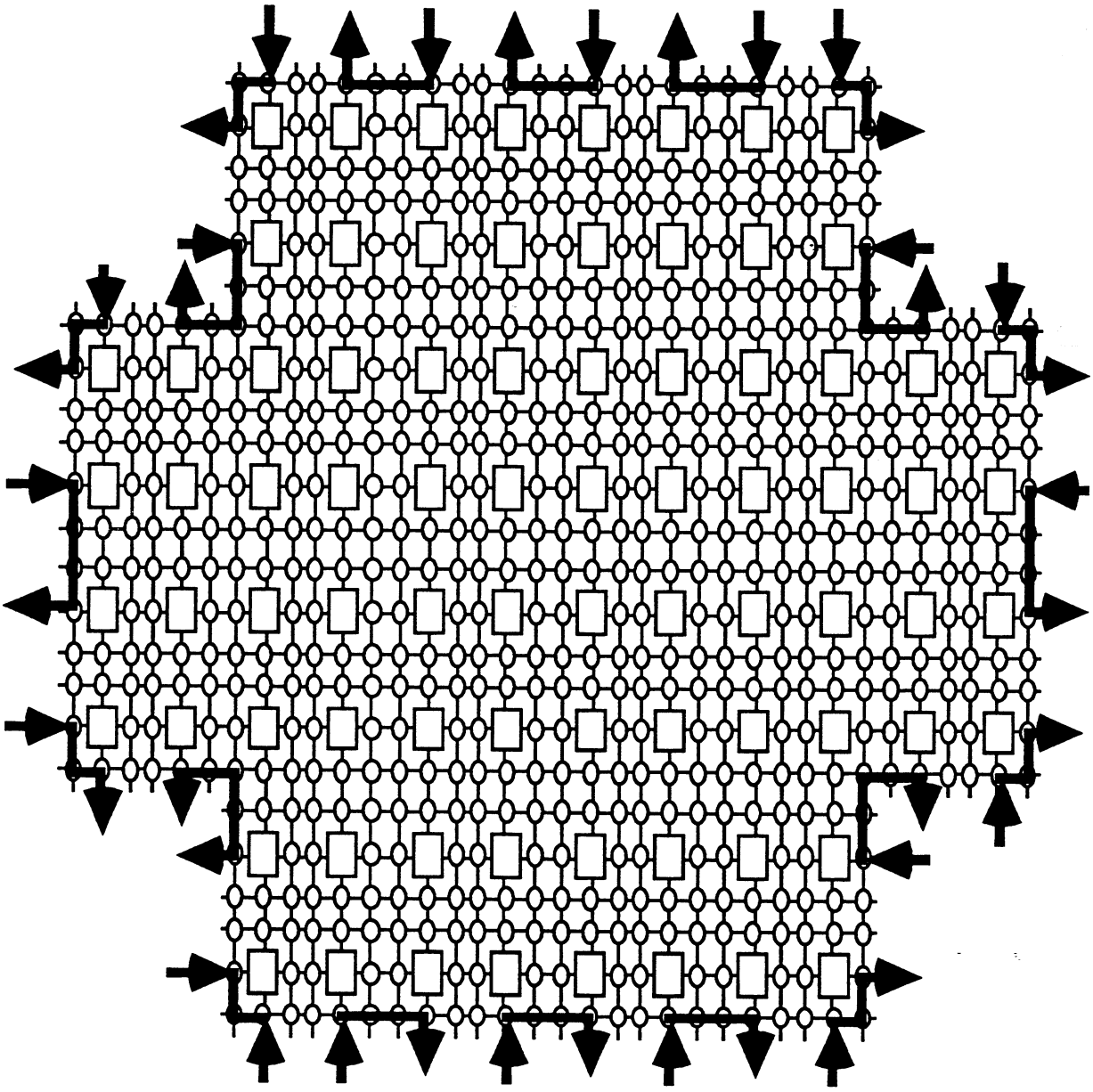


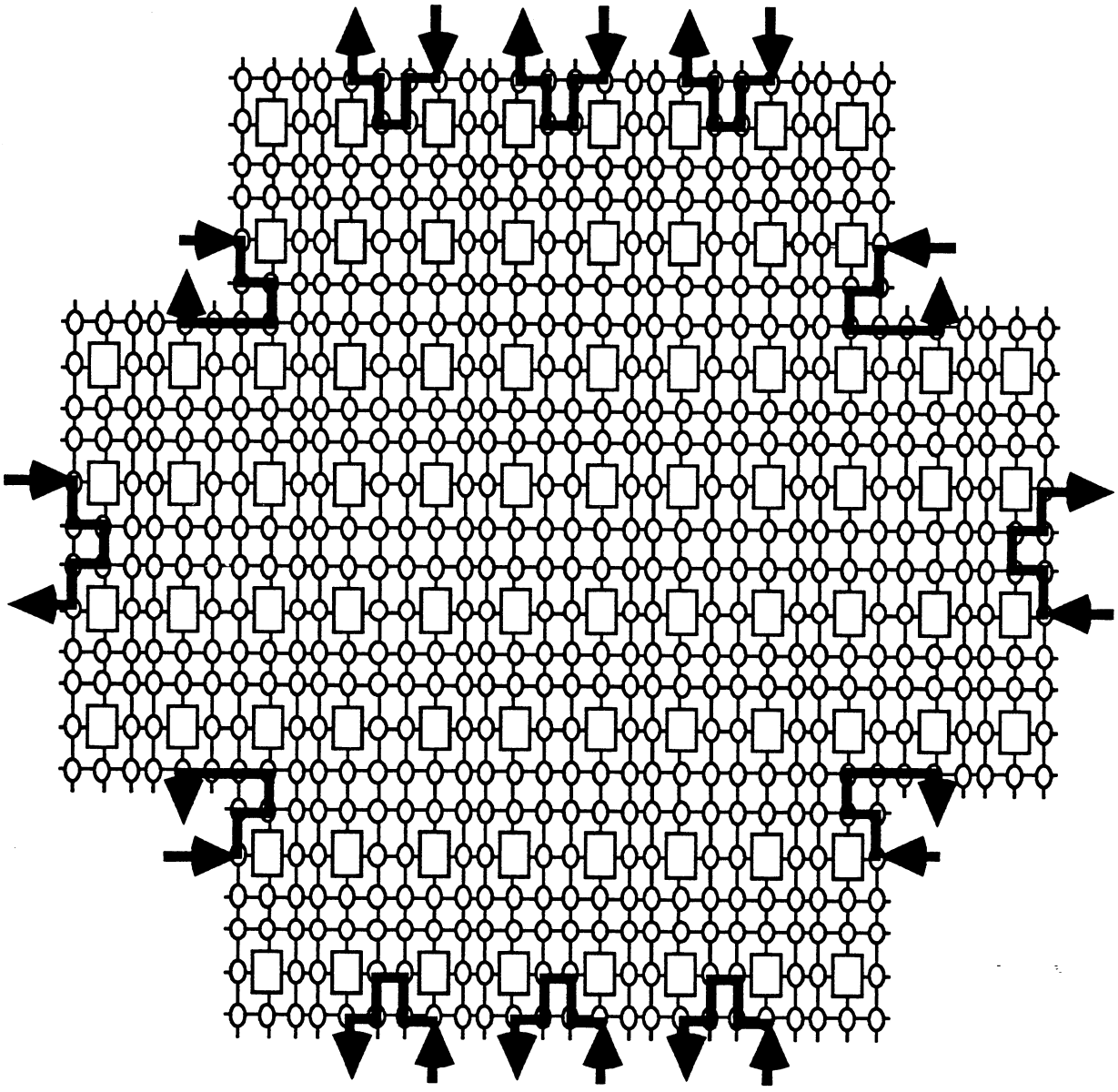












4 INITIALISATION

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 multiplexers, RAM addresses and write controls. These are:

cm=cms	fg=1	alu=ALU	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.

BIBLIOGRAPHIE

[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

[ChLe83] F.R.H Chung, F.T. Leighton and A.L. Rosenberg, " Diogenes : a methodology for designing fault-tolerant VLSI processor arrays,"*Proc. of the 13 Int. Symposium on Fault-Tolerant Computing*, Milano Italy, Jun 83, pp.26-31.

[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

[DiSa89] F. Distante, M.G. Sami, R. Stefanelli," Reconfiguration Techniques in the presence of faulty interconnections," *Proc. Int. Conf. on Wafer Scale Integration,* San Francisco, USA, January 3-5 1989, pp. 378-388.

[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

[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

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

[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

[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.

[ItSu89] Hideo Ito and Nobuyuki Suzuki,"*A Hypercube Design on WSI*", Int. Workshop on Defect and fault tolerance in VLSI Systems, Tampa Florida, USA, October 23-24 1989, pp. 248-259.

[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

[JeKu89] S.N. Jean, S.Y. Kung, "Necessary and Sufficient Conditions for Reconfigurability in Single-Track Switch WSI Arrays," *Proc. Int. Conf. on Wafer Scale Integration*, San Francisco, USA, January 3-5 1989, pp. 401-412.

[JeLo88] L.Jervis, F. Lombardi and D. Sciuto," Orthogonal Mapping : A Reconfiguration Strategy for Fault-Tolerance VLSI/WSI 2D Arrays", *Proc. Int Workshop on Defect and Fault tolerance in VLSI Systems,* Amherst, USA, Oct. 6-7 1988, pp. 7.4.1-7.4.12.

[KiRe89] Jung Hwan Kim, Sudhakar M. Reddy," On the design of Fault-Tolerant Two-Dimensional Systolic Arrays for Yield Enhancement", *IEEE Transactions on Computers,* Vol. 38, N°. 4, April 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

[KoSt88] Israel Koren and Charles H. Stapper, "Yield models for defect-tolerant VLSI circuits: a review," Defect and Tolerance in VLSI Systems, Ed. by Israel Koren, Pub. by Plenum Press, New York, 1988, pp. 1-21.

[Lafo89] L. Laforge," Extremally Fault tolerant Arrays",*Proc. Int. Conf. on Wafer Scale Integration,* San Francisco, USA, January 3-5 1989, pp. 365-378.

[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

[LeiLei85] T. Leighton, C. Leiserson,"Wafer-Scale Integration of Systolic Arrays", *IEEE Trans. on Computers*, Vol. c-34, N°5, May 85.

[LNSS87] F. Lombardi, R. Negrini, M.G. Sami, R. Stefanelli, " Reconfiguration of VLSI arrays : A covering approach", *Proc. of the 17 Int. Symposium on Fault-Tolerant Computing*, Pittsburgh, USA, July 6-8 1987, pp. 251-256

[LoSc87] F. Lombardi and D. Sciuto, "Reconfiguration in WSI Arrays Using Minimum Spanning Trees", *Proc. Int. Compeuro*, 1987, pp. 547-550.

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

[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

[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

[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

[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

[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

[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

[SaSt86a] M. Sami, R. Stefanelli," Reconfigurable architectures for VLSI Processing Arrays," *Proc. of the IEEE*, Vol. 74, N° 5, May 1986, pp. 712-722.

[SaSt86b] M. Sami, R. Stefanelli," Fault-Tolerance and Functional Reconfiguration in VLSI Arrays," *Proc. Int. Conf. Circuits Syst.*, 1986, pp. 643-648.

[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

[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

[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.

[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 Demonstrated 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

Les travaux décrits dans cette thèse ont donnés lieu aux publications suivantes :

E-F. Kouka, J-L. Patry, G. Saucier, " Defect tolerance in a Wafer Scale array for image processing," *Proc. International Workshop on Defect and Fault tolerance in VLSI systems*, Springfield, Ma, USA, October 1988, pp. 8.2-1 8.2-13.

E-F. Kouka, J-L. Patry, G. Saucier, " Defect tolerance in a Wafer Scale array for image processing," *Defect and Fault Tolerance in VLSI Systems,* Volume 1, Edited by Israel Koren, Published by Plenum Press, New York, 1989, pp.327-338 .

E-F. Kouka, J-L. Patry, G.Saucier, " A reconfigurable Wafer Scale array for image processing," *Proc. International Conference on Wafer Scale Integration*, San Francisco, CA, USA, January 1989, pp. 277-288.

E-F. Kouka, J-L. Patry, G. Saucier et Al., " ELSA: A European Large SIMD Array for Image Processing," *Proc. EURO ASIC 89*, Grenoble, France, January 1989, pp 364-380.

E-F. Kouka, J-L Patry, " Reconfiguration de circuits intégrés sur tranche de silicium: une approche expérimentale," *L'onde Electrique,* Mai-Juin 1989, Vol. 69, n° 3, pp 57-70.

J-L. Patry, G. Saucier, " Design of an universal switching network for reconfigurable 2D-arrays," *Proc. IFIP Workshop on Wafer Scale Integration*, Como, Italy, June 1989.

J-L. Patry, G. Saucier, A. Boubekeur, " Practical experiences in the design of a Wafer Scale 2D-array," *Proc. International Workshop on Defect and Fault Tolerance in VLSI Systems,* Tampa, Fl, USA, October 1989, pp 51-63.

A. Boubekeur, J-L. Patry, G. Saucier, " Reconfiguration in ELSA," *Proc. IFIP workshop on parallel architectures on silicon*, Grenoble, France, December 1989, pp 117-132.

A. Boubekeur, J-L. Patry, G. Saucier, " Practical experiences in the design of a Wafer Scale 2D-array," *Defect and Fault Tolerance in VLSI Systems,* Volume 2, Edited by C.H. Stapper, V.K. Jain, and G.Saucier, Published by Plenum Press, New York, 1990, pp 75-87.

E-F. Kouka, J-L. Patry. " Algorithms for restructuring WSI arrays of processors," *Computer Systems Science & Engineering*, Vol. 5, n° 2, April 1990, pp. 95-104.

A. Boubekeur, J-L. Patry, G. Saucier, "Switch Programming in a 2D-Array," *Proc. IEEE Workshop on Defect and Fault Tolerance in VLSI systems*, Grenoble, France, November 1990, pp. 178-187.

A. Boubekeur, J-L. Patry, G. Saucier, J. Trilhe, " Universal Switching Network : application to a WSI SIMD array," *Proc. International Conference on Wafer Scale Integration*, San Francisco,CA, USA, January 1991, pp. 256-262.

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, USA, Nov. 1991, pp. 296-299.

A. Boubekeur, J-L. Patry, G. Saucier, J. Trilhe, " Configuration of a Wafer Scale 2D Array of Single-bit Processor," *To be published in Special Issue of IEEE Computer on Wafer Scale Integration : Architectures and Algorithms*, April 1992.

Grenoble, le 18 Février 1992

ECOLE DOCTORALE

Affaire suivie par SIMEON Michèle
Tél : 76.57. 4525

N/Réf. : JPU/MS

Objet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

Mr A. GUYOT Maître de Conférence

Mr IVEY Professeur Université de SHEFFIELD

Mr PATRY Jean Luc

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 :

"Micro-électronique"

Pour le Président de l'INPG
et par délégation
le Directeur de l'Ecole Doctorale
J.C. LABOUME

Résumé

Cette thèse présente des méthodes et outils de conception de systèmes intégrés sur tranche entière (Wafer Scale Integration). L'application traitée (dans le cadre d'un projet européen ESPRIT) est une architecture, constituée d'un réseau 2D de 6720 processeurs (PE) monobits, destinée au traitement d'image de bas niveau. Pour tolérer les défauts de fin de fabrication, une approche hiérarchisée a été implantée. Au niveau sous-système, une technique de redondance figée a consisté à implanter une colonne de PEs de réserve, destinés à remplacer les PEs défaillants. Au niveau tranche entière, une technique de construction d'une cible maximale n'utilisant que des sous-systèmes s'appuie sur l'implantation d'un réseau de commutateurs permettant d'éviter les sous-systèmes défaillants. Une architecture originale des réseaux de commutateurs, contrôlée à partir des plots externes et des algorithmes efficaces de définition et construction du réseau opérationnel constituent les points forts de cette thèse.

Mots-clés : WSI, rendement, tolérance aux défauts, reconfiguration, réseau de commutateur, SIMD.

Abstract

This thesis presents methods and tools for the design of Wafer Scale Integration systems. The demonstrator implemented within an european ESPRIT project is an SIMD 2D array of monobit processors (PE) dedicated to low level image processing. To cope with end of manufacturing defects, two levels of defect tolerance have been implemented. At a subsystems level, spare columns of PEs have been added to replace defective elements. At the wafer level, a switching network allow to bypass defective subsystems. Configuration algorithms aims at defining the largest array that may be constructed which is then physically established on the wafer. The originality of the switching array and the efficiency of the configuration algorithms are the strong points of this thesis.

Keywords : WSI, yield, defect tolerance, reconfiguration, switching network, SIMD.