Test intégré de processeur facilement testable

Philippe de Choudens

T H E S E

Présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Pour obtenir

Le TITRE de DOCTEUR de 3ème Cycle Informatique . . . . . .

Par

M. . . Philippe de CHOUDENS . . . . . .

. . . . . Test Intégré de Processeur Facilement Testable . . . . . .

. . . . . .

. . . . . .

Soutenue le 14 Novembre 1985 devant la Commission d'Examen

J U R Y

Monsieur GEFFROY PRESIDENT

Madame SAUCIER)

Messieurs LANDRAULT) EXAMINATEURS

GOBERT)

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M. MERMET
Directeur des Etudes et de la formation : Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

TEST INTEGRE DE PROCESSEUR

FACILEMENT TESTABLE

Philippe de CHOUDENS

THESE 3ème CYCLE INFORMATIQUE

REMERCIEMENTS

Je tiens à remercier Monsieur le Professeur GEFFROY ainsi que Monsieur le Professeur LANDRAULT qui, malgré leurs nombreuses occupations ont bien voulu accepter de présider et de faire partie de ce jury.

J'adresse, tout particulièrement, mes remerciements à Madame le Professeur SAUCIER qui a bien voulu m'accueillir dans son Laboratoire et diriger mon travail.

Que Monsieur GOBERT, Chef de l'équipe du Laboratoire d'Electronique et de Physique qui a accepté de patronner cette thèse et sans qui, celle-ci n'aurait pu être possible, soit assuré de ma profonde gratitude.

Je veux enfin remercier l'ensemble du Laboratoire Circuits et Systèmes et de l'Equipe du Laboratoire d'Electronique et de Physique, et tout particulièrement Madame BELLON, Messieurs WEIL-RABAUD, VELAZCO, JAY et GENESTIER, pour m'avoir supporté durant le temps que j'ai passé parmi eux et surtout pour l'aide précieuse qu'ils n'ont cessé de m'apporter tout au long de mon travail.

A Danielle, Henri et Christelle.

TABLE DES MATIERES

L'introduction dans les systèmes de contrôle à haute sûreté de fonctionnement de circuits programmables à très haute intégration (V.L.S.I) pose des problèmes très critiques.

Les applications concernées sont essentiellement le transport terrestre (Train, Métro), l'avionique (Equipements embarqués, Pilotes automatiques), le spatial, le nucléaire (Contrôle de processus de centrale nucléaire) et certains domaines de contrôle de processus industriel.

Le concepteur de tels systèmes éprouvera toujours certaines difficultés avant d'acquérir la confiance nécéssaire à l'utilisation de composants nouveaux dans des applications mettant en jeu des vies humaines (Transports), des investissement élevés (Spatial) ou risquant d'apporter des nuisances graves consécutives à un fonctionnement erronné. Pour de telles applications, il convient de prouver en fin de conception la conformité du système de contrôle aux spécifications initiales; en fin de fabrication l'intégrité du matériel (Composants sans défaillance à l'instant initial) et, au cours de son utilisation, la sécurité du système (Non émission de commandes dangereuses par suite de défaillance du dispositif).

Pour atteindre ce dernier objectif, les architectures utilisées sont des architectures redondantes. L'application est implémentée sur 2, 3 ou 4 "Systèmes indépendants" et, une comparaison des sorties permet d'améliorer la sécurité. Une implémentation "simplex", même munie de tests en ligne puissants pourra très difficilement être considérée comme un dispositif à haute sécurité.

Une implémentation d'un système à haute sûreté de fonctionnement necessite une évaluation minutieuse de sa sécurité en vue de certification, par exemple, or cette évaluation se heurte pour les circuits intégrés actuels du type microprocesseur à des dificultés spéciales.

Ces dernières sont dues à la difficulté d'assurer un test efficace et sûr de ces dispositifs. Dans **(BEL 84)**, l'auteur développe différentes stratégies relatives au test de tels composants. Il apparait que vue leur complexité, un test exhaustif (test par identification) est inenvisageable. De même, un test par distinction élaboré sur une description structurelle du circuit sur laquelle est définie un ensemble de pannes s'avère d'une mise en œuvre délicate. En effet, cette méthode demande une très bonne

connaissance du circuit à son niveau logique, électrique et même parfois au niveau du dessin des masques (Souvent inaccessible à l'utilisateur). Elle demande aussi d'avoir une bonne connaissance de la modélisation des pannes au niveau technologique, ce qui est impossible actuellement dans les technologies n.MOS et c.MOS **(WAD 78) (TIM 83)**.

De plus, si l'on trouvait une modélisation suffisante des défaillances au niveau des transistors ou de la porte, l'élaboration de tests détectant ces défaillances au niveau d'un dispositif comprenant plusieurs dizaines de milliers de transistors serait d'une complexité prohibitive.

On peut donc se demander à juste raison, si une application à haute sûreté de fonctionnement implantée sur des dispositifs qui sont testés en ligne et hors ligne de façon imparfaite pourra être l'objet d'une certification valable.

Considérons à titre d'exemple l'architecture du poste d'aiguillage informatisé (PAI) **(SEV 84)**, implémenté sur 2 microprocesseurs 68000 (Figure 1).

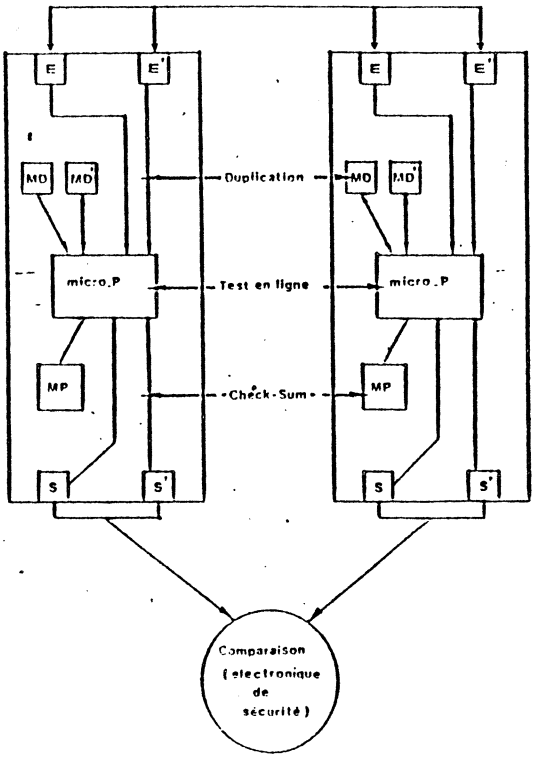


Figure 1

Le P.A.I. est un système basé sur une stratégie de tolérance aux pannes multi-niveaux.

i) Il est composé de 2 unités identiques isolées (Duplex) dont les résultats (émission de commandes) sont constament comparés à l'aide d'un comparateur de sécurité réalisé en électronique hybride.

ii) chaque unité contient des dispositifs interne de détection d'erreurs.

Au sein de chaque dispositif la sécurité est implémentée par des techniques de redondance; On a une duplication des mémoires de données (MD et MD'), des entrées (E et E') et des sorties (S et S') dont les valeurs sont comparées.

Le programme d'application est stocké dans une mémoire programme (MP) dont le contenu est périodiquement testé par réalisation d'un "check-sum".

Le microprocesseur est partiellement testé par l'utilisation d'un détecteur de codes invalides, par l'activation de "chiens de garde" et par l'éxécution régulière de séquences de test (test en oisiveté).

- L'évaluation de l'efficacité des stratégies adoptées dans cette réalisation permetterait d'évaluer la sûreté du système, malheureusement, l'utilisation de V.L.S.I. pose à nouveau le problème du manque d'informations concernant leurs modes de défaillances (difficultés d'observation vu la complexité du problème, absence d'hypothèses de pannes réalistes, ...). La méthode d'évaluation devra donc soit éliminer ces inconnues par prise en compte d'hypothèses pessimistes, soit les faire influer dans la détermination du taux d'insécurité de l'application.

L'approche développée dans **(PIL 82)** va dans le sens de la deuxième solution et, son application au poste d'aiguillage informatisé donne les résultats de la figure 2.

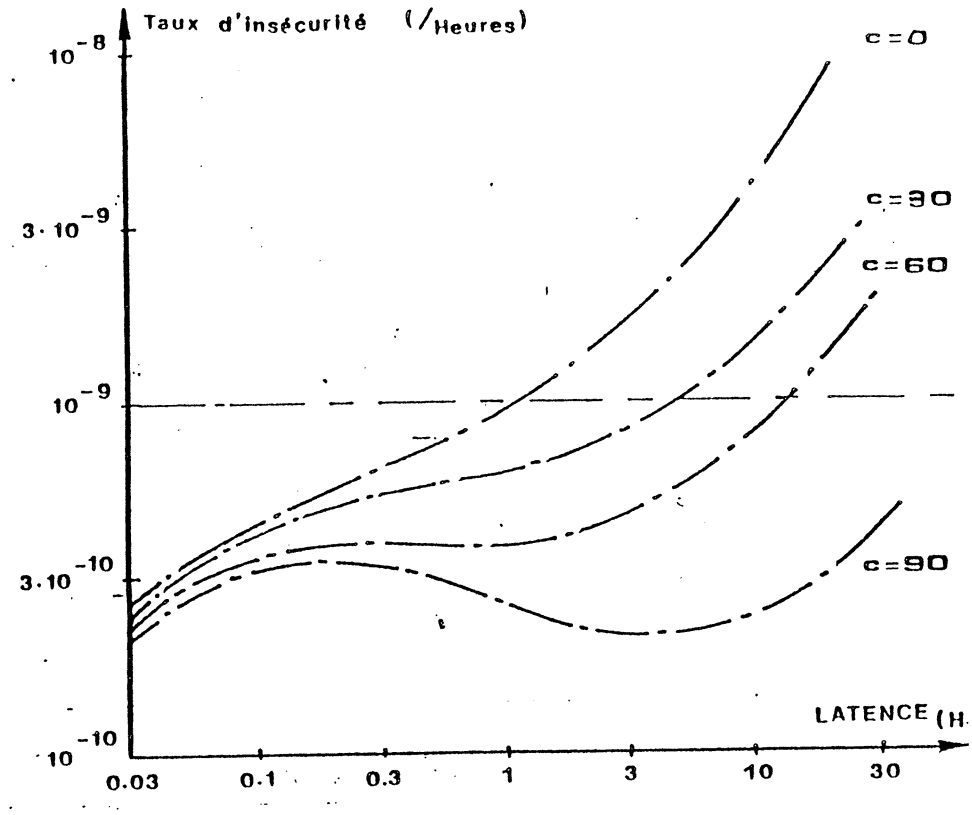


Figure 2

Ces courbes montrent l'évolution du taux d'insécurité du P.A.I. en fonction de deux paramètres mals connus qui sont le taux de couverture c du test en ligne du microprocesseur (autotest par déroulement d'un programme de test totalement éxécuté toutes les trois heures) et la latence des pannes non détectées par ces programmes de test (ces pannes engendreront une erreur dans le programme d'application au bout de H heures, H étant cette latence).

L'examen de ces résultats permet de définir facilement quels sont les caractéristiques attendues pour un microprocesseur assurant le contrôle d'une application a haute sûreté de fonctionnement; On pourrait par exemple exiger :

-Que le taux de couverture des programmes de test soit au moins de 60% .

- Que la latence d'une panne pouvant échapper à ces programmes soit majorable en fonction de la fréquence d'exécution du programme de test (soit par exemple pour le PAI : H < 2 heures).

Le développement de circuits spécifiques destinés aux environnements à haute sécurité devra donc avoir pour objectif principal d'obtenir des produits dans lesquels toute panne sera succeptible d'être détectée rapidement, soit par le programme d'application associé à des

dispositifs de test en ligne puissant, soit par l'éxecution de programmes de test spécifiques destinés à vérifier certaines ressources précises sous utilisées par les programmes d'application ou, dont l'intégrité est vitale
(test en ligne discontinu pendant des périodes d'oisiveté de l'application).

Si nous prenons l'exemple du 68000 utilisé dans l'application du poste d'aiguillage informatisé, on sera gèné pour diverses raisons :

∞ L'application met en jeu des fonctions classiques des automatismes logiques ce qui représente une toute petite proportion des possibilités opérationnelles de ce microprocesseur (environ 30 %) . En particulier, les possibilités d'interruption, sauvegarde du contexte et la plupart des instructions de branchement ne sont pas utilisées . On peut donc craindre à juste raison qu'une panne reste cachée fort longtemps et, que cette latence nous mette dans une situation délicate de double panne (dans les deux dispositifs).

∞ Le test périodique (test en oisiveté) sera difficile à mettre en œuvre. S'il n'est pas trop délicat de balayer les instructions non utilisées par le programme d'application, il n'en sera pas de même pour la logique d'interruption, d'interface asynchrone. Il est d'ailleurs à noter que de tels dispositifs poseront toujours de gros problèmes aux concepteurs d'applications à haute fiabilité; En effet, des mécanismes tels que les interruptions ont la capacité d'influer sur le déroulement du programme d'application d'une façon non controlable temporellement, ainsi des dispositifs de test en ligne basés sur la vérification des flots d'instructions ou sur la mise en place de "chiens de garde" seront délicats à mettre en œuvre.

Il faut aussi rappeler que la couverture des tests sera pénalisée par la méconnaissance des hypothèses de défaillances et de la structure interne des circuits .

A la lueur des constatations précédentes, il apparaît donc necessaire de développer un composant spécifique doté d'un maximum de dispositifs facilitant son immersion dans les applications demandant une grande sureté de fonctionnement.

Quel " cahier des charges " pour un tel microprocesseur ? :

La discussion précédente nous enseigne que :

1) Le microprocesseur ne doit pas être surdimensionné, et être bien adapté à l'application qu'il contrôle :

Dans les domaines de contrôle de processus, il est inutile de disposer de systèmes dont seulement un faible pourcentage des capacités est utilisé; il ne faut pas, non plus, que du fait de sa simplicité, et de son manque de moyen de traitement, l'utilisateur soit contraint de développer des programmes d'application longs, compliqués, et donc impossibles à valider.

Nous nous efforcerons donc de doter notre circuit d'un jeu d'instructions lui conférant les capacités de traitement d'un microprocesseur classique de 16 bits, tout en étant adapté aux automatismes logiques travaillant en temps réel (comme le poste d'aiguillage informatisé).

2) Le microprocesseur doit avoir une latence de panne faible :

C'est le premier objectif pour un circuit à haute sureté de fonctionnement. Toute panne doit être détectée en un temps court, soit par le programme d'application, soit par un programme de test.

Tout sera donc mis en oeuvre pour faciliter la mise en place d'une stratégie de test en ligne efficace.

Afin de définir quels sont les dispositifs les mieux appropriés, nous allons en un premier temps, discuter les différentes méthodes ayant fait l'objet de recherches dans ce domaine. Nous discuterons successivement les approches consistant à intégrer des dispositifs d'autotest dans les circuits, à intégrer les dispositifs de test dans le logiciel d'application, ou à déléguer les opérations de vérification à un coprocesseur supplémentaire.

a) Mise en place de dispositifs de test en ligne intégrés au processeur:

- Une approche codant les informations circulant à l'intérieur du circuit est proposée dans **(MAR 78) (ASH 77) (SMI 83)**.

Un décodeur permet ainsi de détecter toute erreur intervenant dans l'information traitée.

- **(DIA 75)** et Iyengar et Kinney **(IYE 82)** ont étudié la détection de pannes dans les unités de contrôle microprogrammées. L'ensemble des états est partitionné, et une fonction permet d'évaluer, à partir d'un état courant, et de "clés" stockées dans le microprogramme , un ensemble d'états suivants possibles. La comparaison de cette évaluation avec l'état réellement obtenu permet de détecter les pannes survenant dans cette partie du circuit.

- Courtois **(COU 79) (COU 81)** et Courtois et Marchal **(MAR82)** étudient divers mécanismes de détection d'erreurs (code opération invalide, détecteur d'adresse erronnée, détecteur de protection mémoire,...) résultant de collages permanents de lignes de bus ou d'un mauvais fonctionnement du compteur programme pour les microprocesseurs 6800 et 68000.

De telles méthodes sont basées sur une modification profonde de la structure des microprocesseurs. Leur principe utilise des techniques de codages **(WAK 78)(BOS 82)** (code de parité, code de Hamming ...) ne permettant souvent que la détection d'erreurs simples, la mise en place de dispositifs détectant les erreurs doubles ou triples devenant extrémement compliquée. Leur implémentation résulte donc en une augmentation importante de la surface de silicium utilisée pour implanter les organes de stockage des divers codes et leurs dispositifs d'évaluation.

On peut s'interroger sur le bien fondé de telles méthodes, pour une utilisation dans des applications à haute sureté de fonctionnement. En effet, un accroissement de la surface du circuit implique une diminution de sa fiabilité et un risque de défaillance plus important (plus de transistors implique plus de points susceptibles de tomber en panne).

b) Les dispositifs de test sont intégrés au logiciel:

Diverses approches, utilisant des dispositifs intégrés non à l'architecture, mais à son logiciel d'application, ont été développées :

- Dans **(BEL 82)** , les auteurs modèlisent le processeur , par un graphe , ce qui leur permet de définir des tests intégrés aux applications

tournant sur le processeur , et détectant les erreurs pouvant apparaître sur les entrées du circuit .

- Des techniques sont proposées dans **(AYA 79)** et **(KAN 75)** , pour intégrer aux logiciels d'application , un "observateur" qui permet de vérifier la bonne exécution des autres opérations du programme .

- Parhami , Metze et Mili **(PAR 77)** et **(MET 81)** ont , pour leur part , développé des règles permettant l'écriture de programmes s'auto-testant lors de leur exécution .

Toutes ces mèthodes impliquent une modification , non négligeable , des programmes d'application . Ceci accroît donc le temps de calcul , ainsi que l'espace mémoire occupé . Ils ont cependant l'avantage de ne pas demander de modifications au niveau de l'architecture du circuit. Leur mise en oeuvre impose toutefois de modeliser les erreurs pouvant survenir, soit au niveau du composant, ce qui n'est pas aisé, du fait de la méconnaissance du circuit et des hypothéses de fautes dans les technologies utilisées, soit au niveau du logiciel, ce qui est encore certainement plus délicat à obtenir, les erreurs au niveau logiciel étant virtuellemnt impossible à modeliser ou à énumérer.

c) Méthodes consistant à décharger la "responsabilité" du test sur un autre processeur ou circuit spécialisé (Watchdog processor) :

- Mc Cluskey et Namjoo **(MCC 82)** et **(NAM 83)** considérent l'ajout d'un processeur détectant les pannes , logicielles , ou matèrielles , causant un accés mémoire intempestif .

- Dans **(NAM 82)** , **(SRI 82)** et **(SHE 83)** , l'analyse de signature , technique permettant le compactage des informations circulant dans le circuit , a été utilisée . L)es informations étant codées , le résultat attendu , peut être ainsi comparé à la signature obtenue .

- Chavade et Crouzet **(CHA 82)** , s'appuyant sur le travail de Crouzet et Landrault **(CRO 80)** ont développé un circuit , permettant à l'usager de réaliser des systèmes auto-test , à base de microprocesseurs dupliqués.

Ici, on ne demande pas de modifications profondes, ni du circuit au niveau matériel, ni du logiciel d'application. Cependant, la complexité de l'application se trouve accrue par la présence d'un "coprocesseur", ou de dispositifs supplémentaires, destinés à la vérification du flot d'informations. Un tel coprocesseur devra, si l'on désire superviser de façon correcte le déroulement d'une application, posséder une capacité de traitement pratiquement équivallente au processeur contrôlant l'application elle-même. Les problèmes de validation du processeur principal se trouvant donc reportés sur la validation du coprocesseur de contrôle. D'autre part, dans des applications où l'on dispose déja d'une architecture redondante (duplex, triplex ...), est-il vraiment indispensable d'ajouter des dispositifs complexes pour assurer la vérification du bon fonctionnement de l'application ?

Dans l'ensemble des mèthodes présentées précedemment, seules celles citées dans les parties b et c semblent présenter un réel interêt dans l'optique de la réalisation d'un composant du type microprocesseur déstiné à un environnement à haute sureté de fonctionnement.

On s'aperçoit que l'on a interêt à ne pas trop compliquer l'architecture du circuit ou de l'application (par utilisation de composants nouveaux qui s'avereront géniteur de problèmes). Cependant, il apparaît indispensable de faciliter la tâche du programmeur en lui offrant des dispositifs permettant de mettre en place au niveau logiciel une stratégie de test efficace; pour ce faire, les approches citées dans b, combinées à celles développées par Shen et Shuette dans **(SHE 83)** qui consiste à réaliser la compaction des informations circulant dans le système paraissent représenter un bon compromis entre la complexité ajoutée à l'application, sa simplicité de mise en oeuvre et les impératifs de testabilité d'un environnement à haute sureté de fonctionnement.

A) HSURF est un microprocesseur facilitant le test en ligne :

L'architecture interne du microprocesseur HSURF a été étudiée afin d'être compatible avec ce qui a été ennoncé précédemment. C'est ainsi qu'une analyse de signature, réalisée selectivement sur les divers bus, permettra d'accroitre la visibilité interne du circuit.

L'utilisateur peut alors, suivant l'organisation du système utilisant HSURF :

- Soit comparer la signature des deux circuits fonctionnant en parallele de façon synchrone.

- Soit comparer périodiquement les signatures obtenues au cours de l'exécution des programmes d'application à des valeurs précalculées (cas d'un fonctionnement duplex asynchrone par exemple)

B) HSURF est un microprocesseur facilitant le test en oisiveté :

Suivant l'application dans laquelle HSURF fonctionne, le test en ligne permet dans la plupart des cas de réaliser un test complet des ressources internes au microprocesseur . Cependant , la plupart des hypothèses faites sur l'évaluation des systèmes à haute sureté de fonctionnement , ont en commun , de considérer une latence de panne à distribution indépendante du temps : l'environnement doit être analogue à celui que délivrerait un testeur aléatoire . Or , dans la réalité , ce cas de figure est assez rare . Par exemple , le programme d'aterrissage d'un pilote automatique d'avion , n'est sollicité qu'une fois par vol ; de même , un système de commande d'aiguillages de chemins de fer rencontre des configurations pèriodiques de circulation des trains de période 24 heures (ou plus lorsque certaines voies d'une gare ne sont utilisées que trés rarement) .

On sera donc souvent contraint , pour pallier à cette difficulté , et pour éviter que des ressources de l'application restent inutilisées pendant de longues pèriodes , de développer une stratégie de test en ligne discontinue (appelé test en oisiveté) , permettant :

- soit d'éliminer des configurations d'environnement se produisant naturellement trop rarement , en activant ainsi les organes sous utilisés

- soit de tester de la façon la plus complète possible , des organes ou opérateurs sous utilisés , spécifiques du circuit . La possibilité de réaliser un test exhaustif de ces organes , étant , bien entendu , l'idéal .

Le microprocesseur HSURF posséde , dans le but de rendre plus aisé ce test en oisiveté ; un ensemble d'opérateurs testables le plus complétement, facilement et rapidement possible . Pour cela , tout opérateur est rendu totalement accessible : les registres d'entrée et de

sortie de ces opérateurs peuvent être modifiés ou observés , à tout moment , au cours de l'exécution de programmes de test spécifiques à l'opérateur étudié . Cet impératif d'accessibilité a demandé l'implantation d'instructions de test spéciales à HSURF :

- test des registres de la partie contrôle (RI , micro-RI , micro-CO . . .)

- test des opérateurs (U.A.L. , MULT)

C) HSURF, un composant testé à l'instant t=0 :

Les évaluations de systèmes à haute sureté de fonctionnement sont basées sur l'hypothése importante suivante :

" Le composant est sans défaillance à l'instant T=0 "

Il est donc nécessaire de réaliser en fin de fabrication (T=0) un ensemble de tests complets et efficaces. Or, par manque de stratégie favorisant leur réalisation, et vue leur longueur et leur complexité, ceux-ci se trouvent abrégés au maximum. Il devient alors délicat de considérer l'hypothése comme vérifiée.

HSURF posséde un automate de test spécialisé , qui associé à un testeur spécifique , capable de générer certains signaux nécessaires , permet la vérification d'un ensemble de points critiques (minimum vital) . Cet ensemble une fois validé , autorise l'utilisation des dispositifs de test prévus pour la réalisation des tests en ligne et en oisiveté , afin de vérifier le reste du circuit .

Le fait d'intégrer des opérateurs pensés en vue d'un test facile et rapide , a pour but de rendre le temps de test en fin de fabrication acceptable , permettant de garantir ce circuit pour des applications à haute sureté de fonctionnement .

PARTIE 1

Présentation du micro-processeur HSURF

Particularités du microprocesseur HSURF:

Les différentes caractéristiques de HSURF sont **(GEN 84) , (SAU 84) , (JAY 84)** :

- Partie opérative :

Organisation entre deux bus , de tous les registres indépendants et accessibles individuellement . Les opérateurs ont été choisi pour être facilement testables . Toutes les informations circulant dans cette partie opérative , et notemment les bus , peuvent être signées afin de permettre une meilleur observation du circuit .

- Partie contrôle :

Un contrôleur micro-programmé , avec tous ses éléments accessibles (registre de micro-instruction , micro-compteur ordinal . . .), permet l'execution du jeu d'instructions .

- Le jeu d'instructions :

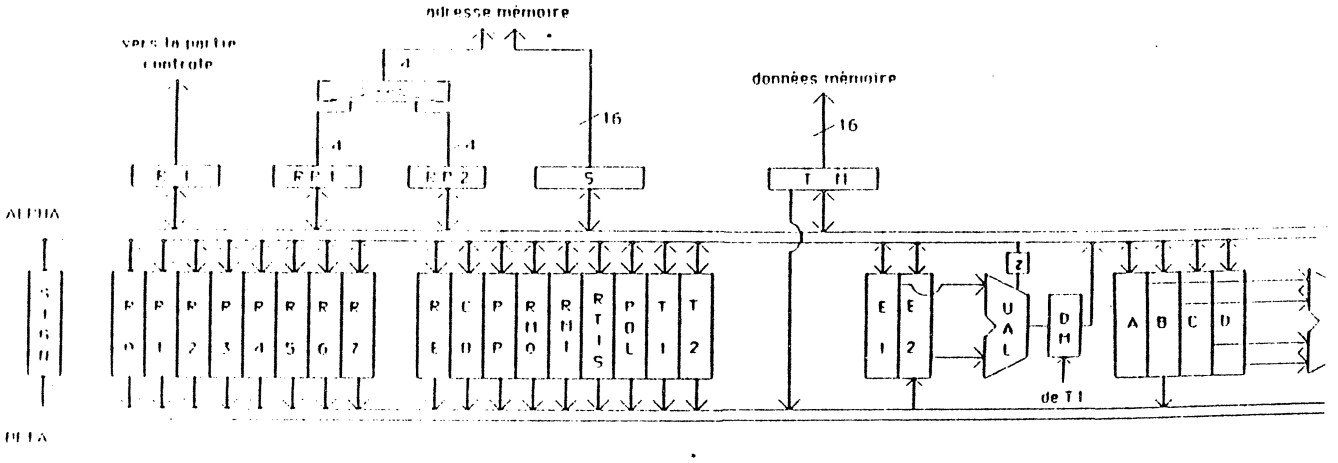
* possibilité de traitement de mots de 16 bits , aussi bien que de bit isolé d'un mot (utilisé dans les automates programmables)

* possibilté de traitement des matrices bidimentionnelles (applications temps-réel portant sur le contrôle de l'évolution de variables) .

* ensemble d'instructions spécialisées facilitant le test du circuit .

1) Partie opérative :

Le dessin de la figure 1 montre cette partie du circuit .



a) Les opérateurs :

- Les registres :

L'utilisateur dispose de 8 registres de travail notés R0 à R7 . De plus , un certain nombre de registres spécialisés existent :

* CO : le compteur ordinal contient l'adresse de l'instruction en cours d'execution .

* PP : le pointeur de pile contient l'adresse du sommet de pile .

* RE : le registre d'état mémorise les valeurs de 10 indicateurs d'état .

* T1 et T2 : sont deux registres tampon , contenant des résultats intermédiaires , lors de l'exécution de certaines instructions .

* E1 et E2 : sont des registres placés à l'entrée de l'U.A.L.

* RI : le registre instruction contient l'instruction en cours d'exécution .

* RM0 et RM1 : sont deux registres pouvant contenir l'adresse d'un descripteur de matrice .

* S et TM : sont des registres d'entrée / sortie avec les bus externes d'adresses et de données .

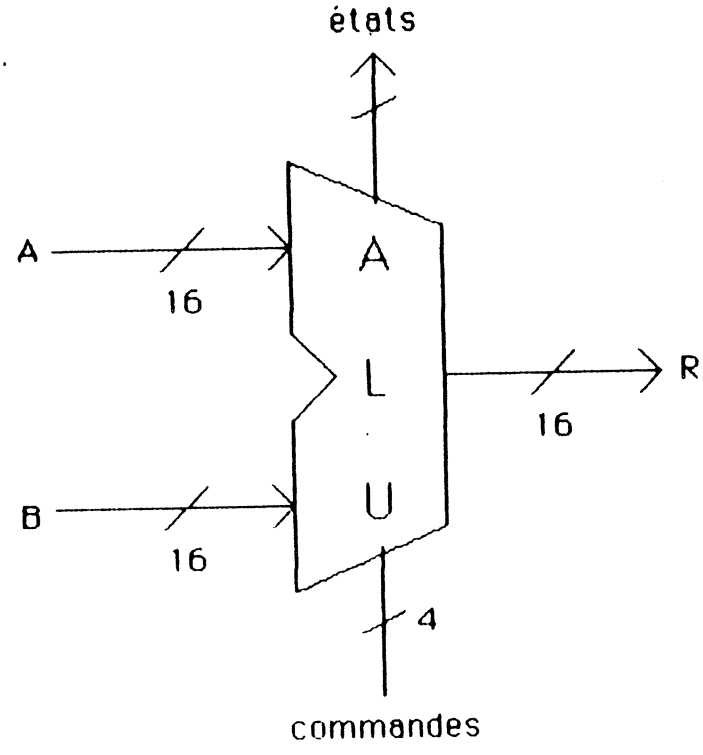
* RP1 et RP2 : contiennent la page de mémoire dans laquelle on est en train de travailler . Ceux-ci permettent d'accroitre l'espace mémoire adressable à 1 M mots (20 bits d'adresse) .

* A , B , C et D : sont les registres tampon , placés à l'entrée du multiplieur .

* SIGN : est un registre à décalage , permettant de réaliser une analyse de signature des informations circulant sur les bus .

* POL et RTIS : contrôlent cette analyse de signature . Ils contiennent respectivement , le polynôme de signature et le type des informations à signer) .

- L' Unité arithmétique et logique :



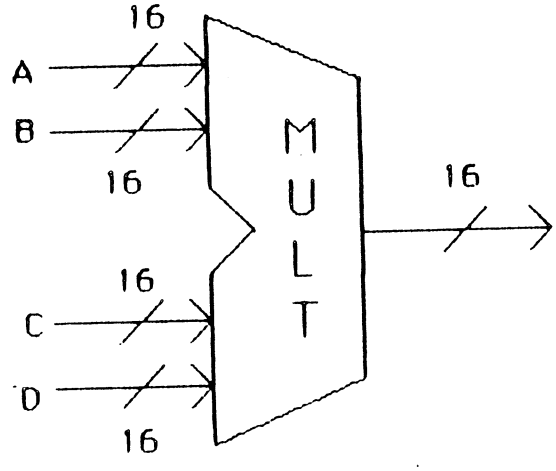
C'est une U . A . L . à deux entrées , traitant des opérandes sur 16 bits , et capable d'effectuer les 16 opérations suivantes :

A + B
A + B + Retenue
A - B
A - B - Retenue
A + 1
A - 1
non A
A et B
A ou B
non (A et B)
non (A ou B)
A disjonction B
Les décalages arithmétiques et logiques à droite et à gauche .

Quatre indicateurs d'état sont générés :

Z : le résultat de l'opération est nul .
N : Le résultat est négatif .
C : Une retenue sortante est générée .
V : Débordement .

- Le multiplieur (MULT) :



Un multiplieur cellulaire est utilisé , pour des raisons qui seront évoquées dans la partie IV . Celui-ci réalise l'opération suivante , sur ses quatre registres d'entrée A , B , C et D :

$$S = A * B + C + D$$

b) Organisation de la mémoire :

La mémoire est divisée en 16 pages , contenant chacune 64 Kmots de 16 bits . Un numéro de page de 0 à 15 , contenu dans un des registres RP1 ou RP2 , sélectionnera la page de travail désirée .

L'un des registre est utilisé pour le programme et les données , tandis que l'autre permet d'accéder aux éléments d'une matrice . Un démultiplexeur se chargeant de sélectionner l'un ou l'autre de ces registres , suivant le cas .

Le traitement des éléments d'une matrice nécessite un ensemble d'informations qui sont regroupées dans un descripteur en mémoire . Ces informations sont :

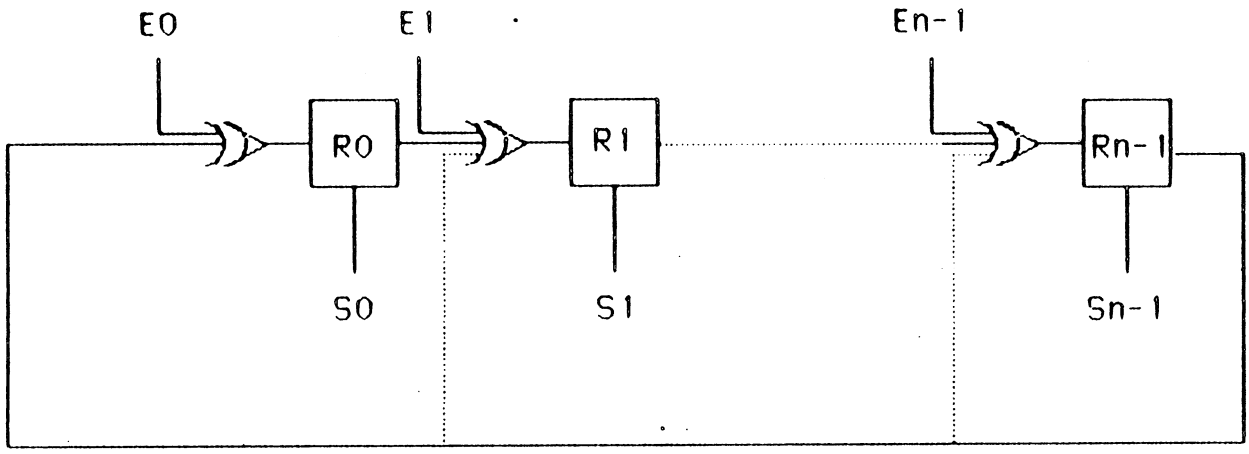
- Une adresse de début de matrice (adresse du premier élément) : numéro de page + adresse du premier élément dans la page .

- La taille de la matrice : nombre de colonnes + nombre de lignes .

Pour accéder à l'un des éléments d'une matrice , il faudra préciser l'adresse du descripteur de la matrice , ainsi que les numéros de ligne et de colonne de l'élément considéré .

c) Analyse de signature **(DAV 78) (SMI 80) (WIL 82) (HAS 84) (JAY 85)** :

L'analyse de signature est une méthode de compactage d'information . Elle est le reste d'une division polynomiale . Cette opération est réalisée par un registre à décalage :



Les sorties S_i donnent le reste de la division polynomiale des entrées E_i par un polynome P tel que :

$$P(X) = X^n + H_{n-1} X^{n-1} + \ldots + H_1 X + 1$$

avec :

$H_i = 1$ si la sortie de R_{n-1} est cablée sur l'entrée de la porte EOR de R_i ($0 < i < n$)

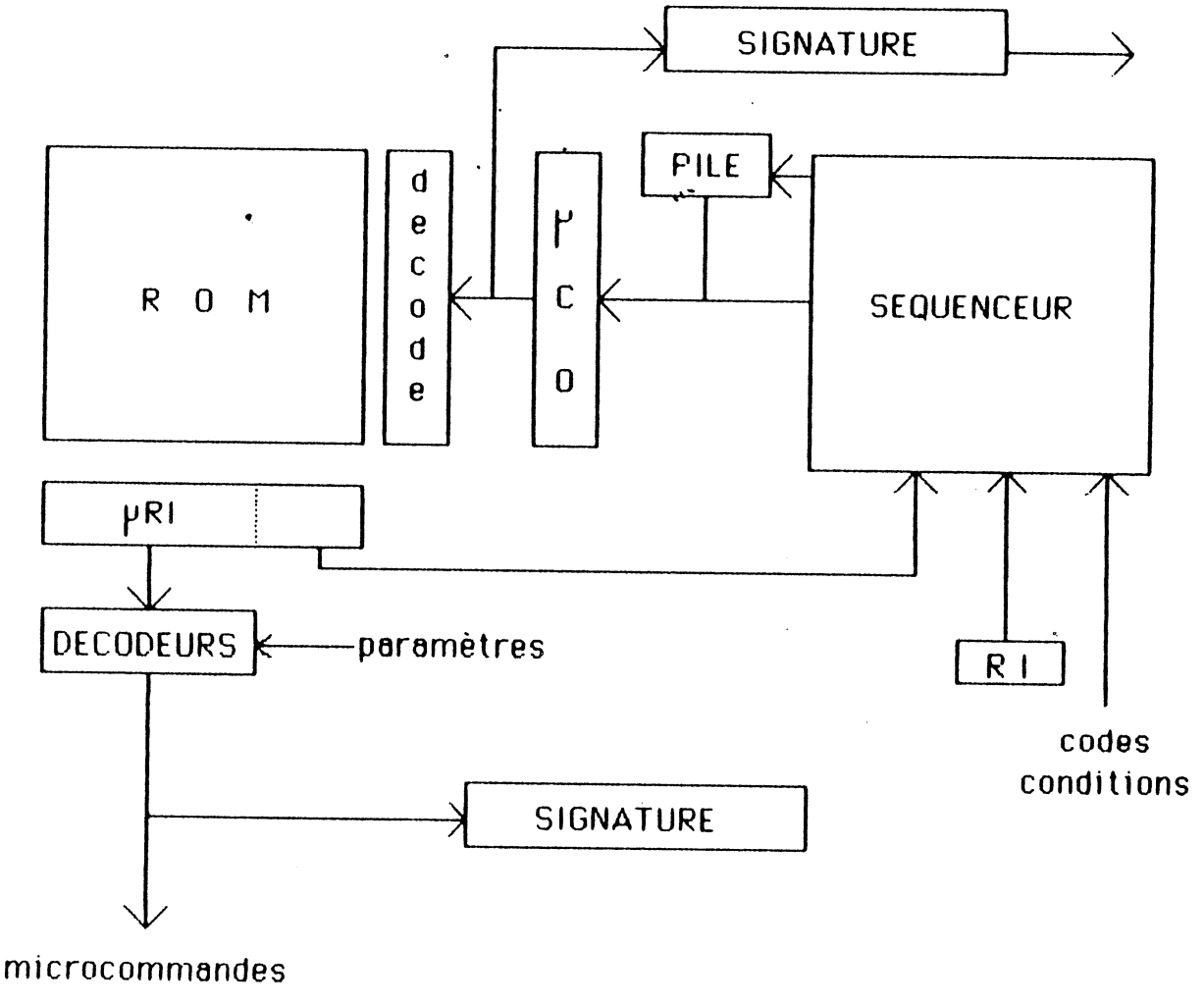
$H_i = 0$ sinon.

Dans la partie opérative du microprocesseur HSURF , le polynôme réalisant la division peut être variable . Ses coefficients H_i sont contenus dans le registre POL , et la signature est effectuée dans le registre SIGN . De plus le type d'informations signées , est laissée libre à l'utilisateur . Le type des informations à signer est codé , et rangé dans le registre RTIS . Ce peut être :

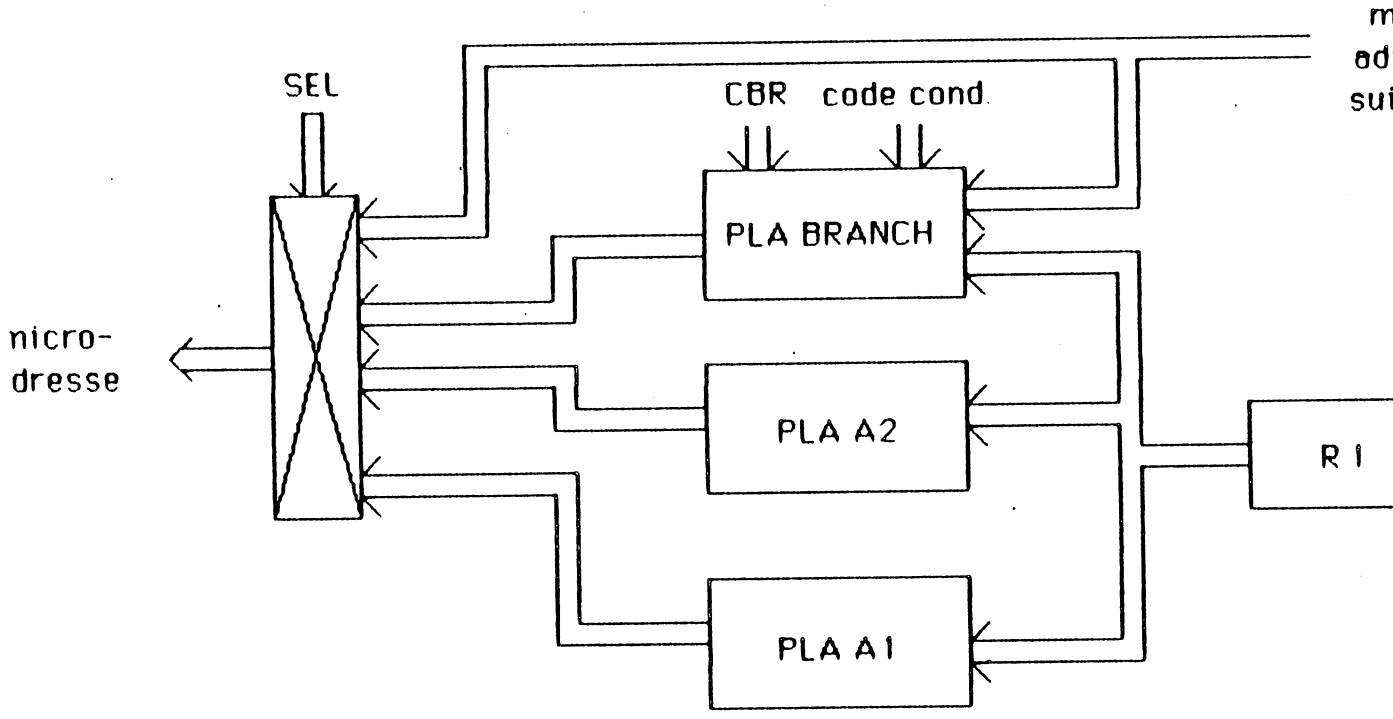
- les adresses
- les données lues
- les données écrites
- les codes instructions
- le bus ALPHA
- le bus BETA
- pas de signature

2) La partie contrôle :

Pour cette partie contrôle , c'est un séquencement réalisé à partir de deux PLAs de décodage des instructions qui est retenu .



Le séquenceur comporte deux PLAs de décodage des instructions , et un PLA de branchement réalisant les tests intervenant dans l'organigramme de contrôle (notemment sur les bascules d'état).



Sequenceur de HSURF

Deux registres d'analyse de signature permettent une meilleur observation des informations circulant dans cette partie du circuit.

SGN : réalise une signature sur les micro-adresses de la ROM
SGC : réalise une signature sur les micro-commandes générées par cette même ROM.

3) Les instructions :

L'ensemble du jeu d'instructions de HSURF est divisé en deux parties :

- instructions "classiques"

- instructions de test

Parmi les instructions classiques, on retrouve :

les opérations arithmétiques **ADD SUB . . .**
les opérations logiques **AND OR EOR . . .**
les opérations de décalage **LSR LSL ASR ASL . . .**
les comparaisons
les transferts **LOAD STORE . . .**
les branchements **BRA BCC . . .**
les sauts
les opérations de manipulation de pile **PUSH PULL . . .**
les opérations sur les matrices **COMPARMAT**

Certaines de ces instructions peuvent, en outre être exécutées en utilisant un dispositif de masquage qui permet de travailler directement sur certains bits d'un mot.

Chacune de ces opérations peut être réalisée avec un mode d'adressage, choisi parmi un ensemble contenant tous les modes d'adressage des opérandes classiques : direct, registre, indirect par registre . . ., ainsi que par un mode spécial, permettant d'atteindre tout élément d'une matrice. Pour celui-ci, il suffit d'indiquer l'adresse du descripteur de la matrice, ainsi que les numéro de ligne et de colonne de l'élément souhaité.

Les instructions de test ont pour rôle de permettre à l'utilisateur d'accéder à tous les registres internes du micro-processeur , et de contrôler les mecanismes d'analyse de signature . Ces instructions sont au nombre de trois :

1) LRS : chargement d'un registre spécialisé avec une valeur immédiate . Les registres spécialisés sont ceux qui sont innaccessibles avec les instructions LOAD et STORE et qui pour la plupart sont des registres tampons (tampon d'entrée/sortie , tampon d'UAL . . .) : T1 , T2 , E1 , E2 , RTIS , SIGN , POL , RI , RP2 , TM , S , A , B , C , D .

2) SRS : sortie d'un registre spécialisé dans un mot mémoire .

3) RAZ : remise à zéro de l'un des registres d'analyse de signature.

Certains registres , comme les registres de micro-instruction ou de micro-compteur ordinal , utilisés lors de l'exécution de toute instruction , ne peuvent être chargés par une instruction de test . De ce fait , un automate interne , décrit dans la partie II , a été inclu dans l'architecture de HSURF , afin de rendre possible entre autre le test de ces registres . Cet automate , spécialisé dans le test sera utilisé fréquemment au cours du test du processeur .

PARTIE 2

Le test de HSURF

1) Stratégie utilisée :

Le test de HSURF va se dérouler suivant la stratégie "start-small" :

- partitionnement du circuit en blocs .

- test de chaque bloc en utilisant, éventuellement , certains des blocs testés auparavant.

Ce type de test peut être utilisé en fin de fabrication, pour la validation des circuits. C'est un test "go - no go" , avec :

- arrêt du test dés qu'une défaillance se produit.
- ou étude d'une possible reconfiguration du circuit, si cela a été prévu lors de sa phase de conception .

2) Déroulement du test :

Le test va se dérouler en deux phases :

- Test sous contrôle d'un automate de test spécialisé, intégré au circuit

- Test utilisant les instructions normales et les instructions spéciales de HSURF.

L'automate spécialisé est chargé, lors du test, de le séquencer et de générer les commandes nécessaires à son exécution.

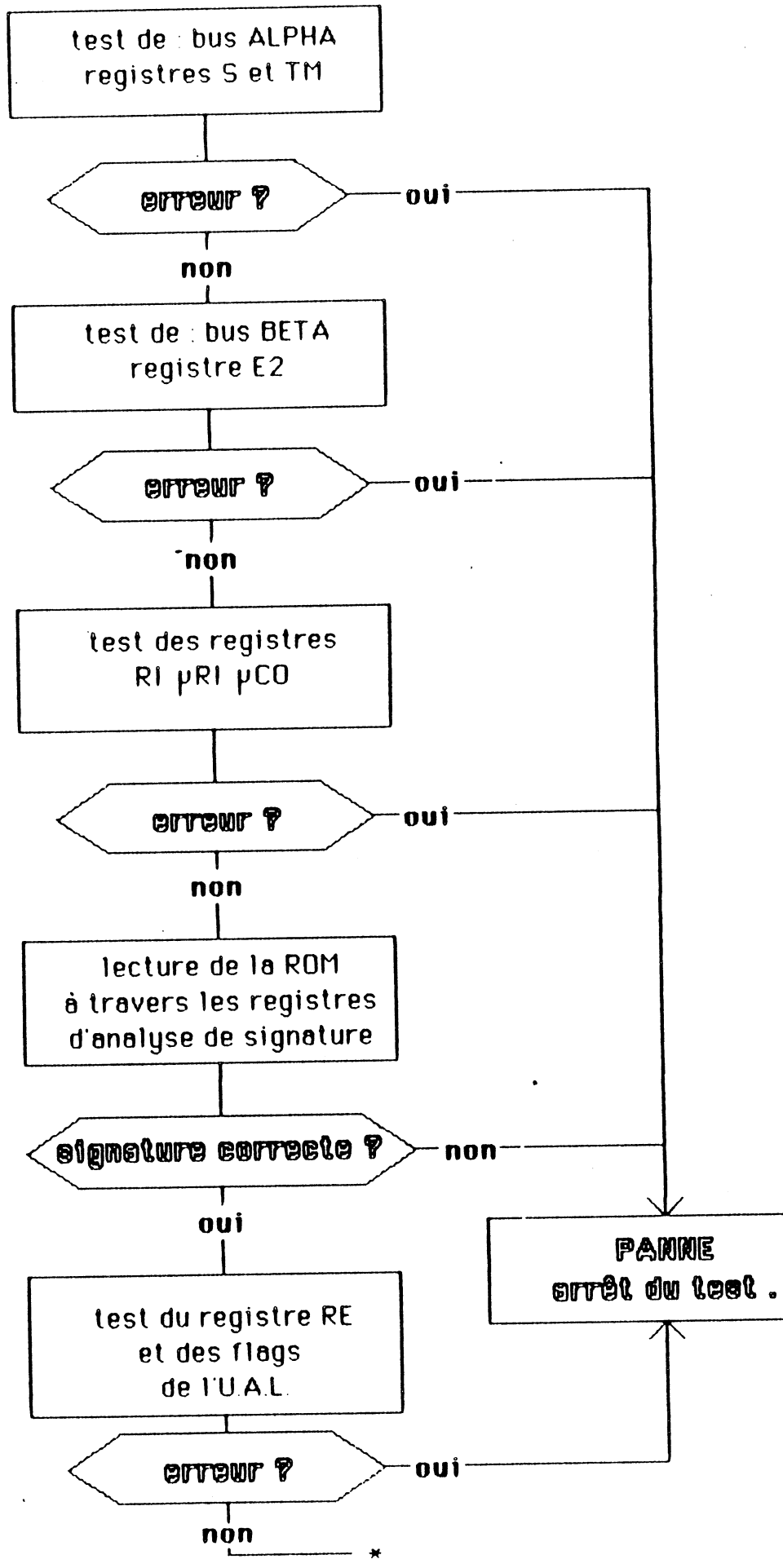
La figure 1 représente un organigramme du test de HSURF par cette méthode.

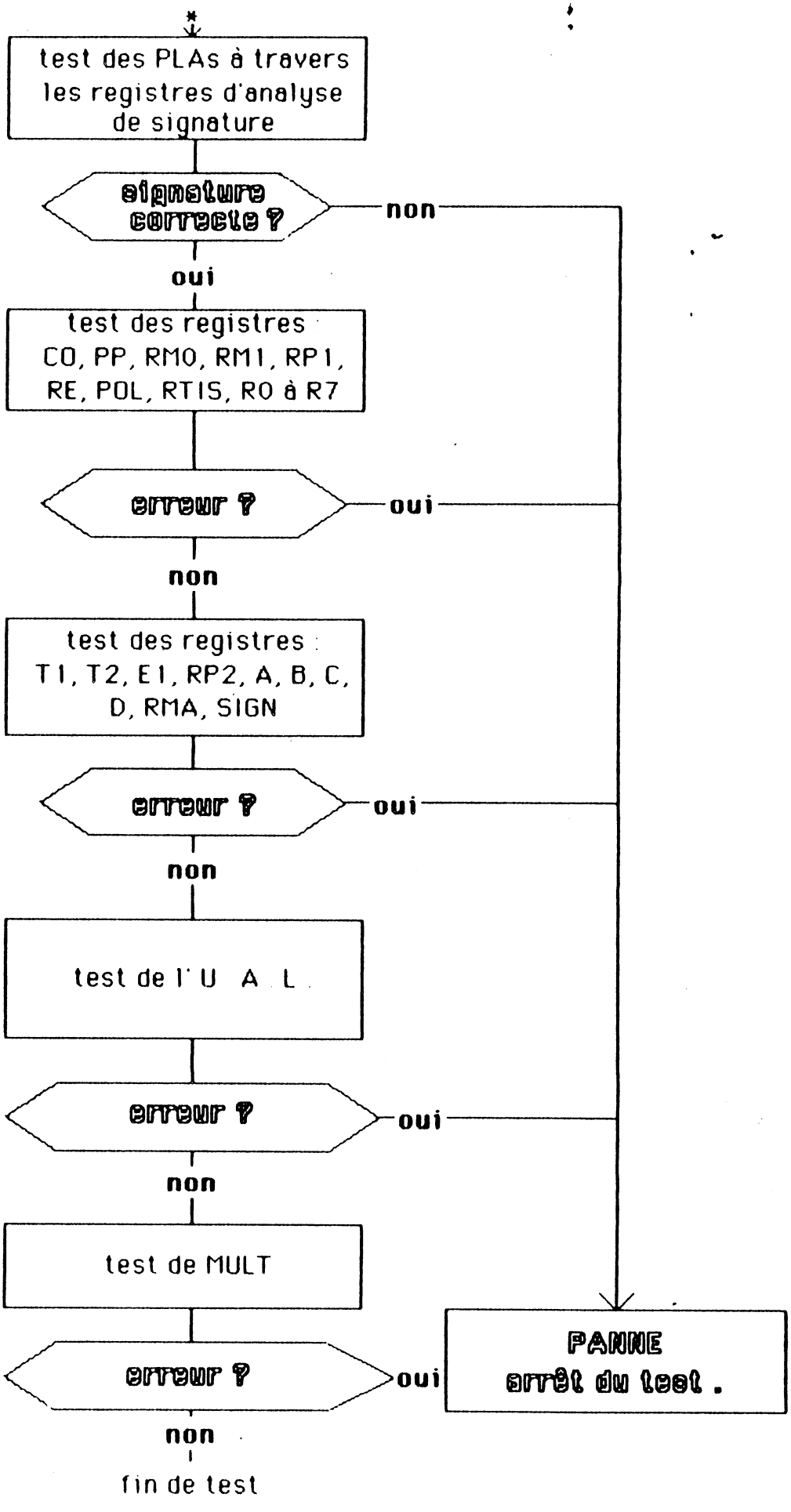
A- Tests sous contrôleur spécial :

a) bloc n° 1 : bus ALPHA,registres S et TM .

a-1 données de test

C'est un test, type test de mémoires vives qui est utilisé : une écriture de suite de "1" et de "0" dans les registres à tester, transitant par le bus ALPHA, détecte les pannes de ce bloc.


test de : bus ALPHA
registres S et TM
erreur ?
oui
non
test de : bus BETA
registre E2
erreur ?
oui
non
test des registres
RI μRI μCO
erreur ?
oui
non
lecture de la ROM
à travers les registres
d'analyse de signature
signature correcte ?
non
oui
test du registre RE
et des flags
de l'U.A.L.
erreur ?
oui
non
PANNE
arrêt du test .


test des PLAs à travers les registres d'analyse de signature
signature correcte ?
non
oui
test des registres CO, PP, RM0, RM1, RP1, RE, POL, RTIS, R0 à R7
erreur ?
oui
non
test des registres T1, T2, E1, RP2, A, B, C, D, RMA, SIGN
erreur ?
oui
non
test de l' U A L
erreur ?
oui
non
test de MULT
erreur ?
oui
non
fin de test
PANNE
arrêt du test .

a-2 chemin utilisé par les données de test :

Une donnée est lue sur le bus données externe, dans le registre TM. Cette valeur est transmise dans le registre S par le bus ALPHA, et est enfin recueillie sur le bus adresses externe. Ce test est répété avec autant de données que nécessaire pour détecter toute panne pouvant apparaître dans ce bloc .

a-3 contrôle du test :

Le contrôle de ce test est effectué par l'automate de test interne, décrit dans le troisième chapitre de cette partie.

b) bloc n° 2 : bus BETA, registre E2

b-1 : données de test

C'est, comme pour le bloc précédent, un test type test de mémoires vives, qui est utilisé pour tester ce registre et ce bus.

b-2 : chemin utilisé par les données de test

Une donnée est lue sur le bus données externe, dans le registre TM, qui a été testé précédemment. Cette valeur est ensuite écrite dans le registre E2, à travers le bus BETA, puis, dans le registre S, à travers le bus ALPHA, et est enfin recueillie sur le bus adresses externe. Comme précédemment, ce test est itéré autant de fois que nécessaire pour tester le registre et le bus.

b-3 : contrôle du test

C'est l'automate de test interne qui contrôle le déroulement des opérations.

c) bloc n° 3 : registre RI

c-1 : données de test

C'est un test type test de mémoires vives qui est utilisé pour valider ce registre.

c-2 : chemin utilisé par les données de test

Une donnée est lue du bus externe dans le registre d'entrée/sortie TM. Cette valeur est écrite dans RI à travers le bus ALPHA. Enfin, aprés avoir transité par le registre S, la donnée est recueillie sur le bus adresses externe.

c-3 : contrôle du test

Le registre RI étant utilisé pour décoder les instructions, il ne peut être chargé ou lu par une instruction particulière sans en modifier sa valeur. C'est donc sous contrôle de l'automate de test interne que ce test est effectué.

d) blocs n° 4 et 5 : registre micro-RI
registre micro-CO

d-1 : données de test

Ces deux registres sont testés comme de la mémoire vive. Ces deux blocs peuvent être testés l'un aprés l'autre dans un ordre indifférent, puisque étant indépendant l'un de l'autre.

d-2 : chemin utilisé par les données de test

Une donnée est lue du bus données externe dans le registre TM, puis écrite dans le registre sous test, via le bus ALPHA. L'analyse de signature est effectuée, l'opération est renouvellée autant de fois que nécessaire, puis le registre de signature est sorti sur le bus adresses externe pour le registre de micro-CO, ou en plusieurs fois sur les deux bus externes pour le registre de micro-RI.

d-3 : contrôle du test

L'automate de test dirige le test de ces deux registres, ainsi que le fonctionnement des registres d'analyse de signature.

e) bloc n° 6 : ROM et INCR

e-1 : données de test

La ROM est lue intégralement. Afin d'accélérer le test, on compacte les sorties à travers un registre d'analyse de signature.

e-2 : chemin utilisé par les données de test

Le registre de micro-CO est initialisé à 0, ainsi que les registres d'analyse de signature. La ROM est lue dans le regisre de micro-RI. Les registres micro-CO et micro-RI sont signés, le registre micro-CO est incrémenté, et l'opération est répétée, tant que le registre micro-CO n'est pas à nouveau nul. Puis les registres d'analyse de signature sont sortis succéssivement sur les bus externes.
Afin de permettre une reconfiguration de la ROM en cas de panne, et ainsi d'accroitre le rendement de la production, une sortie de chaque mot de la ROM sur les bus externes est envisagée. Ce genre de test, trop long, n'est envisagé qu'en cas de détection d'une anomalie par l'analyse de signature.

e-3 : contrôle du test

L'automate de test interne contrôle toutes les opérations, jusqu'à la sortie des registres d'analyse de signature.

B- Test utilisant les instructions générales et spéciales de HSURF

f) bloc n° 7 : registre RE
flags issus de l'UAL

f-1 : données de test

Un test type test de mémoires vives est effectué. Les vecteurs de test sont écrits et lus dans le registre RE et les flags de l'UAL par un programme de test.

f-2 : chemin utilisé par les données de test

Une donnée est lue sur le bus externe dans le registre TM. Elle est ensuite écrite dans RE et les flags de l'UAL, aprés avoir transité par le bus ALPHA. La lecture de ces données est ensuite effectuée à travers le bus ALPHA dans le registre S, puis sur le bus adresses externe.

f-3 : contrôle du test

C'est le contrôleur de HSURF qui gére l'exécution de ce programme de test.

g) blocs n° 8 à 10 : PLA A1
PLA A2
PLA BRANCH

g-1 : données de test

Des vecteurs de test sont calculés, d'aprés la structure interne des PLAs à tester. La méthode de calcul de ces vecteurs est donnée dans la partie III. Puis ces vecteurs, représentant un programme exécutable par le processeur, lui sont appliqués.

g-2 : chemin utilisé par les données de test

Le registre d'analyse de signature entre le registre micro-CO et la ROM est initialisé à 0. Puis le programme de test est exécuté, en signant le registre de micro-CO, afin d'obtenir un compactage des sorties du PLA que l'on veut tester, puis le registre d'analyse de signature est sorti sur les bus externes.

g-3 : contrôle du test

C'est le contrôleur du micro-processeur qui travaille ici. Il est utilisé en fonctionnement normal. Trois programmes sont succéssivement utilisés pour tester chacun des trois PLAs. Comme pour la ROM, en vue uniquement de reconfigurer un PLA si une panne est détectée, une lecture du registre d'analyse de signature sur le registre micro-CO, peut être effectuée aprés chaque instruction représentant un vecteur de test pour le PLA.

h) blocs n° 11 à 25 : registre CO
registre PP
registre RM0
registre RM1
registre RP1
registre POL
registre RTIS
registres R0 à R7

h-1 : données de test

C'est un test type test de mémoires vives qui est utilisé. Les vecteurs de test sont successivement écrits puis lus à l'aide des instructions classiques **LOAD** et **STORE**. Un programme de test peut ainsi être constitué.

h-2 : chemin utilisé par les données de test

C'est un programme de test classique qui est exécuté afin de tester l'ensemble des registres utilisateurs de la partie opérative.

h-3 : contrôle du test

Le contrôleur du processeur étant testé, il est utilisé afin de tester ces registres de la partie opérative.

<u>1) blocs n° 26 à 35 :</u> registre T1
registre T2
registre E1
registre RP2
registre A
registre B
registre C
registre D
registre RMA
registre SIGN

i-1 : données de test

Test type test de mémoires vives. Chaque vecteur est écrit puis lu, à l'aide des instructions spéciales de test : **LRS** et **SRS**.

i-2 : chemin utilisé par les données de test

C'est un programme de test qui est, ici encore, utilisé.

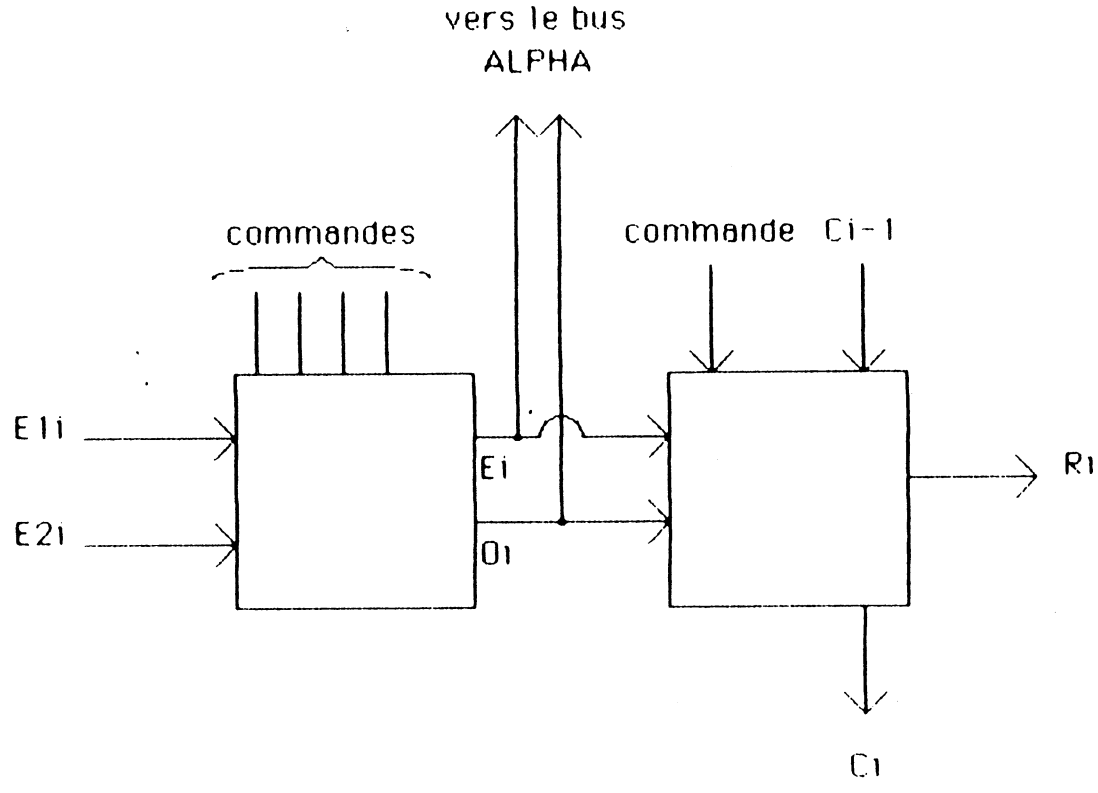
i-3 : contrôle du test

C'est donc le contrôleur de HSURF qui est chargé de l'exécution de ce test

j) bloc n° 36 : registre DM, Unité Arithmétique et Logique

j-1 : données de test

l'Unité Arithmétique et Logique est composée de 16 blocs traitant chacun, un bit des opérandes. Chaque bloc est lui-même scindé en deux morceaux. Les sorties de la première partie sont accessibles via le bus ALPHA.



Ceci permet de réaliser un test exhaustif de chaque bloc de l'UAL. Si l'on considérait un test exhaustif de chaque bit d'une UAL d'un seul bloc, le nombre de vecteurs de test serait : $2^8 = 256$ vecteurs par bit. Pour notre UAL, le nombre de vecteurs de test, dans le pire des cas (cas où l'on ne peut utiliser un vecteur du premier bloc pour tester le second bloc), est :

$$2^6 + 2^4 = 64 + 16 = 80 \text{ vecteurs.}$$

C'est un programme de test qui est écrit, afin d'amener les vecteurs de test jusqu'à l'UAL et lire les informations issues des deux blocs.

j-2 : chemin utilisé par les données de test

Les registres E1 et E2 sont chargés avec les vecteurs de test. L'opération est effectuée. Le bus ALPHA est signé, et l'on compare le résultat de cette signature en fin de test, à la signature précalculée.

j-3 : contrôle du test

C'est le contrôleur de HSURF qui gére ce test.

k) bloc n° 37 : Multiplieur (MULT)

k-1 : données de test

C'est un test exhaustif de chaque cellule du multiplieur qui est réalisé. La méthode est expliquée dans la partie IV.

k-2 : chemin utilisé par les données de test

Les vecteurs nécessaires au test de l'opérateur sont amenés dans les registres A, B, C, D, testés précédemment, grace aux instructions spéciales de test. Le résultat de l'opération est ensuite compactée à travers le registre d'analyse de signature, qui en fin de test, sera sorti sur les bus externes pour être comparé avec une valeur précalculée.

k-3 : contrôle du test

C'est le contrôleur de HSURF qui dirige le test puisqu'un programme de test peut être constitué, à partir des vecteurs de test du multiplieur.

L'ensemble de ces test doit être contrôlé de l'extérieur par un outil de test intelligent, capable de générer les signaux nécessaires au test et d'attendre les résultats afin de les comparer à ceux espérés.

L'ensemble des signaux à générer est :

-Le signal TEST qui déclenche au besoin l'automate de test interne.

-L'ensemble des données et des adresses représentant les programmes de test à appliquer aux divers blocs du circuit.

3) L'automate de test :

a) Définition :

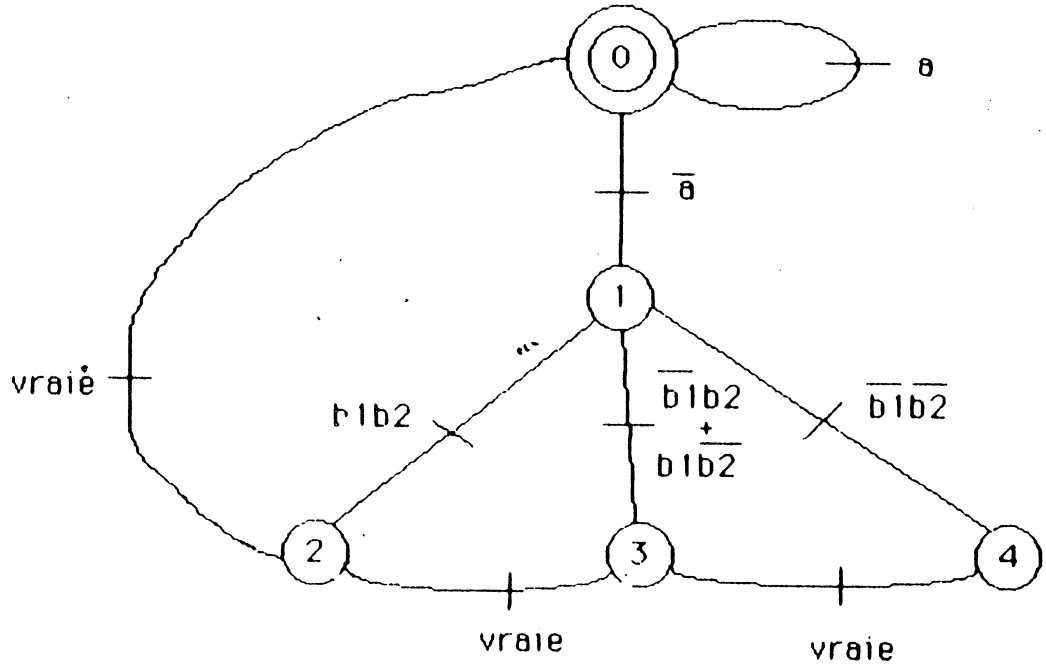
Un automate d'état fini est défini par un ensemble d'états et de transitions conditionnelles entre ces états. A chaque état est associé un ensemble de commandes qui peut être vide. A chaque transition est associée une condition de transition qui, si elle n'est pas satisfaite, bloque l'automate dans l'état précédent.

Un automate d'état fini peut être représenté à l'aide de symboles représentant les états et les transitions :

-Chaque état est représenté par un cercle contenant le numéro de l'état symbolisé.

-Les états sont reliés entre eux par des arcs coupés par des traits représentant les transitions.

-L'un des états, appelé état initial, est symbolisé par un cercle double. C'est dans cet état que l'automate est initialisé à sa mise en route.



ensemble des commandes pour chaque état

état 0

état 1 bus externe --> R1

état 2 R1 --> bus externe

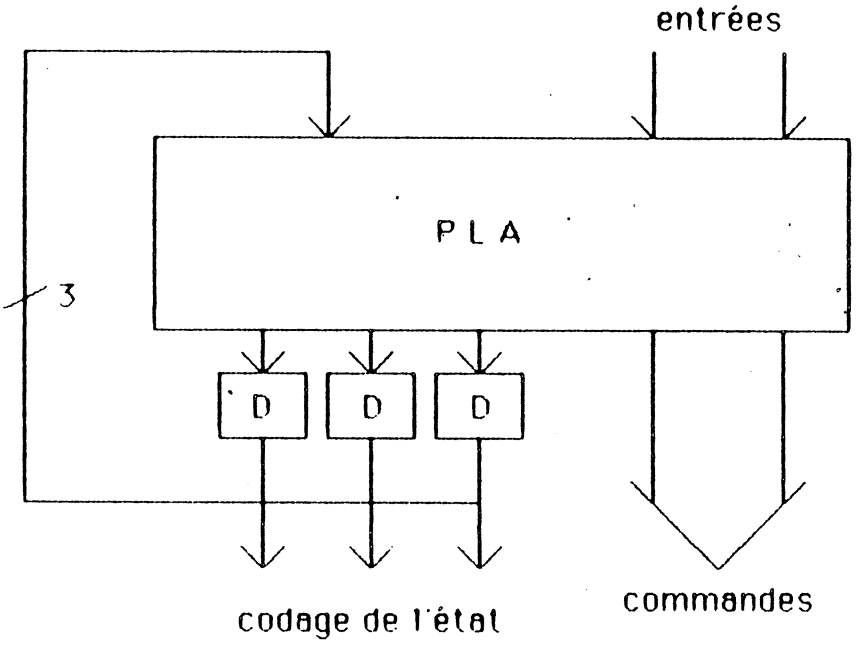
état 3 R1 + 1 --> R1

état 4 R1 * R1 --> R1

Remarque : la condition VRAIE étant toujours réalisée, la transition est franchie chaque fois que l'automate est dans l'un des états qui la précède (ici les états 2, 3 ou 4).

b) Réalisation d'un automate :

Un automate peut être réalisé à l'aide d'un PLA générant les commandes de l'état dans lequel se trouve l'automate, et calculant l'état suivant de l'automate. L'état en cours de l'automate est mémorisé par un ensemble de bascules :



c) L'automate de test de HSURF :

Les figures 2 à 6 représentent le dessin de l'automate interne de test du microprocesseur HSURF.

Le PLA de cet automate a été rendu autotest afin de détecter toute panne pouvant intervenir dans son fonctionnement. Il est composé de:

- 11 entrées :
  - -Les sorties des 6 bascules D mémorisant l'état courrant de l'automate.
  - -Le signal TEST déclenchant et séquençant les divers tests possibles.
  - -Les 3 bits de poids faibles du bus données externe,

utilisés pour choisir le type de test que l'on veut réaliser.

- Le signal micro-CO=0 utilisé pour séquencer la lecture de la ROM.

- 31 sorties

- 6 commandes des bascules D afin de faire passer l'automate à l'état suivant.

- 22 commandes des diverses opérations à réaliser.

- 3 bits de code :

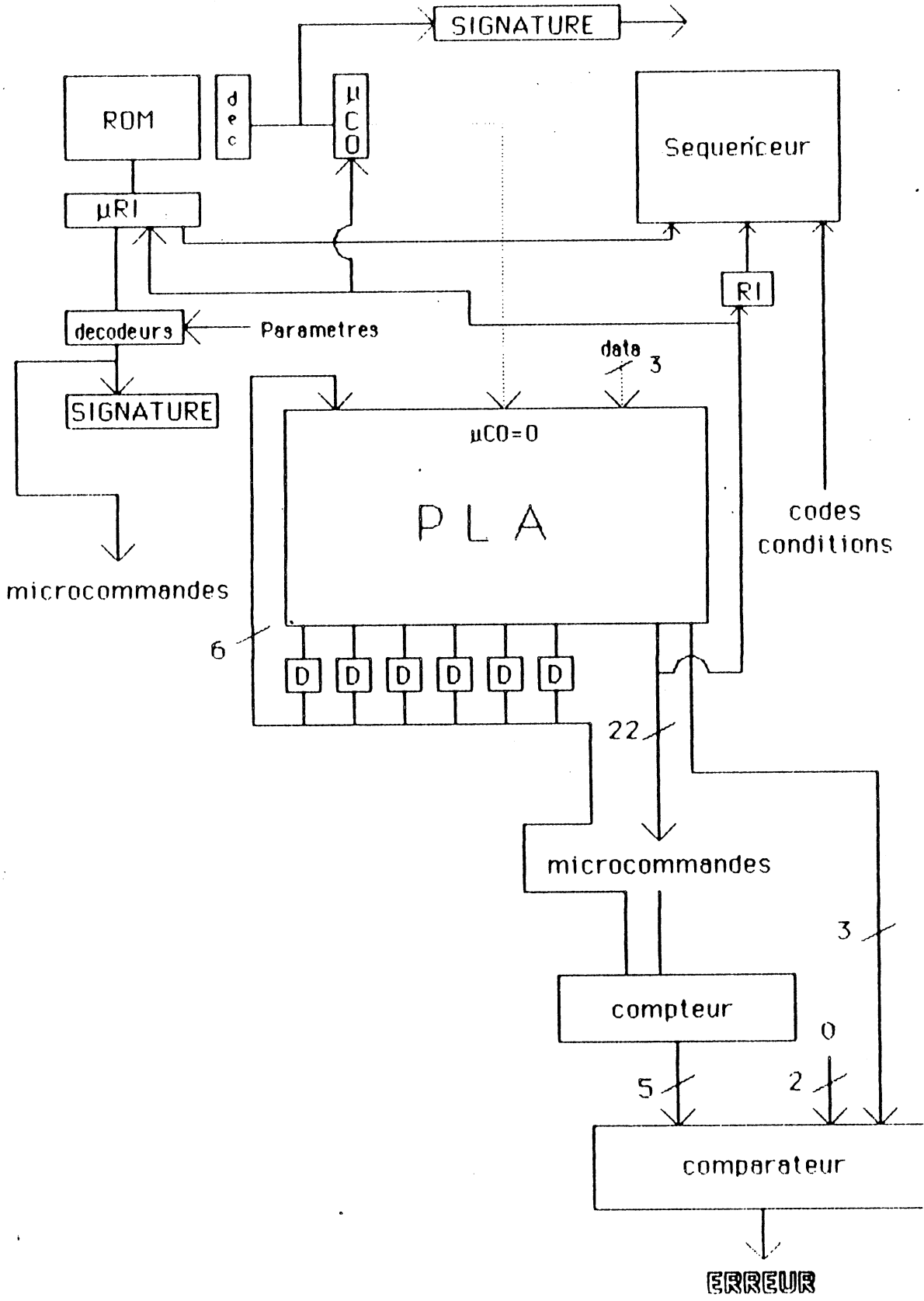
Le code choisi est un code de Berger modifié (MAK 82). Il indique pour tout vecteur d'entrée, le nombre de sorties à 1. Ce nombre étant toujours inférieur à 8, seuls trois bits sont nécessaires pour réaliser ce codage. En outre, le codage a été réalisé de manière à ce que l'apparition d'un état inconnu pour l'automate (bascules D mémorisant un état supérieur à l'état maximum de l'automate : 34), crée aussitôt une panne sur les sorties.

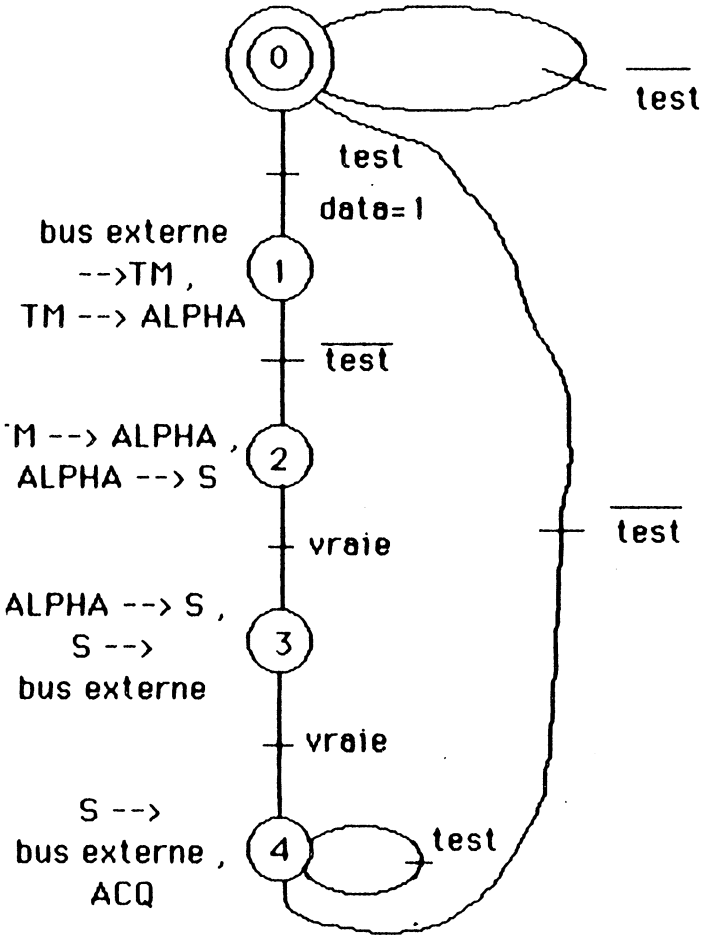
- 112 monomes

La description des monomes et des fonctions est donnée en annexe. La place occupée sur le circuit par le PLA de l'automate est donc:

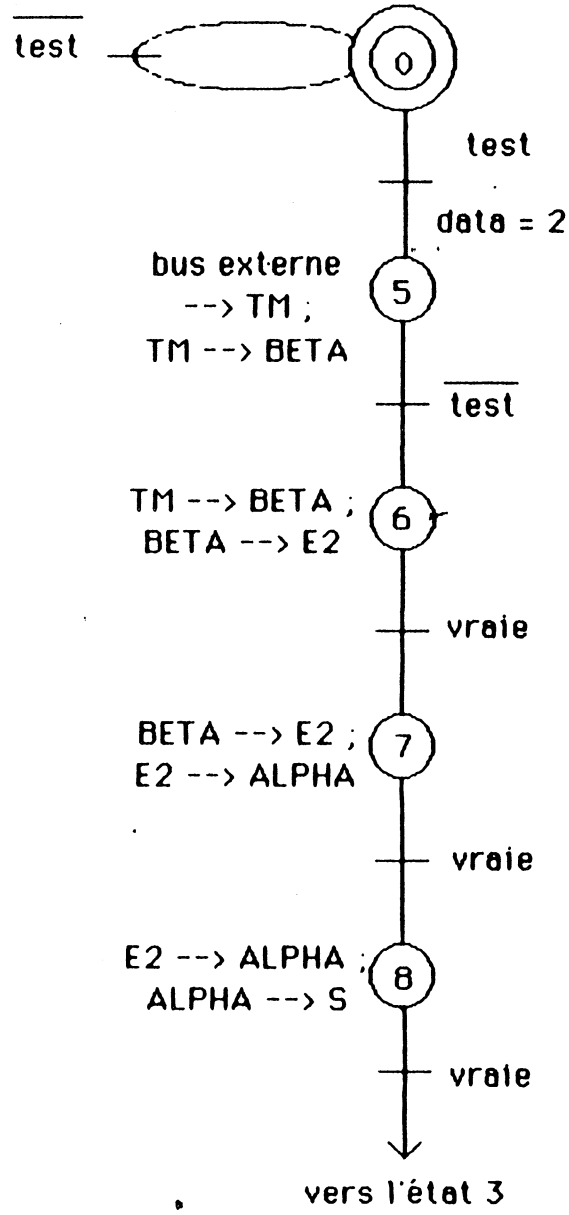
$$112 \times (31 + 11) = 4704 \text{ emplacements de transistors.}$$

Intégration de l'automate de test au microprocesseur HSURF:


SIGNATURE
ROM
dec
µCO
Sequenceur
µRI
RI
decodeurs
Parametres
data
3
SIGNATURE
µCO=0
PLA
codes
conditions
microcommandes
6
D
D
D
D
D
D
22
microcommandes
3
compteur
0
5
2
comparateur
ERREUR

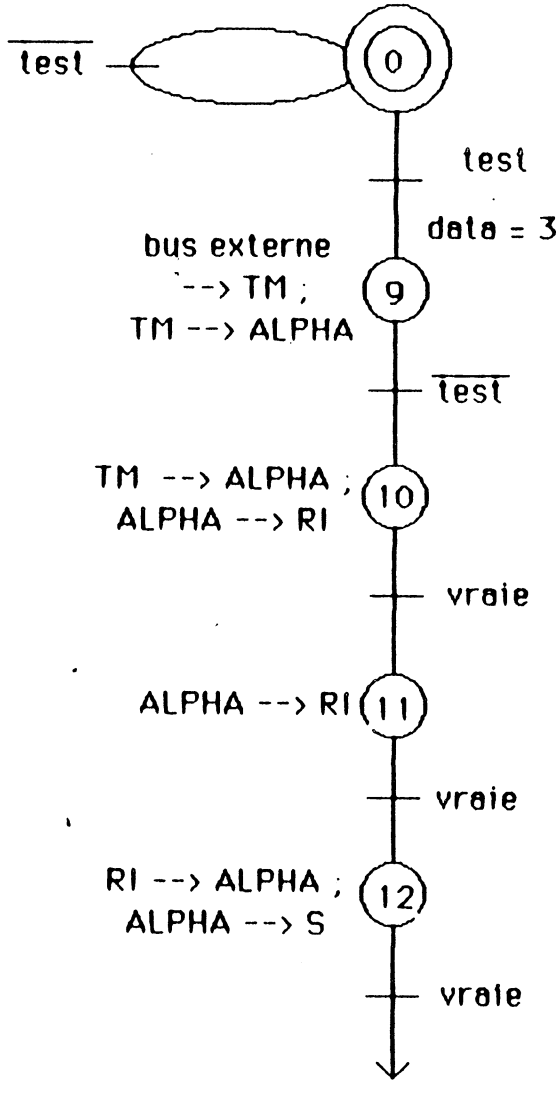


test du bloc n° 1

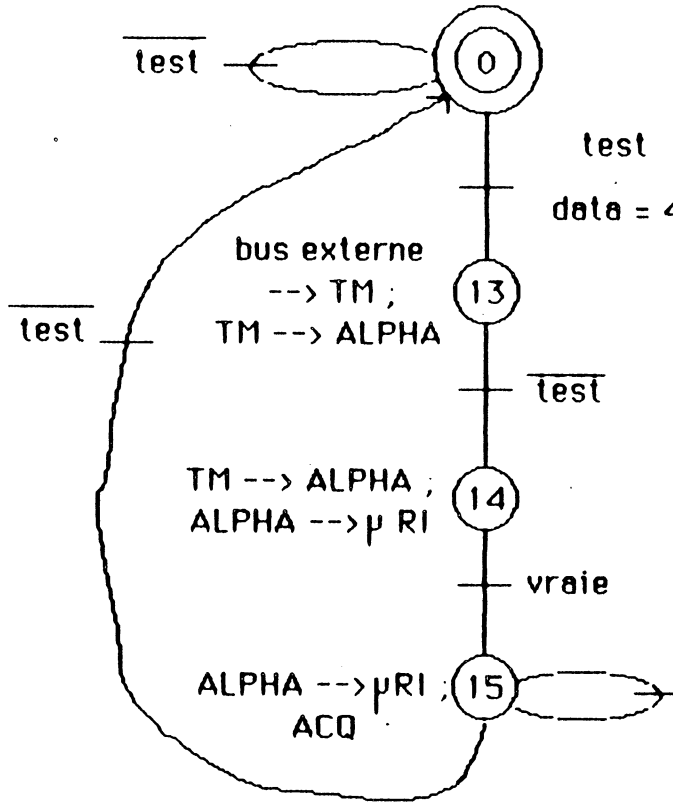


test du bloc n° 2

figure 2



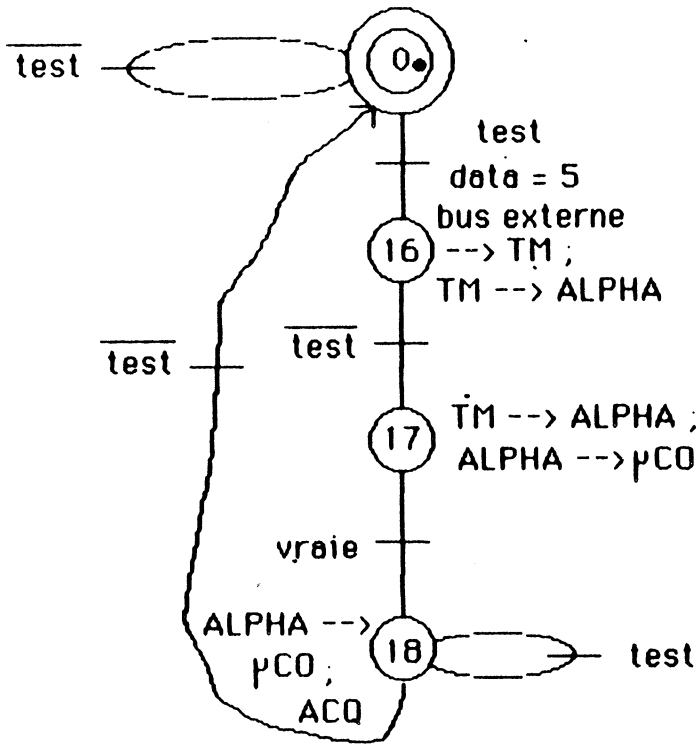
test du bloc n° 3



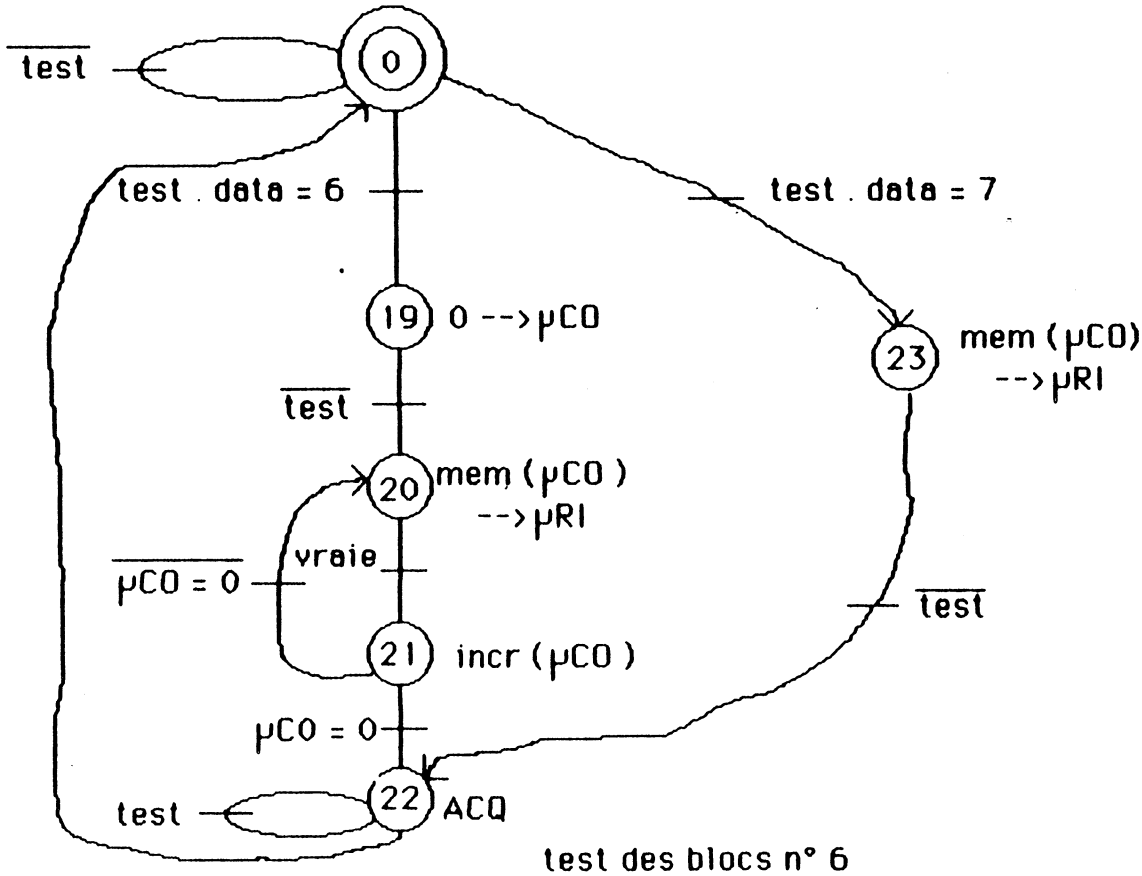
test du bloc n° 4

Figure 3

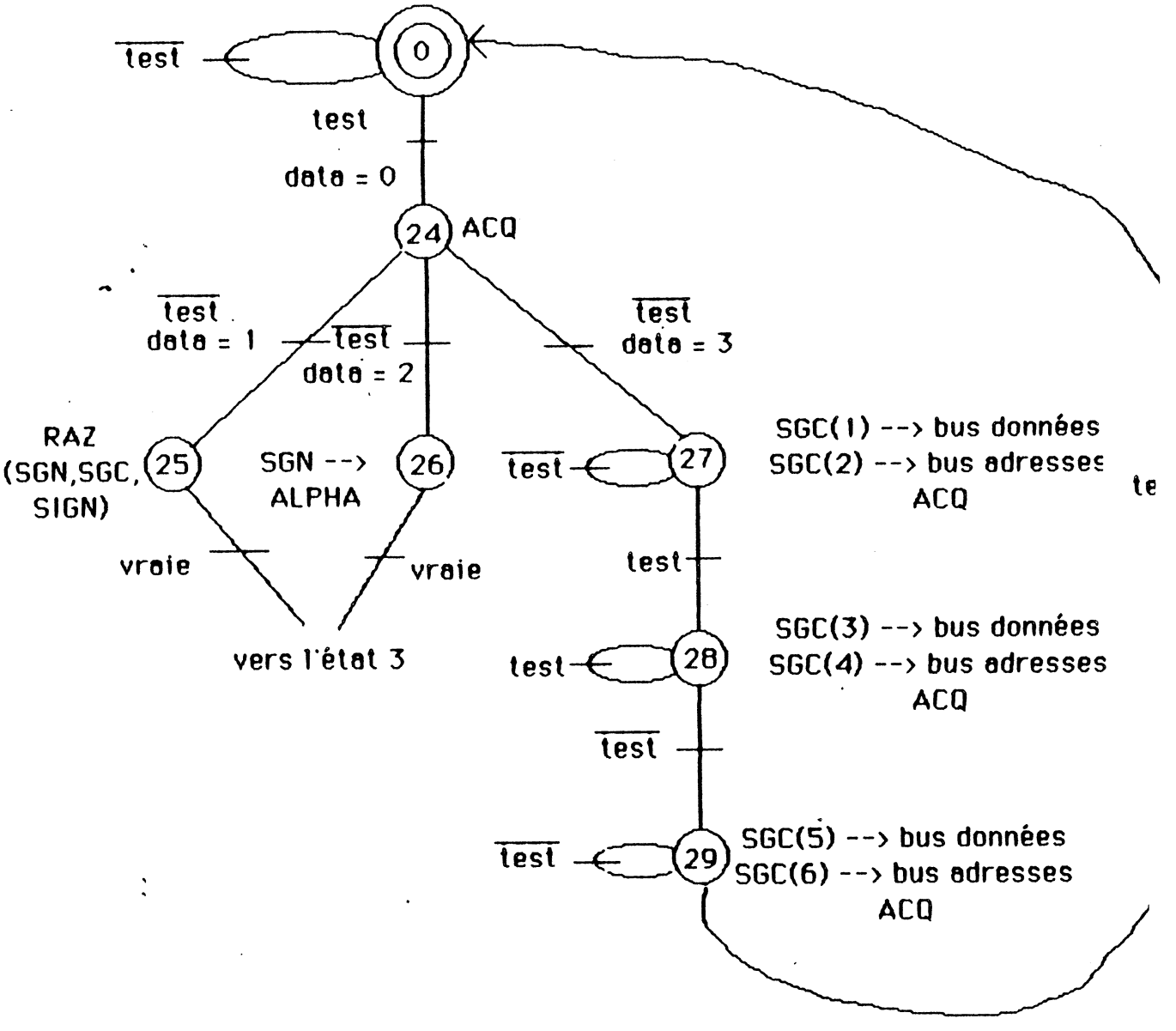
Figure 4



test du bloc n ° 5

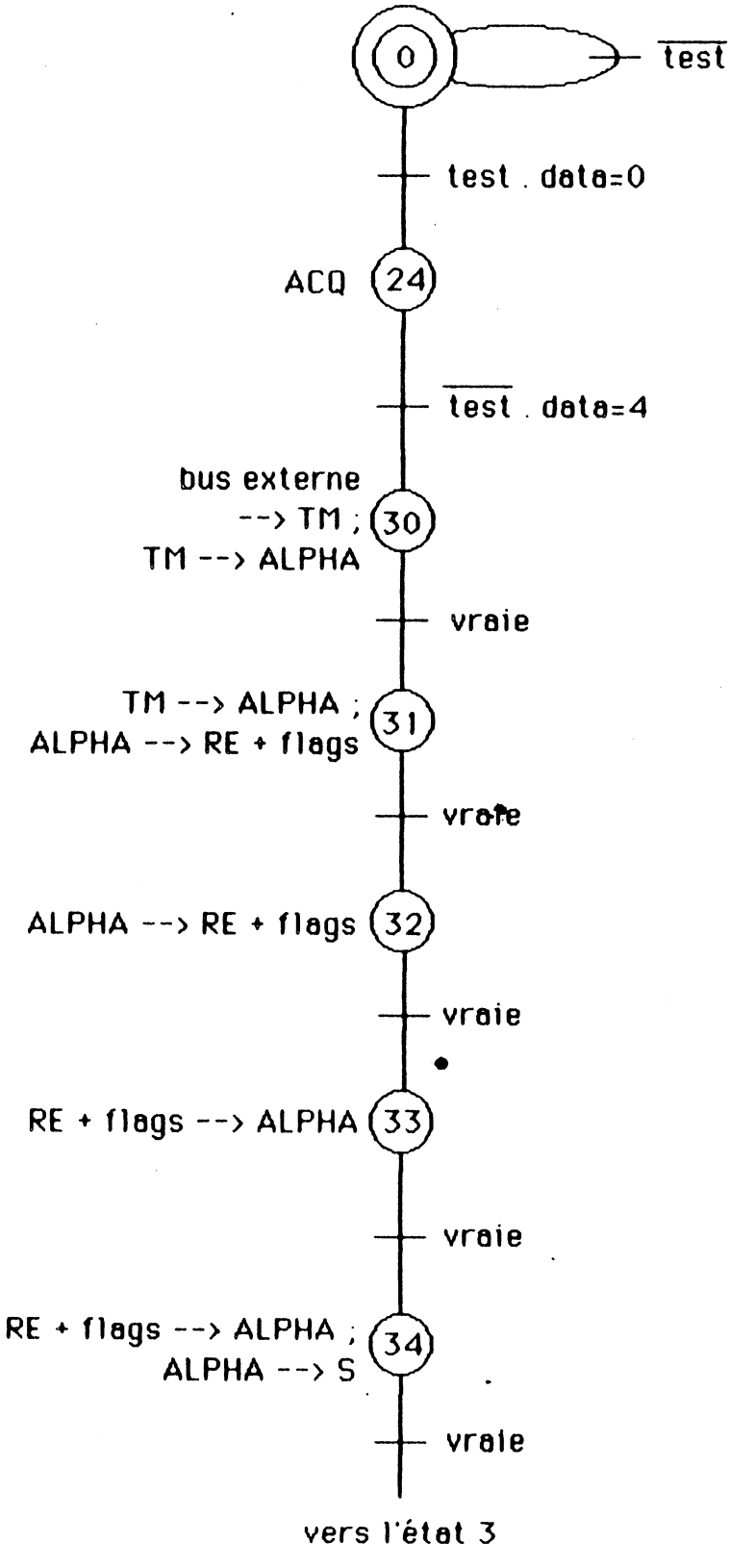


test des blocs n° 6



lecture des registres
d'analyse de
signature

Figure 5



test du bloc n° 7

Figure 6

PARTIE 3

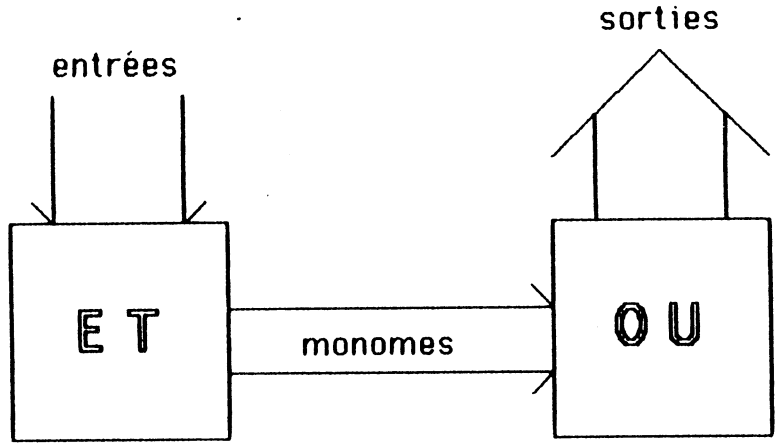
Le test des PLAs

I - Introduction

Un "Programmable Logic Array" (PLA), est un circuit combinatoire réalisant un ensemble de fonctions booléennes, à partir de ses variables d'entrée. Toute fonction logique (booléenne), pouvant s'écrire sous la forme d'une somme de produits de variables, un PLA est constitué de deux étages :

-L'étage "ET", qui fournit, à l'étage "OU", des produits de variable d'entrée (appellés **monomes**)

- L'étage "OU", réalisant les fonctions de sortie du PLA, par somme d'un ou de plusieurs monomes de l'étage "ET".



Les étages "ET" et "OU" sont réalisé à l'aide de portes logiques. Il existe des PLAs fabriqués à l'aide de portes :

- AND pour l'étage "ET", et OR pour l'étage "OU"

- NAND pour les deux étages

- NOR pour les deux étages

Le plus souvent, ce sont des PLAs de cette dernière catégorie qui sont utilisés. On les appelle des PLAs NOR/NOR.

Les PLAs constituent la partie la plus importante des unites de contrôle des microprocesseurs. Par exemple le 68000 possede:

- trois PLAs de decodage des instructions (selection des microprogrammes de traitement des instructions)

- un PLA de parametrage (selection des registres . . .)

- un PLA de branchement (deroutement suivant condition dans les microprogrammes)

La détection des pannes dans un PLA est donc primordiale lors du test de ces circuits . Différentes approches ont été développées , selon que le problème du test est pris en considération avant ou aprés la conception du circuit.

a) Modification de PLA à des fins de test :

Deux stratégies ont été étudiées:

- Modification du PLA pour le rendre autotest. **(MAK 82) , (DON 82)**

Dans **(MAK 82)** G.P.Mak indique comment rendre un PLA autotest par rajout d'un certain nombre de fonctions réalisant un codage sur les sorties du PLA . Un analyseur de ce code **(MAR 78)** indique , à tout moment , si les sorties du PLA sont correctes ou erronnées (fig 1).

- Modification du PLA afin de le rendre facilement testable. **(FUJ 81) , (YAJ 81) , (SAL 81)**

Des PLAs , augmentés d'un minimum de portes logiques , permettent de minimiser le nombre de vecteurs à appliquer sur leurs entrées , pour detecter un ensemble de pannes prédéterminées . L'un des avantages de cette methode est le fait que les vecteurs de test sont indépendants de la structure du PLA (monomes , fonctions de sortie)

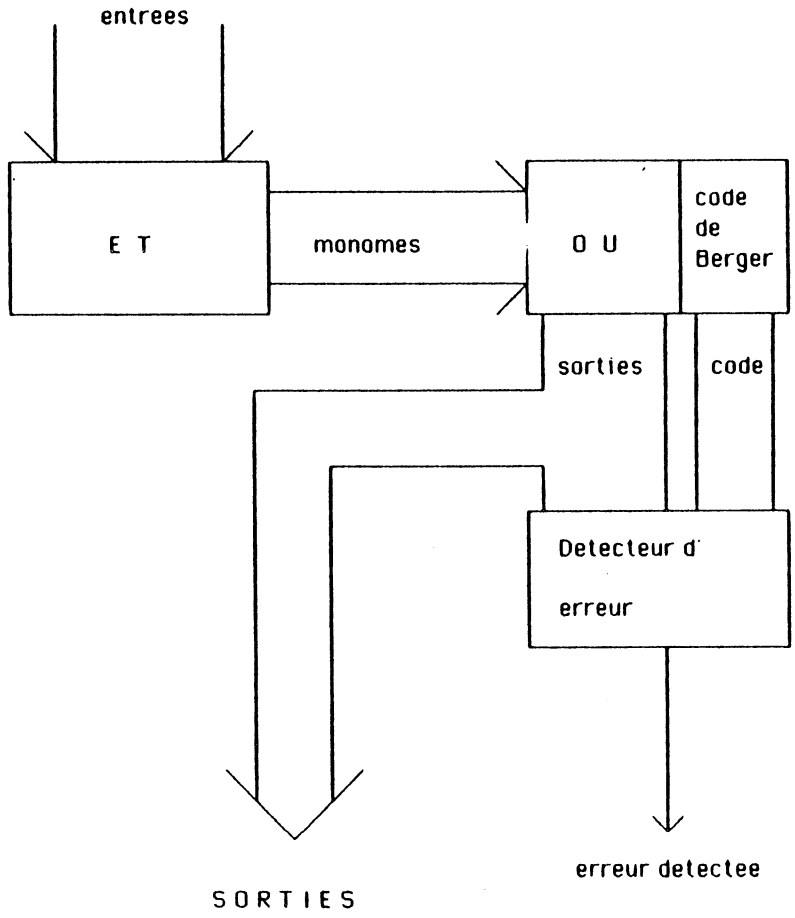
b) Methode de test pour PLA non modifié à des fins de test

La stratégie adoptée est :

- amener sur les entrées du PLA un ensemble de

Figure 1

Exemple de PLA autotest



vecteurs de test déterminés à partir de sa structure interne ,

- observer les sorties du PLA .

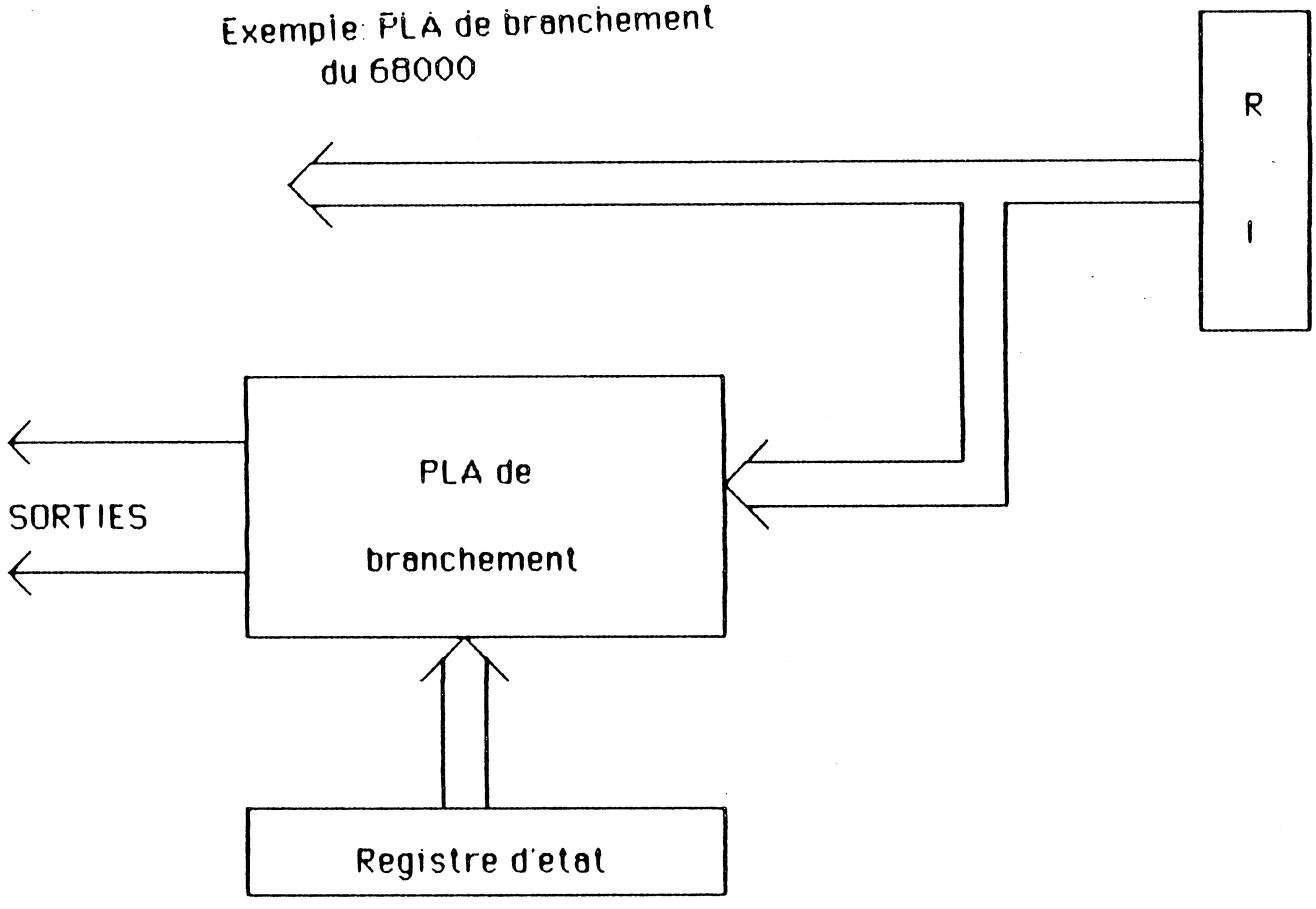
Des methodes de génération de vecteurs de test ont été proposées par exemple par **(SMI 79) , (OST 78) ,(EIC 80) , (SOM 83)** . Nous nous interessons ici au test de PLA déja conçus, sans dispositifs d'autotest, et illustrerons notre étude sur ceux du contrôleur du

microprocesseur MC68000. Nous nous interessons, en particulier, au test de PLAs intégrés dans leur contexte, donc soumis à des contraintes d'environnement.

II - Problèmes liés à l'environnement du PLA testé

a) Contraintes sur les entrées :

En règle générale , les PLAs à tester font partie intégrante de circuits plus importants . Leurs entrées ne sont pas toujours accessibles par les plots externes d'entrées/sorties de ces circuits .



Considérons, par exemple,le PLA de branchement du 68000, qui sert à générer éventuellement , deux bits d'adresse de micro-instruction , afin de choisir entre deux branches de micro-programmes à exécuter . Les entrées de ce PLA ne sont pas directement accessibles , puisque certaines sont connectées au registre d'état de la partie opérative . Le test de ce PLA passera donc par une initialisation du registre d'état , puis de l'exécution d'une instruction faisant intervenir le PLA de branchement dans son séquencement.

b) Propagation des sorties vers l'exterieur :

De même que pour les entrées , les sorties des PLAs ne sont pas toujours observables directement sur les plots externes d'entrées/sorties du circuit . Elles peuvent même être totalement masquées dans certains cas : la figure 2 schématise le fonctionnement du séquenceur du 68000 . L'instruction contenue dans le registre RI fournie les entrées des PLAs de décodage : A1 , A2 et A3 . Dans le cas d'une instruction valide (code opération reconnu) , les sorties d'un seul de ces PLAs sont selectionnées et donnent l'adresse de la première micro-instruction à executer . Dans le cas contraire - instruction inexistante - , l'adresse d'un micro-programme traitant ce cas particulier , sera fournie par un PLA d'exception . La sélection de l'un des PLAs A1 , A2 ou A3 est fonction d'un champ de la dernière micro-instruction executée . Ce champ permet de choisir une nouvelle micro-adresse soit en sortie d'un PLA , soit dans la micro-instruction elle-même .

L'adresse de la première séquence de micro-instruction traitant l'instruction contenue dans le registre RI est donnée par le PLA A1 . La dernière micro-instruction de cette séquence permet de sélectionner , suivant le cas , l'un des PLAs A2 ou A3 pour obtenir l'adresse d'une nouvelle séquence , complétant le traitement effectué précedemment . Dans tous les cas , la fin du traitement de l'instruction renvoie à l'adresse générée par le PLA A1 (fetch) . Quatre types de séquencement apparaissent lors de l'execution du jeu d'instructions du 68000 (fig. 3) .

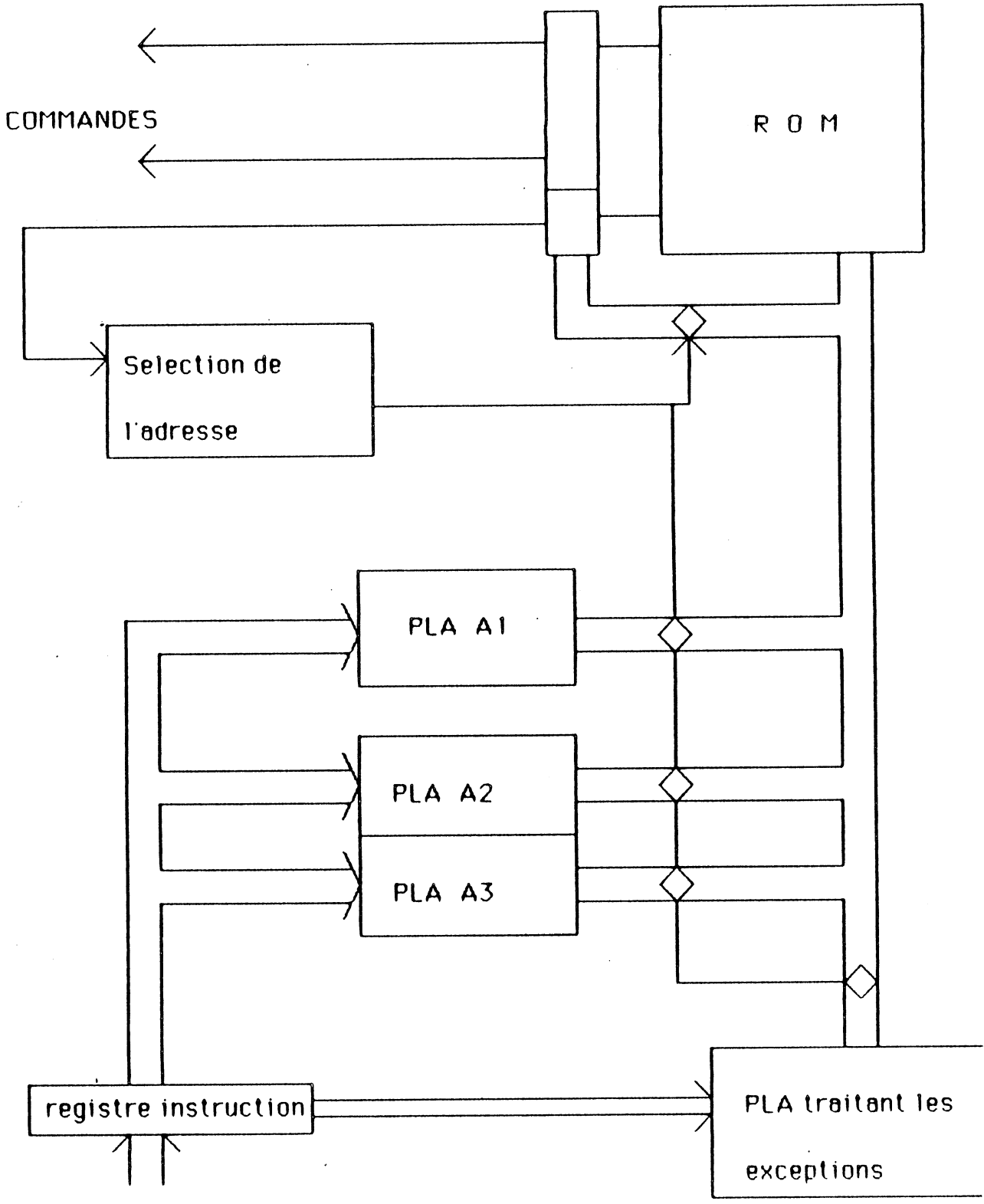
On s'aperçoit que les sorties des PLAs A2 et A3 ne sont pas toujours utilisées , suivant le type d'instruction effectuée .

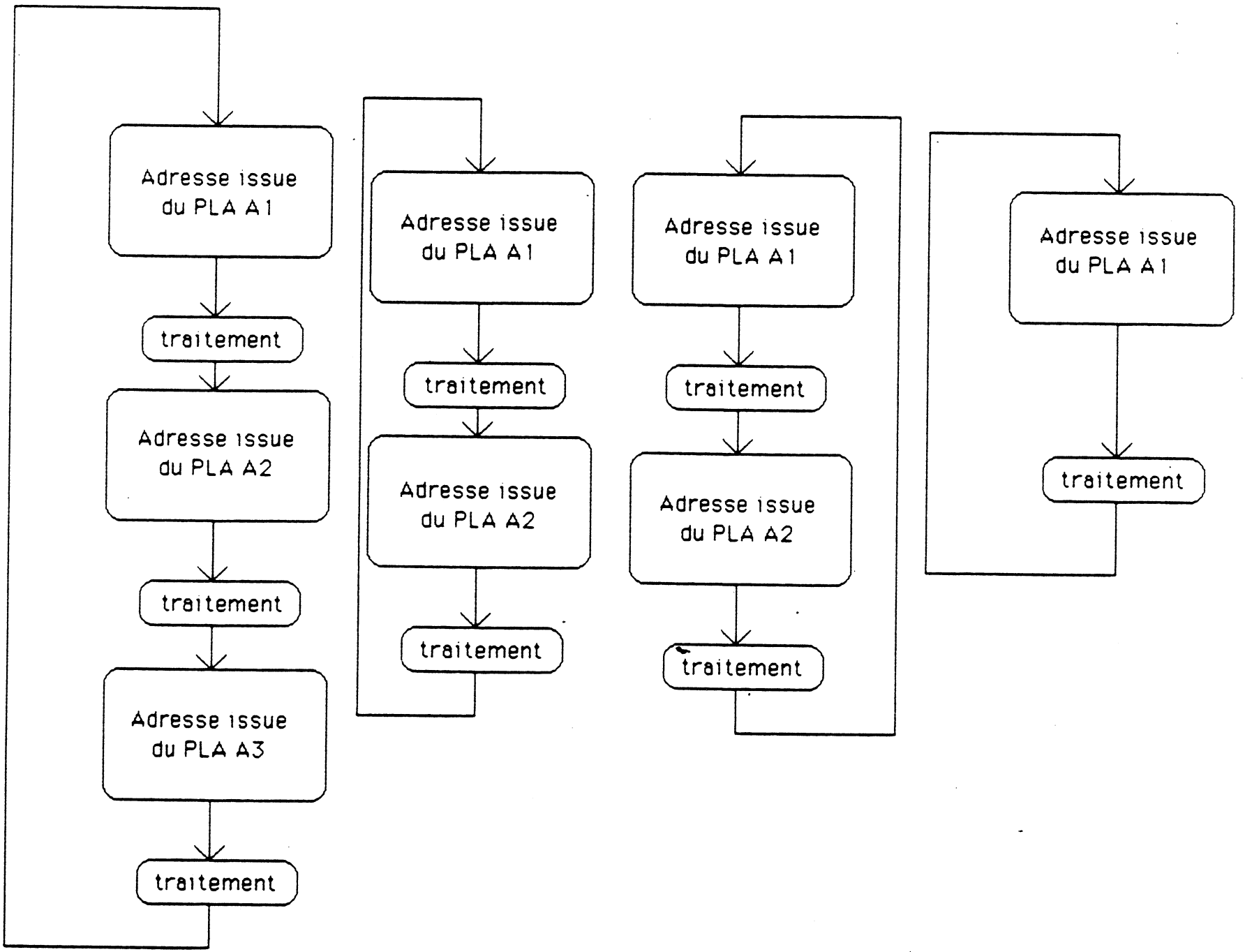
Exemple :

Lors de l'execution des instructions dont le séquencement est du type I ou III , les sorties du PLA A2 ne seront jamais utilisées . Pour certaines valeurs du registre RI , les sorties de ce PLA sont donc totalement inobservables.

Figure 2

Sequenceur du 68000




Adresse issue du PLA A1
traitement
Adresse issue du PLA A2
traitement
Adresse issue du PLA A3
traitement
Adresse issue du PLA A1
traitement
Adresse issue du PLA A2
traitement
Adresse issue du PLA A1
traitement
Adresse issue du PLA A2
traitement
Adresse issue du PLA A1
traitement

En outre , même dans le cas ou les sorties d'un PLA sont sélectionnées , celles-ci fournissent une adresse de la ROM , qui elle même transmet un ensemble de commandes vers la partie opérative . Les sorties des PLAs A1, A2 et A3 sont donc indirectement observables , dans ce cas , puisque seule, une étude de la partie opérative fournira une indication sur ces sorties.

c) Solution envisagée :

Ces problèmes peuvent être partiellement ou totalement résolus , en choisissant les vecteurs de test , parmi un sous-ensemble restreint de vecteurs . Ce sous-ensemble sera choisi afin de permettre un accés relativement aisé aux entrées et aux sorties du PLA .

Exemple :

Pour le test des PLAs de décodage du 68000 , quatre sous-ensembles peuvent être créés , chacun contenant l'ensemble des instructions des types I à IV . La sélection d'un de ces PLAs n'est effectué que pour certaines instructions : le PLA A2 n'est selectionné que pour les instructions des types II et IV . L'union de ces deux sous-ensembles constituera un domaine d'entrée pour le PLA A2 , parmi lequel , on pourra choisir des vecteurs de test arrivant sur ses entrées et sélectionnant ses sorties . Dans ce cas précis , le problème de génération des entrées et de sélection des sorties est résolu . Par contre , la propagation des sorties jusqu'aux plots d'entrées/sorties est plus difficile . Dans la pratique , on prendra l'hypothèse que si une panne apparaît en sortie du PLA choisi , celle-ci fournissant une adresse de micro-programme erronnée , modifiera notablement l'execution de l'instruction en cours et qu'elle sera observable directement ou indirectement sur les sorties externes du circuit .

En théorie , une étude approfondie des pannes et de leurs conséquences , permettrait de réduire encore le nombre des vecteurs du domaine d'entrée , afin de ne garder que ceux qui en cas de panne , rendraient l'observation des sorties du PLA aisée .

Pour des PLAs n'ayant pas de tels problèmes , le domaine d'entrée peut être élargi à l'ensemble exhaustif des vecteurs d'entrée possibles .

III - Méthode de génération de vecteurs de test :

La méthode choisie a donc été : le calcul de vecteurs de test à partir d'une description structurelle et d'un ensemble de vecteurs autorisés - le domaine d'entrée - permettant de détecter les pannes pouvant survenir

dans un PLA . Ces pannes sont **(OST 78) (LOT 85)**:

- les collages aux valeurs logiques "0" ou "1"

- les courts-circuits

- les défauts d'élément

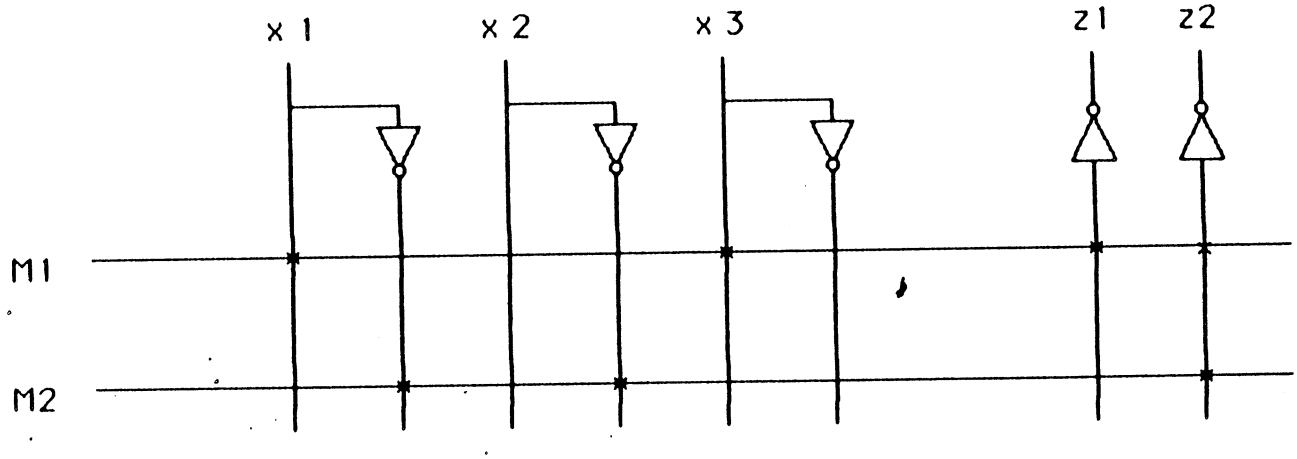
- les "stuck-open" pour les PLAs en technologie CMOS
(RED 84) , (CHA 83) , (CHI 83)

a) Definitions :

* On raisonne sur un PLA NOR/NOR (fig. 4) , mais un raisonnement similaire aurait pu intervenir pour un PLA d'un autre type (NAND/NAND , ET/OU . . .)

* Chaque ligne du PLA correspond à un monome booléen qui apparaît dans les diverses sorties auxquelles il est connecté .

Exemple :



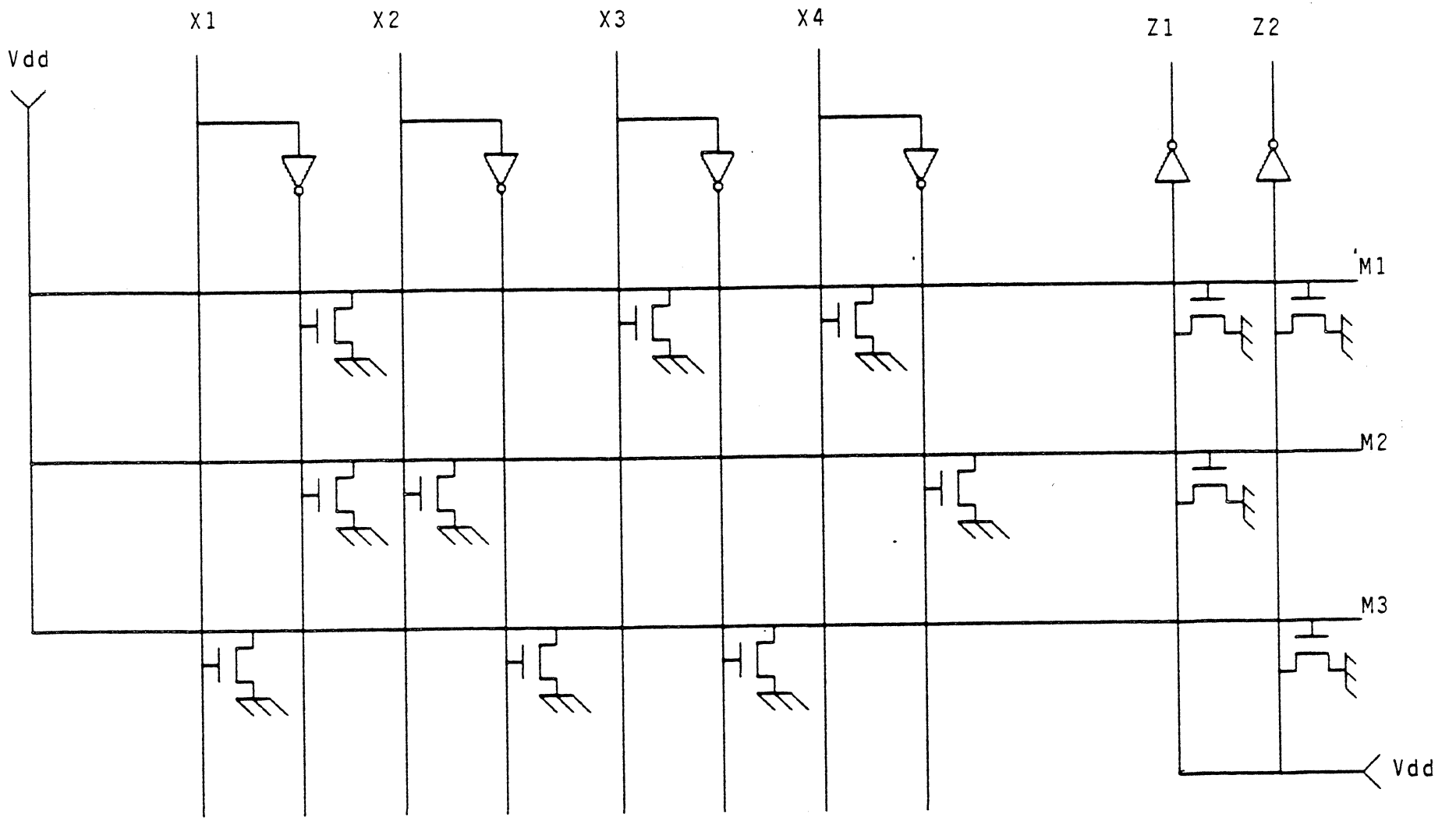
Aux lignes M1 et M2 correspondent les monomes :

$M1 = X_1 \, X_3$

$M2 = \overline{X_1} \, \overline{X_2}$

FIGURE 4

Exemple de PLA NOR/NOR



* On appelle monome adjacent à un monome M_i , tout monome qui ne diffère de M_i que par une variable présente dans l'un, et présente sous sa forme complémentée dans l'autre :

Exemple :

L'ensemble des monomes adjacents au monome $\overline{X_1}\, X_2\, X_4$ est :

- $X_1\, X_2\, X_4$ (adjacent par rapport à la variable X_1)
- $\overline{X_1}\ \overline{X_2}\, X_4$ (adjacent par rapport à la variable X_2)
- $\overline{X_1}\ X_2\ \overline{X_4}$ (adjacent par rapport à la variable X_4)

Le teme "adjacent" a été choisi par analogie avec ce qui se passe lorsque l'on représente les monomes sur un tableau de Karnaugh :

Par exemple : soit le monome $M = X_1 . X_2 . \overline{X_3} . X_4$ sa représentation sur un tel tableau sera :

X1X2 \ X3X4	00	01	11	10
00				
01				
11		M		
10				

L'ensemble de ses monomes adjacents sera :

$$M_1 = \overline{X_1} . X_2 . \overline{X_3} . X_4$$

$$M_2 = X_1 . \overline{X_2} . \overline{X_3} . X_4$$

$$M_3 = X_1 . X_2 . X_3 . X_4$$

$$M_4 = X_1 . X_2 . \overline{X_3} . \overline{X_4}$$

Sur un tableau de Karnaugh, l'ensemble de leurs représentation serait:

X1X2 \ X3X4	00	01	11	10
00				
01		M1		
11	M4	M	M3	
10		M2		

Ces quatre monomes sont bien représentés dans les cases adjacentes à la case représentant le monome M.

* Un monome M_i est <u>couvert</u> par un monome M_j si :

$M_j = M_1 . M_x$ (M_x étant un monome booléen quelconque , éventuellement nul)

* Deux monomes M_i et M_j sont <u>compatibles</u> si , pour chacune des variables d'entrée :

$M_i = M_j$

ou

M_i n'apparaît pas

ou

M_j = n'apparaît pas

Exemple :

Soient les trois monomes suivants :

$$M_1 = X_1 X_3 \overline{X_4}$$

$$M_2 = X_2 X_3$$

$$M_3 = \overline{X_1} X_2 \overline{X_4}$$

M_1 est compatible avec M_2 ,

M_1 est incompatible avec M_3 puisque X_1 apparaît dans M_1 et $\overline{X_1}$ dans M_3

M_2 est compatible avec M_3 .

b) Manifestation fonctionnelle des pannes :

On va démontrer que si l'on vérifie l'absence de tous les monomes adjacents aux monomes des fonctions implémentées sur le PLA , et la présence de tous les monomes du PLA sur chacune des sorties auxquelles ils sont connectés , toutes les pannes sont détectées .

La présence d'un monome est testé par un vecteur compatible avec lui et incompatible avec les autres monomes du PLA à tester . De même , l'absence de monomes adjacents , èst testé par un vecteur compatible avec ce monome , et incompatible avec tous les monomes du PLA .

Les pannes peuvent être classées, en fonction de leurs manifestations fonctionnelles, en cinq catégories :

- 1 - Pannes causant la disparition d'un monome du PLA dans les

fonctions réalisées

- 2 - Pannes causant la disparition de plusieurs monomes du PLA

- 3 - Pannes causant l'apparition d'un monome adjacent à un monome du PLA

- 4 - Pannes causant l'apparition de plusieurs monomes adjacents à des monomes du PLA

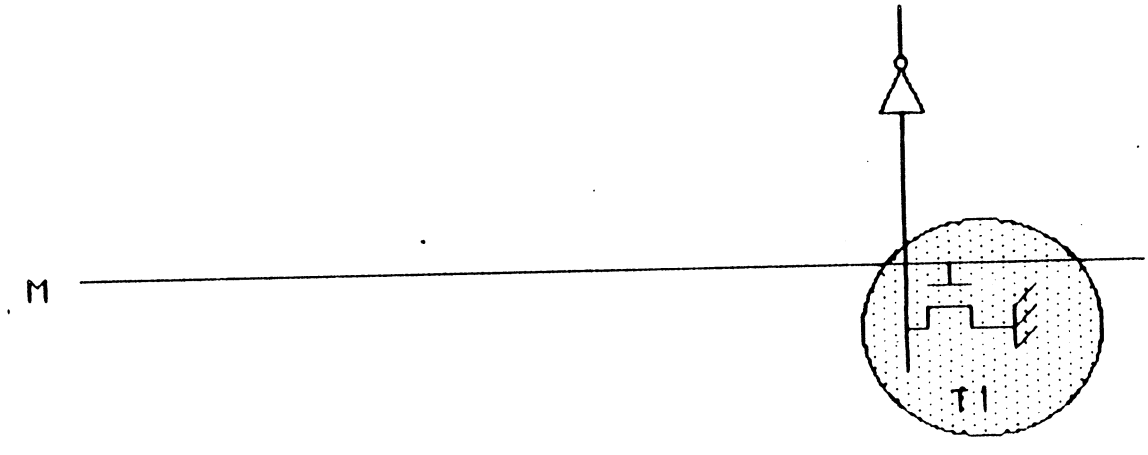
- 5- Pannes causant l'apparition de monomes du PLA dans des fonctions dans lesquelles, ils ne devraient pas apparaître

Remarque : dans les discussions qui vont suivre, Y_i représentera toujours, quel que soit l'indice i, un monome booléen, produit de plusieurs variables booléennes

N_i représentera toujours une somme de monomes booléens.

A- Pannes causant la disparition d'un monome du PLA dans les fonctions réalisées

<u>* Disparition d'un transistor entre une ligne et une sortie :</u>

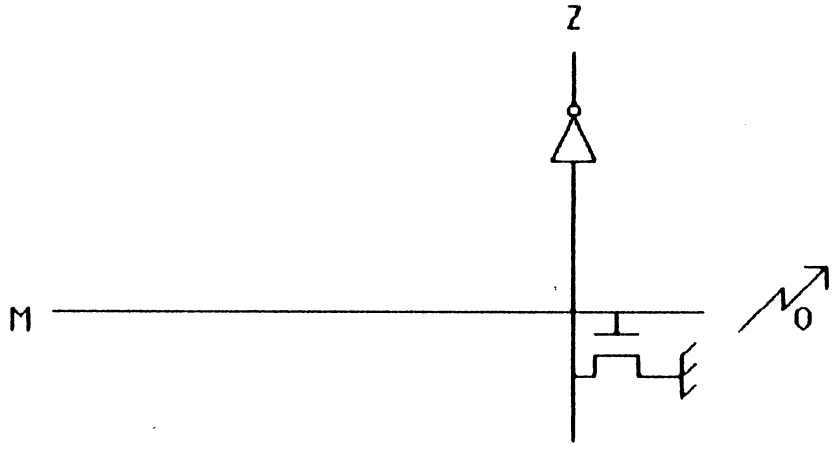


$Z = M + X$

Si le transistor T1 disparait , M disparait de la fonction Z : $Z = X$.

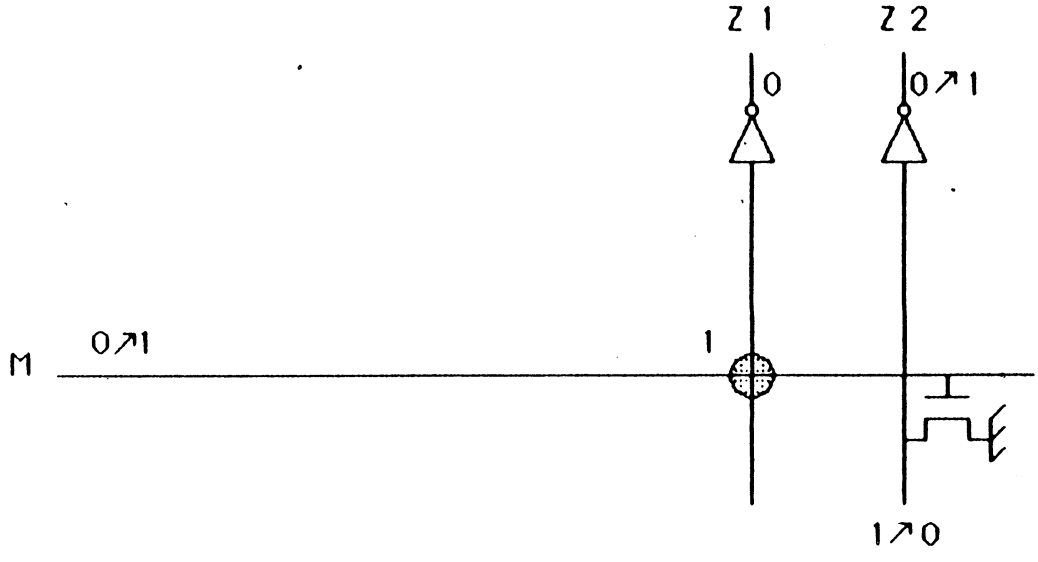
Le test de la présence de M sur toutes les sorties auxquelles il est connecté , fera apparaître ce type d'erreur .

* Collage à 0 d'une ligne :



Si la ligne supportant le monome M est collé à la valeur logique "0" , M disparaît de toutes les fonctions auxquelles il était connecté . Le test de la présence de M dans les monomes du PLA détectera ce type de panne .

* Court-circuit 1 dominant entre une ligne et une ligne de sortie :

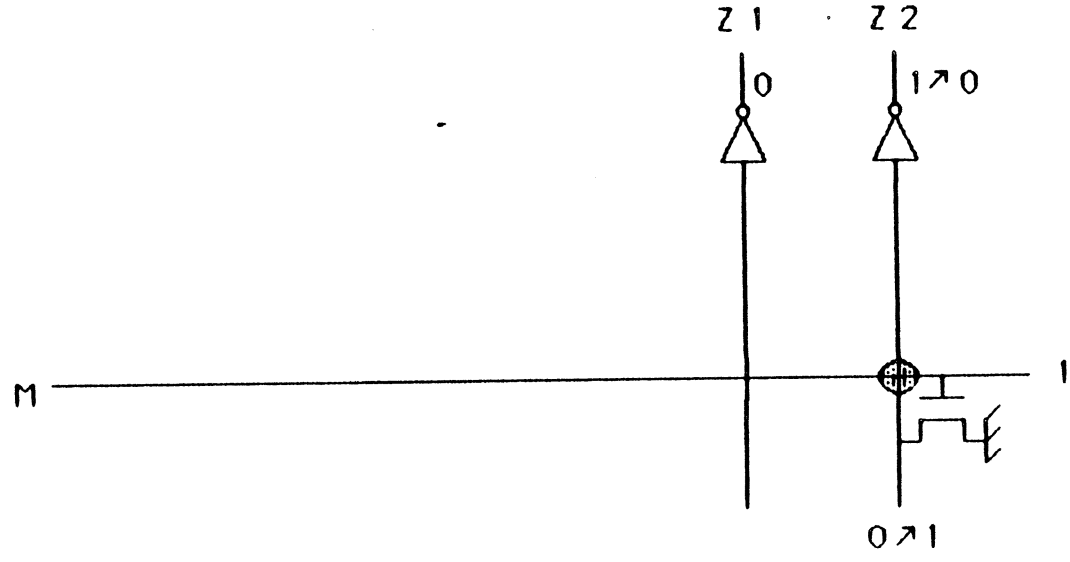


Nous distinguerons deux cas :

a- S'il n'y a pas de connexion entre la ligne et la ligne de sortie fautives:
si l'on teste l'absence de tout monome adjacent à M_1 :
soit V un vecteur de test incompatible avec tous les monomes du PLA et compatible avec un des monomes adjacents à M_1 :
sans panne : $M_1 = 0$ $Z_1 = 0$ $Z_2 = 0$
avec panne entre M_1 et Z_1 : $M_1 = 1$ $Z_1 = 0$ $Z_2 = 1$ détection de la panne sur toutes les sorties connectées à M_1 (ici Z_2)

b- S'il y a une connexion entre la ligne et la ligne de sortie fautives :
Soit V un vecteur testant la présence de M_1 : V est compatible avec M_1 et est incompatible avec les autres monomes du PLA .

sans panne : $M_1 = 1$ $Z_1 = 0$ $Z_2 = 1$
avec panne entre M_1 et Z_2 : $M_1 = 1$ $Z_1 = 0$ $Z_2 = 0$ il y a disparition du monome M_1 dans la fonction Z_2 .

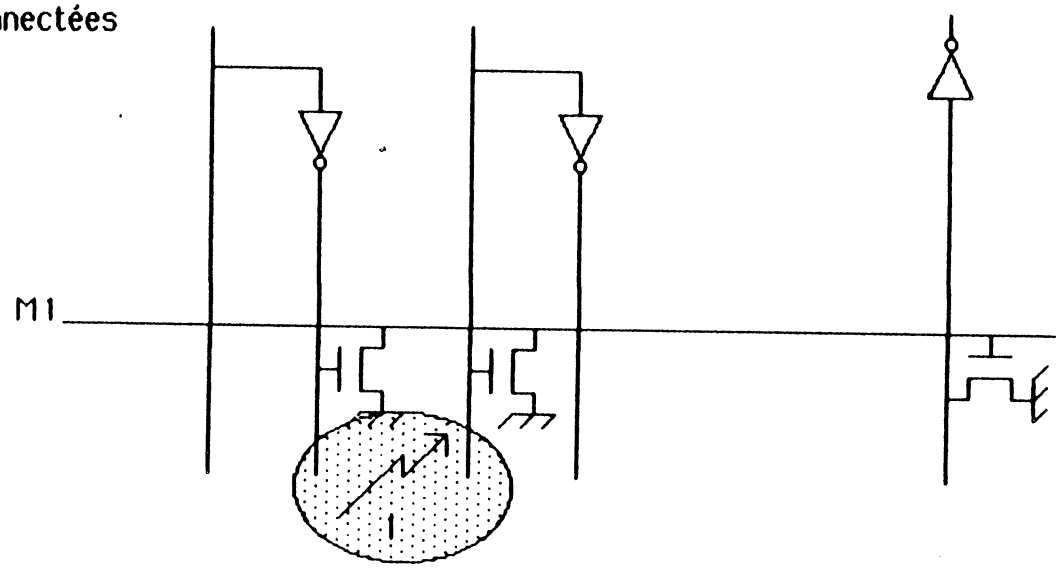


<u>* Court-circuit 1 dominant entre une colonne supportant une variable d'entrée et la ligne supportant le complément de la variable d'entrée précédente :</u>

cas 1 : il n'existe aucune ligne connectée avec l'une ou l'autre des entrées fautives.

La panne est indetectable puisque les deux entrées n'interviennent pas dans le fonctionnement du PLA.

cas 2 : quelle que soit la ligne considérée, soit aucune des colonnes ne lui sont connectées , soit les deux colonnes lui sont connectées



Pour que la panne apparaisse , il faudrait que l'une des entrées fautives soit à la valeur logique "0" , pendant que l'autre est à la valeur logique "1" .

sans panne : $\overline{\quad}$

$X_1 = 0 \quad X_2 = 1$

M = 0 (quel que soit M connecté aux deux entrées)

Z = M + N = 0 + 0 = 0

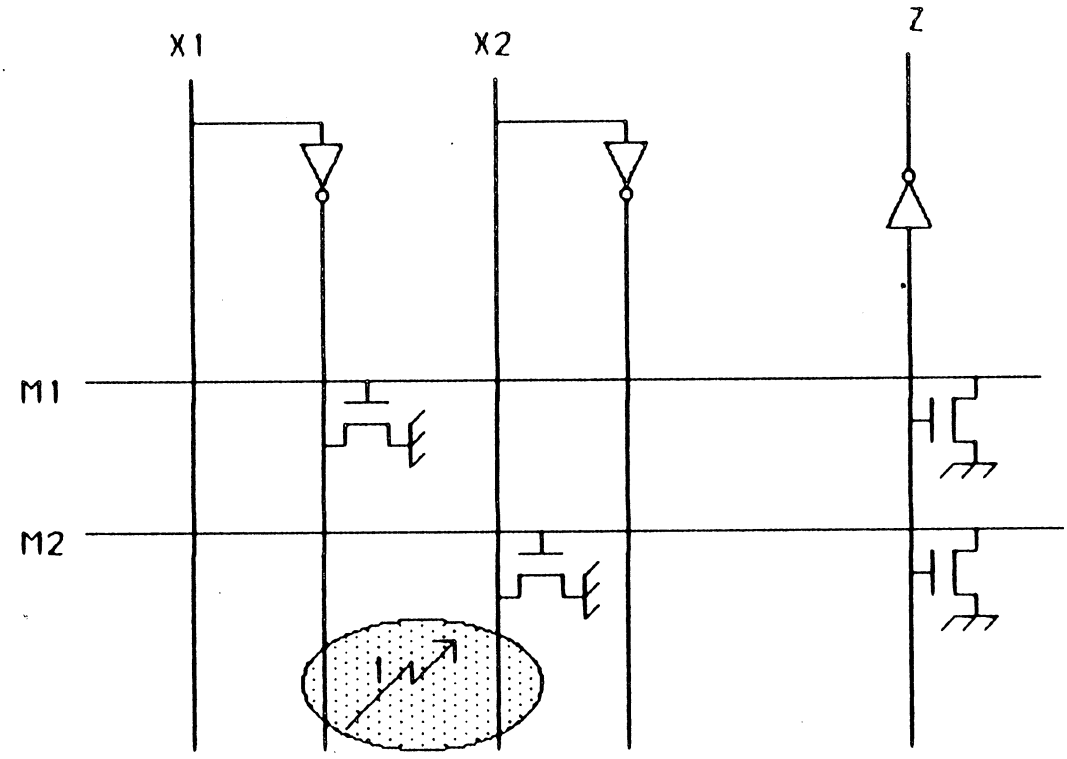
avec panne : $\overline{\quad}$

$X_1 = 1 \quad X_2 = 1$

M = 0

Z = M + N = 0 + 0 = 0 la panne est indétectable.

cas 3 : Il existe au moins une ligne connectée à une seule des entrées fautives.



Soit V un vecteur testant la présence de M_1 , appliqué sur les entrées du PLA.

V est compatible avec M_1 , et est incompatible avec tous les autres monomes du PLA.

sans panne : $X_1 = 1 \quad \overline{X_1} = 0$

$X_2 = 1$ (sinon la panne est indétectable)

$M_1 = 1$

$Z = M_1 + N = 1 + 0 = 1$

avec un court-circuit 1 dominant entre $\overline{X_1}$ et X_2

$X_1 = 1 \quad \overline{X_1} = 0$

$X_2 = 1$

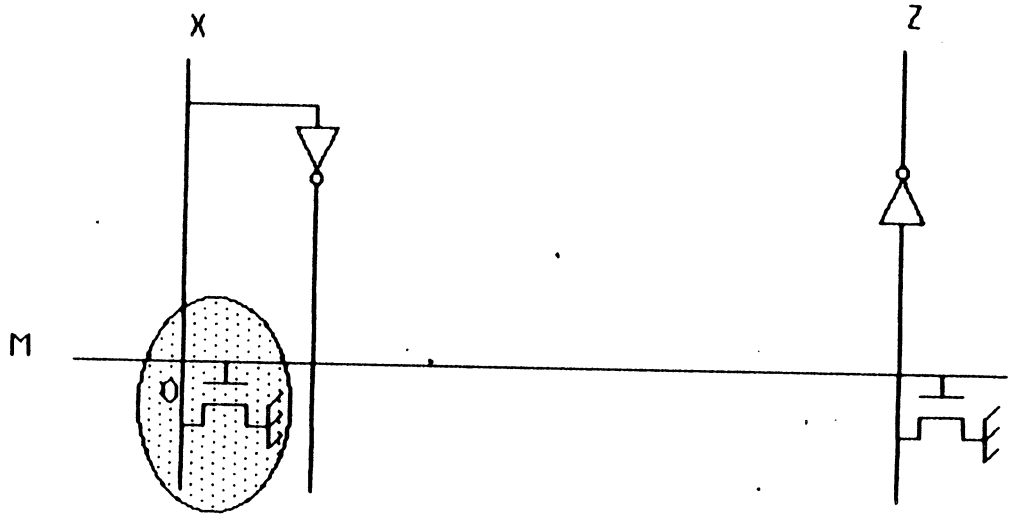
$M_1 = 0$

$Z = M_1 + N = 0 + 0 = 0$ la panne est détectée sur Z

* Court-circuit 0 dominant entre une ligne et une colonne supportant une variable d'entrée :

cas 1 : il existe une connexion entre la ligne et la colonne

supportant la variable d'entrée fautives :



Soit un vecteur V testant la présence du monome M :
V est compatible avec M et est incompatible avec tous les autres vecteurs du PLA.

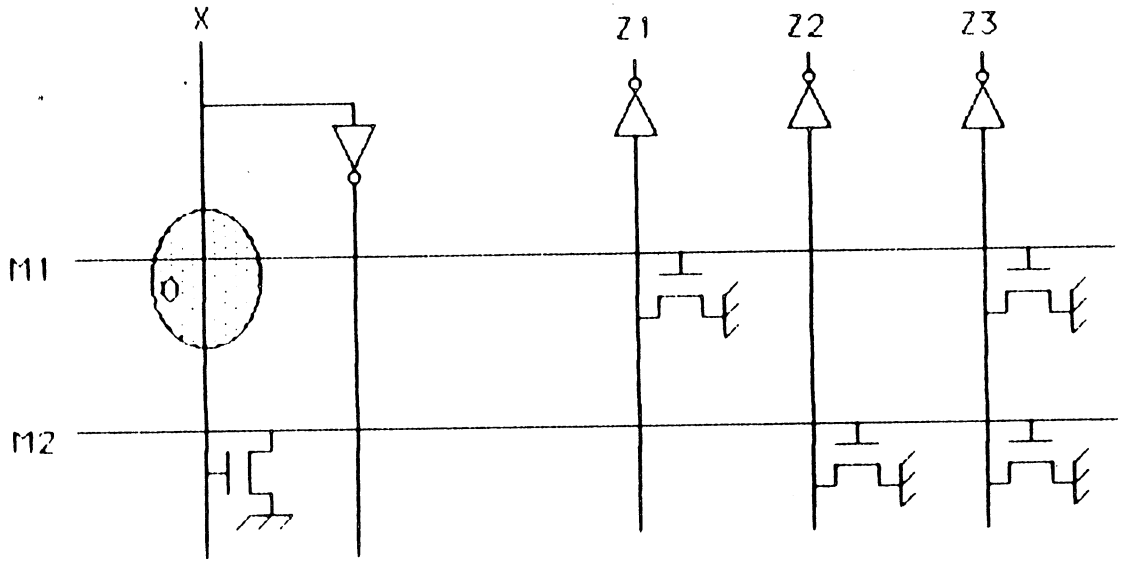
sans panne : X = 0
M = 1
Z = M + N = 1 + 0 = 1

avec un court-circuit 0 dominant entre la ligne supportant le monome M et la colonne supportant la variable X :

X = 0 M = 0
Z = M + N = 0 + 0 = 0 la panne est détectée sur Z.

cas 2 : Il n'y a pas de connexion entre la ligne et la colonne fautives :

cas 2-a : La colonne est connectée à au moins une ligne du PLA.



Soit V un vecteur testant l'absence de monome adjacent à M_2 par rapport à X :
V est incompatible avec tous les monomes du PLA , et est compatible avec le monome adjacent à M_2 par rapport à X .

sans panne : X = 1

$M_1 = 0 \quad M_2 = 0$

$Z_1 = M_1 + N_1 = 0 + 0 = 0$

$Z_2 = M_2 + N_2 = 0 + 0 = 0$

$Z_3 = M_1 + M_2 + N_3 = 0 + 0 + 0 = 0$

avec un court-circuit 0 dominant entre M_1 et X :

$X = 0$

$M_1 = 0 \quad M_2 = 1$

$Z_1 = M_1 + N_1 = 0 + 0 = 0$

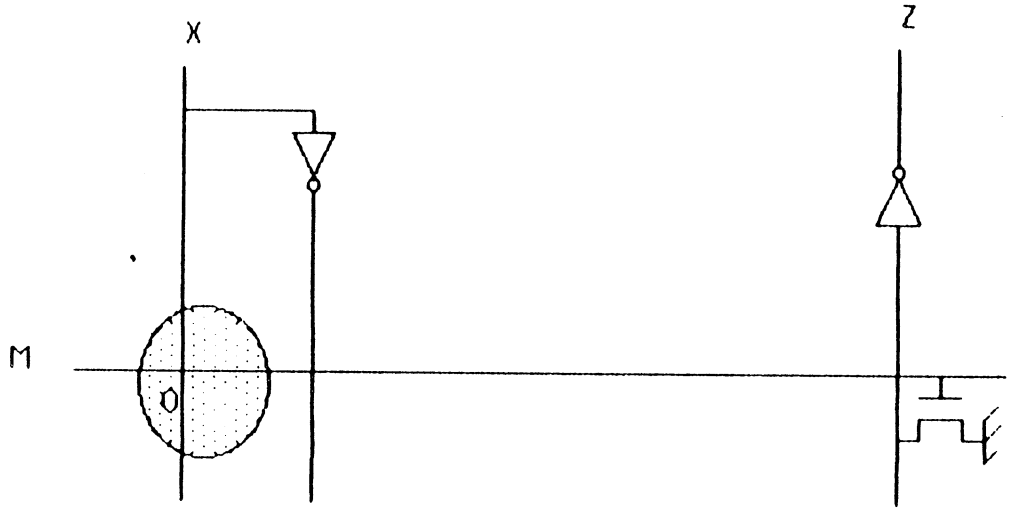
$Z_2 = M_2 + N_2 = 1 + 0 = 1$

$Z_3 = M_1 + M_2 + N_3 = 0 + 1 + 0 = 1$

La panne est détectée sur les sorties Z_2 et Z_3 .

cas 2-b : la colonne fautive n'est connectée à aucune

ligne du PLA .



Soit V un vecteur testant la présence du monome M :
V est compatible avec M , et est incompatible avec les autres monomes du PLA .

sans panne : X = 0 (sinon la panne est indétectable)
M = 1
Z = M + N = 1 + 0 = 1

avec un court-circuit 0 dominant entre M et X :
X = 0
M = 0
Z = M + N = 0 + 0 = 0 la panne est détectée .

B - Pannes causant la disparition de plusieurs monomes dans les fonctions réalisées

<u>* Collage à 0 d'une colonne supportant une sortie :</u>

Z = M + N

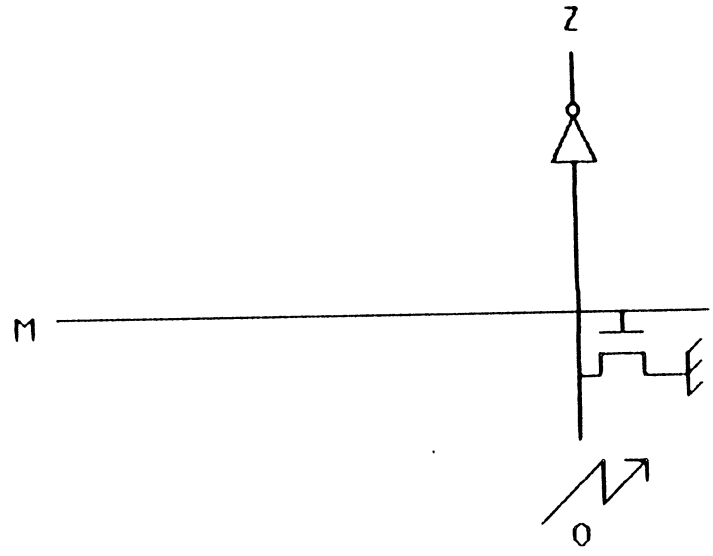
Si la colonne Z est collée à la valeur logique "0" , le test de la présence du monome M (ainsi que de tous les monomes de N) détectera ce type d'erreur.

Soit V un vecteur compatible avec M et incompatible avec les autres

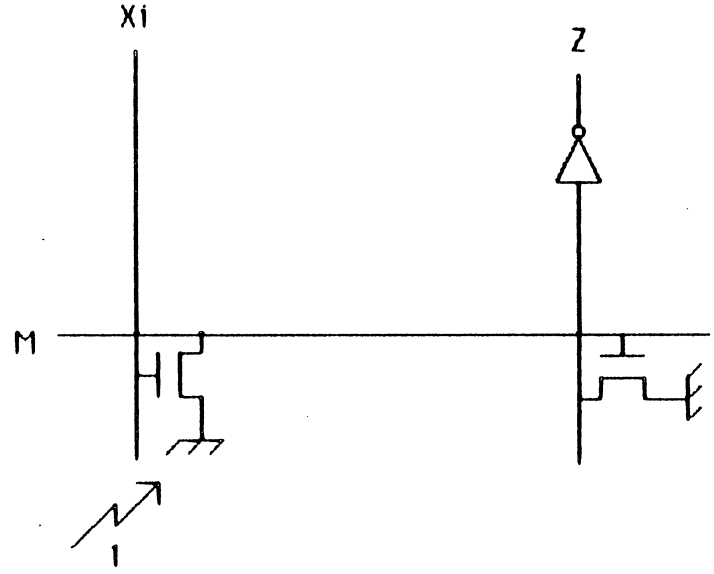
monomes du PLA :

V sur les entrées du PLA ==> M=1

==> Z=1 s'il ya collage de Z à la valeur logique "0" , la panne est détectée . .



<u>* Collage à 1 d'une colonne supportant une entrée :</u>



$M = \overline{X_i} . Y \qquad Z = M + N$

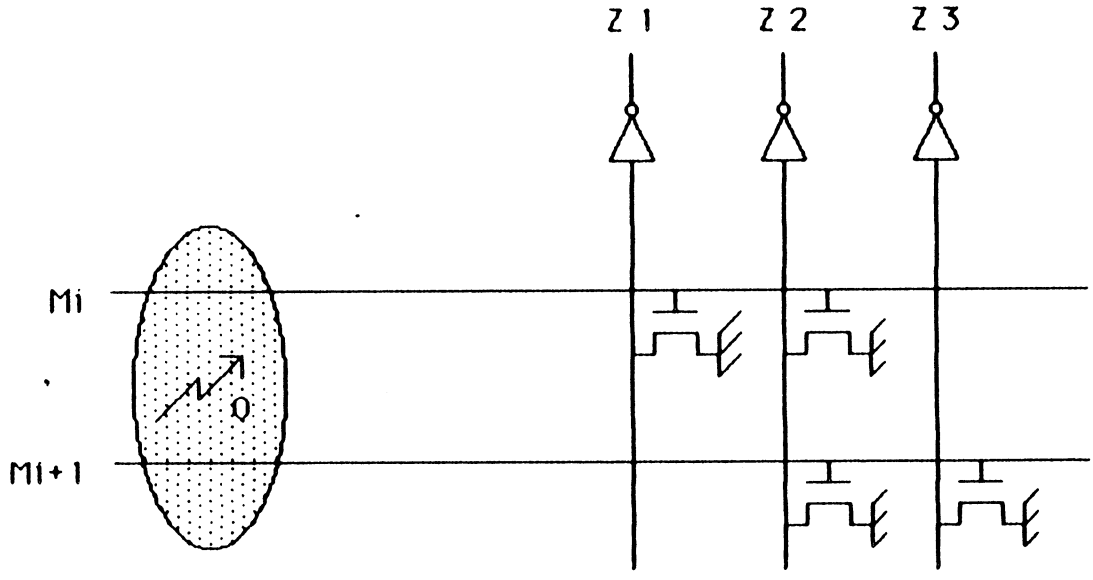
Cette panne provoque le collage à la valeur logique "0" de toutes les lignes auxquelles la colonne est connectée . Un test similaire à celui utilisé pour tester le collage à 0 des lignes détecte ces pannes :

$X_i = 1 \Rightarrow \overline{X_i} = 0$

$\Rightarrow M = 0 \,.\, Y = 0$

$\Rightarrow Z = M + N = 0 + N = N$ Il y a disparition de M dans Z .

* Court-circuit 0 dominant entre deux lignes :



Le court-circuit 0 dominant provoque un ET logique entre les deux lignes en cause :

$$M_i = M_{i+1} = M_i \,.\, M_{i+1}$$

Dans l'exemple : $Z_1 = M_i + N_1 \Rightarrow Z_1 = M_i \,.\, M_{i+1} + N_1$

$Z_2 = M_{i+1} + N_2 \Rightarrow Z_2 = M_i \,.\, M_{i+1} + N_2$

$Z_3 = M_i + M_{i+1} + N_3 \Rightarrow Z_3 = M_i \,.\, M_{i+1} + N_3$

$\Rightarrow Z_j = M_i \,.\, M_{i+1} + N_j$

Dans les trois cas possibles , un vecteur testant la présence de M_{i+1} (respectivement M_i) , en étant incompatible avec M_i (respectivement M_{i+1}) , détecte ce type de panne :

soit V compatible avec M_i , et incompatible avec M_{i+1} et N_j , un vecteur de test appliqué sur les entrées du PLA :

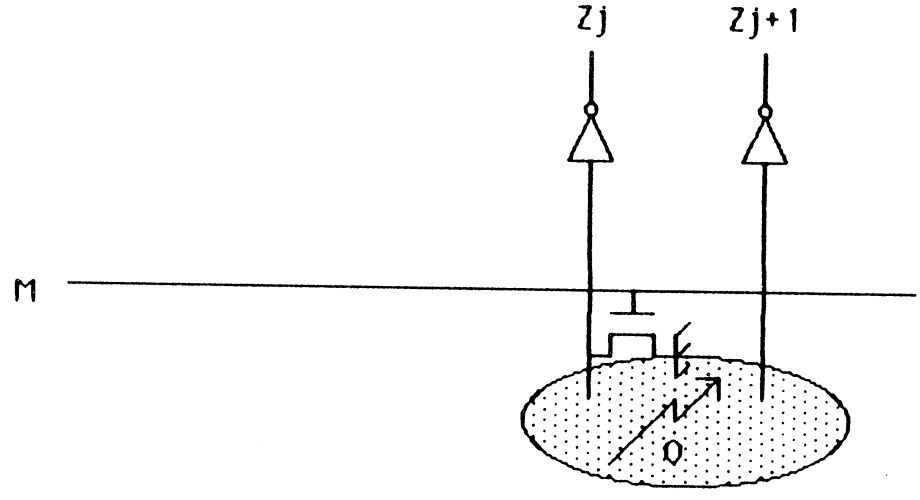
alors $Z_1 = Z_3 = 1 . 0 + 0 = 0$

alors que sans panne , $Z_1 = 1 + 0 = 1$ et $Z_3 = 1 + 0 + 0 = 1$: la panne est détectée sur Z_1 et Z_3 .

Le même raisonnement avec un vecteur V' compatible avec M_{i+1} et incompatible avec M_i et N_j montre que la panne est détectée dans le cas où tous les monomes du PLA sont du type de M_2 .

Il est impossible de trouver un vecteur V répondant aux conditions précédentes si M_i est couvert par M_{i+1} ou réciproquement . Dans ce cas , le PLA contient des monomes redondants et ce type d'erreur est , dans certains cas , indétectable .

* Court-circuit 0 dominant entre deux colonnes supportant des sorties :



$$Z_j = M_i + N_1 \qquad Z_{j+1} = N_2$$

En général , les deux sorties ne sont pas égales , aussi , existe-t-il une ligne connectée à une seule des colonnes fautives . Lors du test de la présence d'un tel monome , celui-ci disparaît dans la fonction où il est connecté :

Soit V un vecteur de test compatible avec M_i et incompatible avec les autres monomes :

alors sans panne : $Z_j = M_i + N_1 = 1 + 0 = 1$

$$Z_{j+1} = N_2 = 0$$

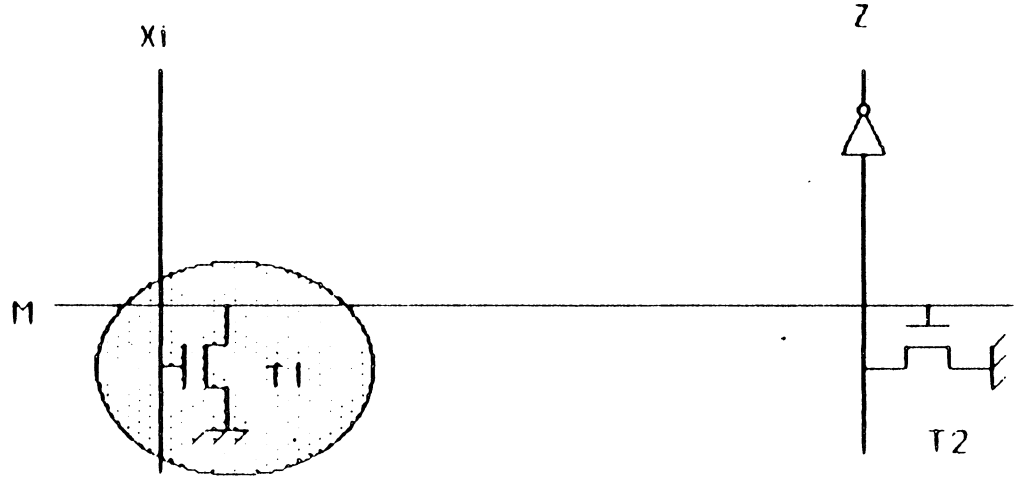
avec la panne : $Z_j = Z_{j+1} = 0$ et la panne est détectée sur Z_j .

<u>* Court-circuit 1 dominant entre une colonne supportant une entrée et la colonne supportant son complément :</u>

Pour les mêmes raisons que précédemment , cette panne est équivalente à un collage à 1 des deux colonnes fautives : X_i et $\overline{X_i}$.

C - Pannes causant l'apparition d'un monome adjacent à l'un des monomes du PLA

<u>* Disparition d'un transistor entre une ligne et une colonne supportant une variable d'entrée :</u>



La disparition du transistor T1 fait disparaître la variable $\overline{X_i}$ du monome M :

$$M = Y = \overline{X_i} \cdot Y + X_i \cdot Y$$
$$Z = M + N + X_i \cdot Y$$

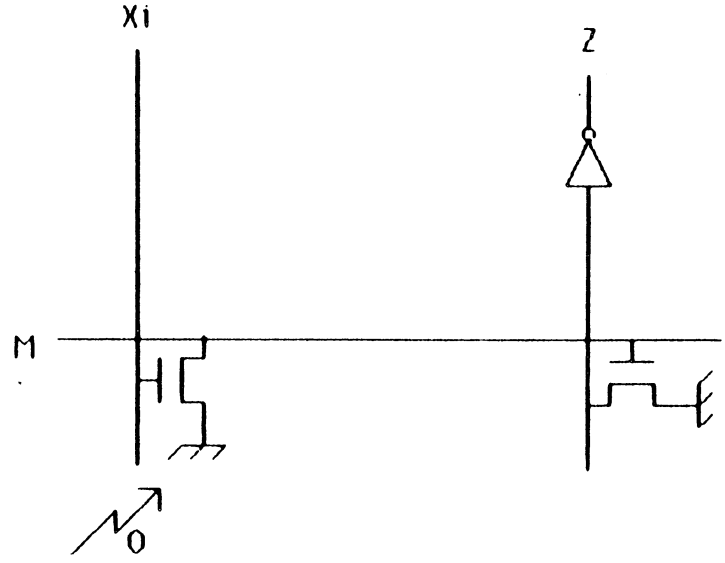
Le monomé $X_i \cdot Y$ apparaît sur la sortie Z or $X_i \cdot Y$ est un des monomes adjacents à $\overline{X_i} \cdot Y$. Le test de l'absence des monomes adjacents à M détectera ce type de pannes.

* Collage à 0 d'une colonne supportant une entrée :

Si une colonne est collée à la valeur logique "0", elle disparaît de toutes les lignes auxquelles elle est connectée. Cette disparition est détectée par le test de l'absence des monomes adjacents aux monomes ainsi modifiés :

Dans l'exemple : $M = \overline{X_i} \cdot Y$ $Z = M + N$

Si la colonne de la variable d'entrée X_i est collée à 0 :

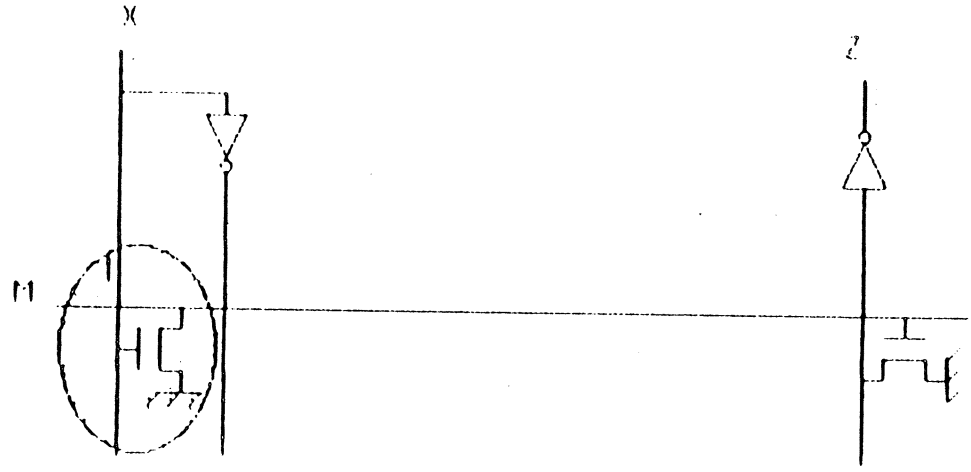


$$X_i = 0 \Rightarrow M = 1 \,.\, Y = Y = \overline{X_i} \,.\, Y + X_i \,.\, Y = M + (X_i \,.\, Y)$$
$$\Rightarrow Z = M + N + (X_i \,.\, Y)$$

Il y a apparition de $X_i \,.\, Y$, qui estle monome adjacent à M par rapport à la variable X_i, dans la fonction Z.

* Court-circuit 1 dominant entre une ligne et une colonne supportant une variable d'entrée :

cas 1 : il existe une connexion entre la ligne et la colonne fautives.



Soit V un vecteur testant l'absence du monome adjacent à M par rapport à X : V est incompatible avec tous les monomes du PLA , mais est compatible avec ce monome adjacent à M .

sans panne : $X = 1$

$$M = \bar{X} . Y = 0 . 1 = 0$$
$$Z = M + N = 0 + 0 = 0$$

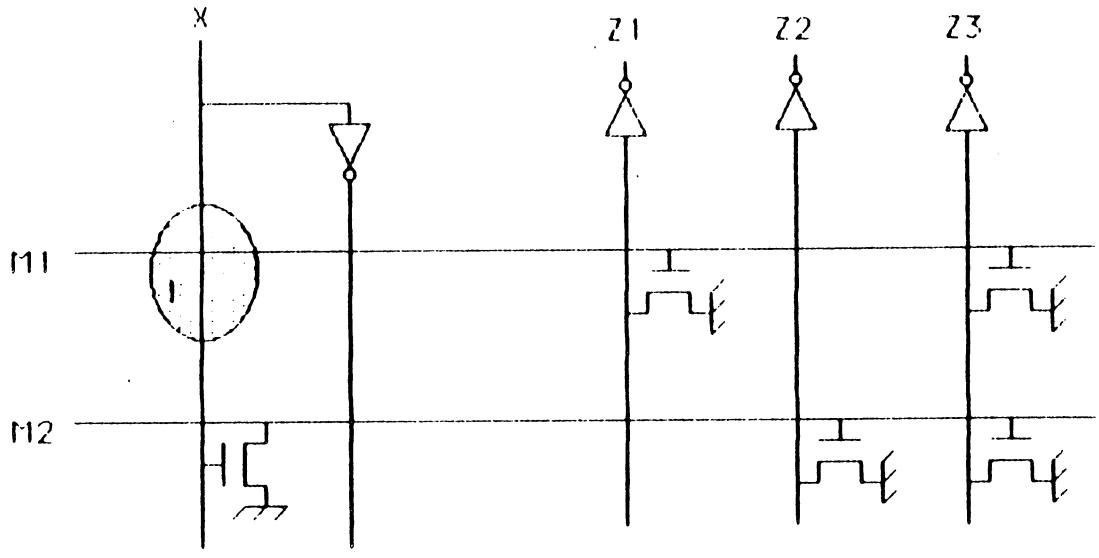
avec un court-circuit 1 dominant entre M et X :

$X = 1$
$M = 1$
$Z = M + N = 1 + 0 = 1$ la panne est détectée .

cas 2 : Il n'existe pas de connexion entre la ligne et la colonne fautives :

cas 2-a : La colonne fautive est connectée à au moins une ligne du PLA .

Soit V un vecteur testant l'absence du monome adjacent à M_2 par rapport à X :
V est incompatible avec tous les monomes du PLA , mais est compatible avec ce monome adjacent à M_2 .



sans panne : $X = 1$

$M_1 = 0 \quad M_2 = 0$

$Z_1 = M_1 + N_1 = 0 + 0 = 0$

$Z_2 = M_2 + N_2 = 0 + 0 = 0$

$Z_3 = M_1 + M_2 + N_3 = 0 + 0 + 0 = 0$

avec un court-circuit 1 dominant entre M_1 et X:

$X = 1$

$M_1 = 1 \quad M_2 = 0$

$Z_1 = M_1 + N_1 = 1 + 0 = 1$

$Z_2 = M_2 + N_2 = 0 + 0 = 0$

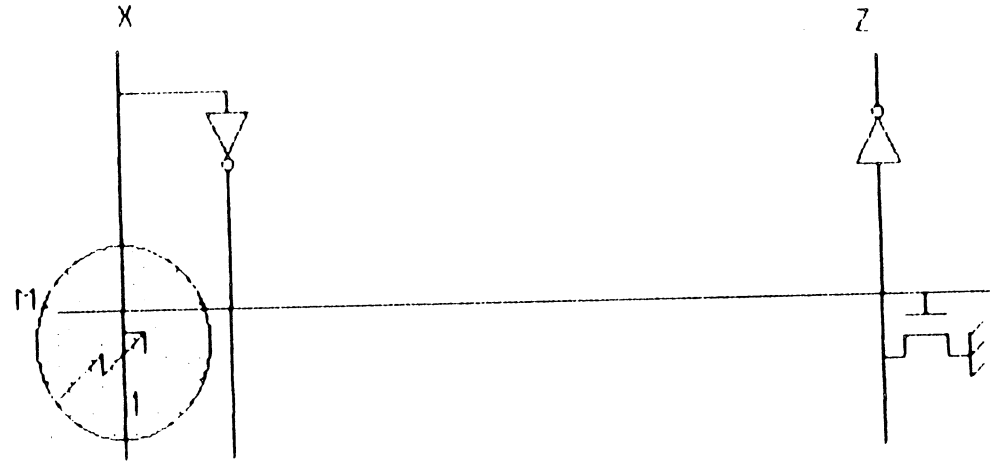
$Z_3 = M_1 + M_2 + N_3 = 1 + 0 + 0 = 0$

La panne est détectée sur les sorties Z_1 et Z_3.

cas 2-b : la colonne fautive n'est connectée à aucune ligne du PLA .

Soit V un vecteur testant l'absence du monome adjacent à M par rapport à X :

V est incompatible avec tous les monomes du PLA , mais est compatible avec ce monome adjacent à M .

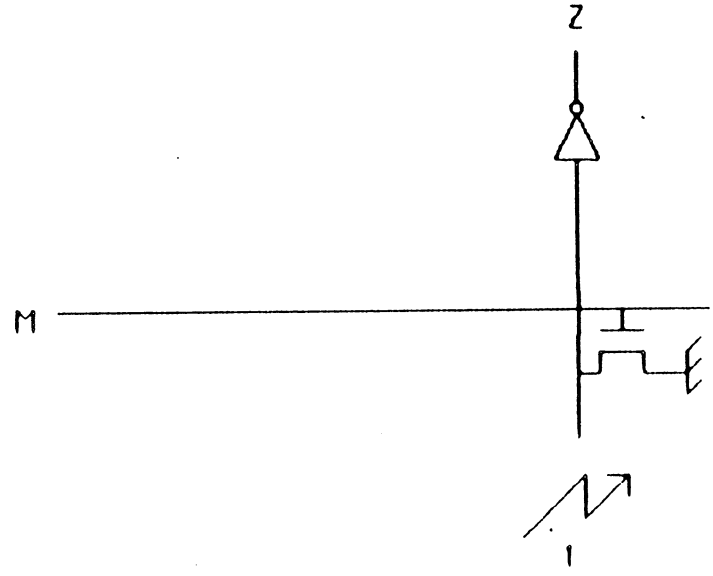


sans panne : X = 1 (sinon la panne est indétectable)
M = 0
Z = M + N = 0 + 0 = 0

avec un court-circuit 1 dominant entre M et X :
X = 1
M = 1
Z = M + N = 1 + 0 = 1 la panne est détectée .

D - Pannes causant l'apparition de plusieurs monomes adjacents à des monomes du PLA

<u>* Collage à 1 d'une colonne supportant une sortie :</u>



Z = M + N

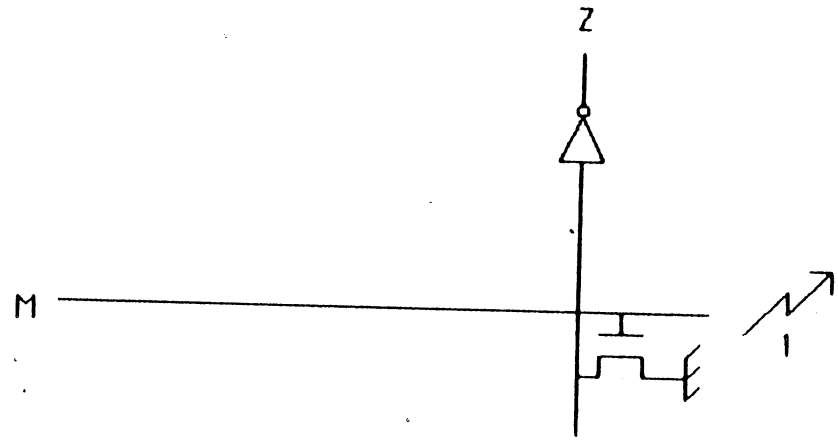
Le test de l'absence de l'un des monomes adjacents à M (ainsi que de tous les monomes adjacents à chacun des monomes de N) détectera ce type de panne : en effet

soit le vecteur M' adjacent à M présent sur les entrées :

M' est incompatible avec tous les monomes du PLA :

==> M = 0 Z = 0 s'il y a collage de la colonne Z à 1 , la panne est détectée .

<u>* Collage à 1 d'une ligne :</u>



Si M est collé à la valeur logique "1" , toutes les colonnes supportant une des sorties auxquelles il est connecté , seront collées à 1 , l'erreur sera donc détectée par le même type de test que pour le collage d'une colonne supportant une sortie à 1 :

Z = M + N

or quel que soit le vecteur de test appliqué sur les sorties du PLA :

M = 1 ==> Z = 1

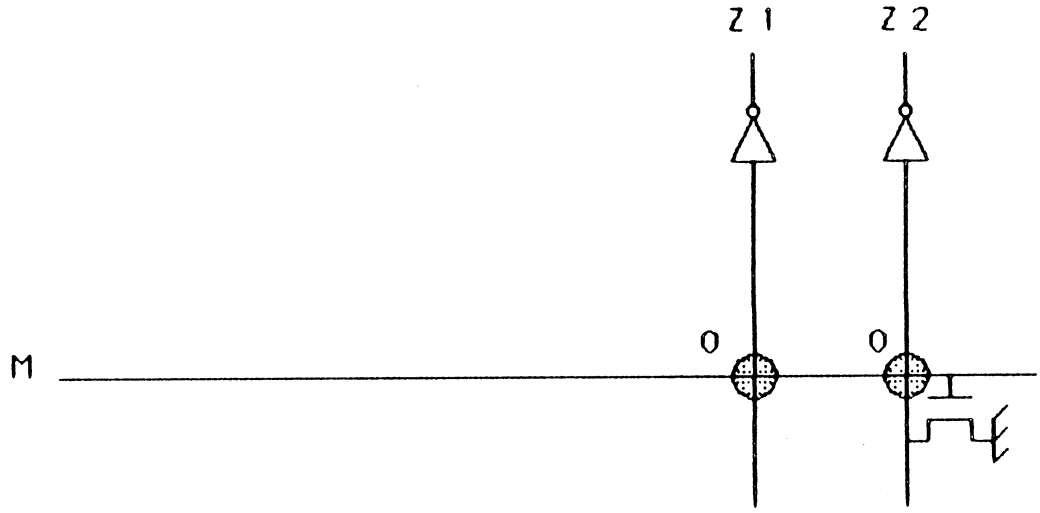
<u>* Court-circuit 0 dominant entre une colonne supportant une variable d'entrée et la colonne supportant son complément :</u>

Les deux valeurs de ces colonnes étant complémentées , l'une est à la

valeur logique "0" , ce qui avec la panne , force l'autre également à 0 .

Cette panne est donc équivallente au collage à 0 des deux colonnes X_i et $\overline{X_i}$

* Court-circuit 0 dominant entre une ligne et une colonne supportant une sortie :



Deux cas peuvent apparaître , suivant qu'il existe , ou pas , une connexion entre la ligne et la colonne fautives :

Si l'on teste l'absence de monome adjacent à M_1 :

Soit V un vecteur de test incompatible avec tous les monomes du PLA , mais compatible avec un des monomes adjacent à M_1 :

Sans panne : $M_1 = 0 \quad Z_1 = 0 \quad Z_2 = 0$

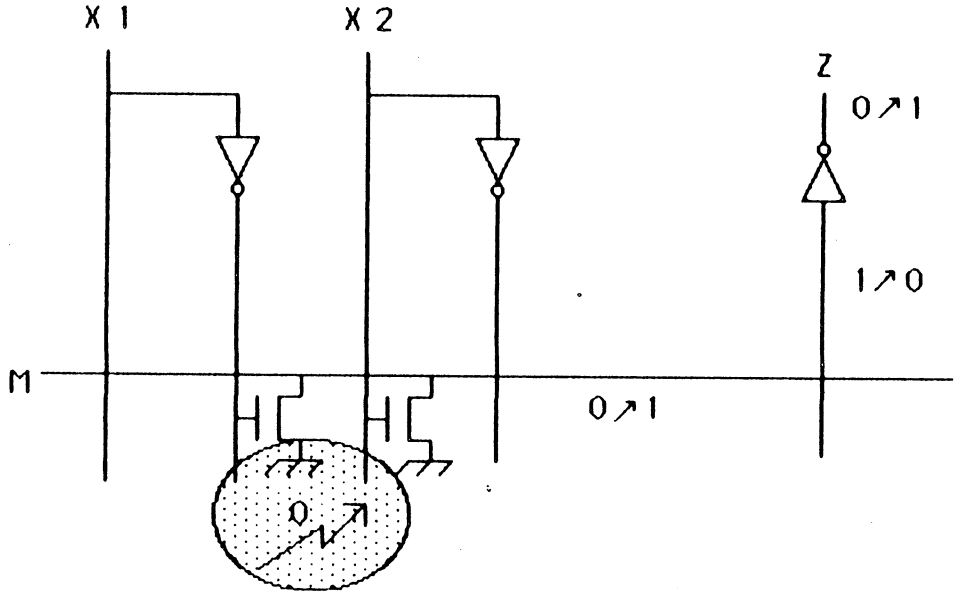
Avec une panne entre M_1 et Z_1 : $M_1 = 0 \quad Z_1 = 1 \quad Z_2 = 0$ détection sur Z_1 .

Avec une panne entre M_1 et Z_2 : $M_1 = 0 \quad Z_1 = 0 \quad Z_2 = 1$ détection sur Z_2 .

* Court-circuit 0 dominant entre une colonne supportant une variable d'entrée et la colonne supportant le complément de la variable d'entrée précédente :

cas 1 : il n'existe aucune connexion entre une ligne et l'une ou l'autre de ces colonnes . Ces entrées n'interviennent pas dans le fonctionnement du PLA , et la panne est indétectable .

cas 2 : il existe une ligne connectée aux deux colonnes fautives :



On teste l'absence du monome adjacent à M_1 par rapport à l'une des deux variables fautives : soit V un vecteur de test incompatible avec tous les monomes du PLA et compatible avec un monome adjacent à M_1 par rapport à X_1

sans panne : $X_1 = 0 \quad \overline{X_1} = 1$

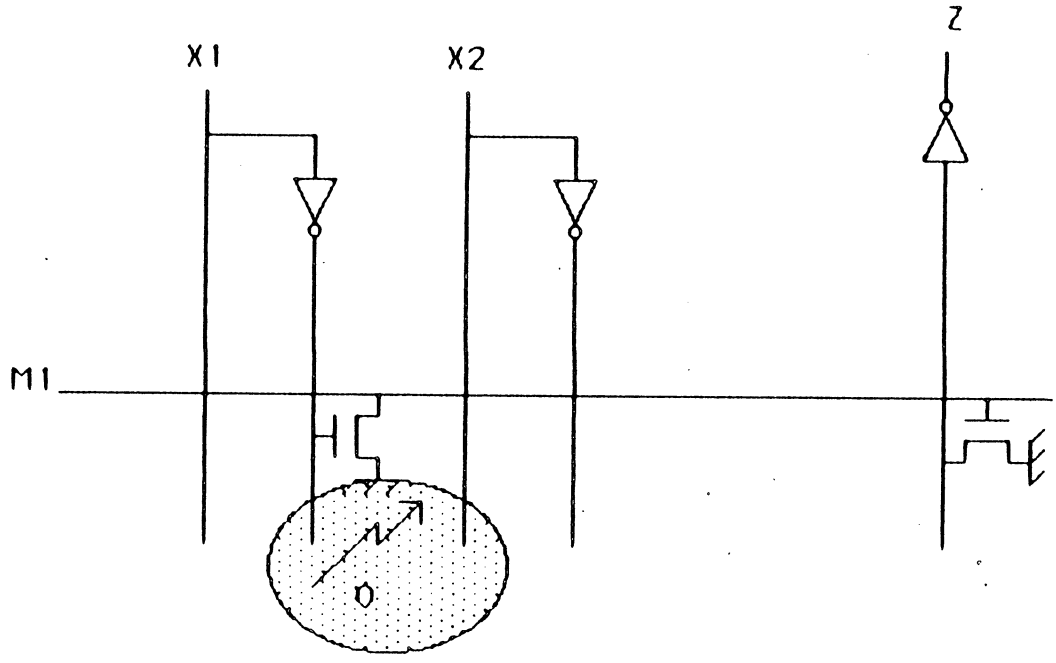
$X_2 = 0 \quad M_1 = 0 \quad Z = M_1 + N = 0 + 0 = 0$

avec panne : $X_1 = 0 \quad \overline{X_1} = 0$

$X_2 = 0 \quad M_1 = 1 \quad Z = M_1 + N = 1 + 0 = 1$

la panne est détectée sur toutes les sorties connectées à un monome connecté aux deux entrées fautives .

cas 3 : une ligne est connectée à une seule des colonnes fautives :



Si l'on teste l'absence du monome adjacent à M_1 par rapport à cette entrée:
soit V un vecteur incompatible avec tous les monomes du PLA , mais compatible avec le monome adjacent à M_1 par rapport à X_1

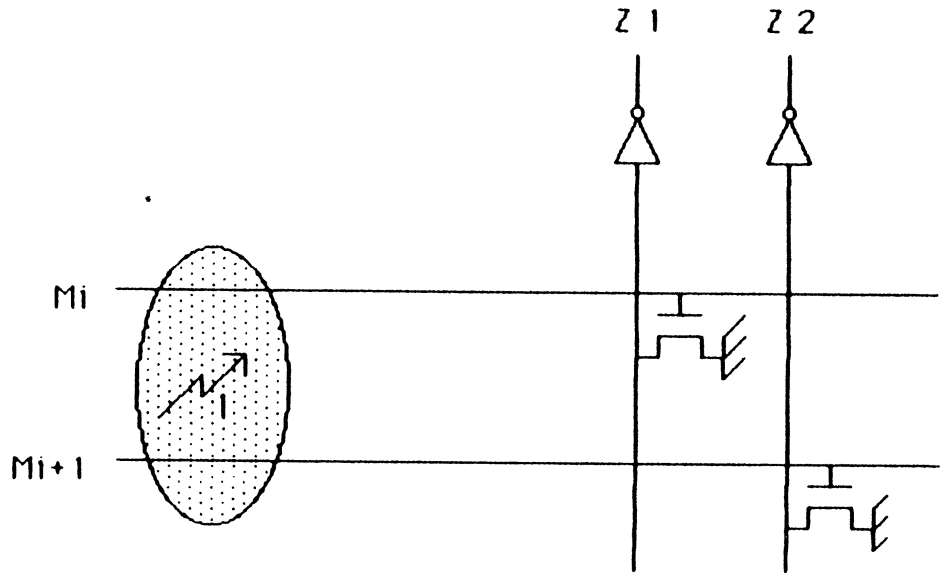
sans panne : $X_1 = 0$ $\overline{X_1} = 1$
$X_2 = 0$ (autrement la panne est indétectable)
$M_1 = 0$ $Z = 0$

avec panne entre la colonne $\overline{X_1}$ et la colonne X_2 :

$X_1 = 0$ $\overline{X_1} = 1$
$X_2 = 0$
$M_1 = 1$ $Z = 1$ la panne est détectée sur Z .

E - Pannes causant l'apparition de monomes du PLA dans des fonctions dans lesquelles ils ne devraient pas apparaître

* Court-circuit 1 dominant entre deux lignes :



Dans le cas général , Z_1 est différente de Z_2 , donc il existe une colonne Z, sur laquelle une seule des lignes M_1 ou M_{1+1} , est connectée .

Supposons :

$$Z_1 = M_1 + N$$

un court-circuit 1 dominant provoque un OU logique entre les deux lignes fautives :

$$M_1 = M_{1+1} = M_1 + M_{1+1}$$

d'où avec la panne :

$$Z_1 = M_1 + M_{1+1} + N$$

Lors du test de la présence de M_{1+1} , la panne est détectée :

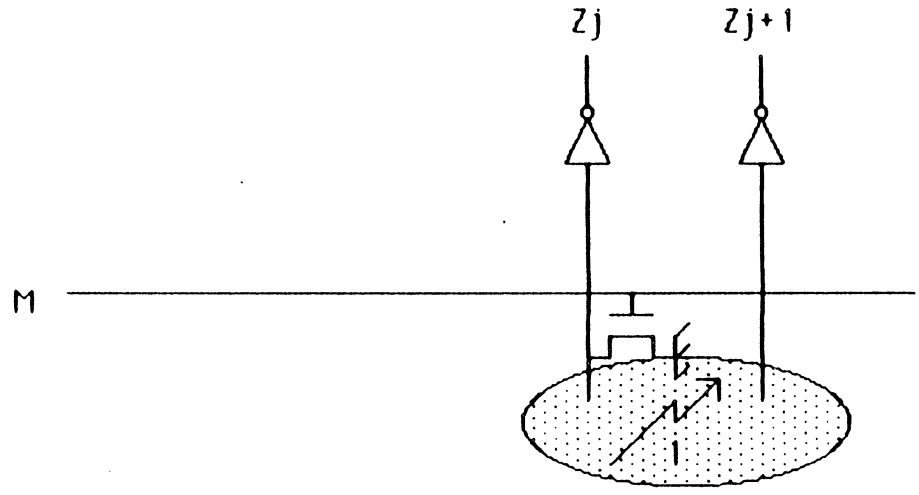
soit un vecteur V compatible avec M_{1+1} , et incompatible avec tout monome M_j ($j \neq 1+1$)

alors $M_{1+1} = 1$ $M_1 = 0$ $N = 0$

sans panne : $Z_1 = M_1 + N = 0 + 0 = 0$

avec panne : $Z_1 = M_1 + M_{1+1} + N = 0 + 1 + 0 = 1$ la panne est détectée .

* Court-circuit 1 dominant entre deux colonnes supportant des sorties :



Il existe , en général , un monome M_i , appartenant à une seule des deux sorties erronnées :

$$Z_j = M_i + N_1$$
$$Z_{j+1} = N_2$$

Soit V , un vecteur de test compatible avec M_i et incompatible avec tous les autres monomes :

sans la panne : $Z_j = M_i + N_1 = 1 + 0 = 1$

$Z_{j+1} = N_2 = 0$

avec la panne : $Z_j = Z_{j+1} = 1$ l'erreur est détectée sur Z_{j+1}

c) La méthode :

De l'étude des conséquences de chaque panne , on choisit de déterminer un ensemble de vecteurs permettant de vérifier pour chacun des monomes du PLA à tester :

- sa présence dans toutes les fonctions auxquelles il appartient .
- l'absence de tous les monomes qui lui sont adjacents dans les fonctions du PLA .

Cet ensemble, permet de couvrir toutes les pannes détectables des cinq classes précédentes .

c-1 Définitions :

* On appelle **représentation binaire** d'un monome à n variables , le vecteur , appartenant à l'ensemble $(0 , 1 , -)^n$, dans lequel , la $i^{ème}$ variable du monome est représentée par :

- "1" si la variable d'entrée i apparaît sous sa forme normale .
- "0" si la variable d'entrée i apparaît sous sa forme complémentée .
- "-" si la variable d'entrée i n'apparaît pas dans le monome .

exemple :

soit le monome $M_1 = X_1 \overline{X_4}$ appartenant à un PLA à 4 entrées .

La représentation binaire de M_1 est : $B_1 = 0 \; - \; - \; 1$

$X_4 \; X_3 \; X_2 \; X_1$

* On appelle **ensemble de représentations décimales** d'un monome d'un PLA à n entrées , l'ensemble construit à partir de sa représentation binaire comme suit :

on calcule l'ensemble exhaustif des représentations binaires de ce monome en remplaçant chaque "-" par

- la valeur "0"
- la valeur "1"

On affecte à chaque bit i de chacune des représentations binaires de cet ensemble , un poids de valeur 2^{i-1} et on le multiplie par le bit i (0 ou 1) ;

puis on effectue la somme des n valeurs ainsi obtenues , afin d'obtenir un ensemble de représentations décimales du monome :

exemple :

$M_1 = X1\,\overline{X4}$

--> Sa représentation binaire est : $B_1 = 0 - - 1$

--> Ensemble exhaustif des représentations binaires de M_1 :

$B_{11} = 0\,0\,0\,1$

$B_{12} = 0\,0\,1\,1$

$B_{13} = 0\,1\,0\,1$

$B_{14} = 0\,1\,1\,1$

--> Ensemble des représentations décimales de M_1 :

$B_{11} = 0\,0\,0\,1 \Rightarrow D_{11} = 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 1 = 1$

$B_{12} = 0\,0\,1\,1 \Rightarrow D_{12} = 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1 = 2 + 1 = 3$

$B_{13} = 0\,1\,0\,1 \Rightarrow D_{13} = 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1 = 4 + 1 = 5$

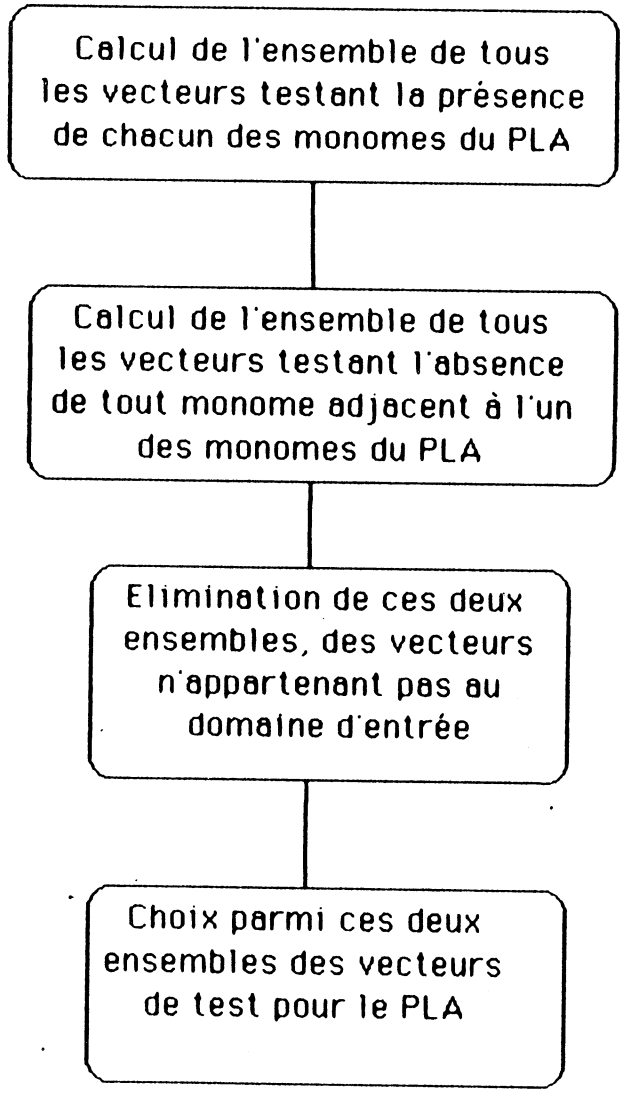
$B_{14} = 0\,1\,1\,1 \Rightarrow D_{14} = 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = 4 + 2 + 1 = 7$

L'ensemble des représentations décimales de M_1 est : (1 , 3 , 5 , 7).

c-2 : Algorithme :

Nous présentons ici, la méthode arithmétique d'après laquelle, un programme a été réalisé. Cette mèthode permet de calculer, pour un PLA donné, un ensemble de vecteurs de test, choisis parmi un ensemble restreint de vecteurs autorisés (appelé **domaine d'entrée** du PLA).

Algorithme général du calcul
de vecteurs de test pour un PLA donné.



c-3 : Méthode arithmétique :

Cette méthode se décompose en plusieurs étapes :

1° Calcul des ensembles de représentations décimales pour chaque monome du PLA .

2° Elimination des représentations décimales apparaissant dans plusieurs ensembles .

3° Calcul des ensembles de représentations décimales de tous les monomes adjacents aux monomes du PLA .

4° Elimination des représentations décimales apparaissant dans les ensembles calculés à létape n° 1 .

5° Elimination des représentations décimales n'appartenant pas au domaine d'entrée que l'on s'est fixé .

6° Choix d'un ensemble de vecteurs de test , tel que pour chaque ensemble calculé précédemment aux étapes n° 1 et 3 , un vecteur au moins appartient aux vecteurs choisis pour le test .

7° Si un ensemble est vide , à la suite des diverses éliminations des étapes n° 2 , 4 et 5 , on étudie les valeurs éliminées , afin de déterminer si la panne est indétectable ou si elle peut être visible sur au moins une sortie . Pour cela, on regarde, pour chacune des valeurs éliminées lors des étapes n° 2 et 4, sur quelles sorties les pannes seront cachées. Si les pannes sont masquées sur toutes les sorties, le vecteur est alors définitivement éliminé. Sinon, il est choisi puisque détectant la panne sur au moins une sortie.

Dans le cas où l'on cherche à détecter la présence d'un monome dans les fonctions auxquelles il appartient, il faudra choisir, si c'est possible, autant de vecteurs que nécessaire, pour que sa présence soit vérifiée dans toutes les fonctions voulues.

Exemple :

Soit le PLA à 4 entrées , 4 monomes et 2 sorties dont les monomes sont :

$M_1 = X_4 \overline{X_3}$ représentation binaire : $B_1 = 1\ 0\ -\ -$

$M_2 = \overline{X_3} X_2$ représentation binaire : $B_2 = -\ 0\ 1\ -$

$M_3 = X_4 X_2$ représentation binaire : $B_3 = 1 - 1 -$

$M_4 = \overline{X_4} \overline{X_3} \overline{X_1}$ représentation binaire : $B_4 = 0\ 0 - 0$

et les sorties sont :

$Z_1 = M_1 + M_2 + M_4$

$Z_2 = M_3 + M_4$

etape n°1: calcul des ensembles de représentations décimales :

M_1 --> (8 , 9 , 10 , 11)

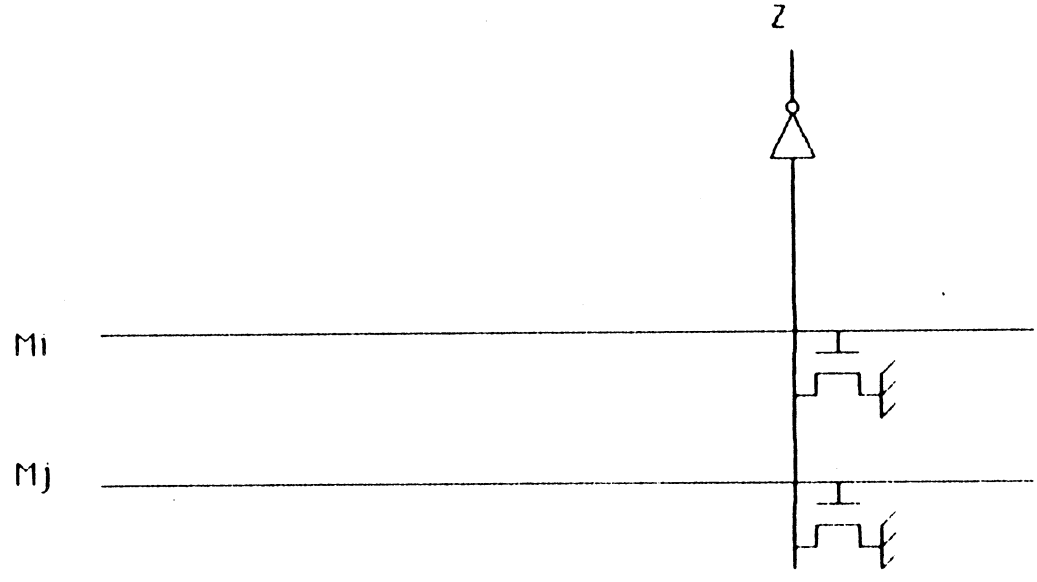
M_2 --> (2 , 3 , 10 , 11)

M_3 --> (10 , 11 , 14 , 15)

M_4 --> (0 , 2)

etape n°2: elimination des représentations décimales apparaissant dans plusieurs ensembles :

si un vecteur apparaissant dans plusieurs ensembles était choisi pour le test , il y aurait risque de masquage de certaines pannes :



Si un vecteur de test a été choisi et appartient aux ensembles des représentations décimales de M_i et M_j , alors :

$M_i = 1 \quad M_j = 1 \Longrightarrow Z = 1$

si l'un des deux transistors disparait , la sortie sera quand même égale à

1 , et la panne sera masquée .

$M_1 = (8, 9, 10, 11) = (8, 9)$
$M_2 = (2, 3, 10, 11) = (3)$
$M_3 = (10, 11, 14, 15) = (14, 15)$
$M_4 = (0, 2) = (0)$

etape n°3: calcul des ensembles de représentation décimales de tous les monomes adjacents aux monomes du PLA :

$M_1 = 10--$ ==> (8, 9)
==> $M_1^1 = 00--$ ==> (0, 1, 2, 3)
==> $M_1^2 = 11--$ ==> (12, 13, 14, 15)

$M_2 = -01-$ ==> (3)
==> $M_2^1 = -11-$ ==> (6, 7, 14, 15)
==> $M_2^2 = -00-$ ==> (0, 1, 8, 9)

$M_3 = 1-1-$ ==> (14, 15)
==> $M_3^1 = 0-1-$ ==> (2, 3, 6, 7)
==> $M_3^2 = 1-0-$ ==> (8, 9, 12, 13)

$M_4 = 00-0$ ==> (0)
==> $M_4^1 = 10-0$ ==> (8, 10)
==> $M_4^2 = 01-0$ ==> (4, 6)
==> $M_4^3 = 00-1$ ==> (1, 3)

Une manière aisée d'obtenir les représentations décimales des monom adjacents à un monome est :

- d'ajouter 2^{i-1} aux représentations décimales de ce monome si $i^{ème}$ variable apparait complémentée dans le monome
- de retrancher 2^{i-1} aux représentations décimales de ce mono si la $i^{ème}$ variable apparait dans le monome .

exemple :

M_1 = 1 0 - - ==> (8 , 9 , 10 , 11)

X_4 = 1 on retranche 2^3 = 8 ==> (0 , 1 , 2 , 3)

X_3 = 0 on rajoute 2^2 = 4 ==> (12 , 13 , 14 , 15)

etape n°4: Elimination des représentations décimales apparaissant dans les ensembles calculés à l'étape n° 1 . Ceci pour les mêmes raisons qu'à l'étape n° 2 (risque de masquage de certaines pannes).

ensemble de l'etape n° 1	ensemble de l'étape n° 2
(8 , 9 , 10, 11)	(0, 1 , 2, 3) (12 , 13 , 14, 15)
(2, 3 , 10, 11)	(6 , 7 , 14, 15) (0, 1 , 8, 9)
(10, 11, 14 , 15)	(2, 3, 6 , 7) (8, 9, 12 , 13)
(0 , 2)	(8, 10) (4 , 6) (1 , 3)

etape n°5: Elimination des représentations décimales n'appartenant pas au domaine d'entrée .

Pour cet exemple , on supposera que le domaine d'entrée correspond à l'ensemble exhaustif de tous les vecteurs possibles .

etape n°6: Choix d'un ensemble de vecteurs de test :

On choisit en priorité les élèments des ensembles ne contenant qu'un seul élèment : ici 3 , 0 , 1 , 1 , 1 .
On a donc déja trois vecteurs de test : 0 , 1 et 3 .

ensembles déja couverts par ces vecteurs :

(8 , 9)	(1)	(12 , 13)	
(3)	(6 , 7)	(1)	
(14 , 15)	(6 , 7)	(12 , 13)	
(0)	( )	(4 , 6)	(1)

On choisit ensuite parmi les ensembles non déja couverts , les vecteurs couvrant un maximum d'ensembles afin de minimiser le nombre de vecteurs de test :

Vecteur	nombre d'ensembles couverts
8	1
9	1
14	1
15	1
12	2
13	2
6	3
7	2
4	1

On ajoute le vecteur 6 à l'ensemble de test et on recommence :

liste des ensembles déja couverts :

(8 , 9)	(**1**)	(12 , 13)	
(**3**)	(**6** , 7)	(**1**)	
(14 , 15)	(**6** , 7)	(12 , 13)	
(**0**)	( )	(4 , **6**)	(**1**)

De la même manière que précédemment , on ajoute à l'ensemble de test le vecteurs : 12 (ou 13) puis 8 (ou 9) , et enfin 14 (ou 15).
l'ensemble de vecteurs de test ainsi constitué est :

(0 , 1 , 3 , 6 , 8 , 12 , 14)

etape n°7: si un ensemble est vide , on étudie le valeurs éliminées :

dans notre exemple , c'est le cas de l'ensemble des représentation décimales du monome adjacent à M_4 par rapport à la variable X_4.

Les deux valeurs contenues dans cet ensemble (8 et 10) ont été éliminée au cours de l'execution de l'algorithme .

*cas du vecteur 10 :

Ce vecteur apparait dans les ensembles relatifs aux monomes M_1 , M_2 e M_3. S'il n'y a pas de pannes :

$$M_1 = 1 \quad M_2 = 1 \quad M_3 = 1 \quad M_4 = 0$$

$$Z_1 = M_1 + M_2 + M_4 = 1 + 1 + 0 = 1$$

$Z_2 = M_3 + M_4 = 1 + 0 = 1$

Si le monome adjacent à M_4 par X_4 apparaît à la suite d'une panne :

$M_1 = 1 \quad M_2 = 1 \quad M_3 = 1 \quad M_4 = 1$

$Z_1 = M_1 + M_2 + M_4 = 1 + 1 + 1 = 1$

$Z_2 = M_3 + M_4 = 1 + 1 = 1$

Les sorties n'ont pas variées , la panne n'est pas détectée .

*cas du vecteur 8 :

Ce vecteur apparaît dans l'ensemble relatif à M_1 :

s'il n'y a pas de pannes :

$M_1 = 1 \quad M_2 = 0 \quad M_3 = 0 \quad M_4 = 0$

$Z_1 = M_1 + M_2 + M_4 = 1 + 0 + 0 = 1$

$Z_2 = M_3 + M_4 = 0 + 0 = 0$

Si le monome adjacent à M_4 par rapport à X_4 apparaît à la suite d'une panne :

$M_1 = 1 \quad M_2 = 0 \quad M_3 = 0 \quad M_4 = 1$

$Z_1 = M_1 + M_2 + M_4 = 1 + 0 + 1 = 1$

$Z_2 = M_3 + M_4 = 0 + 1 = 1$

La panne sera détectée sur la sortie Z2 et le vecteur 8 peut être rajouté à l'ensemble des vecteurs de test . On obtient ainsi l'ensemble des vecteurs de test permettant de tester le PLA :

(0 , 1 , 3 , 6 , 8 , 12 , 14)

Dans certains cas , il se peut , qu'aucun vecteur de test ne puisse être choisi pour détecter certaines pannes . Le taux de couverture des pannes s'en trouvera donc réduit . En règle générale , sauf redondance massive ou domaine d'entrée trés restreint , on obtient une couverture de pannes de l'ordre de 99 % (voir table des résultats expérimentaux) .

d) Le programme :

Un programme (**RDPLA**) calculant automatiquement des vecteurs de test pour un PLA , à partir de cette méthode a été écrit , en Pascal . Il occupe 1500 lignes de programme source (environ 68 Koctet de code compilé) .

Dans (**IBA 75**) il a été démontré , que le temps de calcul des meilleurs algorithmes calculant des vecteurs de test pour un circuit combinatoire , croissait exponentiellement , dans le pire des cas , en fonction du nombre d'entrée et de portes logiques du circuit . Dans un soucis de minimisation du temps de calcul , et de l'espace mémoire occupé , le choix final des vecteurs n'est pas effectué . Un vecteur de test est choisi dans chacun des ensembles constitués . Le gain en temps CPU est appréciable par rapport à l'accroissement du nombre des vecteurs de test (voir table des résultats expérimentaux) .

1 - Les principales procédures du programme :

Le programme est donné en annexe 1 , alors que son organigramme est en figures 9a , 9b , 9c .

Procédure CLASSER : Procédure ordonnant les monomes , suivant :

1° le nombre de ses connexions avec les sorties

2° le nombre de variables d'entrée qui lui sont connectées .

Comme la propagation d'une erreur doit être effective jusqu'à au moins une sortie , les monomes fortement connectés aux sorties , sont susceptibles , plus que les autres , de masquer les effets des pannes . Ceux-ci seront étudiés les premiers , afin de minimiser le temps de calcul .

Procédure CREER-LISTE : Procédure servant à créer un ensemble de vecteurs de test , compatibles avec le monome dont on veut tester la présence , et incompatible avec tous les autres monomes du PLA .

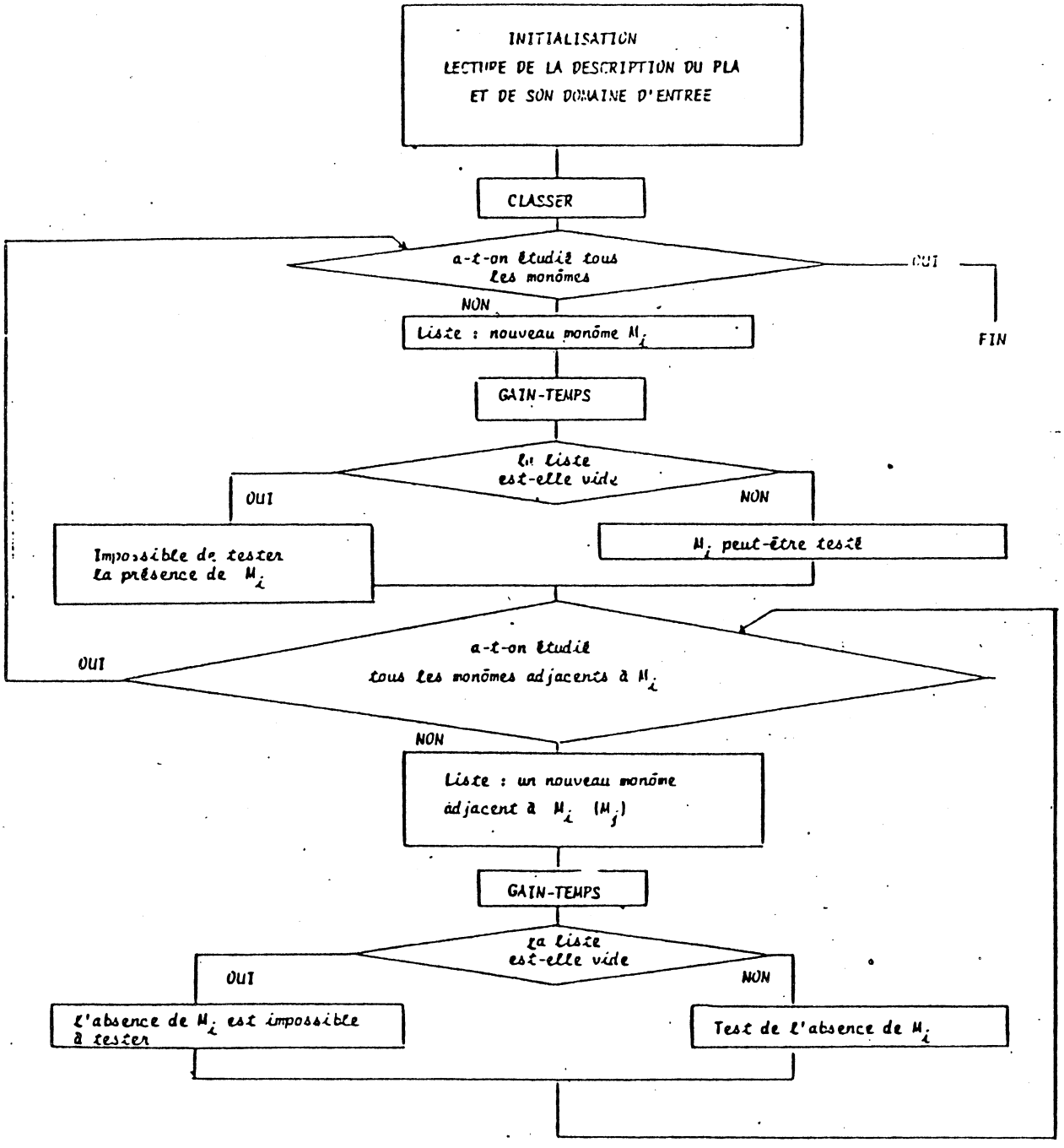
Procédure GAIN-TEMPS : Pour un monome donné , cette procédure récursive , crée un ensemble de vecteurs de test , appartenant au domaine d'entrée . Un tel ensemble croissant exponentiellement , au cours du déroulement de l'algorithme , il est divisé aussitôt que sa taille dépasse une taille de référence . La procédure est alors relancée récursivement avec chacune des parties de l'ensemble ainsi divisé , jusqu'à ce qu'un vecteur de test convenable soit trouvé .

Cette procédure permet de limiter l'espace mémoire utilisé , et de réduire le temps de calcul si un vecteur de test convenable existe .

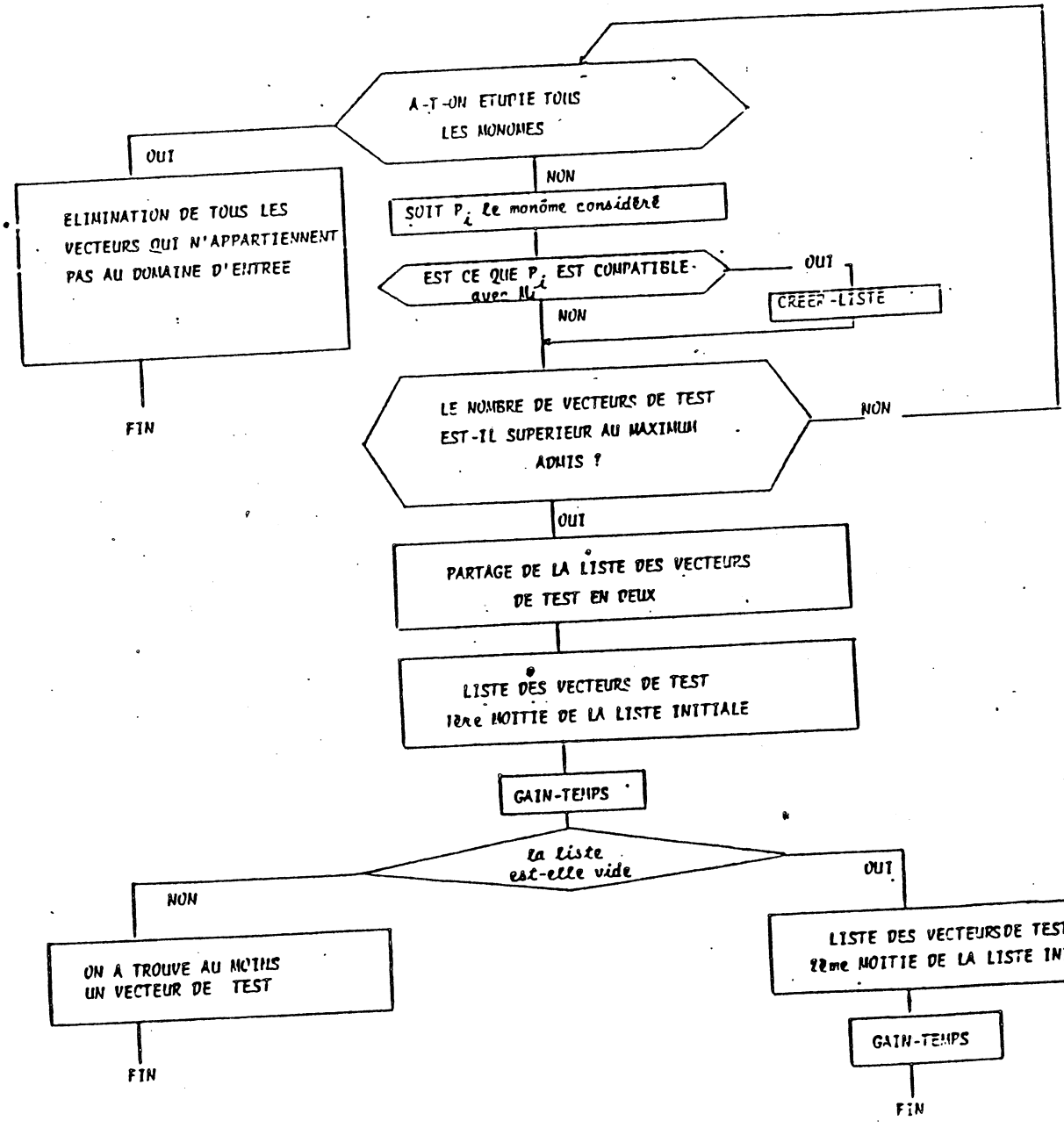
e) Résultats expérimentaux :

La table 5 résume les résultats obtenus sur des PLAs aussi bien fictifs que réels (PLAs du 68000) de tailles variés .

ORGANIGRAMME GENERAL

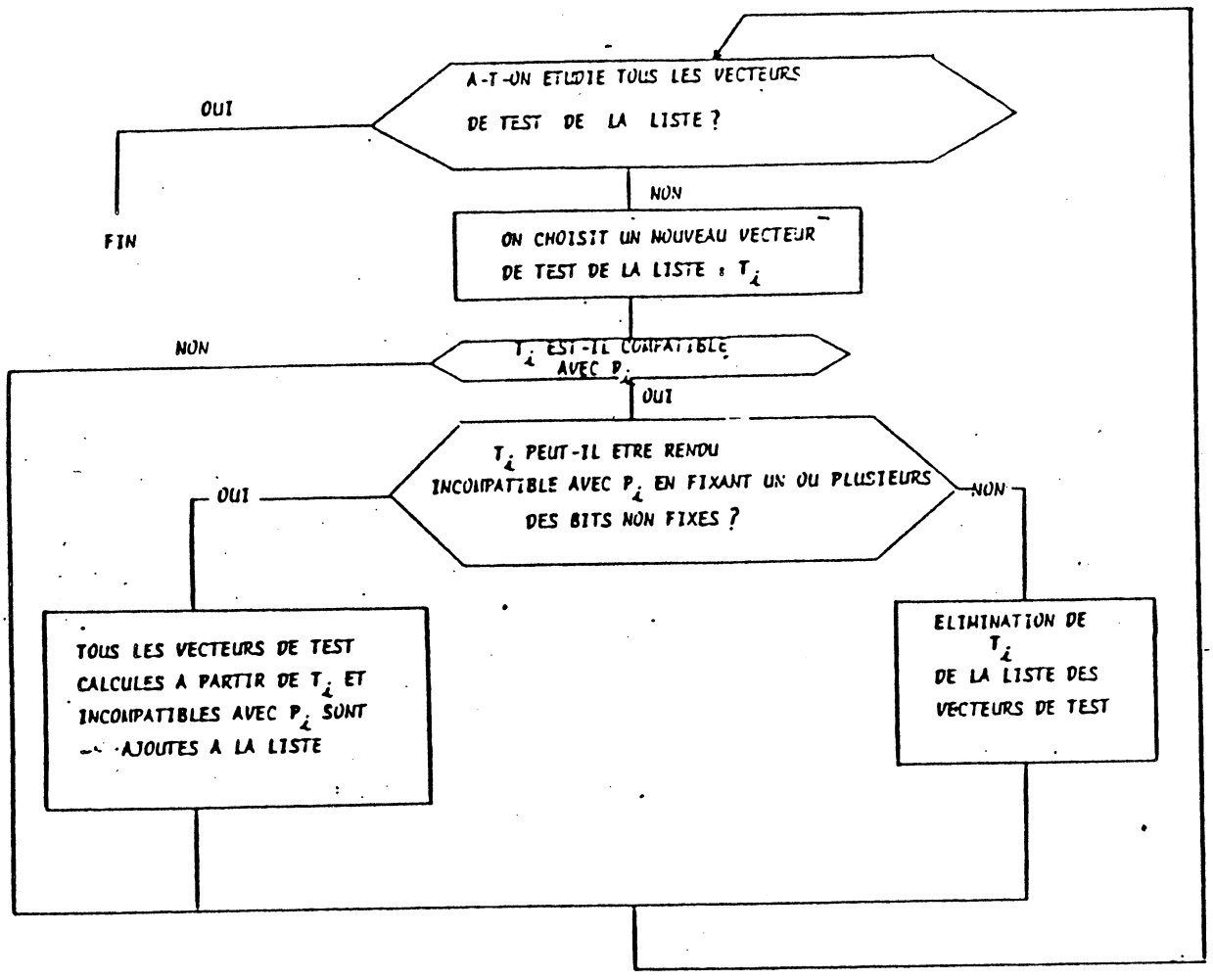


PROCEDURE RECURSIVE UTILISEE POUR DETERMINER UN VECTEUR DE TEST POUR UN MONOME AUSSI VITE QUE POSSI



CREER LISTE

GENERATION DE VECTEURS DE TEST POUR UN MONOME M_i COMPATIBLE AVEC UN MONOME P_i



nb de pannes à détecter	nb de monomes	type NMOS/CMOS	domaine d'entrée	temps de calcul (s)	nb de vecteurs de test	couverture des pannes	commentaires à propos des PLAs testés
128	4	NMOS	100%	0 11	9	99.19%	exemple n°1 (techno NMOS)
128	4	CMOS	100%	0.18	15	99.19%	exemple n°1 (techno CMOS)
593	18	NMOS	100%	0 87	45	98.15%	PLA de branchement du 68000
593	18	CMOS	100%	2.4	55	98.15%	PLA de branchement du 68000
690	18	NMOS	100%	4.31	44	96.8%	exemple n°2
690	18	CMOS	100%	11 86	62	96.8%	exemple n°2
1418	36	NMOS	100%	12.21	90	95.3%	exemple n°3
1418	36	CMOS	100%	147.8	127	95.3%	exemple n°3
1536	46	NMOS	96%	49 58	130	94 82%	PLA A1 de HSURF
1803	37	NMOS	82 81%	7 84	54	93 62%	PLA A2 de HSURF
2197	29	NMOS	56.9%	36 18	92	97 63%	PLA de branchement de HSURF
5699	60	NMOS	100%	71 15	55	99 73%	exemple n°4 sans domaine d'entrée
5699	60	NMOS	17 2%	208 47	61	99.63%	exemple n°4 avec domaine d'entrée
7881	77	NMOS	96%	356.21	98	88.37%	PLA A2 du 68000

Les vecteurs de test obtenus , pour certains de ces PLAs sont donnés en annexe 2.
Une comparaison a été effectuée avec un programme utilisant le même algorithme , sans possibilité de définir un domaine d'entrée , mais calculant un ensemble minimal de vecteurs de test . En moyenne , le gain en temps du programme présenté ici , est de l'ordre de 67 % , alors que le gain en nombre de vecteurs de test de l'autre programme n'est que de 10 %. Le gain de temps est donc appréciable lors de l'étude de gros PLAs (nombre de monomes >= 20).

La comparaison des résultats obtenus avec ceux obtenus par d'autres algorithmes calculant également un ensemble de vecteurs de test pour des PLAs, par une autre approche que celle présentée ici est irréalisable. En effet, la notion de domaine d'entrée, dans lequel on choisit les vecteurs de test n'a pas été développé dans ces autres algorithmes, et fausserait toute comparaison, notemment sur les temps de calcul, la place mémoire occupée et la couverture des pannes.

Afin de localiser toute panne simple pouvant éventuellement se produire dans un PLA, un programme donné en annexe, simulant les effets de l'ensemble des pannes possibles, et éliminant celles n'ayant pu se produire a été écrit. Cette localisation est utile principalement dans le cas où le PLA sous test est reconfigurable **(DAN 85)**. De l'ensemble des pannes possibles, une heuristique permet de déterminer la panne la plus probable, et donc de tenter, en reconfigurant le circuit, de l'annuler.

PARTIE 4

Le multiplieur

I - Choix d'un multiplieur cellulaire :

L'un des buts du micro-processeur HSURF , étant le traitement de matrices bidimentionnelles , le calcul d'adresse de l'un des éléments d'une matrice est obtenue à l'aide d'une multiplication et de deux additions :

Soit la matrice à m+1 lignes et n+1 colonnes :

$$M = \begin{pmatrix} A_{00} & A_{01} & \cdots & A_{0n} \\ A_{10} & A_{11} & \cdots & A_{1n} \\ \vdots & \vdots & & \vdots \\ A_{m0} & A_{m1} & \cdots & A_{mn} \end{pmatrix}$$

Cette matrice sera rangée en mémoire ligne par ligne :

adresse de début : A_{00}

" + 1 : A_{01}

.

.

.

" + n : A_{0n}

" + n + 1 : A_{10}

.

.

.

.

" + m * (n+1) + n : A_{mn}

L'adresse en mémoire de l'élément A_{ij} avec $0 <= i <= m$

$0 <= j <= n$ est obtenue par l'équation :

adr (A_{ij}) = adresse de début + i * nombre de colonnes + j

(ici n+1)

La réalisation de ce calcul pouvait être réalisé de deux manières :

a) Multiplication réalisée par additions et décalage , et controllée par micro-programme :

Exemple d'algorithme pouvant être utilisé pour réaliser une multiplication :

```
début

        (* réalisation de la multiplication de A par B *)

        si B impair alors prod := A ;
                     sinon prod := 0 ;
        tant que B > 1 faire
        début
                B := B div 2 ; (* décalage à droite de B *)
                A := A * 2 ;  (* décalage à gauche de A *)
                si B impair alors prod := prod + A ;
        fin ;
fin .
```

Le microprogramme , exécutant ce type d'algorithme , peut être exécuté par le controleur du microprocesseur , ou à partir d'un controleur spécialisé , activé par le controleur de HSURF . Dans ce deuxième cas , le processeur peut éventuellement effectuer d'autres opérations , durant l'exécution de la multiplication , ceci , afin de réduire le temps d'exécution de l'instruction en cours de traitement .

Les avantages et les inconvénients de cette méthode :

- Avantages :

* Il y a une perte minime de place sur le circuit , puisque l'on utilise principalement l'UAL de la partie opérative .

- Inconvénients :

* Perte de temps : le nombre , souvent non négligeable , d'additions et de décalages nécessaires au déroulement de l'opération occupe un nombre de cycles d'horloge , qui ralentit toute les instructions de traitement des matrices .

* Difficultés pour le test : Il est trés difficile de tester les

bonnes communications entre la partie controle et la partie opérative . Hors cette solution accroit ces communications : test de B > 1 et de B impair .

b) Multiplication réalisée par un opérateur spécialisé : un multiplieur cellulaire : **(DIA 76) , (FRI 73) , (PAR 81) , (FRI) (SHE 83) , (SHE 84)**

Dans cette solution , il suffit de charger les registres en entrée de cet opérateur , et l'on obtient le résultat de l'opération sur ses sorties .

Les avantages et les inconvénients de cette méthode :

- Avantages :

* Rapidité d'exécution : la multiplication est réalisée en une seule microinstruction , quels que soient les opérandes à multiplier .

* Facilité de test : La structure régulière et itérative de tels opérateurs , facilite le calcul de vecteurs de test à leur appliquer pour permettre de les tester efficacement .

- Inconvénients :

* Perte de place : l'opérateur occupe sur le circuit , une place importante qui même réduite au strict minimum , est sans commune mesure avec la place perdue dans la solution précédente .

Notre but étant un microprocesseur facilement testable , tout en étant efficace , la deuxième solution a été choisie , l'augmentation de la surface du circuit , étant acceptable , face à la rapidité des instructions de traitement de matrices , et au test de l'opérateur .

2) Le multiplieur :

a) Principe :

Une multiplication s'effectue par une suite d'additions en "cascade" :
Exemple :

```
                1 0 1 1        (11)
              *                *
                1 1 0 1        (13)
              ---------
                1 0 1 1
            +
              0 0 0 0
            -------------
              0 1 0 1 1
          +
            1 0 1 1
          ---------------
            1 1 0 1 1 1
        +
          1 0 1 1
        -----------------
      1 0 0 0 1 1 1 1        (143)
```

b) Réalisation :

Ces additions successives , peuvent être réalisée , à partir de cellules de type "full-adder" (figure 2) , assemblées de maniére à obtenir les résultats partiels des additions cascadées , ligne par ligne , jusqu'au résulat final (figure 1).

Figure 1

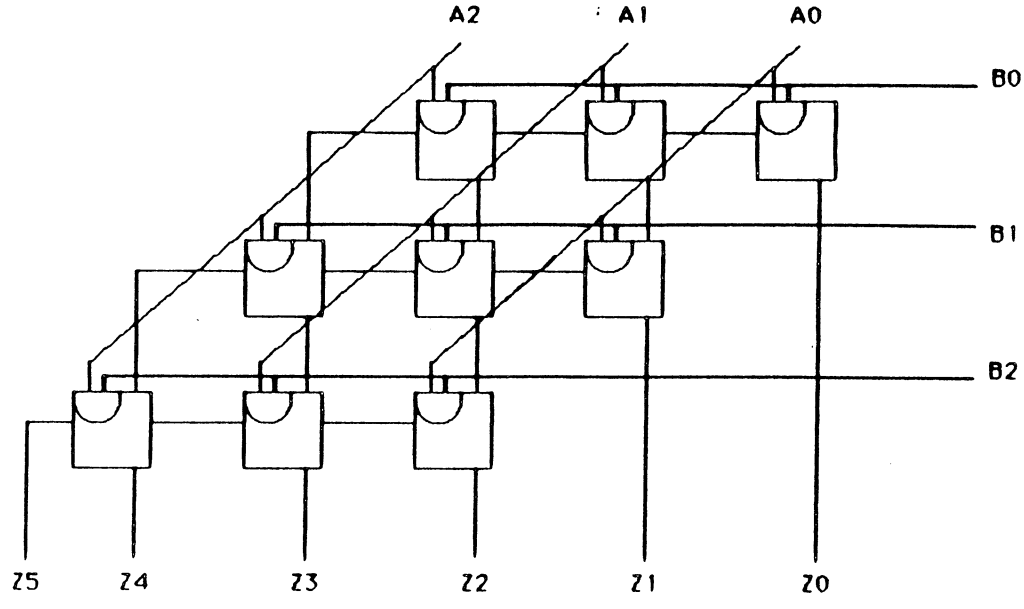


Figure 2 : La cellule de base du multiplieur cellulaire :

$$C_{out} = C_{in} \cdot R_{in} + (A_i \cdot B_j) \cdot C_{in} + (A_i \cdot B_j) \cdot R_{in}$$
$$R_{out} = (A_i \cdot B_j) + C_{in} + R_{in}$$

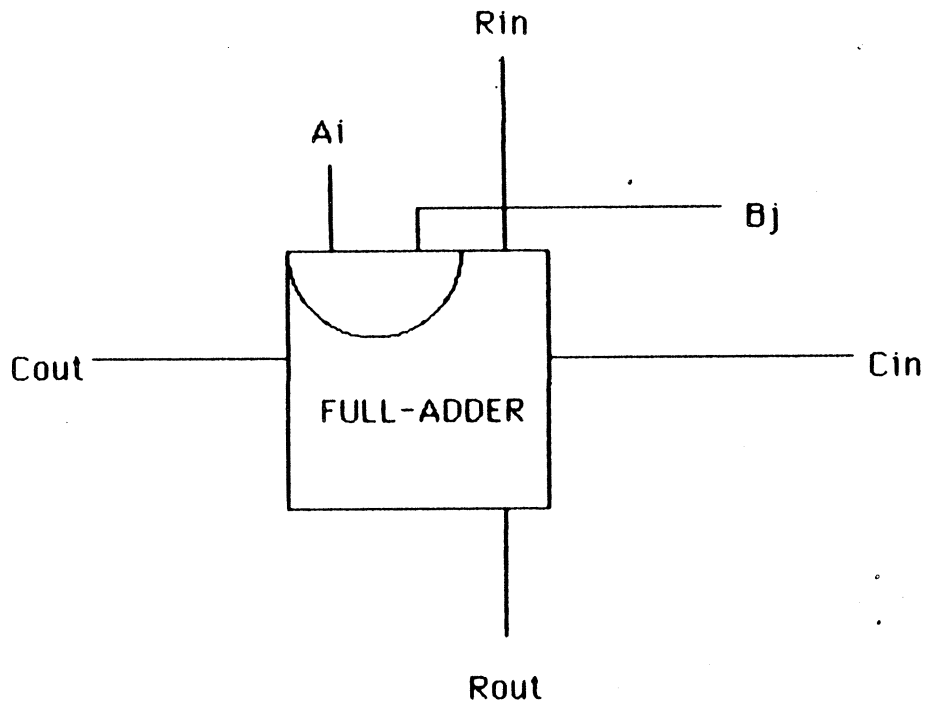


Figure 2

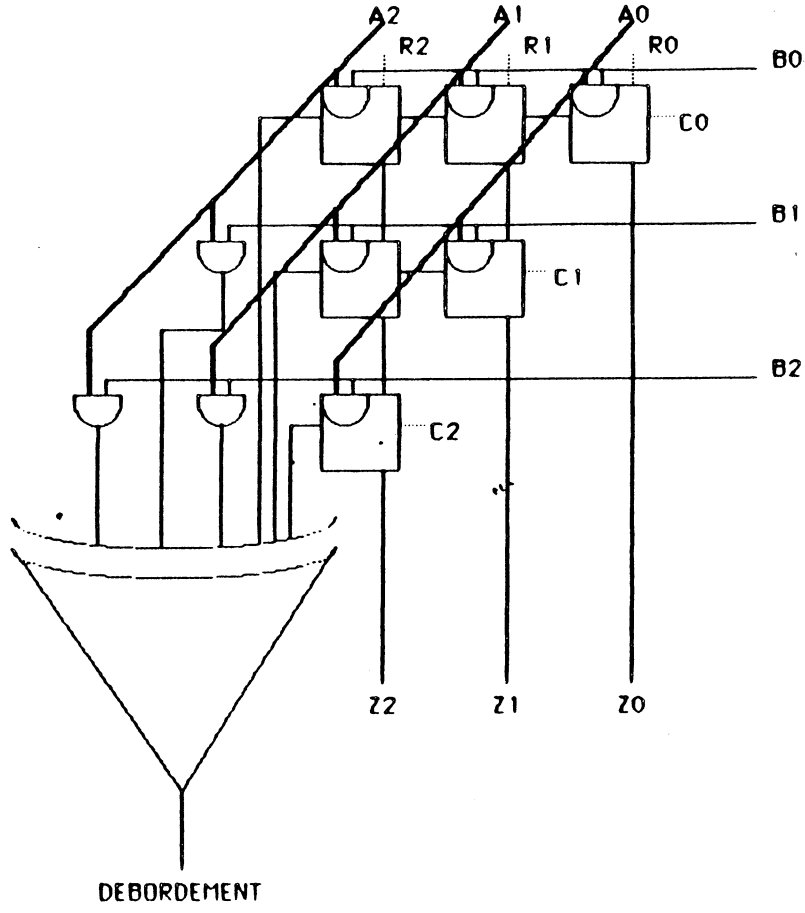
Si toutes les cellules sont identiques (en particulier celles du bord) , en faisant varier les retenues entrantes et les résultats entrants , ce type d'opérateur permet de réaliser la fonction :

$$Z = A * B + C + R$$

Cette fonction , est celle nécessaire au calcul de l'adresse d'un élément de matrice . Le gain en temps est donc accru , par rapport à une multiplication par additions et décalages , suivie obligatoirement de deux nouvelles additions .

Dans ce type d'opérateur , la multiplication de deux nombres de n bits , donne un résultat sur 2*n bits . Pour les besoins de HSURF , les opérandes étant sur 16 bits , le résultat obtenu est sur 32 bits . Mais , du fait de la pagination réalisé par le processeur , là taille des matrices doit être telle , qu'elle tienne , au maximum , sur une page mémoire de 64 Kmots . L'adresse à obtenir n'est donc plus que sur 16 bits , ce qui permet de

sectionner une partie du multiplieur , et de le remplacer par un détecteur de débordement (Figure 3).



Le multiplieur total utilisait 16 * 16 = 256 cellules de base , alors que le multiplieur réduit de HSURF , n'en nécessite que 136
(16 + 15 + ... + 1 = (17 * 16) / 2 = 136 cellule).

3) Test du multiplieur :

Le nombre de bits en entrée du multiplieur est de 4*16 = 64 bits . Un test exhaustif du multiplieur est donc impossible , le temps et la place qu'occuperait un test avec 2^{64} vecteurs d'entrée étant inconcevable . Cependant , chaque cellule du multiplieur ne posséde que 4 entrées . Un test axhaustif de chaque cellule est donc envisageable : la structure régulière du multiplieur permettant de tester simultanément chaque cellule avec une configuration d'entrée . Pour ce test , on suppose que :

-Au plus une cellule est en panne à un instant donné .
- La panne est une panne permanente .

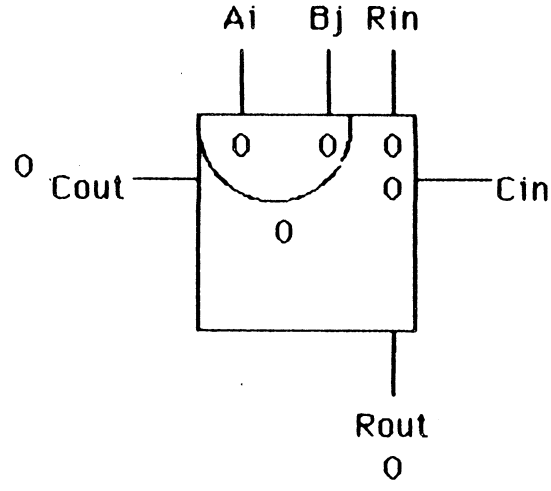
Avec de telles hypothèses , le test exhaustif de chaque cellule permet de détecter toute panne dans le multiplieur .

- Calcul des vecteurs de test :

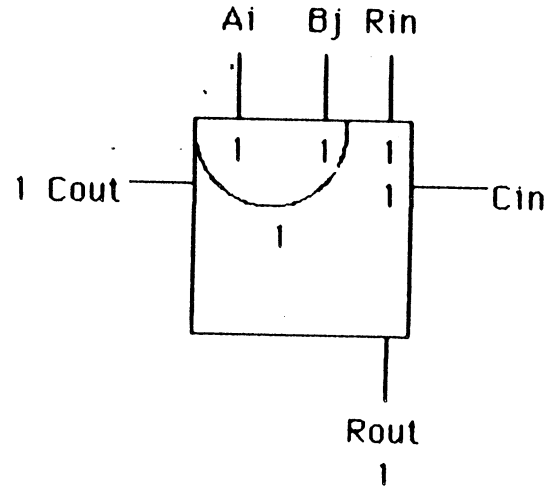
a) Vecteurs se propageant de cellule à cellule:

exemple :

Toutes les entrées à 0 :



Toutes les entrées à 1 :



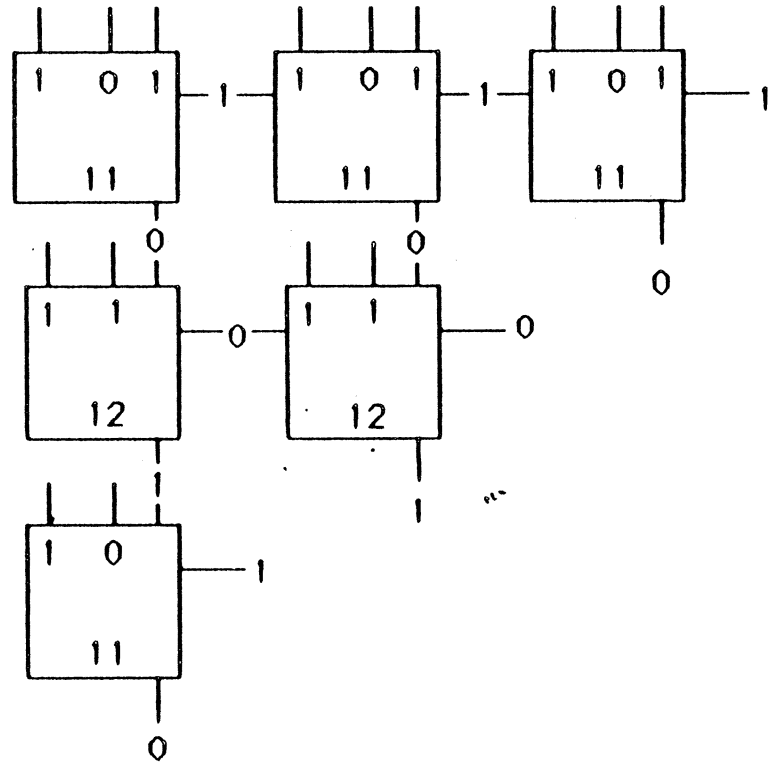
Le tableau suivant , donne l'ensemble des valeurs qui se propagent de cellule à cellule :

A	B	R_{in}	C_{in}
< 00 . . 00 >	< 00 . . 00 >	< 00 . . 00 >	< 00 . . 00 >
< 00 . . 00 >	< 00 . . 00 >	< 11 . . 11 >	< 00 . . 00 >
< 00 . . 00 >	< 11 . . 11 >	< 11 . . 11 >	< 00 . . 00 >
< 11 . . 11 >	< 00 . . 00 >	< 00 . . 00 >	< 00 . . 00 >
< 11 . . 11 >	< 00 . . 00 >	< 11 . . 11 >	< 00 . . 00 >

b) Vecteurs de test se propageant à "une cellule sur deux" :

exemple :

A = < 11 . . 11 > B = < 10 . . 10 > R_{in} = < 11 . . 11 > C_{in} = < 01 . . 01 >



Les cellules des étages impairs (1 , 3 , 5 . . .) sont testées pour les entrées < 1 0 1 1 > , alors que les cellules des étages pairs (2 , 4 , 6 . . .), sont testées pour les entrées < 1 1 0 0 > .

De même , avec les entrées :

A = < 11 . . 11 > B = < 01 . . 01 > R_{in} = < 00 . . 00 > C_{in} = < 10 . . 10 >

les cellules des étages impairs , sont testées pour les entrées < 1 1 0 0 >

et les cellules des étages pairs , sont testées pour les entrées < 1 0 1 1 >. Avec deux vecteurs de test , chaque cellule est testée pour deux configurations de ses entrées . Le tableau suivant indique les vecteurs d'entrée de ce type :

A	B	R_{in}	C_{in}
< 11..11 >	< 10..10 >	< 11..11 >	< 01..01 >
< 11..11 >	< 01..01 >	< 00..00 >	< 10..10 >
< 10..10 >	< 11..11 >	< 10..10 >	< 00..00 >
< 01..01 >	< 11..11 >	< 01..01 >	< 00..00 >

c) Vecteurs de test se propageant à "une cellule sur quatre" :

A	B	R_{in}	C_{in}
< 10..10 >	< 11..11 >	< 01..01 >	< 01..01 >
< 01..01 >	< 11..11 >	< 10..10 >	< 01..01 >
< 10..10 >	< 11..11 >	< 00..00 >	< 10..10 >
< 01..01 >	< 11..11 >	< 00..00 >	< 10..10 >

Avec les 13 vecteurs précédents , chaque cellule est testée pour 13 des 16 valeurs possibles de ses entrées .
Pour les 3 valeurs restantes , le nombre de vecteurs de test est proportionnel à la taille des opérandes .
La valeur d'entrée < 0 0 1 1 > se propage à toutes les cellules d'une même ligne . Il faut donc autant de vecteurs de test qu'il y a de lignes (et donc que de bits dans un opérande) . Les vecteurs d'entrée < 0 0 0 1 > et < 1 0 0 1 > se propagent trés difficilement de cellule à cellule , aussi faut-il un nombre de vecteurs de test qui dépend directement du nombre de cellules à tester .
Il en résulte que le nombre de vecteurs de test dépend de la taille des opérandes à multiplier. La liste de ces vecteurs , pour le multiplieur 16 bits de HSURF , est donné en annexe .

4) Modifications à apporter au multiplieur pour minimiser le nombre de vecteurs pour le test :

Il est possible de réduire le nombre de vecteurs nécessaires pour tester exhaustivement chaque cellule du multiplieur de HSURF . Pour cela , deux conditions sont indispensables :

- Modifier la cellule de base pour favoriser la propagation des valeurs de test .
- Forcer , en cours de fonctionnement , les valeurs des entrées externes C_{in} et R_{in} à 0 .

Sous ces conditions , seuls 16 vecteurs sont nécessaire au test , quelle que soit la taille des opérandes .

* Modifications de la cellule de base :

Seule l'équation de la retenue est modifiée :

$$C_{out} = C_{in} \cdot R_{in} + A_i \cdot C_{in} + \overline{A_i} \cdot \overline{B_j} \cdot C_{in} + A_i \cdot B_j \cdot R_{in}$$

ce qui donne la table de vérité suivante :

Cellule normale : C_{in}

$A_i B_j$ \ $R_{in} C_{in}$	10	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	0

Cellule modifiée : C_{in}

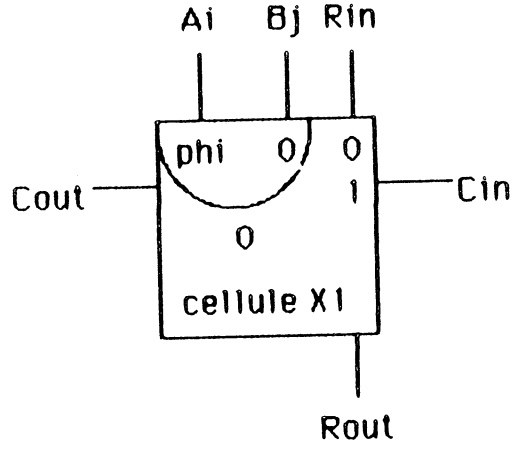
$A_i B_j$ \ $R_{in} C_{in}$	10	01	11	10
00	0	1*	1	0
01	0	0	1	0
11	0	1	1	1
10	0	1*	1	0

Les 16 vecteurs de test sont alors :

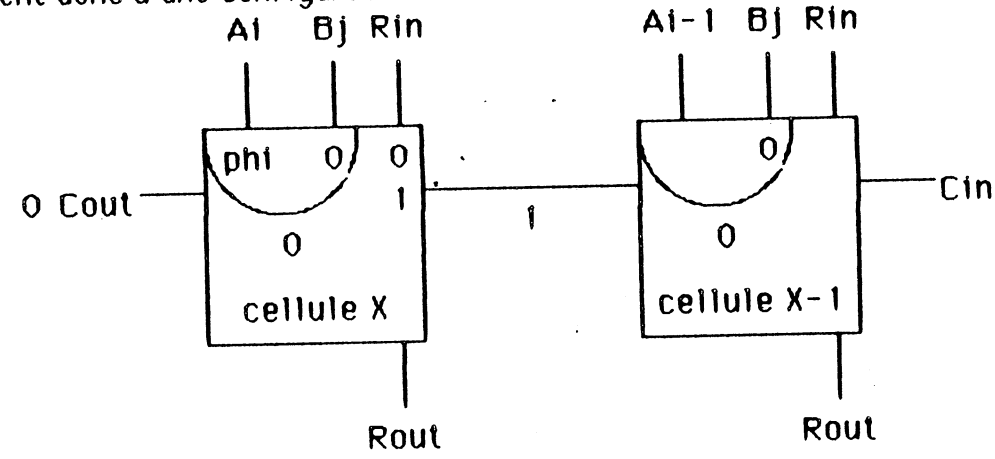
A	B	R_{in}	C_{in}	valeurs des cellules
<00..00>	<00..00>	<00..00>	<00..00>	toutes à <0000>
<00..00>	<00..00>	<11..11>	<00..00>	toutes à <0010>
<00..00>	<00..00>	<11..11>	<11..11>	rangées impaires à <0011> rangées paires à <0001>

A	B	R_{in}	C_{in}	valeurs des cellules
‹00..00›	‹00..00›	‹00..00›	‹11..11›	rangées impaires à ‹0001› ; rangées paires à ‹0011›
‹01..01›	‹11..11›	‹01..01›	‹00..00›	alternance entre : ‹1110› et ‹0101›
‹10..10›	‹11..11›	‹10..10›	‹11..11›	alternance entre : ‹0101› et ‹1110›
‹00..00›	‹11..11›	‹11..11›	‹00..00›	toutes à ‹0110›
‹11..11›	‹00..00›	‹00..00›	‹00..00›	toutes à ‹1000›
‹11..11›	‹00..00›	‹00..00›	‹00..00›	rangées impaires à ‹1001› ; rangées paires à ‹1011›
‹11..11›	‹00..00›	‹11..11›	‹11..11›	rangées impaires à ‹1011› ; rangées paires à ‹1001›
‹11..11›	‹00..00›	‹11..11›	‹00..00›	toutes à ‹1010›
‹11..11›	‹11..11›	‹11..11›	‹11..11›	toutes à ‹1111›
‹10..10›	‹11..11›	‹01..01›	‹01..01›	toutes les cellules sont testées pour : ‹0100›‹0111› ‹1100›‹1110› (1 sur 4 pour chaque vecteur de test).
‹01..01›	‹11..11›	‹10..10›	‹01..01›	
‹10..10›	‹11..11›	‹00..00›	‹10..10›	
‹01..01›	‹11..11›	‹00..00›	‹10..10›	

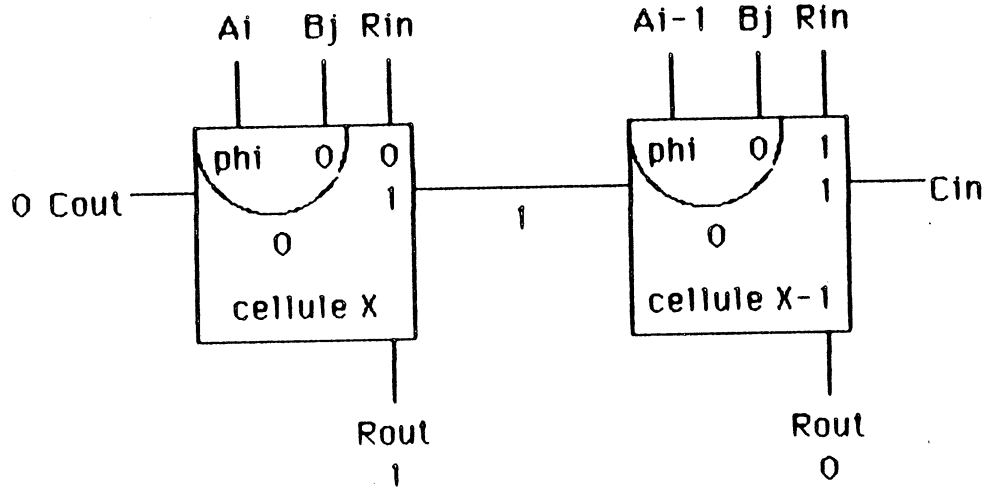
Les modifications apportées à la cellule de base , ne perturbent pas le fonctionnement de l'opérateur , si les entrées externes C_{in} et R_{in} sont nulles : en effet , pour qu'une cellule reçoive une des entrées modifiées : < 0 0 0 1 > ou < 1 0 0 1 > :



Cette cellule ne peut pas être une cellule du bord puisque C_{in} vaut 1 , on en revient donc à une configuration :



les valeurs des entrées telles que $B_j = 0$, avec $C_{out} = 1$ sont : < 0 0 1 1 > ou < 1 0 1 1 > , ce qui donne :



Le problème de la retenue entrante à 1 avec $B_j = 0$ est repoussée à la cellule de droite , et ainsi , de cellule en cellule , jusqu'au bord du multiplieur . La condition $C_{in} = \langle 00 \ldots 00 \rangle$ implique donc que les valeurs $\langle 0\,0\,0\,1 \rangle$ et $\langle 1\,0\,0\,1 \rangle$ n'arriveront jamais sur aucune des cellules du multiplieur en fonctionnement normal . Les modifications apportées à la cellule de base sont transparentes , sauf , au cours du test , lorsque les valeurs de C_{in} et R_{in} seront modifiées .

Conclusion :

Du fait de la condition $R_{in} = C_{in} = \langle 00 \ldots 00 \rangle$, lors de l'usage normal de l'opérateur , la fonction réalisée par le multiplieur est :

$$Z = A * B .$$

On a perdu un des principaux avantages : le calcul direct de l'adresse d'un élément de matrice , et donc la rapidité de calcul . La place occupée sur le circuit est telle que le multiplieur initial a été préféré , même s'il demande un temps de test légèrement plus long .

CONCLUSION:

Les outils développés ici, pour rendre le microprocesseur HSURF facilement testable (opèrateurs facilement testables, mèthode de test des PLAs, automate de test spécialisé, stratégie de test ...), peuvent être réutilisés

- pour tester des microprocesseurs deja conçus (MC68000)

- pour developper d'autres circuits facilement testables

En effet, le programme générant pour un PLA, un ensemble de vecteurs de test choisis parmi un domaine d'entrée restreint, permet de tester tout PLA quelque soit le circuit auquel il est intégré. Ce programme a été validé, non seulement sur les PLAs de HSURF, mais également sur les PLAs de séquencement du MC68000, qui ont un domaine d'entrée très restreint à cause de la difficulté d'observation de leurs sorties.

La stratégie de test, elle même a été appliquée au 68000, avec un partitionnemen moins fin (blocs plus importants).

REFERENCES

(ASH 77) M.ASHJEE & S.REDDY

"On totally self checking checkers for separable codes"

IEEE transaction on computers vol C20 aout 77

pp 737-745

(AYA 79) J.AYACHE , P.AZEMA & M.DIAZ

"Observer : a concept for on-line detection of control errors in concurrent systems"

FTCS 9th juin 79 pp 79-86

(BEL 82) C.BELLON & G.SAUCIER

"Protection against external errors in a dedicated system"

IEEE transaction on computers vol C31 avril 82

pp 311-317

(BEL 84) C.BELLON

"Le test fonctionnel des circuits intégrés complexes"

Thèse pour obtenir le grade de docteur és sciences en informatique, soutenue à l'université de Grenoble Octobre 84

(BOS 82) B.BOSC & T.R.NRAQ

"Theory of unidirectional error correcting/detecting codes"

IEEE transaction on computers Vol C31 juin 82

pp 520-530

(CHA 82) J.CHAVADE & Y.CROUZET
"The P.A.D. : a self-checking LSI circuit for fault detection in micro-computers"
FTCS 12th juin 82 pp 55-62

(CHA 83) R.CHANDRAMOULI
"On testing stuck-open faults"
FTCS 13th Milan juin 1983 pp 258-265

(CHI 83) K.W.CHIANG & Z.G.VRANESIC
"On fault detection in logic networks"
20th DAC Miami juin 83 pp 50-56

(COU 79) B.COURTOIS
"Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunctions"
FTCS 9th juin 1979 pp 71-74

(COU 81) B.COURTOIS
"A methodology for on-line testing of microprocessors"
FTCS 11th juin 1981 pp 272-274

(CRO 80) Y.CROUZET & C.LANDRAULT
"Design of a self checking detection processor"
FTCS 10th octobre 80 pp275-277

(DAN 85) A.DANDACHE

"Etude de structures régulières PLA/ROM dans la partie contrôle de microprocesseurs"

Rapport de recherche n°516 avril 1985

(DAV 78) R.DAVID

"Feedback shift register testing"

FTCS 8th Toulouse juin 78 pp 103-107

(DIA 75) H.DIAZ & J.M. de SOUZA

"Design of self-checking microprogrammed controls"

FTCS 5th 1975 pp 137-142

(DIA 76) F.J.O.DIAZ

"Truth-table verification of an iterative logic array"

IEEE transaction on computers vol. C25 nb 6 juin 76 pp 605-613

(DON 82) H.DONG

"Modified Berger codes for detection of unidirectional errors"

FTCS 12th Santa-Monica juin 82 pp 317-320

(EIC 80) E.B.EICHELBERGER & E.LINDBLOOM

"A heuristic test pattern generator for PLA"

IBM research development Vol 24 n°1 jan 80

(FRI) FRIEDMAN & MENON
"Fault detection in digital circuits"
Iterative logic arrays Chap. 4 pp 106-150

(FRI 73) A.D.FRIEDMAN
"Easily testable iterative systems"
IEEE transaction on computers vol. C22 dec. 73 pp 1061-1064

(FRI 78) A.FRIEDMAN & M.MAROUF
"Efficiently designed self checking checker for only m-out-of-n code"
IEEE transaction on computers vol C27 juin 78 pp 482-490

(FUJ 81) H.FUJIWARA & K.KINOSHITA
"A design of PLA with universal tests"
IEEE transaction on computers Vol C30 n°11 nov 81

(GEN 84) P.GENESTIER & J.F.POIRIER
"Conception d'un microprocesseur pour des applications de haute sécurité : 'HSURF'"
DEA d'informatique , Grenoble , Mars 1984

(HAS 84) S.Z.HASSAN & E.J.Mc CLUSKEY
"Increased fault coverage through multiple signatures"
14th FTCS Kissimmee juin 84 pp 354-359

(IBA 75) O.H.IBARRA & S.K.SAHNI
"Polynomially complete fault detection problems"
IEEE transaction on computers Vol C25 n°3 mars 75

(IYE 82) S.IYENGAR & L.KINNEY
"Concurrent testing of flow of control in simple microprogrammed units"
Proc. international test conference nov 82

(JAY 84) C.JAY , G.SAUCIER & P.GENESTIER
"A testable microprocessor for process control"
Proc. of international conference on computer design New-York oct. 84 pp 284-289

(JAY 85) C.JAY & P.CHAISEMARTIN
"Conception d'un circuit de signature"
1er colloque national conception de circuits à la demande Grenoble mai 85 pp 131-147

(KAN 75) J.KANE & S.YAN
"Concurrent software fault detection"
IEEE transaction software engineering mars 75
pp 87-93

(LOT 85) Z.LOTFI
"Etude et analyse des schemas logiques combinatoires à l'aide de représentations numériques et graphiques"
Thése de docteur es sciences Nancy fev. 85

(MAK 82) G.P.MAK , J.A.ABRAHAM & E.S.DAVIDSON
"The design of PLAs with concurrent error detection"
FTCS 12th Santa-Monica juin 1982 pp 303-310

(MAR 78) M.A.MAROUF & A.D.FRIEDMAN
"Design of self-checking checkers for Berger codes"
FTCS 8th Toulouse juin 78 pp 179-184

(MAR 82) P.MARCHAL & B.COURTOIS
"On detecting the hardware failures disrupting programs in microprocessors"
FTCS 12th juin 1982 pp 249-256

(MCC 82) E.J.Mc CLUSKEY & M.NAMJOO
"Watchdog processors and capability checking"
FTCS 12th juin 1982 pp 245-248

(MET 81) G.METZE & A.MILI

"Self checking programs : an axiomatization of program validation by executable assertions"

FTCS 11th juin 81 pp 118-120

(NAM 82) M.NAMJOO

"Technique for concurrent testing of VLSI processor operation"

International test conference nov 82 pp 461-468

(NAM 83) M.NAMJOO

"Cerebus 16: An architecture for a general purpose watchdog processor"

FTCS 13th juin 83 pp 216-219

(OST 78) D.L.OSTAPKO & S.J.HONG

"Fault analysis and test generation for PLA"

FTCS 8th Toulouse juin 78 pp 83-89

(PAR 77) B.PARHAMI

"The concept of self checking programs"

FTCS 7th juin 77 p 216

(PAR 81) R.PARTHASARATHY & S.M.REDDY
"A testable design of iterative logic array"
IEEE transaction on circuits and systems
Vol. CAS 28 nb 1.1 nov 81 pp 1037-1045

(PIL 82) E.PILAUD
"Conception et validation de systèmes informatiques à haute sureté de fonctionnement"
Thèse pour obtenir le grade de docteur ingénieur, soutenue à Grenoble novembre 82

(RED 84) S.M.REDDY & M.K.REDDY
"Robust tests for stuck-open faults in CMOS combinational logic circuits"
FTCS 14th Kissimmee juin 84 pp 44-49

(SAL 81) K.SALUJA , K.KINOSHITA & H.FUJIWARA
"A multiple fault testable design of PLA"
FTCS 11th Portland juin 81 pp 44-46

(SAU 84) G.SAUCIER & C.JAY

"Conception et utilisation d'un microprocesseur spécialisé de sécurité"

4ème colloque international Fiabilité et Maintenabilité Perros-Guirec France mai 84 pp 561-566

(SEV 84) M.SEVESTRE

"Validation d'un système de sécurité ferroviaire à base de microprocesseur"

4ème colloque int. Fiabilité et Maintenance Perros-Guirrec 1984

(SHE 83) J.P.SHEN & F.J.FERGUSON

"Easily-testable array multipliers"

13th FTCS Milan juin 83 pp 37-40

(SHE 84) J.P.SHEN & F.J.FERGUSON

"The design of easily testable VLSI array multiplier"

IEEE transaction on computers Vol C33 n°6 juin 84 pp 554-560

(SHU 83) M.SHUETTE & J.SHEN

"On-line self-monitoring using signatured instruction streams"

International test conference nov 83 pp 275-282

(SMI 79) J.E.SMITH
"Detection of faults in PLA"
IEEE transaction on computers Vol C28 n°11 nov. 79 PP 845-853

(SMI 80) J.E.SMITH
"Measures of the effectiveness of Fault Signature Analysis"
IEEE transaction on computers Vol. C29 juin 80 pp 510-514

(SMI 83) J.SMITH & P.LAM
"A theory of totally self checking system design"
IEEE transaction on computers sept. 83 pp 831-844

(SOM 83) F.SOMENZI , S.GAI , M.MEZZALAMA & P.PRINETTO
"PART: Programmable array testing based on a partitioning algorithm"
FTCS 13th Milan juin 83 pp 430-433

(SRI 82) T.SRIDHAR & S.THATTE
"Concurrent checking of program flow in VLSI processors"
International test conference nov 82 pp 191-199

(TIM 83) C.TIMOC
"Logical models of physical failures"
ITC 1983 pp 546-553

(WAD 78) P.L.WADSACK

"Technology dependent logics faults"

COMPCON'S 78 1978 pp 124-129

(WAK 78) J.F.WAKERLY

"Error detecting codes, self-checking circuits and applications"

Elsevier North Holland, New-York 1978

(WIL 82) T.W.WILLIAMS & K.P. PARKER

"Design for testability - A survey

IEEE transaction on computers Vol C31 n°1 janvier 82 pp 2-15

(YAJ 81) S.YAJIMA & T.ARAMAKI

"Autonomously testable PLA"

FTCS 11th Portland juin 81

ANNEXES

ANNEXE 1

Description du PLA de l'automate de test de HSURF

```
112 11 31
00--011----
OCCCCO1--1-
0000001-100
0000001-001
00--10-----
--1-10-----
0011111----
001-00-----
-100000----
-10-111----
-1110-0----
-110001-0-1
-10100-----
-11001-----
-110110----
10---0-----
0000001-11-
00--010----
-00-10-----
-011111----
-10-101----
-100111----
-10001-----
-10111-----
-110001-01-
-110001-000
-10101-1---
100--1-----
0000001-010
0000001-111
-001001----
--0101-----
--1110-----
-0-011-----
-00110-----
-01101-----
-100110----
-101101----
-110111----
-1-100-----
-111010----
0000001--00
0000001-011
-01-10-----
-010-1-----
-01101-----
-00111-----
-11-00-----
-111011----
--1011-----
0000001-101
0000001-000
0111010----
-1--00-----
-10-01-----
-10-11-----
-11110-----
-11011-----
-11111-----
```

PLA de 112 monomes

11 entrées

D_6 à D_1 sorties des bascules de mémorisation de l'état

TEST

$\mu CO = 0$

Data 3 à Data 1 : choix du test à réaliser

31 sorties :

D6 à D1 : commandes des 6 bascules

bus externe → TM

TM → ALPHA

ALPHA → S

S → bus externe

ACφ

TM → BETA

BETA → E2

E2 → ALPHA

ALPHA → Ri

Ri → ALPHA

ALPHA → μRi

ALPHA → μCφ

Ø → μCC

mem [μCC] → μRi

incr (μCC)

RAZ : remise à zéro des reg d'analyse de signature

SGN → ALPHA

SGC(1), SGC(2) → bus

SGC(3), SGC(4) → bus

SGC(5), SGC(6) → bus

ALPHA → RE + flags

RE + flags → ALPHA

3 bits de code

1000000000000000000000000110
1100000100000000000000000000
1000000000000000000000000100
1000000000000000000000000100
1000000000000000000000000100
1000000000000000000000000100
1000000000000000000000000100
1100000000000000000000000011
1100000000000000000000000110
1000000000000000000000000000
0100100000000000000000000000
0100001000000000000000000000
0100000000000000000000000100
0111000000000000000000000110
0100100000000000000000000100
0100000000000000000000000100
0100000000000000000000000100
0110000000000000000000000100
0100000000000000000000000100
0110000000000000000000000101
0100000000000000000000000100
0100000000000000000000000011
0010000000000000000000000010
0010000000000000000000000100
0010000000000000000000000100
0010000000000000000000000000
0011000000000000000000000110
0010000000000000000000000000
0010000000000000000000000110
0010000000000000000000000101
0010000000000000000000000011
0010000000000000000000000100
0010000000000000000000000101
0010000000000000000000000100
0010000000000000000000000101
0001000000000000000000000010
0001000000000000000000000010
0001000000000000000000000100
0001000000000000000000000000
0001000000000000000000000101
0001000000000000000000000010
0001000000000000000000000000
0001000000000000000000000011
0001000000000000000000000000
0000100000000000000000000001
0000100000000000000000000010
0000100000000000000000000101
0000100000000000000000000000
0000100000000000000000000000
0000100000000000000000000000
0000100000000000000000000111
0000100000000000000000000100
0000010000000000000000000011
0000010000000000000000000011
0000001000000000000000000000
0000001100000000000000000111
0000000100000000000000100011
0000000100000000000000000000
0000000100000000000000000000
0000000100000000000000000100

ANNEXE 2

Listing du programme de test des PLAs avec domaine d'entrées restreint

```
    PROGRAM testdepla (entree, sortie, input, output) ;

(* ************************************************************************** *
)
(*                                                                            *)
(* PROGRAMME GENERANT UN TEST STRUCTUREL DE PLA A PARTIR DE SA DESCRIPTION    *)
(* DANS LE FICHIER "ENTREE".                                                  *)
(* CETTE DESCRIPTION EST DE LA FORME:                                         *)
(* nbre de monomes du pla   nbre de variables d'entree   nbre de sorties      *)
(* matrice ET du pla                                                          *)
(*          EEEEEE       TTTTTTT                                             *)
(*          E               T                                                *)
(*          E               T                                                *)
(*          EEEE            T                                                *)
(*          E               T                                                *)
(*          E               T                                                *)
(*          EEEEEE          T                                                *)
(*                                                                           *)
(* matrice OU du pla                                                         *)
(*           OOOO         U     U                                           *)
(*          O    O        U     U                                           *)
(*          O    O        U  .  U                                           *)
(*          O    O        U     U                                           *)
(*          O    O        U     U                                           *)
(*           OOOO          UUUUU                                           *)
(* nbre de vecteurs decrivant le domaine d'entree                            *)
(* vecteurs du domaine d'entree                                              *)
(*          DDDDDDD         EEEEEE                                           *)
(*           D     D        E                                                *)
(*           D      D*      E                                                *)
(*           D      D       EEEE                                             *)
(*           D      D       E                                                *)
(*           D     D        E                                                *)
(*          DDDDDDD    *    EEEEEE   *                                       *)
(*                                                                           *)
(* ************************************************************************** *
)
$OPTIONS page$

(* ************************************************************************** *)
(*                                                                            *)
(* EXEMPLE: LE PLA SUIVANT :                                                  *)
(*                                                                            *)
(*      x1    !x1    x2    !x2    x3    !x3    x4    !x4          !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(* -----C-----+-----C-----+-----+-----C-----+-----+-----------+---C---C-  *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(* -----+-----C-----+-----+-----+-----+-----+-----C-----------C---C---C-  *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(* -----+-----+-----+-----C-----C-----+-----C-----+-----------C---C---+-  *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(* -----+-----C-----C-----+-----C-----+-----+-----C-----------+---C---+-  *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(*      !     !      !     !      !     !      !     !            !   !   !   *)
(* -----C-----+-----C-----+-----+-----+-----+-----+-----------C---+---C-  *)
```

```
(*      !      !      !      !      !      !      !      !          !   !   !  *)
(*                                                                 !   !   !  *)
(*                                                               z1  z2  z3  *)
(*                                                                           *)
(*                                                                           *)
(* SERA DECRIT:                                                              *
(* 5  4  3                                                                  *
(* 110-                                                                      *
(* 0--0                                                                      *
(* -011                                                                      *
(* 0110                                                                      *
(* 11--                                                                      *
(*                                                                           *
(* 011                                                                       *
(* 111                                                                       *
(* 110                                                                       *
(* 010                                                                       *
(* 101                                                                       *
(* 1                                                                         *
(* ----                                                                      *
(* tous les vecteurs d'entree possibles peuvent atteindre ce pla             *
(*                                                                           *
(* 2 FICHIERS SONT CREES:                                                    *
(* "SORTIE" REPRESENTE L'ENSEMBLE DES VECTEURS NECESSAIRES POUR TESTER LE    *
(* PLA.                                                                      *
(* "SORTIE_IMP" REPRESENTE L'ENSEMBLE DES MONOMES ET DES MONOMES ADJACENTS   *
(* QUE L'ON NE PEUT PAS TESTER.                                              *
(*                                                                           *
(* ************************************************************************* *
*)
$OPTIONS page$
       $IMPORT
          'compacter (pascal)' : compacter ;
          'traiter_cmos (pascal)' : traiter_cmos ;
          'ecrire_sortie (pascal)' : ecrire_sortie ;
          'principale (pascal)' : principale ;
       $

       $EXPORT
          sortie,
          entree
       $

      CONST
          ref = 40 ;
          max_vecteur = 300 ;
          max_dom_ent = 20 ;
          max_sortie = 30 ;

(* TAILLE MAXIMUM DE LA LISTE CREEE PAR gain_temps, AU DELA DE LAQUELLE, ON
(* SECTIONNE LA LISTE EN 2 MORCEAUX.

       TYPE
          monome = ARRAY [1..max_sortie] OF char ;
          etat = (libre, bouchee, imbouchable) ;
          tabl = ARRAY [1..max_sortie] OF etat ;
          table = ARRAY [1..max_vecteur] OF monome ;
          description = ARRAY [1..100] OF monome ;
          domaine = ARRAY [1..max_dom_ent] OF monome ;
          lis_ptr = @lis ;
          lis = RECORD
```

```
            mon : monome ;
            suiv : lis_ptr ;
            sor : tabl ;
         END ;

      VAR
         un_vecteur_trouve : boolean ;
         nb_pan, nb_pan_ind, nb_pan_ind_de : integer ;
         couv : real ;
         horloge, horloge2 : real ;
         entree, sortie : text ;
         liste_t1, prec_t1, fin_t1 : lis_ptr ;
         ordre, sorti, conex : ARRAY [1..80] OF integer ;
         nbdentrees, nbdesorties, nbdemonomes, i, j : integer ;
         mat_entree, mat_sortie : description ;
         so_tamp : tabl ;
         index_en : integer ;
         en : table ;
         cmos : boolean ;
         reponse : char ;
         sortie_valide : tabl ;
         ms1, ms2 : monome ;
         q, z, liste2, liste_t0 : lis_ptr ;
         prec_t0, fin_t0 : lis_ptr ;
         ok : boolean ;
         mono, mono2 : monome ;
         vect_dom : domaine ;
         nb_vect_dom : integer ;
         sortie_validee_t1, sortie_validee_t0 : boolean ;
         monoadj : monome ;
         n, uu : integer ;
         fini : boolean ;
         compteur : integer ;
$OPTIONS page$
      PROCEDURE ecrire_sortie (index_en, nbdentrees : integer ; en : table) ; E
XTERNAL ;

      PROCEDURE principale (reponse : char) ; EXTERNAL ;

      PROCEDURE traiter_cmos (index_en, nbdentrees, nbdesorties, nbdemonomes :
integer ;
         en : table ;
         mat_entree, mat_sortie : description ;
         vect_dom : domaine ;
         nb_vect_dom : integer) ;
            EXTERNAL ;

      PROCEDURE compacter (n, nbdentrees, index_en : integer ; VAR en : table)
; EXTERNAL ;

      PROCEDURE disp (VAR lib : lis_ptr ; VAR compteur : integer) ; FORWARD ;

      PROCEDURE impmono (monom : monome ; VAR fich : text) ; FORWARD ;

      PROCEDURE trop_de_vecteurs ; FORWARD ;
      PROCEDURE ens_incompatible (VAR lis : lis_ptr ; VAR prec, fin : lis_ptr ;
 VAR sortie_validee : boolean ; VAR compteur : integer) ;
            FORWARD ;

      FUNCTION compatible (mono1, mono2 : monome) : boolean ; FORWARD ;
```

```
$OPTIONS page$
      PROCEDURE gain_temps (mono : monome ; VAR li : lis_ptr ; uu : integer ;
AR compteur : integer ; i : integer ; VAR prec, fin : lis_ptr ; VAR sortie_val
ee : boolean) ;

(* ************************************************************************
*)
(*
(* PROCEDURE RECURSIVE, CREANT LA LISTE DES VECTEURS DE TEST POUR LE MONOME
(* QUE L'ON VEUT TESTER.
(* CETTE LISTE CROISSANT EXPONENTIELLEMENT AVEC LE NOMBRE DE MONOMES, ELLE
(* EST SECTIONNEE EN MORCEAUX JUSQU'A TROUVER UN VECTEUR CORRECT.
(*
(* ************************************************************************
*)

         VAR z, q : lis_ptr ;
            j, zz, auxi, auxi2 : integer ;

         BEGIN
            REPEAT
               BEGIN
                  j := ordre [uu] ;
                  IF j <> i THEN
                     BEGIN
                        mono2 := mat_entree [j] ; ms2 := mat_sortie [j] ;
                        IF compatible (mono, mono2) THEN ens_incompatible (1
prec, fin, sortie_validee, compteur) ;
                     END ;
                  uu := uu + 1 ;
                  IF ((compteur > ref + 1) AND (uu < nbdemonomes)) THEN
                     BEGIN
                        z := li ;
                        FOR zz := 1 TO (compteur DIV 2) - 1 DO z := z^.suiv
                        q := z^.suiv ;
                        z^.suiv := NIL ; z := NIL ;
                        auxi := compteur DIV 2 ;
                        auxi2 := compteur - auxi ;
                        gain_temps (mono, li, uu, auxi, i, prec, fin, sorti
lidee) ;
                        IF li = NIL THEN
                           BEGIN
                              li := q ;
                              gain_temps (mono, li, uu, auxi2, i, prec, fin
rtie_validee) ;
                           END ELSE
                           disp (q, compteur) ;
                        uu := nbdemonomes + 1 ;
                     END ;
               END UNTIL uu > nbdemonomes ;
         END ;
$OPTIONS page$
      PROCEDURE classer (i, k : integer) ;

(* ************************************************************************
*)
(*
(* PROCEDURE DE CLASSEMENT DES MONOMES DU PLA, EN FONCTION :
(*         1)-DU NOMBRE DE CONNEXIONS AVEC LES SORTIES
(*         2)-DU NOMBRE DE SES VARIABLES SIGNIFICATIVES
```

```
(*                                                                              *)
(* ******************************************************************************
*)

          VAR n, n2, p, p2, j, z : integer ;

             trifini : boolean ;

          BEGIN
             j := 1 ; trifini := (j = i) ;
             WHILE NOT trifini DO
                BEGIN
                   IF sorti [j] < k THEN trifini := true ELSE
                      BEGIN
                         IF ((sorti [j] = k) AND (conex [i] < conex [ordre [j]])
) THEN trifini := true ELSE j := j + 1 ;
                      END ;
                   IF j = i THEN trifini := true ;
                END ;
             IF j < i THEN
                BEGIN
                   n := sorti [j] ; p := ordre [j] ;
                   FOR z := j + 1 TO i - 1 DO
                      BEGIN
                         n2 := sorti [z] ; p2 := ordre [z] ;
                         sorti [z] := n ; ordre [z] := p ;
                         n := n2 ; p := p2 ;
                      END ;
                   sorti [i] := n ; ordre [i] := p ;
                END ;
             ordre [j] := i ; sorti [j] := k ;
          END ;
$OPTIONS page$
       PROCEDURE nouv_q (VAR q : lis_ptr ; VAR compteur : integer) ;

(* ******************************************************************************
*)
(*                                                                              *)
(* ACQUISITION D'UN NOUVEL ENREGISTREMENT, AVEC TEST DE L'ESPACE MEMOIRE        *)
(*                                                                              *)
(* ******************************************************************************
*)

          BEGIN
             compteur := compteur + 1 ;
             new (q) ;
             IF q = NIL THEN writeln (sortie, 'ERREUR: PLUS DE PLACE!') ;
          END ;
$OPTIONS page$
       FUNCTION compatible ;

(* ****************************************************************************** *
)
(*                                                                              *)
(* FONCTION QUI TESTE SI mono1 ET mono2 SONT COMPATIBLES                        *)
(* (SI POUR TOUT 0<i<nbre d'entrees ALORS                                       *)
(*          (mono1[i]=mono2[i]) OU (mono1[i]='-') OU (mono2[i]='-'))            *)
(*                                                                              *)
(* ****************************************************************************** *
)
```

```
          VAR i : integer ;

          BEGIN
             compatible := true ;
             i := 1 ;
             REPEAT
                IF ((mono1 [i] <> mono2 [i]) AND (mono1 [i] <> '-') AND (mono2 [
i] <> '-')) THEN compatible := false ;
                i := i + 1 ;
             UNTIL i > nbdentrees ;
          END ;
$OPTIONS page$
       PROCEDURE acq_domaine_entree ;

(* ************************************************************************** *
)
(*                                                                            *)
(* ACQUISITION DES VECTEURS AUTORISES EN ENTREE DU PLA A TESTER                *)
(*                                                                            *)
(* ************************************************************************** *
)

          VAR
             i, j : integer ;

          BEGIN
             readln (entree, nb_vect_dom) ;
             FOR i := 1 TO nb_vect_dom DO
                BEGIN
                   FOR j := 1 TO nbdentrees DO
                      read (entree, vect_dom [i, j]) ;
                   readln (entree) ;
                END ;
          END ;
$OPTIONS page$
       PROCEDURE ajuster (monom : lis_ptr ;
          VAR liste_finale : lis_ptr ;
          VAR compteur : integer) ;

          VAR
             i, j : integer ;
             ok : boolean ;
             mono2 : monome ;
             q, prec : lis_ptr ;

          BEGIN
             liste_finale := NIL ; prec := NIL ;
             compteur := 0 ;
             FOR i := 1 TO nb_vect_dom DO
                BEGIN
                   ok := true ;
                   mono2 := monom^.mon ;
                   FOR j := 1 TO nbdentrees DO
                      IF vect_dom [i, j] <> '-' THEN
                         BEGIN
                            IF vect_dom [i, j] <> mono2 [j] THEN
                               BEGIN
                                  IF mono2 [j] = '-' THEN
                                     mono2 [j] := vect_dom [i, j] ELSE
```

```
                                ok := false ;
                            END ;
                        END ;
                    IF ok THEN
                        BEGIN
                            new (q) ;
                            q^ := monom^ ;
                            q^.mon := mono2 ;
                            IF prec = NIL THEN
                                liste_finale := q ELSE
                                prec^.suiv := q ;
                            prec := q ;
                            compteur := compteur + 1 ;
                        END ;
                END ;
        END ;
$OPTIONS page$
    PROCEDURE acq_mono_adj (mono : monome ; p : integer ; VAR monoadj : monom
e ; VAR fini : boolean ; VAR n : integer) ;

(* ************************************************************************
*)
(*                                                                        *)
(* CALCUL D'UN MONOME ADJACENT AU PARAMETRE mono, A PARTIR DE L'INDICE p, ET  *)
(* LE RANGE DANS monoadj.                                                 *)
(* S'IL N'EXISTE PLUS DE MONOME ADJACENT APRES LE RANG p, LE PARAMETRE fini   *)
(* EST MIS A VRAI, SINON n REPRESENTE L'INDICE QUI DIFFERE ENTRE mono ET      *)
(* monoadj.                                                               *)
(*                                                                        *)
(* ************************************************************************
*)

        VAR term : boolean ;

        BEGIN
            monoadj := mono ;
            n := p + 1 ; term := (n > nbdentrees) ;
            WHILE NOT term DO
                BEGIN
                    IF mono [n] <> '-' THEN
                        BEGIN
                            term := true ;
                            CASE mono [n] OF
                                '0' : monoadj [n] := '1' ;
                                '1' : monoadj [n] := '0' ;
                            END ;
                        END ELSE
                        BEGIN
                            n := n + 1 ;
                            IF n > nbdentrees THEN term := true ;
                        END ;
                END ;
            fini := (n > nbdentrees) ;
        END ;
$OPTIONS page$

    PROCEDURE creer_liste·(m1, m2 : monome ; VAR liste, fin : lis_ptr ; VAR o
k : boolean ; z : lis_ptr ; VAR sortie_validee : boolean ; VAR compteur : intege
r) ;
```

```
(* ************************************************************************
*)
(*
(* CREATION DE LA LISTE DES VECTEURS COMPATIBLES AVEC m1 ET INCOMPATIBLES
(* AVEC m2.
(* SI CELA EST IMPOSSIBLE,MET LE PARAMETRE ok A FAUX.
(*
(* ************************************************************************
*)

          VAR
             i, k : integer ;
             ok2 : boolean ;

          BEGIN
             ok := false ;
             fin := NIL ;
             liste := NIL ;
             FOR i := 1 TO nbdentrees DO
                BEGIN
                   IF ((m1 [i] = '-') AND (m2 [i] <> '-')) THEN
                      BEGIN
                         ok := true ;
                         nouv_q (q, compteur) ; q@.suiv := NIL ;
                         q@.sor := z@.sor ;
                         q@.mon := m1 ;
                         CASE m2 [i] OF
                            '1' : q@.mon [i] := '0' ;
                            '0' : q@.mon [i] := '1' ;
                         END ;
                         IF liste = NIL THEN liste := q ELSE fin@.suiv := q
                         fin := q ;
                      END ;
                END ;

(* m1 EST COMPATIBLE AVEC m2 *)
(* ON CHERCHE UNE SORTIE CONNECTEE AU MONOME m1 QUI N'EST *)
(* PAS MASQUEE PAR LE MONOME m2 *)

             ok2 := true ;
             FOR i := 1 TO nbdentrees DO
                IF ((z^.sor [i] = imbouchable) AND (ms2 [i] = '1')) THEN ok
false ;
             IF ok2 THEN
                BEGIN
                   sortie_valide := z^.sor ;
                   i := 1 ;
                   REPEAT
                      BEGIN
                         IF sortie_valide [i] <> bouchee THEN
                            BEGIN
                               IF ms2 [i] = '1' THEN sortie_valide [i] := t
e ELSE
                                  BEGIN
                                     sortie_valide [i] := imbouchable ;
                                     ok := true ; sortie_validee := true ;
                                     k := i + 1 ;
                                     WHILE k <= nbdesorties DO
                                        BEGIN
                                           IF ms1 [k] = ms2 [k] THEN sorti
```

```
de [k] := bouchee ;
                                        k := k + 1 ;
                                     END ;
                                  nouv_q (q, compteur) ;
                                  WITH q^ DO
                                     BEGIN
                                        suiv := NIL ;
                                        mon := m1 ;
                                        sor := sortie_valide ;
                                     END ;
                                  sortie_valide := z^.sor ;
                                  IF fin = NIL THEN liste := q ELSE fin^.suiv
 := q ;
                                  fin := q ;
                               END ;
                            END ;
                         i := i + 1 ;
                      END UNTIL i > nbdesorties ;
                END ;

          END ;
$OPTIONS page$
       PROCEDURE ens_incompatible ;

(* ************************************************************************************
*)
(*                                                                                    *)
(* CHAQUE VECTEUR DE LA LISTE POINTEE PAR LE PARAMETRE "lis" EST                      *)
(*       -SOIT RENDU INCOMPATIBLE AVEC mono2 SI C'EST POSSIBLE                        *)
(*       -SOIT ELIMINE DE LA LISTE SINON.                                             *)
(*                                                                                    *)
(* ************************************************************************************
*)

          BEGIN
             z := lis ; prec := NIL ;
             sortie_validee := false ;
             WHILE z <> NIL DO
                BEGIN
                   IF compatible (z^.mon, mono2) THEN
                      creer_liste (z@.mon, mono2, liste2, fin, ok, z, sortie_val
idee, compteur) ELSE
                         BEGIN
                            ok := true ;
                            new (q) ;
                            compteur := compteur + 1 ;
                            WITH q^ DO
                               BEGIN
                                  mon := z^.mon ;
                                  suiv := NIL ;
                                  sor := z@.sor ;
                               END ;
                            liste2 := q ;
                            fin := q ;
                         END ;
                      IF ok THEN
                         BEGIN
                            IF prec = NIL THEN lis := liste2
                            ELSE prec@.suiv := liste2 ;
                            fin@.suiv := z@.suiv ;
```

```
                        prec := fin ;
                        z@.suiv := NIL ; dispose (z) ; compteur := compteur - 1
;
                        z := fin@.suiv ;
                     END ELSE
                     BEGIN

(* ELIMINATION DU VECTEUR POINTE PAR "z" *)

                        IF prec = NIL THEN
                           BEGIN
                              q := lis ; lis := z@.suiv ;
                              q@.suiv := NIL ; dispose (q) ; compteur := compte
ur - 1 ;
                              IF sortie_validee THEN z := NIL ELSE z := lis ;
                           END
                        ELSE
                           BEGIN
                              prec@.suiv := z@.suiv ;
                              q := z ;
                              IF sortie_validee THEN z := NIL ELSE z := z@.sui
;
                              q@.suiv := NIL ; dispose (q) ; compteur := compt
ur - 1 ;
                           END ;
                     END ;
               END ;
         END ;
$OPTIONS page$
      PROCEDURE acq_mat_entree ;

(* ************************************************************************
*)
(*
(* ACQUISITION DE LA MATRICE "ET" DU PLA
(* LE SYMBOLE "1" PRESENT DANS UN MONOME SIGNIFIE QUE LA VARIABLE D'ENTREE
(* APPARAIT DANS LE MONOME.
(* LE SYMBOLE "0" PRESENT DANS UN MONOME SIGNIFIE QUE LA VARIABLE D'ENTREE
(* APPARAIT COMPEMENTEE DANS LE MONOME.
(* LE SYMBOLE "-" PRESENT DANS UN MONOME SIGNIFIE QUE LA VARIABLE D'ENTREE
(* N'APPARAIT PAS DANS LE MONOME
(*
(* ************************************************************************
*)

         VAR i, j, k : integer ;

         BEGIN
            FOR i := 1 TO nbdemonomes DO
               BEGIN
                  k := 0 ;
                  FOR j := 1 TO nbdentrees DO
                     BEGIN
                        read (entree, mat_entree [i, j]) ;
                        IF mat_entree [i, j] <> '-' THEN k := k + 1 ;
                     END ;
                  readln (entree) ;
                  conex [i] := k ;
               END ;
         END ;
```

```
$OPTIONS page$
        PROCEDURE acq_mat_sortie ;

(* ************************************************************************
*)
(*                                                                        *)
(* ACQUISITION DE LA MATRICE "OU" DU PLA                                  *)
(* SEULS LES SYMBOLES "0" OU "1" PEUVENT Y FIGURER:                       *)
(* "0": LE MONOME N'EST PAS CONNECTE A LA SORTIE.                         *)
(* "1": LE MONOME EST CONNECTE A LA SORTIE.                               *)
(*                                                                        *)
(* ************************************************************************
*)

           VAR i, j, k : integer ;

           BEGIN
              FOR i := 1 TO nbdemonomes DO
                 BEGIN
                    k := 0 ;
                    FOR j := 1 TO nbdesorties DO
                       BEGIN
                          read (entree, mat_sortie [i, j]) ;
                          IF mat_sortie [i, j] = '1' THEN k := k + 1 ;
                       END ;
                    readln (entree) ;
                    classer (i, k) ;
                 END ;
           END ;
$OPTIONS page$
        PROCEDURE disp ;

(* ************************************************************************
*)
(*                                                                        *)
(* ELIMINATION DE LA LISTE POINTEE PAR LE PARAMETRE "lib"                 *)
(* AFIN DE LIBERER UN MAXIMUM D'ESPACE MEMOIRE.                           *)
(*                                                                        *)
(* ************************************************************************
*)

           VAR
              z : lis_ptr ;

           BEGIN
              WHILE lib <> NIL DO
                 BEGIN
                    z := lib ;
                    lib := lib@.suiv ;
                    z@.suiv := NIL ;
                    dispose (z) ; compteur := compteur - 1 ;
                 END ;
           END ;
$OPTIONS page$
        PROCEDURE impmono ;

(* ************************************************************************
*)
(*                                                                        *)
(* IMPRESSION DU MONOME "monom" DANS LE FICHIER "fich".                   *)
```

```
(*
(* ************************************************************************
*)

          VAR
             i : integer ;

          BEGIN
             FOR i := 1 TO nbdentrees DO
                BEGIN
                   write (fich, monom [i]) ;
                END ;
             writeln (fich) ;
          END ;
$OPTIONS page$
       PROCEDURE traiter_def_so (sort : tabl ; i : integer) ;

(* ************************************************ *)
(*                                                  *)
(* TRAITEMENT DES DEFAUTS D'ELEMENT SUR LES SORTIES  *)
(*                                                  *)
(* ************************************************ *)

          VAR
             zz, nn : integer ;
             q : lis_ptr ;

          BEGIN
             FOR nn := 1 TO nbdesorties DO
                BEGIN
                   IF ((sort [nn] = bouchee) AND (ms1 [nn] = '1')) THEN
                      BEGIN
                         new (q) ;
                         WITH q^ DO
                            BEGIN
                               mon := mono ;
                               suiv := NIL ;
                            END ;
                         liste_t0 := q ;
                         FOR zz := 1 TO nbdesorties DO
                            IF ms1 [zz] = '1' THEN
                               liste_t0^.sor [zz] := libre ELSE
                               liste_t0^.sor [zz] := bouchee ;
                         liste_t0^.sor [nn] := imbouchable ;
                         uu := 1 ; prec_t0 := NIL ;
                         fin_t0 := NIL ;
                         compteur := 1 ;
                         un_vecteur_trouve := false ;
                         gain_temps (mono, liste_t0, uu, compteur, i, prec_
in_t0, sortie_validee_t0) ;
                         IF liste_t0 <> NIL THEN
                            BEGIN
                               en [index_en] := liste_t0^.mon ;
                               index_en := index_en + 1 ;
                               IF index_en > max_vecteur THEN
                                  trop_de_vecteurs ;
                               FOR zz := nn + 1 TO nbdesorties DO
                                  IF ((sort [zz] = bouchee) AND (liste_t0^.
zz] <> bouchee)) THEN
                                     sort [zz] := liste_t0^.sor [zz] ;
```

```
                              disp (liste_t0, compteur) ;
                           END ELSE
                           BEGIN
                              writeln (sortie, 'IMPOSSIBLE DE TESTER LES DEFAUT
S D"ELEMENT ENTRE LA SORTIE', nn : 3, ' ET LA LIGNE', i : 4) ;
                              nb_pan_ind := nb_pan_ind + 1 ;
                              IF un_vecteur_trouve THEN
                                 nb_pan_ind_de := nb_pan_ind_de + 1 ;
                           END ;
                     END ;
               END ;
         END ;
$OPTIONS page$
      PROCEDURE trop_de_vecteurs ;

         VAR
            i : integer ;

         BEGIN
            FOR i := 1 TO index_en - 1 DO
               IF en [i, 1] <> '2' THEN compacter (i, nbdentrees, index_en, en)
 ;
            i := 1 ;
            WHILE i <= index_en - 1 DO
               BEGIN
                  IF en [i, 1] = '2' THEN
                     BEGIN
                        en [i] := en [index_en - 1] ;
                        index_en := index_en - 1 ;
                     END ELSE
                     i := i + 1 ;
               END ;
            writeln (sortie, '******************') ;
         END ;
$OPTIONS page$
      BEGIN

(* ************************************************************************
*)
(*                                                                        *)
(*                     DEBUT DU PROGRAMME PRINCIPAL                       *)
(*                     ----------------------------                       *)
(*                                                                        *)
(* ************************************************************************
*)

         reset (entree) ;
         rewrite (sortie) ;
         writeln ('C = calcul de vecteurs de test') ;
         writeln ('M = calcul de vecteurs de test pour un PLA en technologie CM
OS') ;
         writeln ('D = detection et verdict de pannes') ;
         writeln ('S = simulation du fonctionnement normal d"un PLA') ;
         REPEAT
            read (reponse) ;
         UNTIL ((reponse = 'c') OR (reponse = 'm') OR (reponse = 'd') OR (repon
se = 's') OR
            (reponse = 'C') OR (reponse = 'M') OR (reponse = 'D') OR (reponse =
 'S')) ;
         IF ((reponse = 'd') OR (reponse = 's') OR (reponse = 'D') OR (reponse
```

```
= 'S')) THEN
            principale (reponse) ELSE
            BEGIN
               rewrite (sortie) ;
               horloge := clock ;
               cmos := ((reponse = 'm') OR (reponse = 'M')) ;
               read (entree, nbdemonomes, nbdentrees, nbdesorties) ;
               readln (entree) ;
               acq_mat_entree ; readln (entree) ; acq_mat_sortie ;
               acq_domaine_entree ;
               index_en := 1 ;
               nb_pan_ind := 0 ;
               nb_pan_ind_de := 0 ;
               nb_pan := 0 ;
               FOR i := 1 TO nbdemonomes DO
                  FOR j := 1 TO nbdentrees DO
                     IF mat_entree [i, j] <> '-' THEN
                        nb_pan := nb_pan + 1 ;
               FOR i := 1 TO nbdemonomes DO
                  FOR j := 1 TO nbdesorties DO
                     IF mat_sortie [i, j] = '1' THEN nb_pan := nb_pan + 1 ;
               nb_pan := nb_pan + 4 * nbdesorties + 8 * nbdentrees + nbdemono
s * (2 * nbdentrees + 2 * nbdesorties + 4) - 6 ;
               FOR i := 1 TO nbdemonomes DO
                  BEGIN
                     mono := mat_entree [i] ; ms1 := mat_sortie [i] ;
                     new (q) ;
                     WITH q@ DO
                        BEGIN
                           mon := mono ;
                           suiv := NIL ;
                        END ;
                     FOR j := 1 TO nbdesorties DO
                        IF ms1 [j] = '1' THEN q^.sor [j] := libre ELSE
                           q^.sor [j] := bouchee ;
                     ajuster (q, liste_t0, compteur) ;
                     uu := 1 ; prec_t0 := NIL ; fin_t0 := NIL ;
                     un_vecteur_trouve := false ;
                     gain_temps (mono, liste_t0, uu, compteur, i, prec_t0, f
t0, sortie_validee_t0) ;
                     IF liste_t0 <> NIL THEN
                        BEGIN
                           en [index_en] := liste_t0^.mon ;
                           index_en := index_en + 1 ;
                           IF index_en > max_vecteur THEN
                              trop_de_vecteurs ;
                           so_tamp := liste_t0^.sor ;
                           disp (liste_t0, compteur) ;
                           traiter_def_so (so_tamp, i) ;
                        END ELSE
                        BEGIN
                           write (sortie, 'IMPOSSIBLE DE TESTER LA PRESENCE
MONOME:') ;
                           impmono (mono, sortie) ;
                           FOR j := 1 TO nbdentrees DO
                              IF mono [j] <> '-' THEN
                                 BEGIN
                                    nb_pan_ind := nb_pan_ind + 1 ;
                                    IF un_vecteur_trouve THEN
                                       nb_pan_ind_de := nb_pan_ind_de + 1 ;
```

```
                                    END ;
                              END ;
                           acq_mono_adj (mono, 0, monoadj, fini, n) ;
                           WHILE NOT fini DO
                              BEGIN
                                 new (q) ; WITH q^ DO
                                    BEGIN
                                       mon := monoadj ; suiv := NIL ;
                                    END ;
                                 FOR j := 1 TO nbdesorties DO
                                    IF ms1 [j] = '1' THEN q^.sor [j] := libre ELSE q^
.sor [j] := bouchee ;
                                 ajuster (q, liste_t1, compteur) ;
                                 IF liste_t1 <> NIL THEN
                                    BEGIN
                                       uu := 1 ;
                                       un_vecteur_trouve := false ;
                                       gain_temps (monoadj, liste_t1, uu, compteur, i
, prec_t1, fin_t1, sortie_validee_t1) ;
                                       IF liste_t1 <> NIL THEN
                                          BEGIN
                                             en [index_en] := liste_t1^.mon ;
                                             index_en := index_en + 1 ;
                                             IF index_en > max_vecteur THEN
                                                trop_de_vecteurs ;
                                             disp (liste_t1, compteur) ;
                                          END ELSE
                                          BEGIN
                                             write (sortie, 'IMPOSSIBLE DE TESTER LA
PRESENCE DU MONOME ADJACENT: ') ;
                                             impmono (monoadj, sortie) ;
                                             nb_pan_ind := nb_pan_ind + 1 ;
                                             IF un_vecteur_trouve THEN
                                                nb_pan_ind_de := nb_pan_ind_de + 1 ;
                                          END ;
                                    END ELSE
                                    BEGIN
                                       write (sortie, 'LE MONOME ADJACENT SUIVANT N"A
PPARTIENT PAS AU DOMAINE D"ENTREE:') ;
                                       impmono (monoadj, sortie) ;
                                       nb_pan_ind := nb_pan_ind + 1 ;
                                       nb_pan_ind_de := nb_pan_ind_de + 1 ;
                                    END ;
                                 acq_mono_adj (mono, n, monoadj, fini, n) ;
                              END ;
                        END ;
                     horloge := (clock - horloge) / 1000000 ;
                     horloge2 := clock ;
                     FOR i := 1 TO index_en - 1 DO
                        IF en [i, 1] <> '2' THEN compacter (i, nbdentrees, index_en,
en) ;
                     IF cmos THEN
                        traiter_cmos (index_en, nbdentrees, nbdesorties, nbdemonomes,
 en, mat_entree, mat_sortie, vect_dom, nb_vect_dom) ELSE
                        ecrire_sortie (index_en, nbdentrees, en) ;
                     horloge2 := (clock - horloge2) / 1000000 ;
                     writeln (sortie) ;
                     writeln (sortie, '**********************************************
**********') ;
                     writeln (sortie, '*
```

```
        *') ;
                writeln (sortie, '* TEMPS DE CALCUL DES VECTEURS DE TEST : ', h
rloge : 6 : 2, 's CPU   *') ;
                writeln (sortie, '*
        *') ;
                IF cmos THEN
                   BEGIN
                      writeln (sortie, '* TEMPS DE CALCUL DES VECTEURS CMOS
 ', horloge2 : 6 : 2, 's CPU   *') ;
                      writeln (sortie, '*
              *') ;
                   END ELSE
                   BEGIN
                      writeln (sortie, '* TEMPS DE COMPACTAGE DES VECTEURS
 ', horloge2 : 6 : 2, 's CPU   *') ;
                      writeln (sortie, '*
              *') ;
                   END ;
                couv := (nb_pan - nb_pan_ind) * 100 / nb_pan ;
                writeln (sortie, '* COUVERTURE DES PANNES
 ', couv : 6 : 2, '% *') ;
                writeln (sortie, '*
        *') ;
                couv := (nb_pan - nb_pan_ind + nb_pan_ind_de) * 100 / nb_pan ;
                writeln (sortie, '* COUVERTURE DES PANNES SANS DOMAINE D"ENTRE
 ', couv : 6 : 2, '% *') ;
                writeln (sortie, '*
        *') ;
                writeln (sortie, '**********************************************
**********') ;
             END ;
      END.
```

```
    PROGRAM compact (sortie) ;

(* ************************************************************************
*)
(*                                                                        *)
(* PROGRAMME QUI A PARTIR D'UN FICHIER "ENTREE" COMPACTE SES VECTEURS , AFIN  *)
(* D'OBTENIR UN ENSEMBLE MINIMAL DE VECTEURS.                             *)
(*                                                                        *)
(* ************************************************************************
*)
       $IMPORT
          'sortie (pascal)' : sortie ;
       $

       $EXPORT
          compacter,
          ecrire_sortie
       $

       CONST
          max_vecteur = 300 ;
max_sortie = 30 ;

       TYPE
          monome = ARRAY [1..max_sortie] OF char ;
          table = ARRAY [1..max_vecteur] OF monome ;

       VAR
          sortie : text ;

       PROCEDURE compacter (n, nbdentrees, index_en : integer ;
          VAR en : table) ;

          VAR
             i, k : integer ;

          FUNCTION compat (n, i : integer) : boolean ;

(* ************************************************************************
*)
(*                                                                        *)
(* FONCTION DETERMINANT SI LES VECTEURS No n ET i SONT COMPATIBLES , ET DONC  *)
(* COMPACTABLES.                                                          *)
(*                                                                        *)
(* ************************************************************************
*)

             VAR j : integer ;

             BEGIN
                compat := true ;
                FOR j := 1 TO nbdentrees DO
                   IF ((en [n, j] <> en [i, j]) AND (en [n, j] <> '-') AND (en [
i, j] <> '-')) THEN compat := false ;
             END ;
          BEGIN
             FOR i := n + 1 TO index_en - 1 DO
                BEGIN
                   IF en [i, 1] <> '2' THEN
                      BEGIN
```

```
                        IF compat (n, i) THEN
                           BEGIN
                              FOR k := 1 TO nbdentrees DO
                                 IF en [n, k] = '-' THEN en [n, k] := en [i
;
                              en [i, 1] := '2' ;
                           END ;
                     END ;
               END ;
         END ;
$OPTIONS page$
      PROCEDURE ecrire_sortie (index_en, nbdentrees : integer ;
         en : table) ;

(* ************************************************************************
*)
(*
(* ECRITURE DE L'ENSEMBLE DES VECTEUR COMPACTES.
(*
(* ************************************************************************
*)

         VAR i, j, nb : integer ;

         BEGIN
            writeln (sortie, 'NUMERO   VECTEUR') ;
            writeln (sortie) ;
            nb := 1 ;
            FOR i := 1 TO index_en - 1 DO
               BEGIN
                  IF en [i, 1] <> '2' THEN
                     BEGIN
                        write (sortie, nb : 6, ' : ') ;
                        FOR j := 1 TO nbdentrees DO
                           write (sortie, en [i, j]) ;
                        writeln (sortie) ; nb := nb + 1 ;
                     END ;
               END ;
         END ;
$OPTIONS page$
      BEGIN
      END.
```

```
PROGRAM cmos (sortie) ;

   $IMPORT
      'sortie (pascal)' : sortie
      $

   $EXPORT
      traiter_cmos
   $

   CONST
      max_vecteur = 300 ;
      max_dom_ent = 20 ;
      max_sortiee = 30 ;

   TYPE
      monome = ARRAY [1..max_sortiee] OF char ;
      table = ARRAY [1..max_vecteur] OF monome ;
      description = ARRAY [1..100] OF monome ;
      domaine = ARRAY [1..max_dom_ent] OF monome ;

   VAR
      sortie : text ;

   PROCEDURE traiter_cmos (index_en, nbdentrees, nbdesortiees, nbdemonomes :
integer ;
      en : table ;
      mat_entree, mat_sortie : description ;
      vect_dom : domaine ;
      nb_vect_dom : integer) ;

      TYPE
         typ_sortiee = ARRAY [1..max_sortiee] OF char ;
         ptr_typ_vec_test = ^typ_vec_test ;
         typ_vec_test = RECORD
            vect : monome ;
            sort : typ_sortiee ;
            suiv : ptr_typ_vec_test ;
            no : integer ;
         END ;

      VAR
         tete_test, tete_finale, fin_finale : ptr_typ_vec_test ;
         sortiee : typ_sortiee ;

      FUNCTION appartenance (VAR mono : monome) : boolean ;

         VAR
            i, j : integer ;
            mono2 : monome ;
            ok : boolean ;

         BEGIN
            i := 1 ;
            REPEAT
               BEGIN
                  ok := true ;
                  mono2 := mono ;
                  FOR j := 1 TO nbdentrees DO
                     IF vect_dom [i, j] <> '-' THEN
```

```
                    BEGIN
                       IF vect_dom [i, j] <> mono2 [j] THEN
                          BEGIN
                             IF mono2 [j] = '-' THEN
                                mono2 [j] := vect_dom [i, j] ELSE
                                ok := false ;
                          END ;
                    END ;
              IF ok THEN
                 BEGIN
                    i := nb_vect_dom ;
                    mono := mono2 ;
                 END ;
              i := i + 1 ;
           END
        UNTIL i > nb_vect_dom ;
        appartenance := ok ;
     END ;

  PROCEDURE imp_sortiee (sor : typ_sortiee ; VAR fich : text) ;

     VAR
        z : integer ;

     BEGIN
        write (fich, '          ') ;
        FOR z := 1 TO nbdesortiees DO
           write (fich, sor [z]) ;
        writeln (fich) ;
        writeln (fich) ;
     END ;

  PROCEDURE sortiee_barre (sor : typ_sortiee ; VAR barre : typ_sortie
;

     VAR
        i : integer ;

     BEGIN
        FOR i := 1 TO nbdesortiees DO
           BEGIN
              IF sor [i] = '0' THEN barre [i] := '1'
              ELSE barre [i] := '0' ;
           END ;
     END ;

  FUNCTION sortiee_complement (sor : typ_sortiee ;
     VAR vecteur, precedent : ptr_typ_vec_test) : boolean ;

     VAR
        i : integer ;
        fin, egal : boolean ;
        barre : typ_sortiee ;

     BEGIN
        vecteur := tete_test ;
        precedent := NIL ;
        fin := (vecteur = NIL) ;
        sortiee_complement := false ;
        WHILE NOT fin DO
```

```
            BEGIN
               sortiee_barre (sor, barre) ;
               egal := true ;
               FOR i := 1 TO nbdesortiees DO
                  IF vecteur^.sort [i] <> barre [i] THEN egal := false ;
               IF egal THEN
                  BEGIN
                     fin := true ;
                     sortiee_complement := true ;
                  END ELSE
                  BEGIN
                     precedent := vecteur ;
                     vecteur := vecteur^.suiv ;
                     IF vecteur = NIL THEN fin := true ;
                  END ;
            END ;
      END ;

   FUNCTION incompatible (a, b : monome) : boolean ;

      VAR
         i : integer ;

      BEGIN
         i := 1 ;
         incompatible := false ;
         REPEAT
            IF ((a [i] = '-') OR (b [i] = '-')) THEN i := i + 1
            ELSE
               BEGIN
                  IF a [i] = b [i] THEN i := i + 1
                  ELSE
                     BEGIN
                        i := nbdentrees + 1 ;
                        incompatible := true ;
                     END ;
               END ;
         UNTIL i > nbdentrees ;
      END ;

   FUNCTION poss_rendre_compatible (a : monome ;
      VAR b : monome) : boolean ;

      VAR
         i : integer ;

      BEGIN
         poss_rendre_compatible := true ;
         i := 1 ;
         REPEAT
            IF a [i] = '-' THEN
               i := i + 1 ELSE
               BEGIN
                  IF b [i] = '-' THEN b [i] := a [i]
                  ELSE
                     BEGIN
                        IF b [i] <> a [i] THEN
                           BEGIN
                              i := nbdentrees ;
                              poss_rendre_compatible := false ;
```

```
                                END ;
                            END ;
                        i := i + 1 ;
                    END ;
                UNTIL i > nbdentrees ;
            END ;

        FUNCTION calcul_alpha (VAR alpha : ptr_typ_vec_test ;
            sor : typ_sortiee) : boolean ;

            VAR
                ok : boolean ;
                el, res : ptr_typ_vec_test ;
                z : integer ;
                memo : monome ;
                barre : typ_sortiee ;

            PROCEDURE deriver (el : ptr_typ_vec_test ;
                VAR res : ptr_typ_vec_test ;
                i, j, k : integer ;
                VAR ok : boolean ;
                memo : monome) ;

                LABEL 9999 ;


                BEGIN
                    WHILE i <= nbdesortiees DO
                        BEGIN
                            IF barre [i] = '0' THEN
                                BEGIN
                                    WHILE j <= nbdemonomes DO
                                        BEGIN
                                            IF mat_sortie [j, i] = '0' THEN
                                                j := j + 1 ELSE
                                                BEGIN
                                                    IF incompatible (mat_entree [j
vect) THEN

                                                        j := j + 1 ELSE
                                                        BEGIN
                                                            WHILE k <= nbdentrees DO
                                                                BEGIN
                                                                    IF el^.vect [k] <>
EN

                                                                        k := k + 1 ELSE
                                                                        BEGIN
                                                                            IF mat_entre
] = '-' THEN

                                                                                k := k +
                                                                                BEGIN
                                                                                    IF mat

 [j, k] = '0' THEN

                                                                                        el^

k] := '1' ELSE

                                                                                        el^

k] := '0' ;

                                                                                    derive

res, i, j + 1, 1, ok, memo) ;

                                                                                    IF ok

TO 9999 ELSE
```

```
                                                                    BEGIN
                                                                       elʌ.vec
t [k] := '-' ;
                                                                       k := k
+ 1 ;
                                                                    END ;
                                                                 END ;
                                                              END ;
                                                           END ;
                                                        GOTO 9999 ;
                                                     END ;
                                                  END ;
                                               END ;
                                            i := i + 1 ;
                                            j := 1 ;
                                            k := 1 ;
                                         END ELSE
                                         BEGIN
                                            WHILE j <= nbdemonomes DO
                                               BEGIN
                                                  IF mat_sortie [j, i] = '0' THEN
                                                     j := j + 1 ELSE
                                                     BEGIN
                                                        memo := elʌ.vect ;
                                                        IF NOT poss_rendre_compatible (mat_en
tree [j], elʌ.vect) THEN
                                                           BEGIN
                                                              elʌ.vect := memo ;
                                                              j := j + 1 ;
                                                           END ELSE
                                                           BEGIN
                                                              deriver (el, res, i + 1, 1, 1,
ok, memo) ;
                                                              IF ok THEN GOTO 9999 ELSE
                                                                 BEGIN
                                                                    elʌ.vect := memo ;
                                                                    j := j + 1 ;
                                                                 END ;
                                                           END ;
                                                     END ;
                                               END ;
                                            GOTO 9999 ;
                                         END ;
                                   END ;
                                IF appartenance (elʌ.vect) THEN
                                   ok := true ;
                                resʌ := elʌ ;
9999 :                       END ;

                          BEGIN
                             new (res) ;
                             resʌ.suiv := NIL ;
                             new (el) ;
                             elʌ.suiv := NIL ;
                             sortiee_barre (sor, barre) ;
                             elʌ.sort := barre ;
                             FOR z := 1 TO nbdentrees DO
                                BEGIN
                                   elʌ.vect [z] := '-' ;
                                   memo [z] := '-' ;
```

```
      END ;
    ok := false ;
    deriver (el, res, 1, 1, 1, ok, memo) ;
    calcul_alpha := ok ;
    alpha := res ;
    alpha^.no := 0 ;
  END ;

PROCEDURE impvect (v : monome) ;

  VAR
    i : integer ;

  BEGIN
    FOR i := 1 TO nbdentrees DO
      write (sortie, v [i]) ;
  END ;

PROCEDURE ecrire_test ;

  VAR
    z : ptr_typ_vec_test ;
    i : integer ;

  BEGIN
    writeln (sortie, 'NUMERO   VECTEUR            SORTIE') ;
    writeln (sortie) ;
    WHILE tete_finale <> NIL DO
      BEGIN
        write (sortie, tete_finale^.no : 3, ' :  ') ;
        impvect (tete_finale^.vect) ;
        IF nbdentrees < 18 THEN
          FOR i := nbdentrees TO 18 DO write (sortie, ' ') ;
        FOR i := 1 TO nbdesortiees DO
          write (sortie, tete_finale^.sort [i]) ;
        writeln (sortie) ;
        z := tete_finale ;
        tete_finale := z^.suiv ;
        z^.suiv := NIL ;
        dispose (z) ;
      END ;
  END ;

PROCEDURE calcul_sortiees (n : ptr_typ_vec_test) ;

  VAR
    i, j : integer ;

  BEGIN
    FOR i := 1 TO nbdesortiees DO
      n^.sort [i] := '0' ;
    FOR i := 1 TO nbdemonomes DO
      BEGIN
        IF NOT incompatible (n^.vect, mat_entree [i]) THEN
          BEGIN
            FOR j := 1 TO nbdesortiees DO
              IF mat_sortie [i, j] = '1' THEN
                n^.sort [j] := '1' ;
          END ;
      END ;
```

```
   END ;

PROCEDURE lire_vect_test ;

   VAR
      i, j, num : integer ;
      dernier, n : ptr_typ_vec_test ;

   BEGIN
      dernier := NIL ;
      num := 1 ;
      FOR i := 1 TO index_en - 1 DO
         BEGIN
            IF en [i, 1] <> '2' THEN
               BEGIN
                  new (n) ;
                  IF dernier = NIL THEN tete_test := n
                  ELSE dernier^.suiv := n ;
                  dernier := n ;
                  WITH n^ DO
                     BEGIN
                        suiv := NIL ;
                        no := num ;
                        FOR j := 1 TO nbdentrees DO
                           BEGIN
                              vect [j] := en [i, j] ;
                              IF vect [j] = '-' THEN
                                 vect [j] := '0' ;
                           END ;
                     END ;
                  calcul_sortiees (n) ;
                  num := num + 1 ;
               END ;
         END ;
   END ;

PROCEDURE generer_test ;

   VAR
      vecteur, precedent, vecteur_mem, alpha : ptr_typ_vec_test ;
      fini : boolean ;
      z : integer ;
      barre : typ_sortiee ;
      deja_calcule, dernier_calcule, w, q : ptr_typ_vec_test ;
      trouve : boolean ;

   BEGIN
      IF calcul_alpha (alpha, tete_test^.sort) THEN
         BEGIN
            new (deja_calcule) ;
            deja_calcule^ := alpha^ ;
            tete_finale := alpha ;
            alpha^.suiv := tete_test ;
            tete_test := tete_test^.suiv ;
            alpha^.suiv^.suiv := NIL ;
            fin_finale := alpha^.suiv ;
         END ELSE
         BEGIN
            writeln (sortie, 'IL EST IMPOSSIBLE DE TROUVER UN VECTEUR
D"ENTREE INITIALISANT LES SORTIES A: ') ;
```

```
                    sortiee_barre (tete_testʌ.sort, barre) ;
                    imp_sortiee (barre, sortie) ;
                    tete_finale := tete_test ;
                    tete_test := tete_testʌ.suiv ;
                    tete_finaleʌ.suiv := NIL ;
                    fin_finale := tete_finale ;
                    new (deja_calcule) ;
                    WITH deja_calculeʌ DO
                       BEGIN
                          sort := barre ;
                          no := 1 ;
                          suiv := NIL ;
                       END ;
                 END ;
              dernier_calcule := deja_calcule ;
              sortiee := fin_finaleʌ.sort ;
              fini := false ;
              REPEAT
                 BEGIN
                    IF sortiee_complement (sortiee, vecteur, precedent) THEN
                       BEGIN
                          fin_finaleʌ.suiv := vecteur ;
                          fin_finale := vecteur ;
                          IF precedent = NIL THEN
                             tete_test := vecteurʌ.suiv ELSE
                             precedentʌ.suiv := vecteurʌ.suiv ;
                          vecteurʌ.suiv := NIL ;
                          sortiee := vecteurʌ.sort ;
                       END ELSE
                       BEGIN
                          IF tete_test = NIL THEN
                             fini := true ELSE
                             BEGIN
                                vecteur_mem := tete_test ;
                                tete_test := tete_testʌ.suiv ;
                                vecteur_memʌ.suiv := NIL ;
                                trouve := false ;
                                w := deja_calcule ;
                                sortiee_barre (vecteur_memʌ.sort, barre) ;
                                WHILE ((w <> NIL) AND (NOT trouve)) DO
                                   BEGIN
                                      trouve := true ;
                                      FOR z := 1 TO nbdesortiees DO
                                         IF barre [z] <> wʌ.sort [z] THEN
                                            trouve := false ;
                                      IF NOT trouve THEN
                                         w := wʌ.suiv ;
                                   END ;
                                IF NOT trouve THEN
                                   BEGIN
                                      IF calcul_alpha (alpha, vecteur_memʌ.sor
t) THEN
                                         BEGIN
                                            new (q) ;
                                            qʌ := alphaʌ ;
                                            qʌ.no := 0 ;
                                            fin_finaleʌ.suiv := alpha ;
                                            fin_finale := alpha ;
                                         END ELSE
                                         BEGIN
```

```
                                        writeln (sortie, 'IL EST IMPOSSIBL
E DE TROUVER UN VECTEUR D"ENTREE INITIALISANT LES SORTIES A: ') ;
                                        sortiee_barre (vecteur_mem^.sort,
barre) ;

                                        imp_sortiee (barre, sortie) ;
                                        new (q) ;
                                        q^.no := 1 ;
                                        q^.sort := barre ;
                                        q^.suiv := NIL ;
                                     END ;
                                  IF deja_calcule = NIL THEN
                                     deja_calcule := q ELSE
                                     dernier_calcule^.suiv := q ;
                                  dernier_calcule := q ;
                                  fin_finale^.suiv := vecteur_mem ;
                                  fin_finale := vecteur_mem ;
                                  sortiee := vecteur_mem^.sort ;
                               END ELSE
                               BEGIN
                                  IF w^.no = 0 THEN
                                     BEGIN
                                        new (alpha) ;
                                        alpha^ := w^ ;
                                        alpha^.suiv := NIL ;
                                        fin_finale^.suiv := alpha ;
                                        fin_finale := alpha ;
                                     END ;
                                  fin_finale^.suiv := vecteur_mem ;
                                  fin_finale := vecteur_mem ;
                                  sortiee := vecteur_mem^.sort ;
                               END ;
                         END ;
                   END ;
             END ;
          UNTIL fini ;
       END ;

    BEGIN
       lire_vect_test ;
       generer_test ;
       ecrire_test ;
    END ;

BEGIN
END.
```

ANNEXE 3

Exemples d'exécutions du programme de test des PLAs

29 10 9

nb de mooves, nb d'entrées, nb de sorties

description du PLA

plage ET

plage OU

```
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------11---00000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------00---00000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------01---10000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------01---01000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------01---00100
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------01---00010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------01---00001
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ---------10---00000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ----------1-00110
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ----------000011
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ----------1-00101
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: ----------1-00101
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:-0101---1-----001
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:-1001---1-----001
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:-1111---1-----001
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:-1100---1-----001
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT:--1101---0-----00110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:-1101---1-----011
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT:----------0--00111
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1110--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0010--------010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --0111--------01000
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0110--------110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0110--------011
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0110--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1111--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0011--------010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --0110--------01000
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0111--------110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0111--------011
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0111--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0000--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1101--------010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --1010--------01000
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --1001--------01000
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1000--------110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1000--------011
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1000--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0001--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1101--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1011--------010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --1000--------01000
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1001--------110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1001--------011
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1001--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--0010--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1110--------010
IMPOSSIBLE DE TESTER LA PRESENCE DU MONOME ADJACENT: --1000--------01000
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1011--------010
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1010--------110
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1010--------011
LE MONOME ADJACENT SUIVANT N'APPARTIENT PAS AU DOMAINE D'ENTREE:--1010--------010
NUMERO   VECTEUR

     1 : --1-----00---00000
     2 : --1------10---00000
     3 : --1------01---00000
     4 : --0101---00---10000
     5 : -0-01110-00---01000
     6 : --1-----000-1-00100
```

```
 7 : --1-----0-0-1-00001
 8 : 0-1---1-0-00-1-00001
 9 : 0-1000---11-1000000
10 : --0101---10---10000
11 : 0-0111-1-10---01000
12 : -1101---110-1-00100
13 : -1101---110-0000010
14 : 0-1----1-10-0-00001
15 : --0101---11---10000
16 : --0111---11---01000
17 : --1------11-1-00100
18 : --1-----11--000010
19 : 0-1-------11--00001
20 : 1-1----------00001
21 : 0-0100-------10001
22 : 0-010000------01001
23 : 0-1---------11-00101
24 : 0-1--------10100011
25 : 1-0101-------10101
26 : 1-010110------01001
27 : 1-1---------00-00101
28 : 1-1--------00-00011
29 : 1-1001---11-00100001
30 : --0101-------010010
31 : --0111-------101011
32 : --1------0---1000110
33 : --1-------------100010
34 : --0101-------110010
35 : --0111-------101001
36 : --1------0---0100110
37 : --1-------------00011
38 : --0101------1-10011
39 : 0-010110-----1-01011
40 : --1------0--11-00111
41 : --0111------0-10011
42 : 0-0101-1----0-01011
43 : -1101---1--11-00111
44 : --1---------0-00100
45 : --0101------1-10100
46 : --0101------1-01100
47 : --0111------0-10100
48 : --0101------0-01100
49 : --1-------------00101
50 : --0101------1-10101
51 : 0-010110-----1-01101
52 : --0101------0-10101
53 : 0-0111-1----0-01101
54 : -1101---1--0--00110
55 : --0111---0-------10110
56 : --0101--0------01110
57 : -1101---1-------10110
58 : --0101-----1--10111
59 : --0101-----1--01111
60 : --0101--1--1--00110
61 : --1-------------00111
62 : --0101-----0--10111
63 : --0101-----0--01111
64 : --0110--------01000
65 : --0100--------01000
66 : 0-0110---11-0000000
```

```
67 : 0-0110?0-------01?01
68 : --0101--------01000
69 : 0-0111---11-0000000
70 : 0-011110------01001
71 : --1000--------01000
72 : 0-100010------01001
73 : --1001--------01000
74 : 0-100110------01001
75 : --1010--------01000
76 : --1010---11---00000
77 : 0-101010------01001
78 : 0-0100---11-0000000
79 : --0100--1--11011---
80 : 0-1100---11-0000000
81 : 0-0000---11-0000000
82 : 0-0101---11-0000000
83 : 0-1101---11-0000000
84 : 0-0001---11-0000000
85 : 0-010110------01001
86 : 0-010100------01001
87 : 0-010111------01001
88 : 0-010110------11001
89 : 0-1-----1------01001
90 : 1-0101-1------01001
91 : 0-0101-1------11001
92 : --0101--------11--0
```

```
*****************************************************
*                                                   *
* TEMPS DE CALCUL DES VECTEURS DE TEST :  23.33s CPU  *
*                                                   *
* TEMPS DE COMPACTAGE DES VECTEURS     :   7.85s CPU  *
*                                                   *
* COUVERTURE DES PANNES                :  97.63%  *
*                                                   *
* COUVERTURE DES PANNES SANS DOMAINE D'ENTREE:  99.09%  *
*                                                   *
*****************************************************
```

Example no. 2.

```
  5 - 3
  01--1
  1--00
  001--
  10--1
  --001

  010
  110
  101
  110
  001
  1
  -----
```

NUMERO DU VECTEUR

Technologie NMOS

1 : 011-1
2 : 11101
3 : 00011
4 : 01100
5 : 1--00
6 : 10110
7 : 001--
8 : 101-1
9 : 00001
10 : 0-000

****

* TEMPS DE CALCUL DES VECTEURS DE TEST : 0.12s CPU *

* TEMPS DE COMPACTAGE DES VECTEURS : 0.09s CPU *

* COUVERTURE DES PANNES : 100.00% *

* COUVERTURE DES PANNES SANS DOMAINE D'ENTREE: 100.00% *

****

Technologie CMOS

NUMERO	VECTEUR	SORTIE
0 :	001--	101
1 :	01101	010
2 :	00100	101
0 :	10001	111
3 :	11101	000
0 :	10001	111
3 :	00011	000
0 :	10001	111
4 :	01100	000
0 :	00001	001
5 :	10000	110
0 :	00001	001
2 :	10101	110
0 :	10001	111
6 :	10110	000
0 :	10001	111
10 :	00000	000

```
*****************************************************************
*                                                               *
* TEMPS DE CALCUL DES VECTEURS DE TEST :        0.19s CPU       *
*                                                               *
* TEMPS DE CALCUL DES VECTEURS CMOS    :        0.29s CPU       *
*                                                               *
* COUVERTURE DES PANNES                           : 100.00%     *
*                                                               *
* COUVERTURE DES PANNES SANS DOMAINE D'ENTREES: 100.00%         *
*                                                               *
*****************************************************************
```

ANNEXE 4

Les vecteurs de test du multiplieur cellulaire de HSURF

LISTE DES VECTEURS DE TEST DU MULTIPLIEUR CELLULAIRE DU MICROPROCESSEUR HSURF

FFFF	B= 5555	C= 0000	D= AAAA
FFFF	B= AAAA	C= FFFF	D= 5555
AAAA	B= FFFF	C= 5555	D= 5555
5555	B= FFFF	C= AAAA	D= 5555
AAAA	B= FFFF	C= 0000	D= AAAA
5555	B= FFFF	C= 0000	D= AAAA
0000	B= 0000	C= 0000	D= 0000
0000	B= 0000	C= FFFF	D= 0000
0000	B= FFFF	C= FFFF	D= 0000
FFFF	B= 0000	C= 0000	D= 0000
FFFF	B= 0000	C= FFFF	D= 0000
FFFF	B= FFFF	C= FFFF	D= FFFF
5555	B= FFFF	C= 5555	D= 0000
AAAA	B= FFFF	C= AAAA	D= FFFF
0000	B= 0000	C= FFFF	D= FFFF
0000	B= 0000	C= FFFF	D= 0002
0000	B= 0000	C= FFFF	D= 0004
0000	B= 0000	C= FFFF	D= 0008
0000	B= 0000	C= FFFF	D= 0010
0000	B= 0000	C= FFFF	D= 0020
0000	B= 0000	C= FFFF	D= 0040
0000	B= 0000	C= FFFF	D= 0080
0000	B= 0000	C= FFFF	D= 0100
0000	B= 0000	C= FFFF	D= 0200
0000	B= 0000	C= FFFF	D= 0400
0000	B= 0000	C= FFFF	D= 0800
0000	B= 0000	C= FFFF	D= 1000
0000	B= 0000	C= FFFF	D= 2000
0000	B= 0000	C= FFFF	D= 4000
0000	B= 0000	C= FFFF	D= 8000
0001	B= 0000	C= 0000	D= FFFF
0002	B= 0000	C= 0001	D= FFFF
0000	B= 0000	C= 0003	D= 0002
0000	B= 0000	C= 0007	D= 0002
0000	B= 0000	C= 000F	D= 0002
0000	B= 0000	C= 001F	D= 0002
0000	B= 0000	C= 003F	D= 0002
0000	B= 0000	C= 007F	D= 0002
0000	B= 0000	C= 00FF	D= 0002
0000	B= 0000	C= 01FF	D= 0002
0000	B= 0000	C= 03FF	D= 0002
0000	B= 0000	C= 07FF	D= 0002
0000	B= 0000	C= 0FFF	D= 0002
0000	B= 0000	C= 1FFF	D= 0002
0000	B= 0000	C= 3FFF	D= 0002
0000	B= 0000	C= 7FFF	D= 0002
0000	B= 0000	C= 0007	D= 0004
0000	B= 0000	C= 000F	D= 0004
0000	B= 0000	C= 001F	D= 0004
0000	B= 0000	C= 003F	D= 0004
0000	B= 0000	C= 007F	D= 0004
0000	B= 0000	C= 00FF	D= 0004
0000	B= 0000	C= 01FF	D= 0004
0000	B= 0000	C= 03FF	D= 0004
0000	B= 0000	C= 07FF	D= 0004
0000	B= 0000	C= 0FFF	D= 0004

A= 0000 B= 0000 C= 1FFF D= 0004
A= 0000 B= 0000 C= 3FFF D= 0004
A= 0000 B= 0000 C= 7FFF D= 0004
A= 0000 B= 0000 C= 000F D= 0008
A= 0000 B= 0000 C= 001F D= 0008
A= 0000 B= 0000 C= 003F D= 0008
A= 0000 B= 0000 C= 007F D= 0008
A= 0000 B= 0000 C= 00FF D= 0008
A= 0000 B= 0000 C= 01FF D= 0008
A= 0000 B= 0000 C= 03FF D= 0008
A= 0000 B= 0000 C= 07FF D= 0008
A= 0000 B= 0000 C= 0FFF D= 0008
A= 0000 B= 0000 C= 1FFF D= 0008
A= 0000 B= 0000 C= 3FFF D= 0008
A= 0000 B= 0000 C= 7FFF D= 0008
A= 0000 B= 0000 C= 001F D= 0010
A= 0000 B= 0000 C= 003F D= 0010
A= 0000 B= 0000 C= 007F D= 0010
A= 0000 B= 0000 C= 00FF D= 0010
A= 0000 B= 0000 C= 01FF D= 0010
A= 0000 B= 0000 C= 03FF D= 0010
A= 0000 B= 0000 C= 07FF D= 0010
A= 0000 B= 0000 C= 0FFF D= 0010
A= 0000 B= 0000 C= 1FFF D= 0010
A= 0000 B= 0000 C= 3FFF D= 0010
A= 0000 B= 0000 C= 7FFF D= 0010
A= 0000 B= 0000 C= 003F D= 0020
A= 0000 B= 0000 C= 007F D= 0020
A= 0000 B= 0000 C= 00FF D= 0020
A= 0000 B= 0000 C= 01FF D= 0020
A= 0000 B= 0000 C= 03FF D= 0020
A= 0000 B= 0000 C= 07FF D= 0020
A= 0000 B= 0000 C= 0FFF D= 0020
A= 0000 B= 0000 C= 1FFF D= 0020
A= 0000 B= 0000 C= 3FFF D= 0020
A= 0000 B= 0000 C= 7FFF D= 0020
A= 0000 B= 0000 C= 007F D= 0040
A= 0000 B= 0000 C= 00FF D= 0040
A= 0000 B= 0000 C= 01FF D= 0040
A= 0000 B= 0000 C= 03FF D= 0040
A= 0000 B= 0000 C= 07FF D= 0040
A= 0000 B= 0000 C= 0FFF D= 0040
A= 0000 B= 0000 C= 1FFF D= 0040
A= 0000 B= 0000 C= 3FFF D= 0040
A= 0000 B= 0000 C= 7FFF D= 0040
A= 0000 B= 0000 C= 00FF D= 0080
A= 0000 B= 0000 C= 01FF D= 0080
A= 0000 B= 0000 C= 03FF D= 0080
A= 0000 B= 0000 C= 07FF D= 0080
A= 0000 B= 0000 C= 0FFF D= 0080
A= 0000 B= 0000 C= 1FFF D= 0080
A= 0000 B= 0000 C= 3FFF D= 0080
A= 0000 B= 0000 C= 7FFF D= 0080
A= 0000 B= 0000 C= 01FF D= 0100
A= 0000 B= 0000 C= 03FF D= 0100
A= 0000 B= 0000 C= 07FF D= 0100
A= 0000 B= 0000 C= 0FFF D= 0100
A= 0000 B= 0000 C= 1FFF D= 0100
A= 0000 B= 0000 C= 3FFF D= 0100
A= 0000 B= 0000 C= 7FFF D= 0100

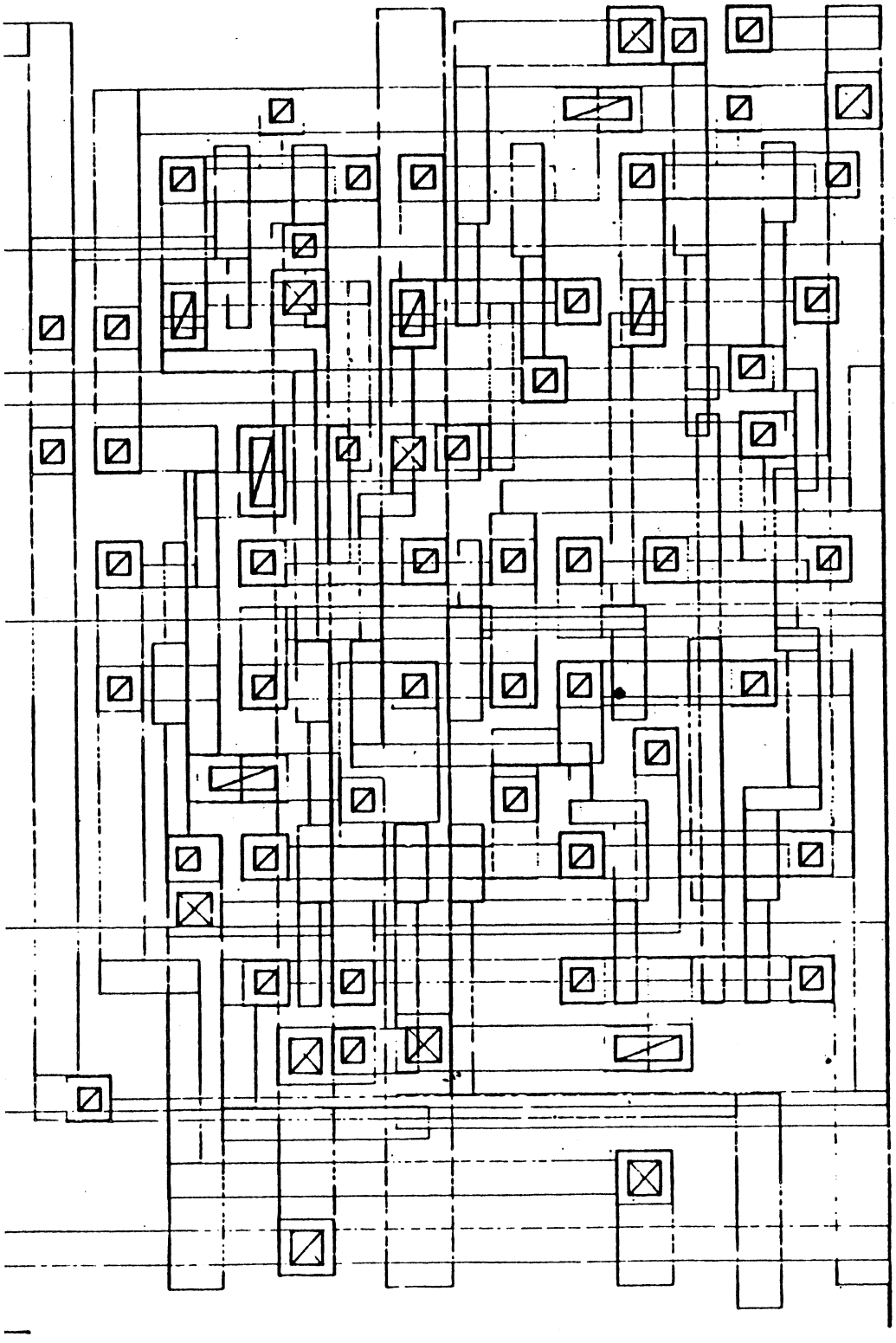
A= 0000 B= 0000 C= 03FF D= 0200
A= 0000 B= 0000 C= 07FF D= 0200
A= 0000 B= 0000 C= 0FFF D= 0200
A= 0000 B= 0000 C= 1FFF D= 0200
A= 0000 B= 0000 C= 3FFF D= 0200
A= 0000 B= 0000 C= 7FFF D= 0200
A= 0000 B= 0000 C= 07FF D= 0400
A= 0000 B= 0000 C= 0FFF D= 0400
A= 0000 B= 0000 C= 1FFF D= 0400
A= 0000 B= 0000 C= 3FFF D= 0400
A= 0000 B= 0000 C= 7FFF D= 0400
A= 0000 B= 0000 C= 0FFF D= 0800
A= 0000 B= 0000 C= 1FFF D= 0800
A= 0000 B= 0000 C= 3FFF D= 0800
A= 0000 B= 0000 C= 7FFF D= 0800
A= 0000 B= 0000 C= 1FFF D= 1000
A= 0000 B= 0000 C= 3FFF D= 1000
A= 0000 B= 0000 C= 7FFF D= 1000
A= 0000 B= 0000 C= 3FFF D= 2000
A= 0000 B= 0000 C= 7FFF D= 2000
A= 0003 B= 0000 C= 0003 D= 0002
A= 0007 B= 0000 C= 0007 D= 0002
A= 000F B= 0000 C= 000F D= 0002
A= 001F B= 0000 C= 001F D= 0002
A= 003F B= 0000 C= 003F D= 0002
A= 007F B= 0000 C= 007F D= 0002
A= 00FF B= 0000 C= 00FF D= 0002
A= 01FF B= 0000 C= 01FF D= 0002
A= 03FF B= 0000 C= 03FF D= 0002
A= 07FF B= 0000 C= 07FF D= 0002
A= 0FFF B= 0000 C= 0FFF D= 0002
A= 1FFF B= 0000 C= 1FFF D= 0002
A= 3FFF B= 0000 C= 3FFF D= 0002
A= 7FFF B= 0000 C= 7FFF D= 0002
A= 0003 B= 0000 C= 0007 D= 0004
A= 0007 B= 0000 C= 000F D= 0004
A= 000F B= 0000 C= 001F D= 0004
A= 001F B= 0000 C= 003F D= 0004
A= 003F B= 0000 C= 007F D= 0004
A= 007F B= 0000 C= 00FF D= 0004
A= 00FF B= 0000 C= 01FF D= 0004
A= 01FF B= 0000 C= 03FF D= 0004
A= 03FF B= 0000 C= 07FF D= 0004
A= 07FF B= 0000 C= 0FFF D= 0004
A= 0FFF B= 0000 C= 1FFF D= 0004
A= 1FFF B= 0000 C= 3FFF D= 0004
A= 3FFF B= 0000 C= 7FFF D= 0004
A= 0003 B= 0000 C= 000F D= 0008
A= 0007 B= 0000 C= 001F D= 0008
A= 000F B= 0000 C= 003F D= 0008
A= 001F B= 0000 C= 007F D= 0008
A= 003F B= 0000 C= 00FF D= 0008
A= 007F B= 0000 C= 01FF D= 0008
A= 00FF B= 0000 C= 03FF D= 0008
A= 01FF B= 0000 C= 07FF D= 0008
A= 03FF B= 0000 C= 0FFF D= 0008
A= 07FF B= 0000 C= 1FFF D= 0008
A= 0FFF B= 0000 C= 3FFF D= 0008
A= 1FFF B= 0000 C= 7FFF D= 0008
A= 0003 B= 0000 C= 001F D= 0010

A= 0007 B= 0000 C= 003F D= 0010
A= 000F B= 0000 C= 007F D= 0010
A= 001F B= 0000 C= 00FF D= 0010
A= 003F B= 0000 C= 01FF D= 0010
A= 007F B= 0000 C= 03FF D= 0010
A= 00FF B= 0000 C= 07FF D= 0010
A= 01FF B= 0000 C= 0FFF D= 0010
A= 03FF B= 0000 C= 1FFF D= 0010
A= 07FF B= 0000 C= 3FFF D= 0010
A= 0FFF B= 0000 C= 7FFF D= 0010
A= 0003 B= 0000 C= 003F D= 0020
A= 0007 B= 0000 C= 007F D= 0020
A= 000F B= 0000 C= 00FF D= 0020
A= 001F B= 0000 C= 01FF D= 0020
A= 003F B= 0000 C= 03FF D= 0020
A= 007F B= 0000 C= 07FF D= 0020
A= 00FF B= 0000 C= 0FFF D= 0020
A= 01FF B= 0000 C= 1FFF D= 0020
A= 03FF B= 0000 C= 3FFF D= 0020
A= 07FF B= 0000 C= 7FFF D= 0020
A= 0003 B= 0000 C= 007F D= 0040
A= 0007 B= 0000 C= 00FF D= 0040
A= 000F B= 0000 C= 01FF D= 0040
A= 001F B= 0000 C= 03FF D= 0040
A= 003F B= 0000 C= 07FF D= 0040
A= 007F B= 0000 C= 0FFF D= 0040
A= 00FF B= 0000 C= 1FFF D= 0040
A= 01FF B= 0000 C= 3FFF D= 0040
A= 03FF B= 0000 C= 7FFF D= 0040
A= 0003 B= 0000 C= 00FF D= 0080
A= 0007 B= 0000 C= 01FF D= 0080
A= 000F B= 0000 C= 03FF D= 0080
A= 001F B= 0000 C= 07FF D= 0080
A= 003F B= 0000 C= 0FFF D= 0080
A= 007F B= 0000 C= 1FFF D= 0080
A= 00FF B= 0000 C= 3FFF D= 0080
A= 01FF B= 0000 C= 7FFF D= 0080
A= 0003 B= 0000 C= 01FF D= 0100
A= 0007 B= 0000 C= 03FF D= 0100
A= 000F B= 0000 C= 07FF D= 0100
A= 001F B= 0000 C= 0FFF D= 0100
A= 003F B= 0000 C= 1FFF D= 0100
A= 007F B= 0000 C= 3FFF D= 0100
A= 00FF B= 0000 C= 7FFF D= 0100
A= 0003 B= 0000 C= 03FF D= 0200
A= 0007 B= 0000 C= 07FF D= 0200
A= 000F B= 0000 C= 0FFF D= 0200
A= 001F B= 0000 C= 1FFF D= 0200
A= 003F B= 0000 C= 3FFF D= 0200
A= 007F B= 0000 C= 7FFF D= 0200
A= 0003 B= 0000 C= 07FF D= 0400
A= 0007 B= 0000 C= 0FFF D= 0400
A= 000F B= 0000 C= 1FFF D= 0400
A= 001F B= 0000 C= 3FFF D= 0400
A= 003F B= 0000 C= 7FFF D= 0400
A= 0003 B= 0000 C= 0FFF D= 0800
A= 0007 B= 0000 C= 1FFF D= 0800
A= 000F B= 0000 C= 3FFF D= 0800
A= 001F B= 0000 C= 7FFF D= 0800
A= 0003 B= 0000 C= 1FFF D= 1000

```
A= 0007  B= 0000  C= 3FFF  D= 1000
A= 000F  B= 0000  C= 7FFF  D= 1000
A= 0003  B= 0000  C= 3FFF  D= 2000
A= 0007  B= 0000  C= 7FFF  D= 2000
A= 0000  B= FFFF  C= 0000  D= 0000
A= FFFF  B= FFFF  C= 0000  D= FFFF
A= 0000  B= 0000  C= 0000  D= FFFF
A= 0000  B= 0000  C= 0001  D= FFFF
```

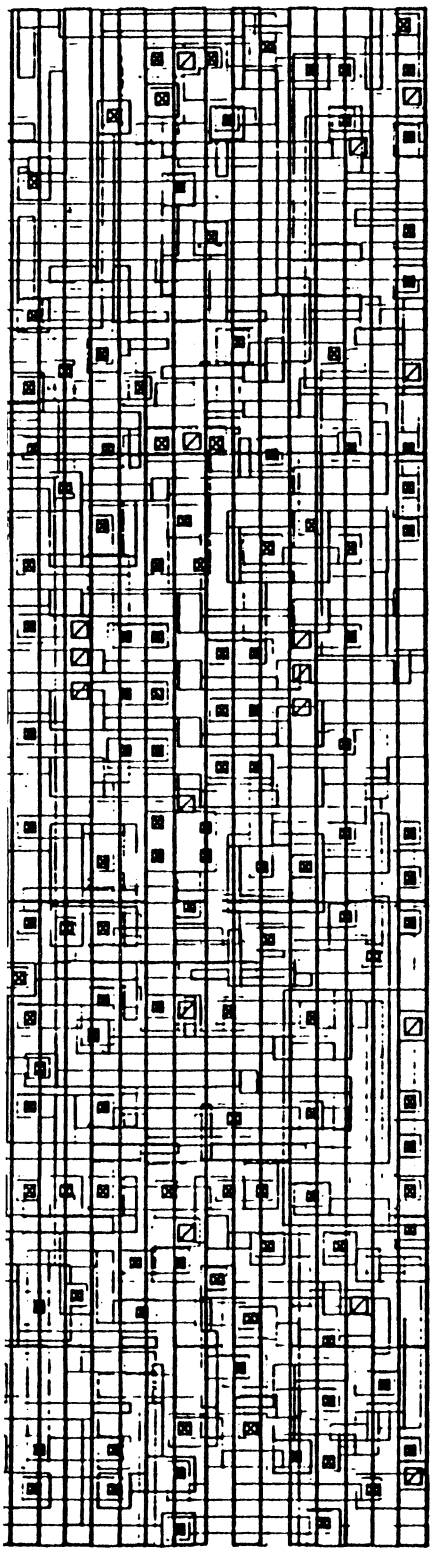
ANNEXE 5

Dessin au micron du multiplieur cellulaire de HSURF



ANNEXE 6

Dessin au micron de l'UAL de HSURF



AUTORISATION de SOUTENANCE

les dispositions de l'article 3 de l'arrêté du 16 avril 1974

les rapport de présentation de Madame le Professeur SAUCIER

Monsieur Philippe DE CHOUDENS

autorisé à présenter une thèse en soutenance en vue de l'obtention du titre DOCTEUR de TROISIEME CYCLE, spécialité "Informatique".

Fait à Grenoble, le 18 octobre 1985

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

RESUME DE THESE

L'importance des applications utilisant des circuits à très ha intégration (transports, spatial, nucléaire...) rend le probl de la fiabilité de fonctionnement de ces circuits particulièrem crucial. Ce travail a donc pour objet de développer un t permettant d'assurer la sécurité de fonctionnement de ces VL La première partie montre l'intérêt dans un tel contexte, d processeur facilement testable ; la deuxième partie dévelo pour de tels microprocesseurs une stratégie de test. Dans troisième partie, est traité le problème de la définition vecteurs de test des PLAs. Enfin dans la dernière partie, test est développé pour un multiplieur itératif.

MOTS-CLES

Circuits à très haute intégration ; Fiabilité ; Test ; VLS Processeur facilement testable ; Microprocesseurs ; Vecte de test ; PLAs ; Multiplieur.