Étude d'un optimiseur logique de P.L.A.

Roland Krasicki

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'Ingénieur

en

Informatique

par

Roland KRASICKI

ETUDE D'UN OPTIMISEUR LOGIQUE DE P.L.A.

SOUTENU LE : 28 Octobre 1983

JURY

Président : MM. L. BOLLIET & Y. RANCHIN

Membres : MM. F. ANCEAU
S. CHUQUILLANQUI
J.P. MOREAU

L'étude présentée dans ce mémoire a été réalisée dans le cadre de la Formation Professionnelle.
La rémunération était assurée par le Ministère du Travail par l'intermédiaire de la Direction du Travail et de la Main-d'Oeuvre.
Ce travail s'est effectué dans l'Equipe de Recherche en Architecture des Ordinateurs du Laboratoire IMAG.

Je tiens à remercier

Monsieur Y. RANCHIN, Professeur du Conservatoire National des Arts et métiers qui m'a fait l'honneur d'accepter mon dossier de thèse,

Monsieur L. BOLLIET, Professeur à l'Université de Grenoble, pour m'avoir incité et aidé à préparer cette thèse, ainsi que pour l'honneur qu'il m'a fait en acceptant de présider le Jury de cette thèse,

Monsieur F. ANCEAU, Professeur à l'Institut National Polytechnique de Grenoble, responsable de l'Equipe de Recherche en Architecture des Ordinateurs pour m'avoir accepté dans son équipe, et pour m'avoir prodigué ses conseils, ses encouragements et sa confiance afin de mener à bien ce travail,

Monsieur J. Y. MOREAU, de la société THOMSON-EFCIS d'avoir bien voulu participer au Jury de cette thèse,

Messieurs S. CHUQUILLANQUI, T. PEREZ-SEGOVIA, auteurs de l'optimiseur topologique "PAOLA", pour m'avoir conseillé et aidé tout au long de ce travail,

Tous les membres de l'équipe de Recherche en Architecture des Ordinateurs et en particulier MM A. JERRAYA, J.P. SCHOELKOPF, A. GUYOT et R. GALLAIS pour leur aide précieuse et leurs critiques constructives,

MMes H. DIAZ et C. CHALAND pour leur amabilité et l'aide qu'elles m'ont apporté à résoudre tous les problèmes administratifs,

Melle S. DIDNEE qui a su mettre en page, dactylographier et corriger le présent mémoire,

Monsieur D. IGLESIAS, du service de reprographie de l'IMAG, qui a assuré le tirage de ce document.

à Sylviane
mes parents
ma famille

ABSTRACT

We present a study of a PLA logical optimizer.
The first part is a resume of the functioning and of the use of PLA's. Then, the topological optimization is studied. The various topological optimization methods are described.
The PAOLA system, which is used for topological optimization is succinctly described.
In the third part, the various logical optimization methods are described. The need of sub-optimal methods is evoked.
The fourth part describes the realization of a sub-optimal logical optimizer, and its results are given.

KEY-WORDS

Classic methods for logical optimization - Sub-optimal methods for logical optimization - Logical optimization - Topological optimization - Programmed logic arrays.

RESUME

Nous présentons l'étude d'un optimiseur logique de PLA.
La première partie de cette étude est un rappel du fonctionnement et de l'utilisation des PLA.
Ensuite, l'optimisation topologique est étudiée. Les différentes méthodes d'optimisation topologique sont décrites.
Le système PAOLA, qui permet d'effectuer cette optimisation topologique est succinctement décrit.
Dans une troisième partie, les diverses méthodes classiques d'optimisation logique sont abordées et étudiées. La nécessité des méthodes sous-optimales est évoquée.
La quatrième partie décrit la réalisation d'un optimiseur logique sous-optimal, et les performances de cet optimiseur sont données.

MOTS-CLE

Méthodes classiques d'optimisation logique - Méthodes sous-optimales d'optimisation logique - Optimisation logique - Optimisation topologique - PLA - Réseaux logiques programmés

ETUDE D'UN OPTIMISEUR LOGIQUE

DE PLA

ETUDE D'UN OPTIMISEUR LOGIQUE DE PLA

INTRODUCTION

L'essor récent de la micro-informatique et des techniques qui en découlent est en train de se faire ressentir dans notre vie actuelle, à tous les niveaux. Aux grands ordinateurs des années 60 ont succédé les super-minis qui, par l'abondance des mémoires, la précision et la vitesse des calculs, sont arrivés naturellement dans les bureaux d'études, où les produits nouveaux sont inventés. L'avénement des circuits intégrés, et plus spécialement des circuits intégrés VLSI (Very Large Scale Integration), à très grande intégration a rendu nécessaire l'emploi d'outils adaptés à la conception et au test de tels circuits. Un concepteur devant dessiner un tel circuit à haut degré d'intégration (100.000 composants) aurait ainsi du travail pour une soixantaine d'années, auxquelles soixante autres années s'ajouteraient, représentant le temps de mise au point du circuit. L'emploi d'outils de Conception Assistée par Ordinateur (CAO) s'avère alors indispensable à ces concepteurs, afin de gérer, archiver, trier, dessiner et analyser l'ensemble des informations nécessaires à la réalisation d'un tel circuit.

Un système intégré de CAO comportera en outre des outils informatiques tels que les outils de simulation, de manipulation de la structure de l'objet en création, des logiciels de tracé de circuits imprimés, des outils interactifs d'implantation et de dessin des masques pour les circuits

intégrés, des programmes de génération de séquences de tests, etc...

Un tel système intégré est en cours de réalisation au Laboratoire d'Architecture des Ordinateurs de l'IMAG. L'activité de recherche dans le cadre de la réalisation d'un tel système consiste à trouver les méthodes (et les outils qui en découlent) susceptibles de réduire notablement le coût de conception d'un microprocesseur ou d'un circuit intégré complexe, défini par son comportement. Pour ce faire, un langage (IRENE) est développé, qui permet de décrire sans ambigüité le comportement et la structure fonctionnelle d'un futur circuit intégré.

La description en langage IRENE est alors analysée de manière à en déduire les spécifications de la partie opérative et de la partie contrôle du futur circuit. Les spécifications de la partie opérative permettent la génération de son masque de fabrication. Les spécifications de la partie contrôle et le choix de son type d'architecture permettent d'en déduire le contenu des parties répétitives (réseaux logiques programmables : PLA, mémoire morte : ROM).

Les très gros PLA sont alors optimisés topologiquement à l'aide d'un programme appelé PAOLA. Un évaluateur topologique appelé TESS permet d'estimer à priori la taille et la forme du futur circuit, à partir des spécifications; on peut dès lors construire le plan de masse du futur circuit, et en déduire les contraintes de forme et de connexion, concernant les différents blocs composant ce circuit.

Les circuits intégrés actuels font de plus en plus appel à des structures régulières tels les PLA, lesquels deviennent alors de plus en plus importants dans les circuits intégrés produits. Comme ces structures sont parmi celles qui sont le plus facilement optimisables en terme de place occupée, la réduction de la taille et donc du coût d'un tel circuit passera par une optimisation des PLA qu'il contient.

Cette optimisation peut se faire selon 2 voies différentes : une optimisation topologique, qui permet un gain de place en gagnant sur la surface inoccupée du PLA, et une optimisation logique, qui permet de réduire cette taille en supprimant du PLA toutes les fonctions qui sont répétées, donc redondantes. Un optimiseur de PLA se composera ainsi d'un optimiseur logique, qui doit déjà réduire le nombre de composants du PLA, et en aval de celui-ci d'un optimiseur topologique, qui permettra au PLA d'occuper une surface moindre, car mieux agencée topologiquement.

L'optimiseur topologique PAOLA existe et fonctionne de manière satisfaisante. Le but de mes travaux a été d'étudier l'optimisation logique, puis de réaliser et de faire fonctionner un optimiseur logique de PLA.

Cette étude, après avoir rappelé les concepts de base des réseaux logiques programmables nous expliquera le fonctionnement de l'optimiseur logique PAOLA. Puis nous étudierons les principales méthodes d'optimisation logique, et enfin, nous décrirons la réalisation et le fonctionnement de l'optimiseur logique.

CHAPITRE I

LES PLA (Programmable Logic Array)

1.1. Introduction aux réseaux logiques.

Les circuits intégrés VLSI (Very Large Scale Integration) sont devenus à l'heure actuelle si complexes que des techniques de conception sont indispensables pour assurer un fonctionnement logique et électrique correct, tout en gardant une période de conception relativement courte. La complexité croissante des systèmes VLSI a rendu l'usage de structures régulières de plus en plus fréquent, car cet usage permet de réduire de façon notable le temps de conception <CHU-83>. Parmi ces structures régulières, les réseaux logiques sont simples et très utilisés. Les PLA (Programmable Logic Array ou Réseau Logique Programmable) en particulier, se sont révélés comme étant des outils très efficaces, car ils peuvent être programmés pour implémenter les fonctions booléennes multi-sorties, et de par leur structure régulière, leur conception reste très simple et très automatisable pour un concepteur de circuit intégré.

1.1.1. Généralités sur la conception des circuits intégrés.

Un aspect important du concepteur de circuits logiques est sa recherche permanente du meilleur compromis possible entre la performance et le coût unitaire <FLE-75>.

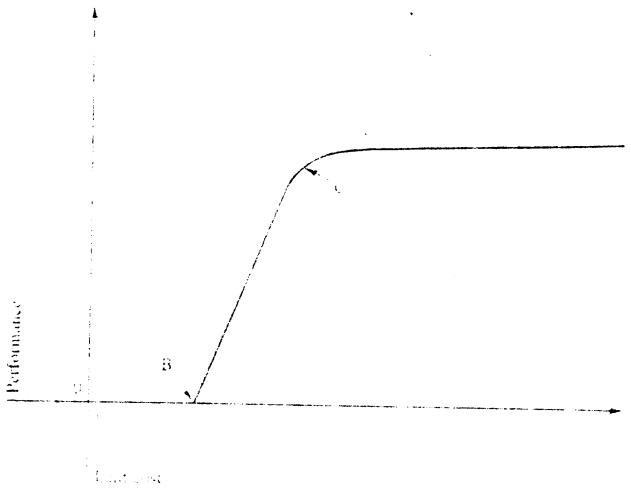


Figure 1.1 Courbe coût-performance des circuits logiques

Le type de courbe qui conditionne le choix du concepteur est illustré par la figure 1.1. Deux points importants sont mis en évidence :

B, la base, ou seuil du coût, qui décale la courbe à droite ou à gauche

C, le point d'inflexion de la courbe.

Dans le passé, quand les circuits étaient faits à partir des composants discrets, ce point d'inflexion était aisé à définir. En effet, le coût dépendait directement du nombre de composants utilisés. La base de la courbe, le point B était facile à déterminer lui aussi, car il était étroitement lié au temps total de conception de ce circuit.

L'avénement de la LSI (Large Scale Integration) et de la VLSI (Very Large Scale Integration) a fait changer les facteurs de cette courbe de façon spectaculaire. Le coût, tout d'abord, doit être considéré comme directement lié à la surface totale de silicium. Ce facteur détermine la forme de la courbe, mais non pas son déplacement B. Ce dernier est affecté par la durée de conception manuelle, les coûts de conception automatisée, etc... En particulier, plus les "puces" deviennent importantes en surface, plus le point B tend à se rapprocher de l'origine. Ainsi, plus la "puce" devient importante et complexe, plus la quantité de "puces" produites est importante et plus le coût de production tend à baisser.

Un problème apparaît cependant à ce niveau. En effet, un circuit, une fois terminé et réalisé est "personnalisé". Mais il se peut qu'à ce stade, tous les paramètres de son fonctionnement ne soient pas définis, car on anticipe, en fait, son utilisation ultérieure. Une solution à ce problème est l'utilisation d'une technologie, qui permet de personnaliser ce circuit, une fois terminé, et quand tous ses paramètres de fonctionnement et d'environnement sont définis. Cette notion est une motivation très forte pour l'utilisation des réseaux logiques.

Les "réseaux logiques" sont considérés dans ce cas comme une utilisation particulière des structures de mémoires mortes, pour effectuer des opérations logiques.

1.1.2. Les réseaux logiques.

Un réseau logique fonctionne, dans son principe, de la manière suivante : on présente les bits en entrée des structures de mémoire (c'est-à-dire les bits d'adresse). La reconnaissance de cette entrée entraîne le processus d'extraction d'un résultat associé pré-déterminé dans le réseau. Comme les fonctions générées par un tel réseau ne dépendent que de la "personnalité" de ce réseau (c'est-à-dire son contenu), la logique peut, en principe, être changée de la même façon que la mémoire est écrite.

Un réseau logique est contenu dans une structure en forme de mémoire, dans le sens que les variables sont utilisées comme des bits d'adresse, alors que les bits de sortie ont été sélectés et combinés de façon à ne donner qu'un seul bit qui est la valeur de la fonction rangée dans le réseau, pour la combinaison particulière des bits d'entrée.

1.1.3. Problèmes antérieurs des réseaux logiques.

Jusqu'à une période récente, les réseaux logiques n'étaient pas assez intéressants pour être largement utilisés. Nous allons voir quelles sont les raisons de cette désaffection :

1 - La technique des réseaux logiques occupait trop de place pour être compétitive. Avec des puces de silicium relativement petites, la quantité de logique que l'on pouvait placer à l'intérieur était trop faible.

Ce problème était aggravé par le fait que les techniques permettant de minimiser la taille de tels réseaux n'existaient pas.

2 - Les réseaux logiques étaient inconnus, ou presque, des concepteurs. En plus du délai nécessaire à l'assimilation d'une nouvelle technique, cette technique ne leur paraissait pas beaucoup plus intéressante. De plus, les concepteurs craignaient que ces réseaux occupent une surface plus grande que celle de la logique cablée conventionnelle.

3 - Les réseaux logiques étaient mal adaptés aux technologies existantes. Nous avons vu que les réseaux logiques ressemblent aux mémoires à semi-conducteurs. Mais les technologies conçues pour les utilisations de mémoires ne sont pas nécessairement adaptées à une utilisation logique. Par exemple, une mémoire doit avoir une vitesse de lecture et d'écriture la plus grande possible. Cependant, pour de la logique, la vitesse d'écriture n'est pas primordiale, car seule la lecture est importante. Ainsi, certains réseaux logiques sont utilisés uniquement en lecture, comme c'est le cas pour les ROM (Read Only Memory).

4 - La souplesse d'utilisation des réseaux logiques était souvent contrebalancée par la difficulté de changer les connexions entre éléments. Aussi, la possibilité de changer facilement la fonction d'un réseau logique avait peu de valeur face au temps nécessaire à la modification des connexions.

Cependant, la complexité croissante des circuits intégrés a conduit les concepteurs à réviser leurs notions en ce qui concerne les réseaux logiques, et à prendre en compte, de plus en plus, les avantages qui leur étaient ainsi offerts.

1.1.4. L'utilisation actuelle des réseaux logiques.

1 - Comme la complexité des machines va croissant, et comme de plus en plus de fonctions sont intégrées dans les circuits, il paraît évident qu'il ne sera pas possible de faire un de ces circuits de façon définitive, en ayant tout prévu à l'avance. Aussi, l'intérêt des concepteurs envers les réseaux logiques s'accroit, car ceux-ci leur permettent de concevoir un circuit, puis ensuite de programmer la logique du circuit de façon à ce que les fonctions que l'on en attend soient remplies.

2 - Les techniques de conception ont beaucoup évolué. En effet, la conception d'un circuit logique important, par des méthodes classiques nécessite beaucoup de compétence, d'habileté, et de temps, surtout si ce circuit est important et intègre des fonctions complexes. A l'heure actuelle, de nombreux outils d'aide à la conception ont été développés pour aider les concepteurs dans leur travail.

3 - Les progrès technologiques ont rendu l'utilisation des réseaux logiques très intéressante. Les densités d'intégration des éléments ont augmenté de façon spectaculaire et ont rendu

l'utilisation des réseaux logiques très compétitive, grâce à leur faible consommation, la réduction de la place occupée, et la complexité des fonctions remplies.

4 - La souplesse des interconnexions est maintenant possible. Aussi, les différents réseaux seront plus facilement implémentables sur les circuits, car ils auront une occupation du circuit mieux répartie.

5 - Enfin, le temps nécessaire à la conception d'un réseau logique est devenu de plus en plus faible, grâce au développement des outils d'aide à la conception, comme nous l'avons vu en -2-. Ceci entraîne un impact sur le plan économique, car le coût de conception décroît, et le concepteur est plus libre dans son travail.

Après avoir étudié les avantages et les inconvénients des réseaux logiques, nous allons tenter de concevoir un tel réseau.

1.1.5. Conception des réseaux logiques.

Nous utiliserons un additionneur 2 bits pour illustrer le paragraphe suivant.

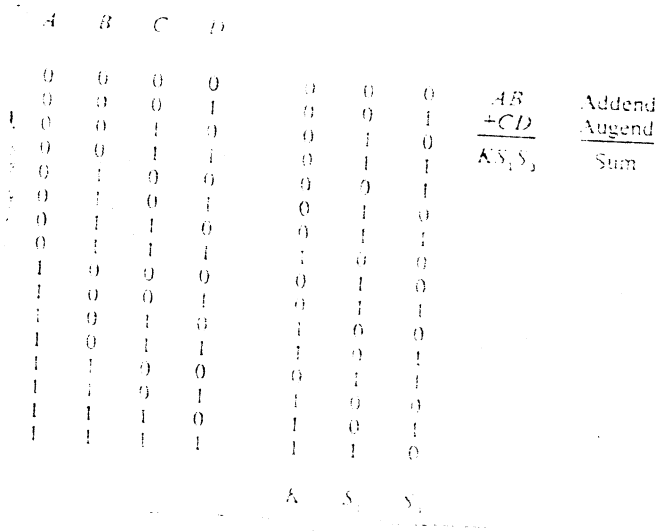


Figure 1.2 Table de vérité d'un additionneur 2 bits

L'additionneur 2 bits accepte 4 variables d'entrée A,B,C et D, et génère 3 sorties SO,Sl et K qui sont respectivement les poids faibles, les poids forts et la retenue de la somme des 2 entrées. La figure 1.2 nous montre la table de vérité de cet additionneur.

Une possibilité de faire cet additionneur serait de faire une mémoire conventionnelle, à accès aléatoire. L'interrogation de cette mémoire par une combinaison de 4 bits en entrée, par l'intermédiaire d'un décodeur 4 bits permettrait de "trouver" la ligne de sortie correspondante, dans la table de vérité. Ce type de table est limité à de petits problèmes, car le nombre de lignes du tableau est de 2^n , n étant le nombre de variables en entrée. Ainsi, pour 16 entrées, 65000 lignes environ seraient nécessaires pour contenir la solution à ce problème. Chaque terme de la table, en entrée, est appelé un produit, c'est-à-dire une fonction ET (un produit) de toutes les variables.

L'équation booléenne de chaque sortie peut être obtenue en faisant le ou les mintermes pour lesquels on trouve un "1" dans la table des sorties. Ainsi, l'équation booléenne pour la retenue, par exemple, contient les mintermes 0111, 1010, 1011, 1101, 1110 et 1111, ce qui équivaut à $\bar{A}BCD$, $A\bar{B}C\bar{D}$, $A\bar{B}CD$, $AB\bar{C}D$, $ABC\bar{D}$ et ABCD. On peut alors écrire, après simplification selon les règles d'algèbre booléenne, les équations des sommes S0, S1 et de la retenue K :

$$S0 = B\bar{D} \cup \bar{B}D \qquad (1)$$

$$S1 = \bar{A}\bar{B}C \cup A\bar{B}\bar{C} \cup \bar{A}C\bar{D} \cup A\bar{C}\bar{D} \cup \bar{A}BC\bar{D} \cup ABCD \qquad (2)$$

$$K = AC \cup ABD \cup BCD \qquad (3)$$

Le symbole $\cup$ représente l'opération OU logique. Les informations stockées dans la table de vérité vue plus haut peuvent être rangées et accédées d'une manière tout à fait différentes. Au lieu d'avoir un seul décodeur dans lequel 4 entrées génèrent 16 lignes, on pourrait utiliser 4 décodeurs à une seule entrée.

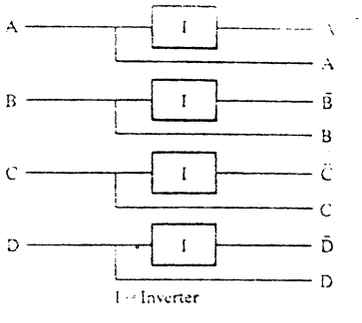


Figure 1.3

Quatre décodeurs à une entrée.

La figure 1.3 nous montre que le nombre de lignes nécessaires dans ce cas passe de 16 à 8 lignes. Il apparaît alors que si le nombre d'entrées augmente, le gain en nombre de lignes devient proportionnellement plus important. Si l'on utilise 2 lignes pour une table avec adressage conventionnel, cette structure nécessite 2 n lignes.

En reprenant l'exemple des 16 entrées, le nombre de lignes nécessaires passe de 65000 à 32 seulement. La fonction donnée par une colonne quelconque est obtenue en sélectionnant l'entrée appropriée dans chaque décodeur puis en effectuant un ET logique de toutes ces entrées. Pour cette raison, on appelle cette partie la partie ET (AND array).

On remarque que chaque variable peut prendre indifféremment les couples de valeurs 0-1, 1-0 ou 1-1, 0-0. Le couple 0-0 n'est pas possible mais il peut être interprété comme une "valeur d'inhibition" assignée à la variable, car il force la valeur du monôme à 0, indépendamment des autres variables.

De même, pour le couple 1-1, nous avons une situation indifférente (Don't Care), pour cette variable particulière. Ceci équivaut à éliminer cette variable, par simplification booléenne, car cette variable n'aura aucune influence sur le ET logique pour cette colonne.

De même, 0-1 correspond à X = 1 et 1-0 correspond à X = 0.

Toutes les fonctions logiques peuvent être représentées par des OU logiques de plusieurs produits, ce qui

fait qu'une seconde table qui effectue le OU logique des produits pour obtenir la fonction désirée est nécessaire. Cette seconde table est appelée la partie OU (OR array) du PLA.

Si l'on se réfère à une telle structure, nous pouvons alors illustrer le profil de l'additionneur 2 bits.

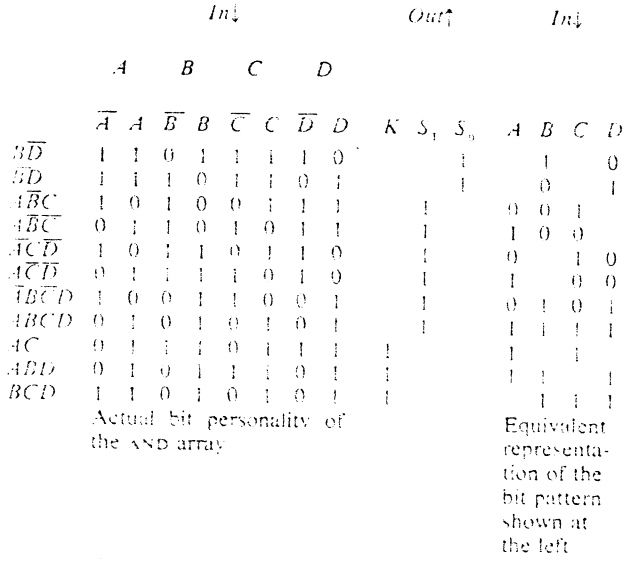


Figure 1.4

Additionneur 2 bits avec décodeur 1 entrée

La figure 1.4 nous montre ce profil. Pour simplifier la représentation de l'information, dans la partie ET, nous adopterons les conventions suivantes :

un 0 correspond à l'état 1-0, un 1 correspond à l'état 0-1, un espace correspond à l'état 1-1.

Notons au passage que l'état 0-0 n'est pas représenté, parce que la sortie du mot concerné serait toujours à zéro. Les 1 de la partie OU correspondent aux produits de la partie ET sur lesquels on doit effectuer le OU logique, pour générer la fonction désirée. Les blancs de la partie OU ne correspondent pas aux

"Don't care" mais ils signifient qu'aucune connexion n'existe entre la partie OU et la partie ET.

Nous remarquerons que la taille de la partie ET est de 88 bits, celle de la partie OU est de 33 bits. Au total 121 bits sont utilisés avec cette méthode, au lieu des 48 bits nécessaires avec une méthode conventionnelle. Malgré celà, cette dernière méthode présente des avantages non négligeables.

Premièrement, le coût total de l'ensemble des décodeurs à une entrée est beaucoup plus faible que celui d'un décodeur unique.

Deuxièmement, on remarque sur la figure 1.4 que la lecture des expressions booléennes est plus simple et plus aisée. Cet avantage n'est pas directement appréciable sur un petit exemple, mais pour des problèmes plus importants, la génération de la table de vérité pour les concepteurs s'en trouve facilitée.

Ces raisons expliquent pourquoi les décodeurs à une entrée de ce type sont très populaires auprès des concepteurs.

D'autres méthodes sont également employées pour réaliser ces décodeurs, mais le problème est de savoir quelle est la meilleure façon de réaliser ces décodeurs.

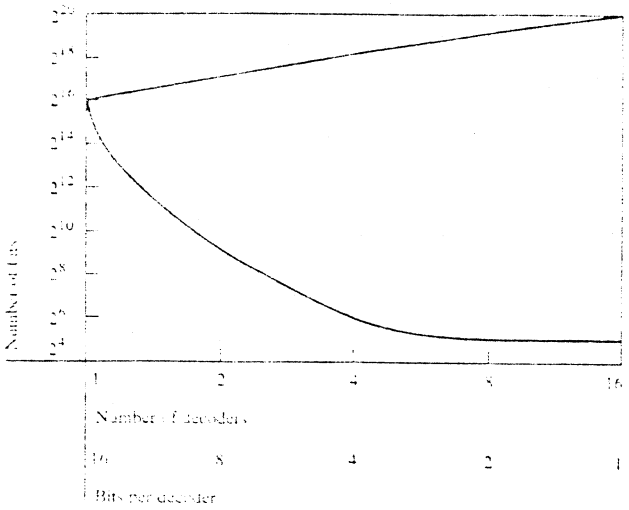


Figure 1.5

Représentation d'une fonction de 16 variables en fonction du nb de bits et de décodeurs

La figure 1.5, qui compare le nombre minimum et maximum de bits nécessaires à la réalisation de n'importe quelle fonction de 16 variables, pour la partie ET donne une réponse à cette question. Nous constatons que le nombre de bits minimum et maximum est de 2^{16} dans les deux cas. Mais par contre, le maximum possible est de 2^{20} pour le décodeur à une entrée alors que le minimum est de 2^{5} . Cet exemple nous montre à quel point le choix du concepteur d'un circuit, dès l'origine, peut influencer sa réalisation, son coût et ses performances ultérieures.

Après l'étude théorique de la conception d'un réseau logique, nous allons examiner leur implémentation physique.

1.1.6. Implémentation physique des réseaux logiques.

Au vu de ce qui précède, nous pouvons dire qu'un réseau logique nécessite, en plus de la fonction de décodage, un

support permettant de stocker les bits, un opérateur faisant le ET des entrées et un opérateur faisant le OU du résultat de ce ET logique.

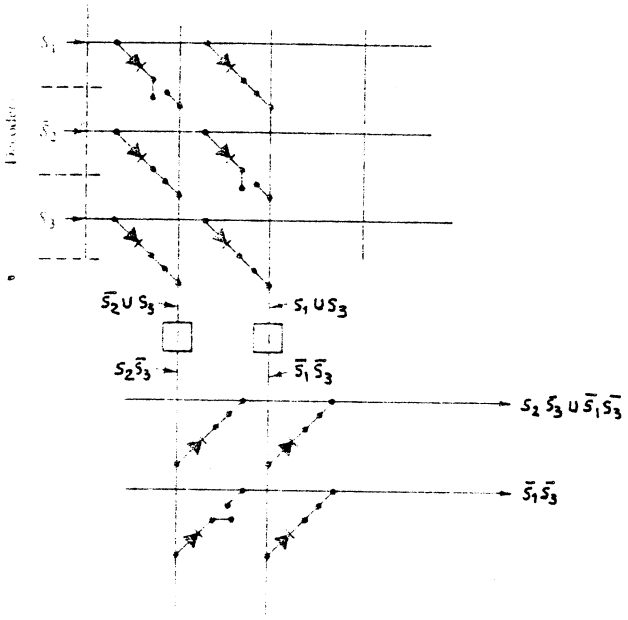


Figure 1.6

Implémentation idéale des opérations ET et OU dans un réseau logique

La manière la plus simple de faire le ET et le OU est illustrée par la figure 1.6 où l'état des bits est symbolisé par des interrupteurs ouverts ou fermés, correspondant aux 0 et 1, respectivement. Les signaux provenant des différentes lignes de décodeurs sont amenés aux inverseurs (I) situés juste avant la partie OU. Un interrupteur fermé seulement (1 logique) est suffisant pour activer une ligne de mots, car le signal est alors inversé. Ceci équivaut à faire le ET logique du complément des sorties des décodeurs.

Pour un support accessible en lecture seulement, les "interrupteurs" peuvent être facilement réalisés comme dans la

figure 1.6. Par contre, pour de la logique accessible en lecture/écriture, l'élément idéal pour réaliser les interrupteurs n'est pas disponible. Une solution serait une cellule de mémoire servant à commuter chaque intersection à 0 ou à 1. Si l'on ajoute une porte ET à chaque intersection, la même fonction que celle de la figure 1.6 exécutée par les diodes et les interrupteurs, sera remplie.

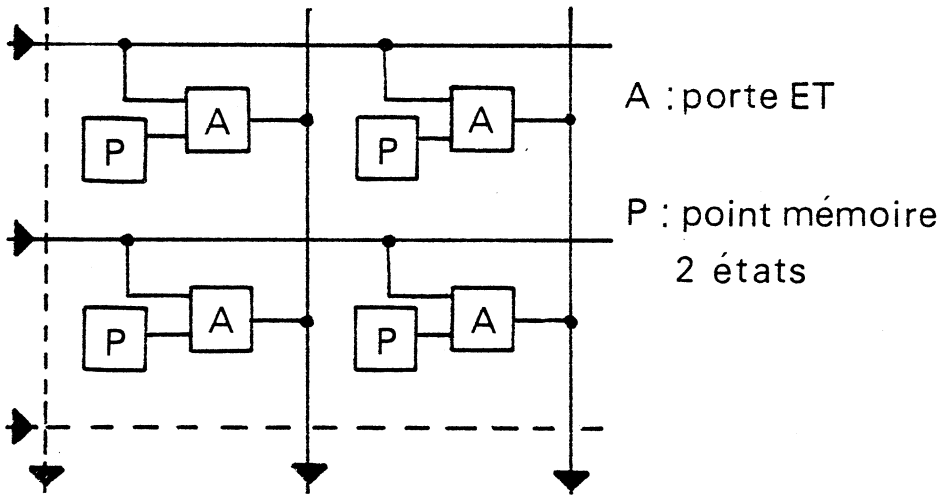


Figure 1.7 Représentation d'un réseau à l'aide de points mémoire et de portes ET.

La figure 1.7 nous montre une réalisation possible d'un tel dispositif où P est un point mémoire à 2 états (0 ou 1), et A une porte ET logique.

Une approximation de cet interrupteur idéal, à l'heure actuelle est donnée par la technologie N-MOS (canal N, metal-oxyde semi-conductor). L'interrupteur est remplacé par le canal du transistor N-MOS. En appliquant un potentiel V supérieur au seuil du transistor, au moyen de la grille, le transistor devient

passant (l'interrupteur est fermé : état logique 1). La figure 1.8 nous montre comment est implémenté un tel transistor dans un réseau logique.

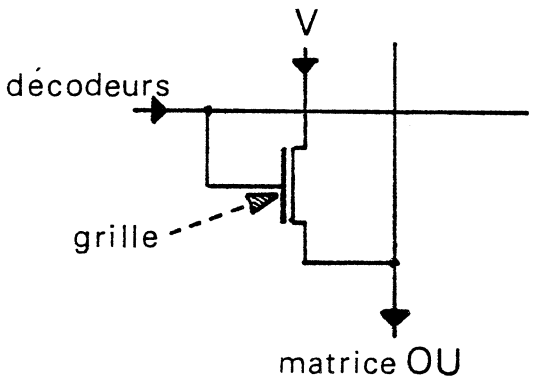


Figure 1.8

Utilisation de transistors N-MOS pour implémenter un réseau logique

Les progrès actuels de la technologie ont introduit l'utilisation d'autres types de transistors, les C-MOS (Complementary MOS) qui ont l'avantage sur la technologie N-MOS d'avoir une consommation très faible, en état de fonctionnement, ce qui est très intéressant dans le cas actuel de circuits comportant un nombre de transistors de plus en plus élevé (il est à l'heure actuelle de l'ordre de 10^5 transistors par puce).

Cependant, dans les PLA, les transistors sont de type NMOS, les transistors CMOS étant utilisés pour réaliser des fonctions telles que les amplificateurs, inverseurs, etc...

Pour résumer, nous pourrons considérer qu'un réseau logique se compose de trois parties, comme le montre la figure 1.9

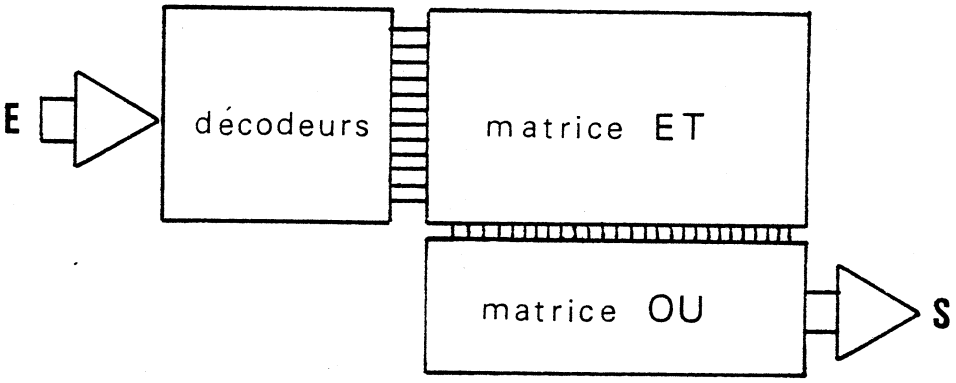


Figure 1.9 Les trois parties de base d'un réseau logique

Ces trois parties sont le décodeur, la partie ET et la partie OU. Nous retrouvons cette forme dans tous les circuits sous forme de réseaux logiques, et plus spécialement dans les PLA (Programmable Logic Array).

1.2. Généralités sur les PLA.

Les PLA sont un type particulier de réseaux logiques, mais leur utilisation peut dépendre de la fonction que ce PLA doit remplir <PER-79>.

1.2.1. Les PLA réalisant des fonctions logiques

Un PLA sera vu de l'extérieur comme un bloc réalisant des fonctions logiques, telles que $S_i = F_i$ (e1,e2,...,en)

avec F_i = fonction logique

S_i = état des sorties du bloc

e1,e2,...,en = état des entrées du bloc.

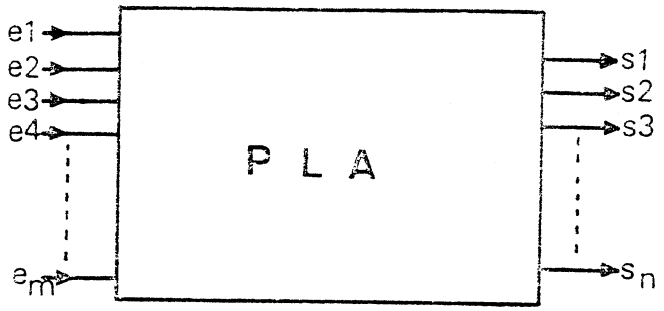


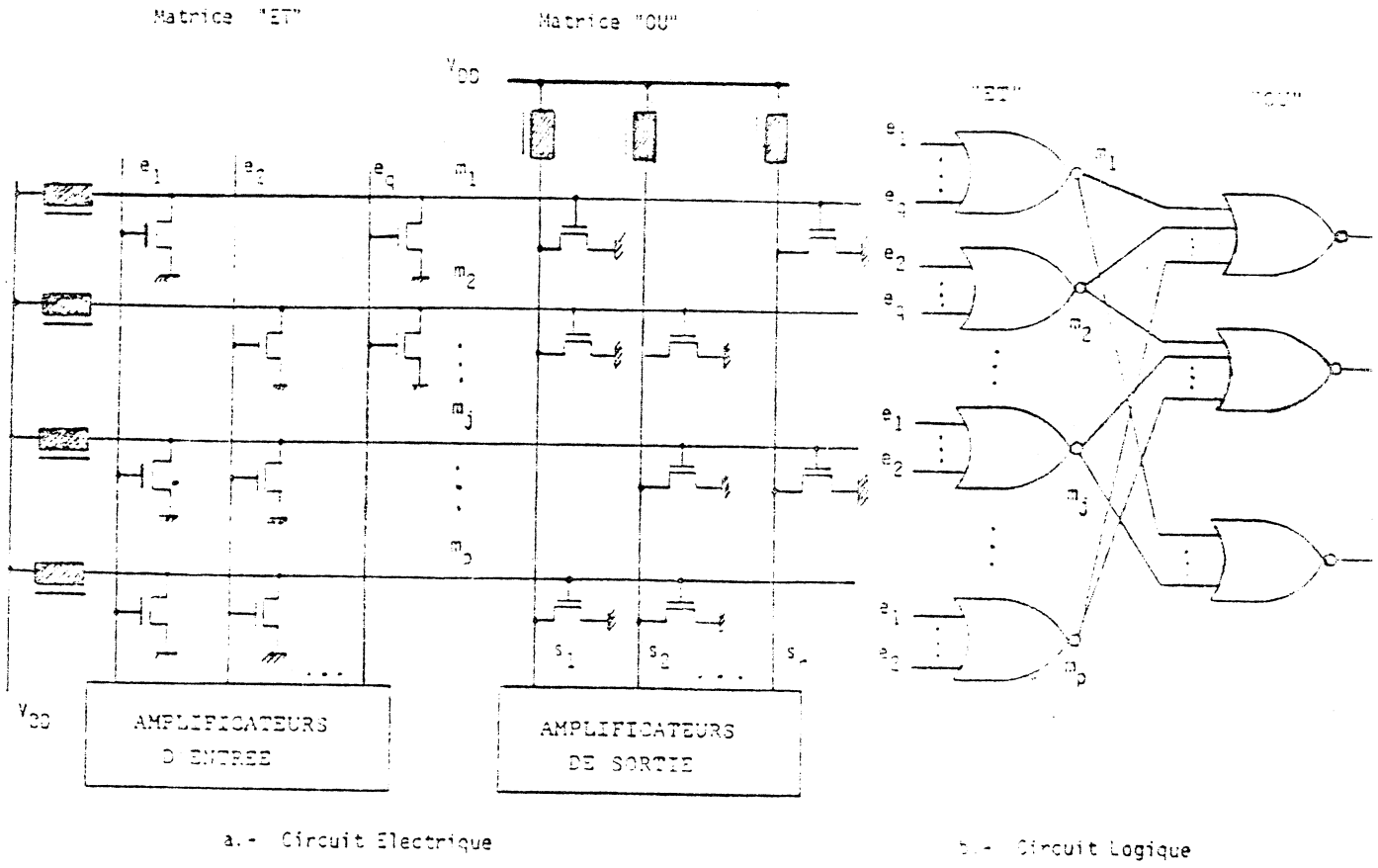
Figure 1.10

Le PLA vu comme fonction logique

Toute fonction logique peut se découper en somme de produits (ou produit de sommes) des entrées et des entrées complémentées. Les PLA pourront réaliser les fonctions logiques voulues sous une de ces deux formes. Un PLA remplissant une (des) fonction(s) logique(s) sera composé :

- D'un amplificateur à sorties directes et complémentées,
- D'une matrice réalisant le ET logique des entrées et de leurs compléments,
- D'une matrice réalisant le OU logique des sorties de la première matrice.

La réalisation électronique des fonctions ET et OU sera :



a.- Circuit Electrique

b.- Circuit Logique

Figure 1.11 Structure classique des "PLA".

1.2.2. Les PLA utilisés comme mémoire morte.

Le PLA peut être considéré comme une mémoire morte organisée en m mots de n bits, n étant le nombre de sorties du PLA. Dans ce cas, m devra être inférieur à 2^K , K étant le nombre d'entrées. Dans une telle mémoire, plusieurs mots peuvent être activés au même moment, et l'information obtenue en sortie est le OU bit à bit des mots activés.

La figure 1.12 nous montre l'organisation d'un PLA faisant fonction de mémoire morte.

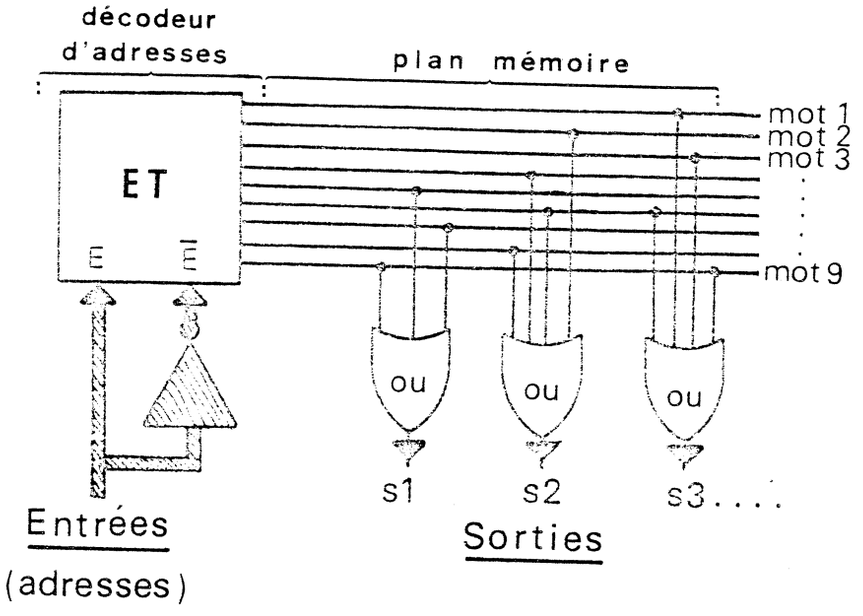


Figure 1.12 PLA utilisé en mémoire morte

A l'intersection entre une ligne de bits et une ligne de mots, il existera ou non une connexion selon l'information contenue dans ce point mémoire.

Pour une mémoire, la fonction OU est réalisée par chaque ligne de bits, mais à un instant donné, un seul mot peut être activé. Dans les PLA, les points qui ne doivent pas activer les lignes de bits n'ont pas de connexion physique avec la ligne de bits.

Un PLA est donc une ROM pour laquelle les relations entre les adresses des mots et leur contenu ont été relevées, condensées sous forme canonique et implantées dans des matrices. Ceci permet d'avoir un gain en surface par rapport à une ROM qui est relativement important, mais au détriment du coût de conception qui est plus élevé.

1.2.3. Les PLA utilisés comme mémoire associative.

Dans ce cas, on considère que le PLA a plusieurs mots qui peuvent être activés par la même configuration des entrées. Les sorties de ce PLA correspondent à la somme logique, bit à bit, des données associées à chacun des mots activés.

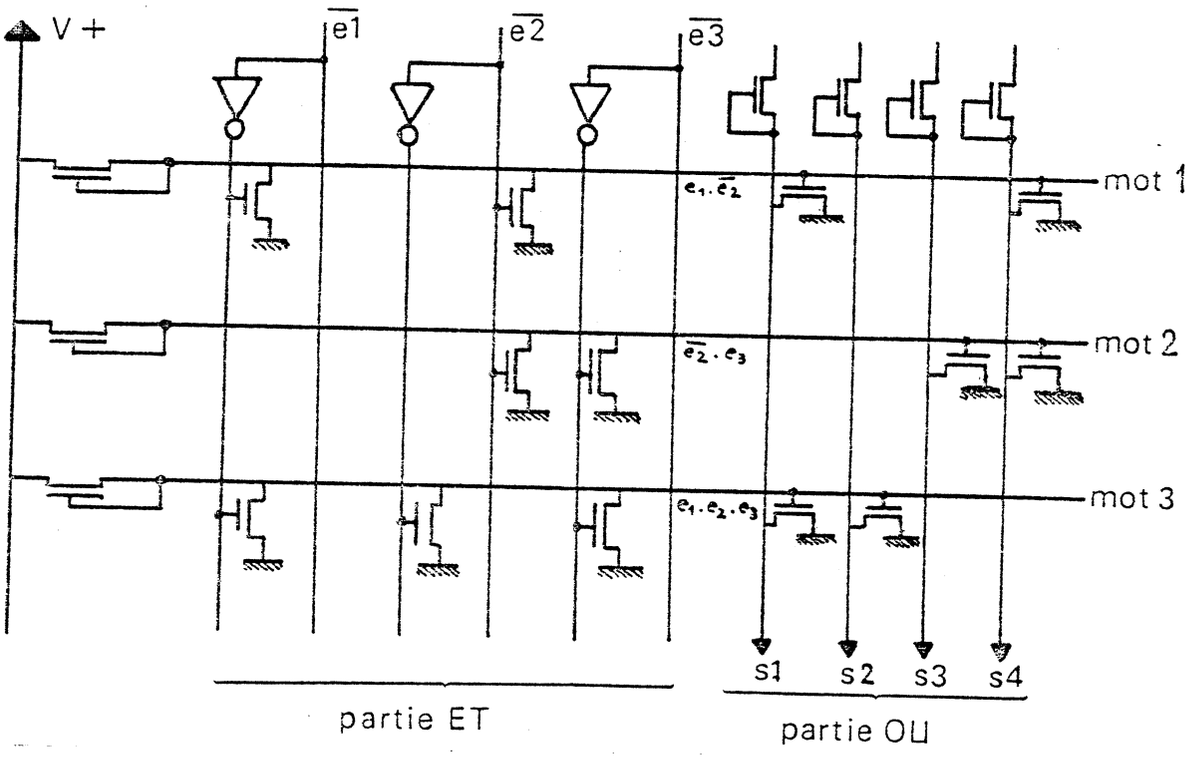


Figure 1.13 PLA utilisé comme mémoire associative

Chaque entrée est d'abord associée à son inverse, dans le décodeur, avant d'attaquer la première matrice. Dans celle-ci se fait la comparaison des entrées avec les configurations programmées. Plusieurs configurations peuvent être reconnues pour la même entrée. A chacune de ces configurations (mots) sont associées plusieurs informations représentant les sorties de la mémoire associative.

Ceci revient à dire que dans un tel PLA, plusieurs "mots" peuvent être reconnus en même temps, et la sortie correspondante sera le OU, bit à bit, des données associées aux mots reconnus.

1.2.4. Les PLA dans la partie contrôle des micro-processeurs.

L'évolution des techniques de conception des circuits intégrés a rendu l'utilisation des PLA dans la partie contrôle des micro-processeurs très courante, mais la fonction remplie par ces PLA varie d'une réalisation à l'autre.

Ces fonctions principales sont le décodage des instructions, le séquencement des instructions, la génération et la validation des micro-commandes.

1.2.4.1. Décodage des instructions.

Dans ce cas, le PLA détermine les commandes, ou la famille d'instructions. Les entrées du PLA sont l'instruction elle-même, et quelques bascules dont l'état peut modifier l'interprétation des instructions. La sortie du PLA sera schématiquement la décomposition de l'instruction en éléments directement assimilables par la partie opérative du micro-processeur (n° de l'accumulateur dans lequel s'effectue l'opération, type d'adressage, type d'opération, valeur à stocker, etc...).

1.2.4.2. Séquencement des instructions.

A ce niveau, on peut retrouver deux types d'approches distinctes selon le type de la partie contrôle de la machine (type MEALEY ou type MOORE).

Si la partie contrôle fonctionne comme une machine de MEALEY (fig. 1.14), le PLA détermine la séquence à suivre ainsi que les micro-commandes. En entrée du PLA, on doit présenter l'instruction, les signaux de contrôle et les bascules d'états.

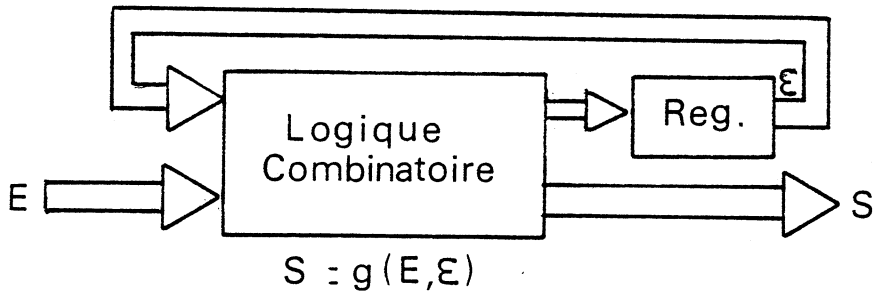


Figure 1.14 Machine de MEALEY

Le nombre de sorties d'un tel PLA est en général très important, ce qui entraîne des PLA de grande taille utilisés seulement dans des machines à algorithme relativement simple.

Ce problème est en partie résolu par les parties contrôle fonctionnant comme une machine de MOORE (fig. 1.15).

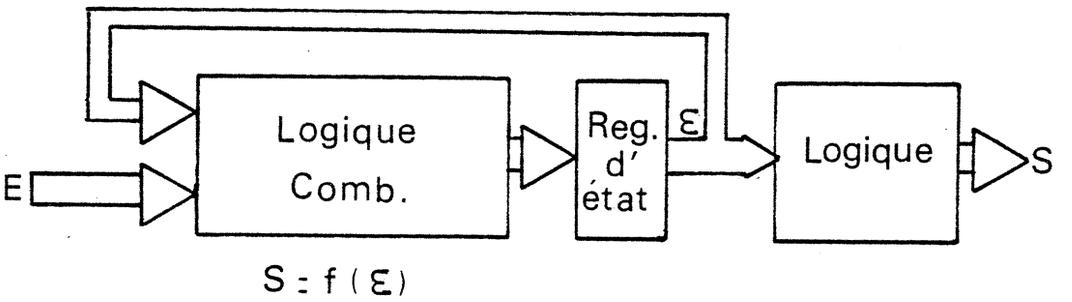


Figure 1.15 Machine de MOORE

Les adresses des micro-instructions sont alors confondues avec l'état de la machine, ce qui fait qu'on peut utiliser le PLA conjointement à un compteur de micro-programme et un incrémenteur pour déterminer si la séquence est linéaire, ou si l'on doit effectuer un branchement.

Cependant, pour des réalisations complexes, les machines de MEALEY seront les plus utilisées, car leur implémentation, surtout pour des PLA de grande taille, sera plus aisée que celle de machines de MOORE.

1.2.4.3. Génération et validation de micro-commandes.

Ce type de PLA accepte en entrée les commandes à valider ainsi que les signaux de temps issus de l'horloge.

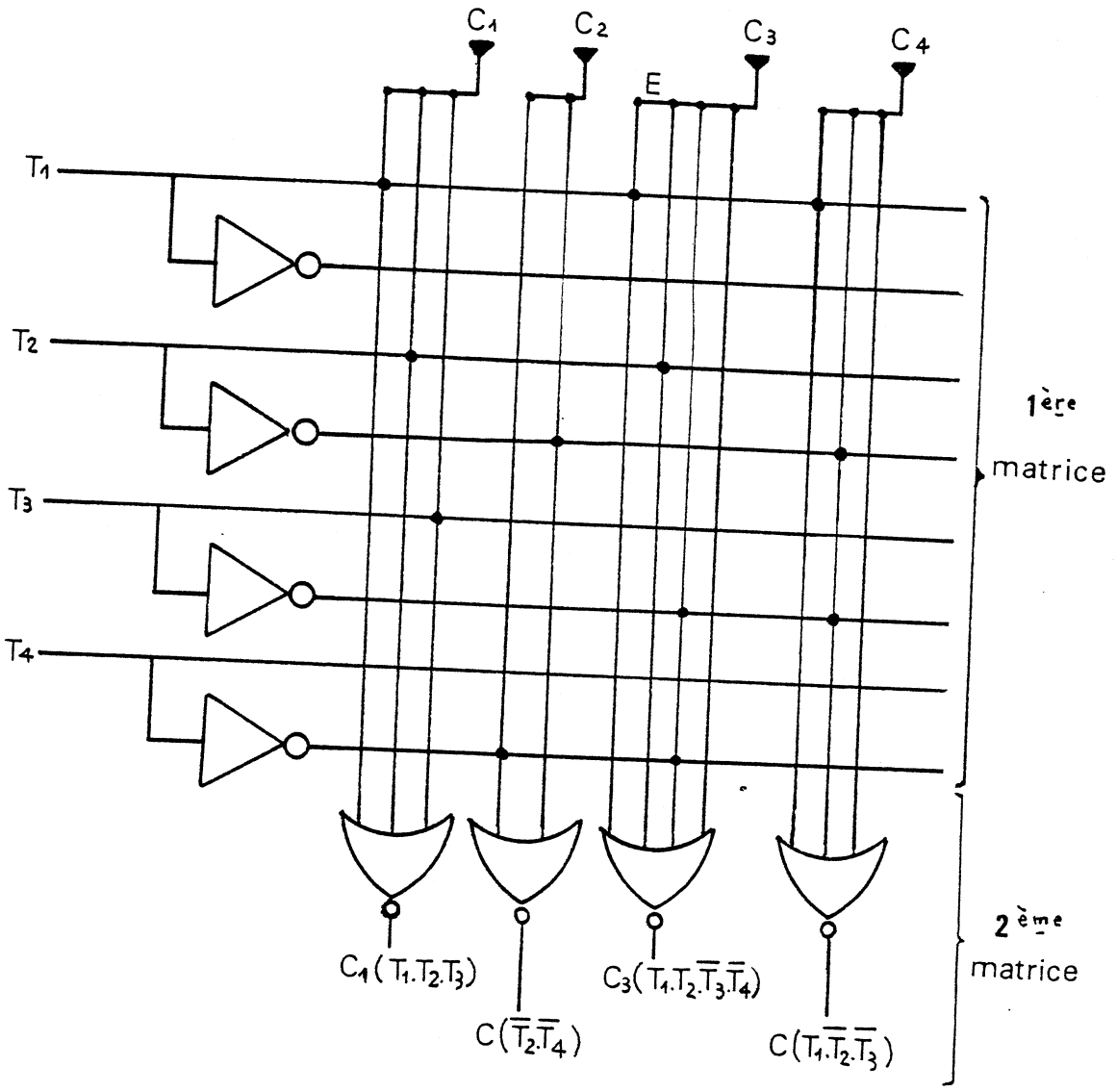


Figure 1.16 PLA de validation de micro-commandes

Les signaux de temps parcourent toute la première matrice, alors que les signaux de commande sont plus localisés. La forme de la seconde matrice dépend étroitement de la distribution des entrées par rapport aux sorties.

1.2.5. Autres fonctions réalisées par les PLA.

Nous avons étudié ci-dessus les fonctions "classiques" remplies par les PLA, mais dans la conception de circuits intégrés, les PLA sont parfois amenés à remplir d'autres fonctions:

- Remplacement de mémoires mortes dans les machines micro-programmées.

- Génération de valeurs immédiates pour l'utilisation de fonctions arithmétiques complexes.

- Interface entre la partie contrôle et la partie opérative d'un micro-processeur lorsque les commandes sont générées sous forme de code.

De tout ceci, il ressort que l'utilisation des PLA dans la conception de circuits intégrés est de plus en plus répandue. Cependant subsiste un inconvénient notable à cette utilisation : la surface occupée par les PLA est généralement supérieure à la logique anarchique qu'ils remplacent. Le problème sera alors de trouver une forme de PLA qui soit la plus petite possible et qui s'adapte au mieux à la surface disponible dans le circuit intégré. Les espaces inutilisés, aussi bien à l'intérieur du PLA qu'entre les blocs du circuit, doivent être réduits à leur dimension minimale, en ajustant la forme de chaque bloc à sa destination finale <ANC-82>.

Nous allons voir dans le chapitre 2 les différentes méthodes d'optimisation topologique des PLA.

CHAPITRE II

L'OPTIMISATION TOPOLOGIQUE DES PLA

2.1. Généralités sur l'optimisation topologique.

Les PLA ont, nous l'avons vu au chapitre I, de gros avantages sur la logique anarchique. De plus, ils peuvent être générés de façon automatique, et leur dessin, également, peut être fait de façon automatique. Par contre, les PLA ont un inconvénient majeur : ils sont en général assez "creux", c'est-à-dire qu'une grande partie de leur surface est inutilisée. Il est donc primordial de tenter de réduire cette surface inutilisée d'une part, et aussi de tenter d'adapter les PLA aux blocs qui les entourent, dans un circuit intégré.

Les PLA classiques sont en effet composés de deux matrices ET et OU, matrices rectangulaires. Les sorties sont alors parallèles aux entrées, et la taille du PLA dépend directement du nombre d'entrées, de sorties et de monômes, ainsi que des règles de dessin du PLA.

La surface d'un tel PLA peut s'exprimer par la formule

$$S = (e + s) \cdot m \cdot k$$

avec e = nombre d'entrées

s = nombre de sorties

m = nombre de monômes

k est un paramètre dépendant des règles de dessin.

Les inconvénients des PLA sont donc :

- Le faible remplissage de la seconde matrice, en général, donc une surface inoccupée très importante,

- La distribution des entrées et des sorties très peu pratique, surtout pour de gros PLA, donc la connexion avec d'autres blocs du circuit très malaisée,

- La surface du PLA qui s'accroît proportionnellement au nombre d'entrées, de sorties, de monômes.

Par contre, les PLA ont aussi des avantages :

- La conception, la génération et le dessin sont très facilités par la possibilité d'une automatisation de ces tâches,

- La personnalisation du PLA est très simple si elle ne modifie pas le nombre de monômes. En effet, une reprogrammation de PLA ne nécessite que la modification d'un seul masque,

- La compréhension et le décodage, enfin, d'un PLA est très simple pour un concepteur.

L'utilisation des PLA étant devenue très courante dans la conception et la réalisation de circuits intégrés, il a donc fallu trouver des méthodes permettant de minimiser la surface occupée par les PLA. Pour celà, différentes approches ont été effectuées, que nous allons voir dans le paragraphe suivant.

2.2. Les différentes méthodes de minimisation topologique des PLA.

2.2.1. Optimisation triangulaire.

L'optimisation triangulaire peut être réalisée sur un PLA dont les entrées et les sorties sont perpendiculaires entre elles. Nous avons vu que la seconde matrice d'un PLA est en général très creuse. Le principe de la méthode sera alors d'ordonner les monômes, donc les sorties, de telle sorte que les emplacements inutilisés soient regroupés dans un coin de la matrice. Cet espace vide pourra alors être supprimé, ou employé à un autre usage (fig. 2.1).

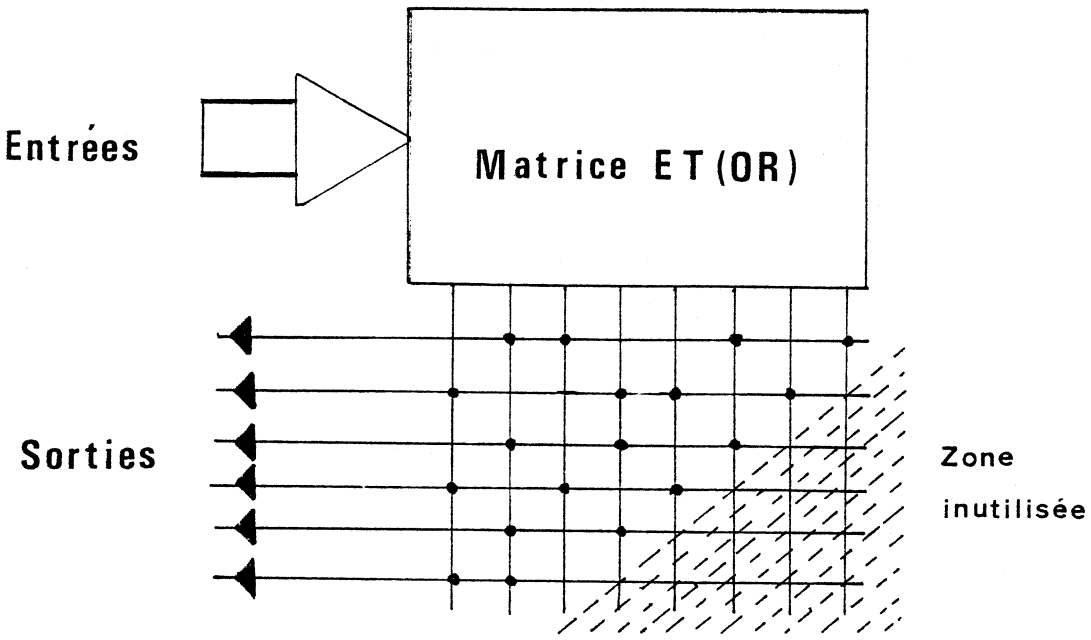


Figure 2.1 Principe de l'optimisation triangulaire

Ce type de minimisation admet trois variantes :

- L'arrangement des monômes. Les monômes sont ordonnés de telle façon que ceux qui n'ont pas d'interconnexion avec les sorties soient regroupés du côté de ces sorties, de telle sorte qu'un coin de la matrice soit libéré.

- L'arrangement des sorties. Cette variante ressemble à la méthode précédente, mis à part le fait qu'au lieu d'ordonner les monômes, on ordonne les sorties. Cette variante est en général associée à la précédente.

- L'arrangement par poids. Chaque point de PLA se voit associé un indice correspondant au nombre d'interconnexions du

monôme et de la sortie correspondante. Chaque monôme reçoit ainsi un poids, correspondant à la somme de tous les indices des points qui lui sont rattachés. Les monômes sont alors rangés de manière croissante, selon leur poids.

Enfin, au lieu d'affecter un poids aux monômes, on peut faire la même opération avec les sorties.

Ces trois méthodes peuvent être combinées, et il a été prouvé que les combinaisons diverses de ces trois méthodes donnaient des résultats différents (fig. 2.2).

Cependant, la place libérée dépend entièrement du contenu de la seconde matrice. De plus, les impératifs de conception influent sur le choix de la méthode à employer :

- Si l'ordre des sorties est imposé, par les blocs avoisinants par exemple, on ne pourra agir que sur l'ordonnancement des monômes.

- Par contre, si l'ordre des sorties n'est pas imposé, on pourra les ordonner elles aussi, et la place ainsi libérée sera plus importante.

L'exemple de la figure 2.2 montre que l'optimisation possible obtenue avec cette méthode de triangulation peut varier de façon notable selon la variante employée. Mais il faut garder à l'esprit que la meilleure optimisation en terme de place libérée n'est pas forcément la meilleure pour le concepteur. En effet,

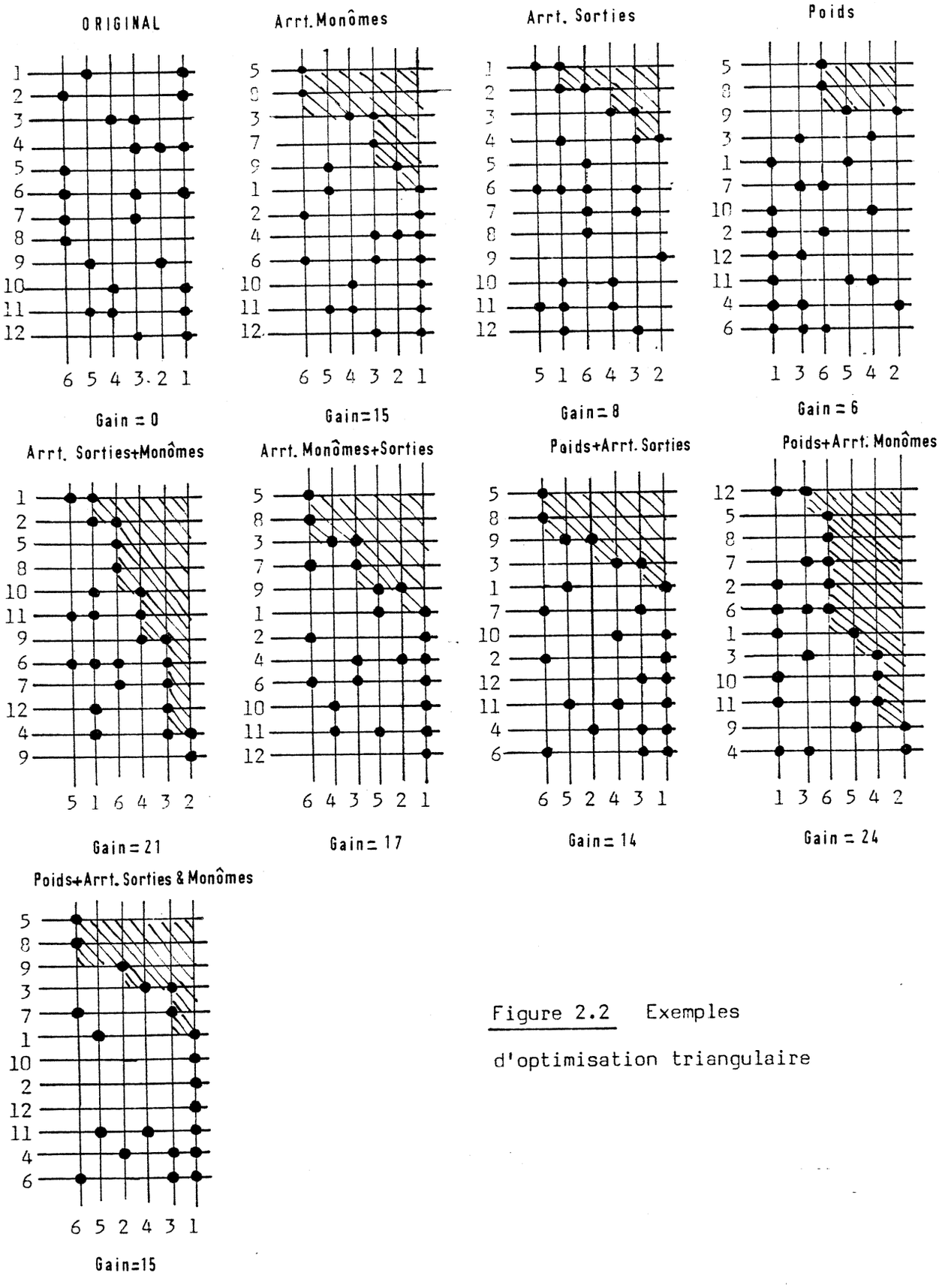


Figure 2.2 Exemples d'optimisation triangulaire

l'utilisation ultérieure de la surface ainsi libérée est un critère de choix primordial : si cette place libérée doit être utilisée pour faire des interconnexions entre blocs, par exemple, une bande étroite serait préférable; si l'on veut implanter un peu de logique dans la place libérée, une zone plus conséquente, comme un triangle, est alors préférable. Ceci explique que d'autres méthodes doivent être envisagées.

2.2.2. Optimisation par linéarisation de la seconde matrice.

Dans le cas de PLA ayant un grand nombre de sorties, il sera nécessaire de générer les sorties le plus près possible de leur utilisation ultérieure, afin de diminuer le nombre et la longueur des interconnexions. Ceci est visible dans un PLA de décodage des instructions d'un micro-processeur, par exemple, où les entrées du PLA sont les codes opérations, et les sorties les commandes de la partie opérative. La forme du PLA est alors une forme allongée (fig. 2.3).

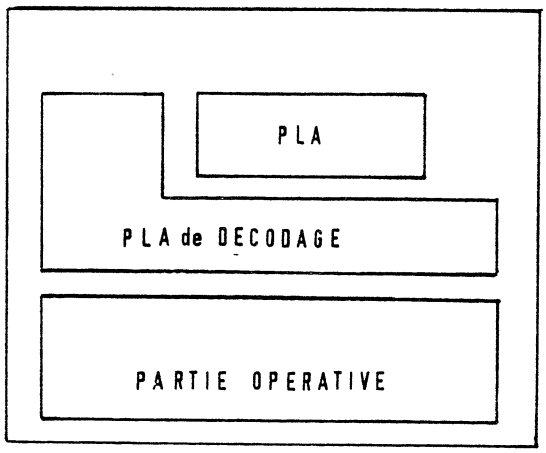


Figure 2.3

Emplacement des PLA dans le Z80 .

Le résultat de l'optimisation "linéaire" est donc un PLA de forme allongée. La contrepartie de cette méthode d'optimisation est que le gain de largeur est compensé par la perte en longueur. Le gain effectif en surface dépend alors étroitement du nombre de monômes communs à plusieurs sorties (fig. 2.4).

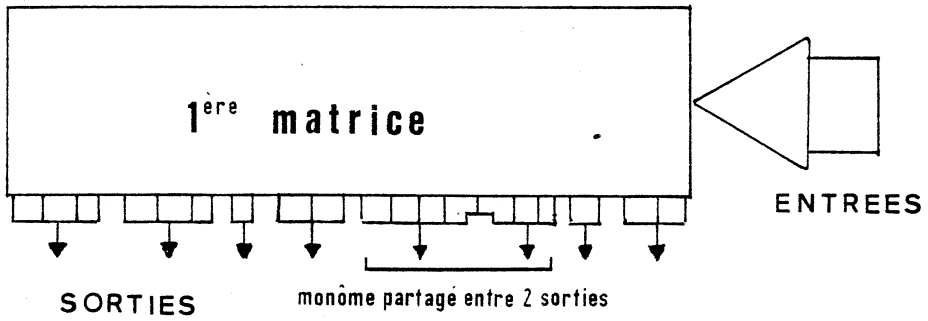


Figure 2.4.

PLA linéaire, avec monômes communs à plusieurs sorties.

Ce type d'optimisation présente de sérieux avantages : la génération automatique de PLA est très simple à réaliser, ainsi que le contrôle du résultat par le concepteur. En effet, ce dernier peut pratiquement lire les équations de sortie du PLA dans les dessins des masques de fabrication.

Par contre, cette méthode comporte aussi un inconvénient de taille : la reprogrammation d'un tel PLA nécessite la reprise de tous les masques de fabrication, car cette reprogrammation entraîne automatiquement une variation de la longueur du PLA.

2.2.3. Optimisation en "lignes brisées".

Cette méthode est dérivée de la méthode précédente, mais permet d'atteindre un degré d'optimisation plus poussé, dans le cas où le nombre de monômes communs à plusieurs sorties est important. Le principe est le suivant : sans répéter les monômes communs, on étale la seconde matrice sur plusieurs niveaux (au lieu d'un seul pour l'optimisation linéaire).

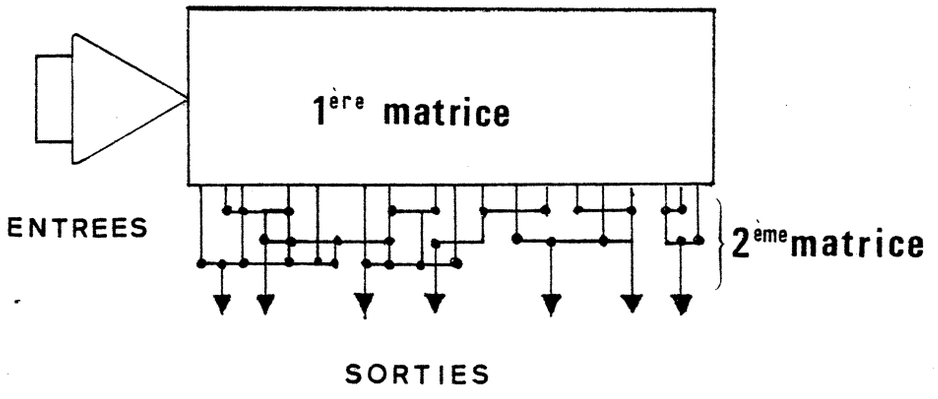


Figure 2.5 PLA en lignes brisées

Ce type d'optimisation est nettement plus complexe que les autres: en effet, la détermination du niveau de chaque sortie, ainsi que de la position de chaque monôme est très délicate. On doit alors tenir compte de l'ordre des sorties, de leur compatibilité, de leur distance, afin de faciliter les interconnexions avec d'autres blocs. De plus, le gain en surface est plus aléatoire, car il dépendra entièrement des fonctions à générer.

A partir de cette idée de base, deux idées ont été retenues pour améliorer l'optimisation.

2.2.3.1. Optimisation en lignes brisées à plusieurs niveaux.

On part du résultat obtenu par optimisation linéaire, puis on essaye de regrouper les monômes en fonction du nombre de sorties séparant leur sortie respective. Si les deux sorties sont voisines, on affecte une distance de 0 à ces deux monômes, puis on continue pour une distance de 1, puis 2... jusqu'à n.

Les monômes de distance 0 sont alors simplifiés, et le monôme restant, s'il attaque les deux sorties restantes sur le même niveau permet d'avoir alors un gain en surface, dépendant des règles de dessin du PLA. Cette procédure est répétée pour le niveau 1, 2... n, jusqu'à ce que l'on ait obtenu un gain en surface suffisant. Le PLA présentera alors un ensemble de sorties et une seconde matrice illustrés par la figure 2.6.

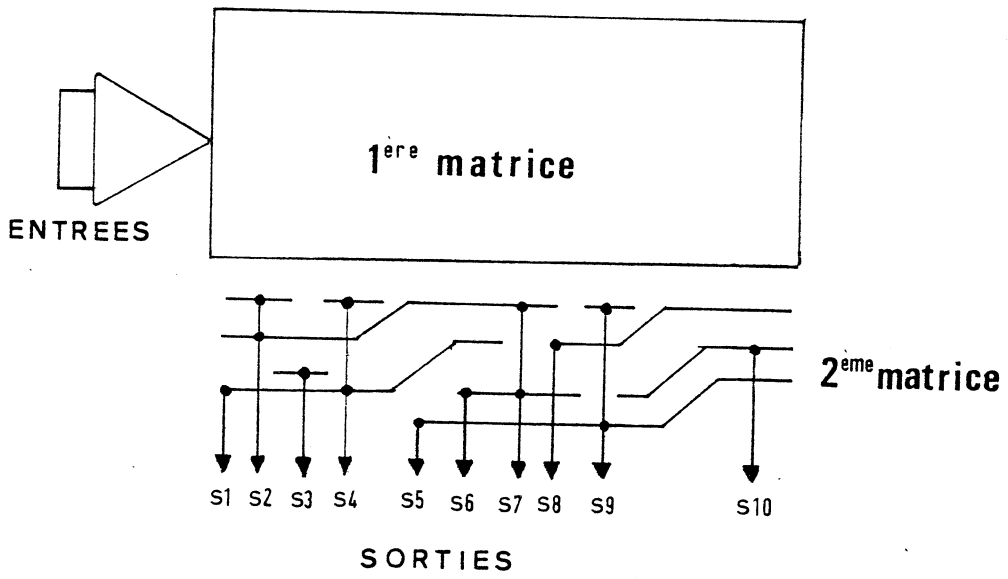


Figure 2.6 PLA en lignes brisées à plusieurs niveaux

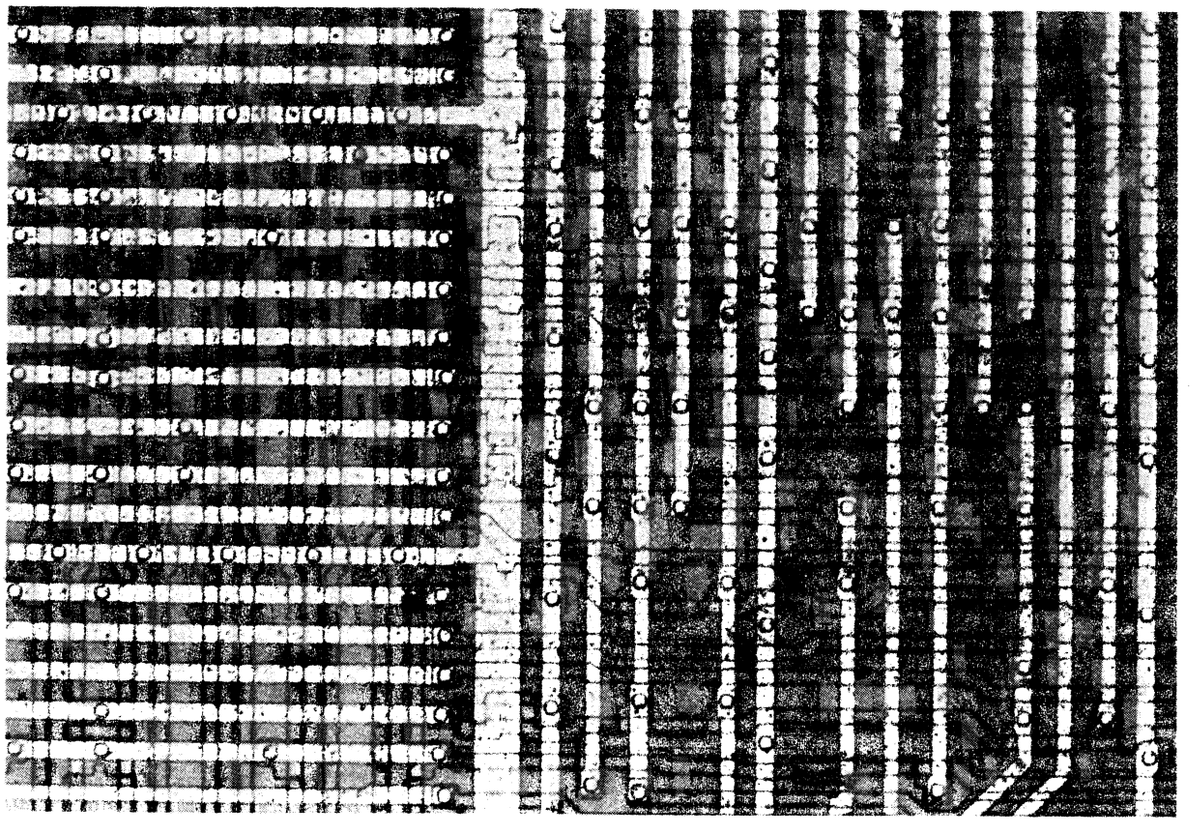


Fig. 2.5 bis. PLA en lignes brisées à plusieurs niveaux du Z80

Un exemple de ce type d'optimisation nous est donné par la photo ci-dessus. Ce PLA est celui de génération des commandes du Z80. Il reconnaît environ 100 monômes et génère 45 sorties dans une matrice OU optimisée par brisure de lignes sur 14 niveaux différents.

Un autre PLA de génération des commandes du Z80 est optimisé selon une autre méthode, dite optimisation à partir de la forme "classique".

2.2.3.2. Optimisation à partir de la forme "classique"

Cette méthode part de la forme classique du PLA, mais dans ce cas, on place les sorties perpendiculairement aux entrées. Pour ce faire, les monômes sont arrangés de telle sorte que les sorties n'ayant aucun monôme commun soient placées sur le même niveau (fig. 2.7).

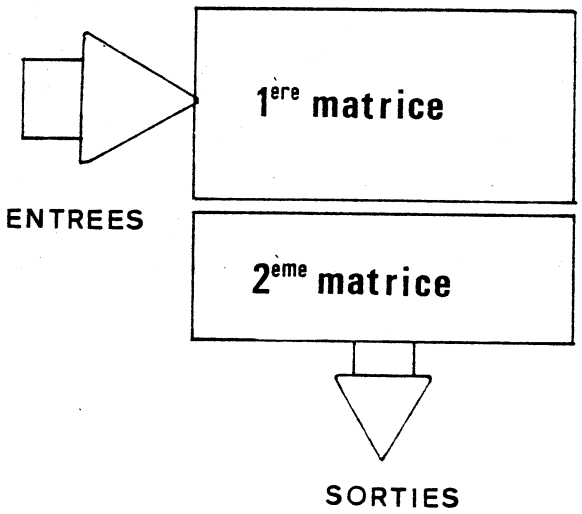


Figure 2.7
PLA en lignes brisées de forme "classique"

Cette méthode, très simple à réaliser, est utilisable aussi bien sur la seconde matrice que sur la première. Par contre, elle entraîne un faible gain de place.

2.2.4. Influence de la technologie et de la fonction des PLA dans l'optimisation topologique.

Les méthodes d'optimisation topologique des PLA sont nombreuses, et bien que le gain en surface varie avec la méthode employée, il arrive que des méthodes différentes aboutissent à des degrés d'optimisation semblables. Mais la technologie de

réalisation des PLA va, elle aussi, influer sur l'optimisation en matière de gain de place. En effet, selon le type de technologie choisi, la place occupée par le PLA sera plus ou moins importante. Le type de technologie employé influera également sur la forme du PLA. Enfin, n'importe quel type d'optimisation ne sera pas compatible avec n'importe quelle technologie. Le choix final du concepteur sera alors fonction des contraintes de départ, et sera souvent un compromis entre divers impératifs. La fonction remplie par le PLA sera, elle auusi, déterminante pour ce qui est de la possibilité d'optimiser les PLA. Ainsi, par exemple, pour un PLA qui comporte peu d'entrées et peu de sorties, même si sa taille est importante, la possibilité de l'optimiser en réorganisant ses sorties est importante, et le gain de surface qui en découle peut être élevé. Par contre, un PLA de validation de micro-commandes, par exemple, qui comporte un grand nombre d'entrées et sorties imposées, laissera une plus faible marge de liberté, et une possiblité d'optimisation beaucoup plus faible qu'avec l'exemple précédent.

Parmi toutes les méthodes d'optimisation étudiées ci-dessus, il en est une qui a été réalisée et qui fonctionne de manière satisfaisante; cette méthode d'optimisation topologique a été réalisée au Laboratoire d'Architecture des Ordinateurs de l'IMAG et se nomme PAOLA. Le paragraphe suivant va nous faire découvrir, de manière succinte, les principes, la réalisation et les résultats de cette méthode.

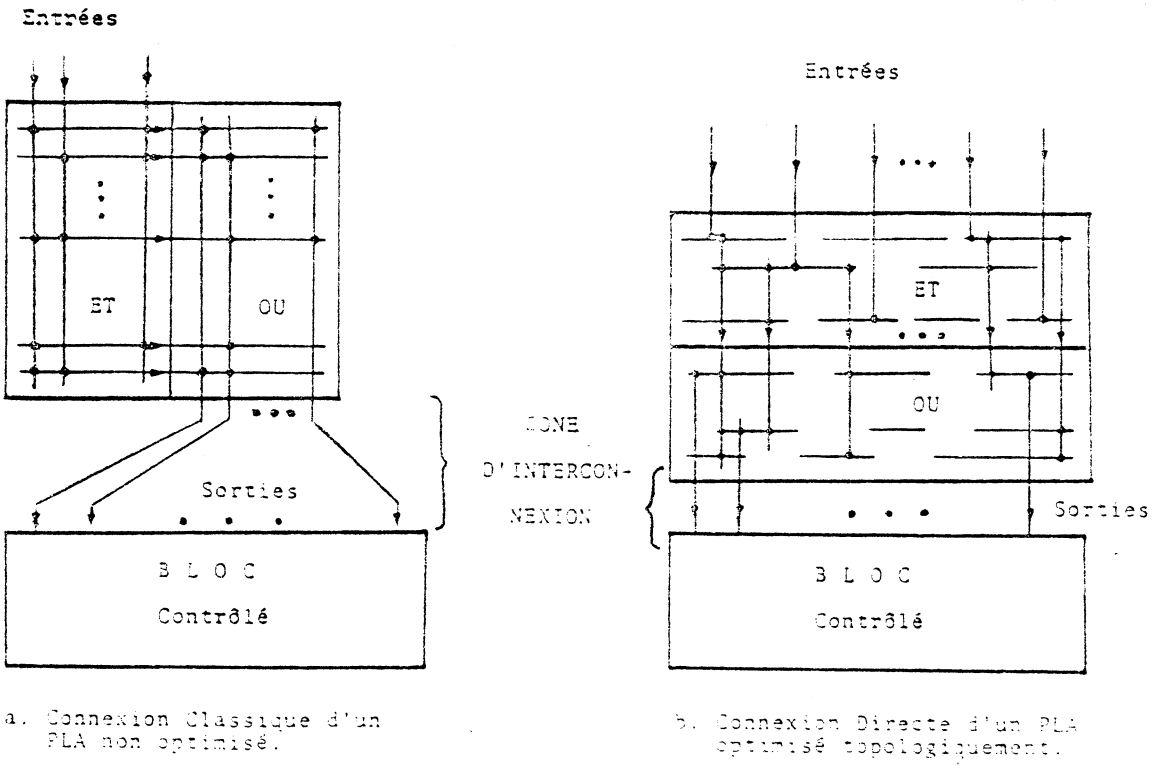
2.3. PAOLA, un optimisateur topologique de PLA <CHU-82>.

2.3.1. Principe de PAOLA.

PAOLA (PLA Automatic Optimization and LAyout) est un système CAO destiné à l'optimisation topologique puis au dessin automatique des masques de PLA complexes.

Dans les PLA classiques, la perte de place en terme de surface utilisée est dûe à deux choses :

- le faible remplissage de la matrice OU,
- la mauvaise répartition des lignes d'entrées-sorties, entraînant l'existence d'une nappe d'interconnexions occupant un espace important (fig. 2.8 a).



a. Connexion Classique d'un PLA non optimisé.

b. Connexion Directe d'un PLA optimisé topologiquement.

Figure 2.8 Connexion d'un PLA et un bloc

Nous avons vu avec les méthodes précédentes qu'il est possible de réduire ces inconvénients, qui deviennent critiques lorsque le PLA a une taille importante.

- Premièrement en réduisant la surface du PLA par brisure des lignes d'entrées-sorties et en plaçant plusieurs de ces lignes dans chaque colonne.

- Deuxièmement en plaçant les bornes de connexion des entrées-sorties sur toute la longueur du PLA,au lieu de les placer sur ses extrêmités, afin de réduire la surface d'interconnexion (fig. 2.8 b).

2.3.2. Description du système d'optimisation.

Le processus d'optimisation topologique permet d'obtenir une nouvelle structure des matrices ET (OU) en permettant aussi un assouplissement du tracé des connexions entre les segments d'entrée (sortie) et leurs bornes.

Ce processus se décompose en 3 étapes :

- Réordonnancement des monômes,
- Duplication des monômes,
- Compactage des matrices.

La première et la dernière étape peuvent être appliquées indifféremment aux matrices ET et OU d'un PLA, mais nous ne verrons ici que le cas de la matrice OU.

2.3.2.1. Réordonnancement des monômes.

Cette étape consiste à ordonner les monômes en minimisant la distance entre les monômes connectés à une même sortie <PER-80>. Ceci est fait grâce à une méthode heuristique, réalisant l'ordonnancement des monômes par rapport aux coordonnées du barycentre des sorties qu'ils excitent. Le résultat est une matrice ayant une structure proche des matrices diagonales (fig. 2.9 b).

2.3.2.2. Duplication des monômes.

La connexion d'un PLA aux blocs qui l'environnent peut nécessiter l'allongement du PLA à la taille des blocs. Cet allongement est obtenu en dédoublant (dupliquant) un monôme en deux, chacun des monômes obtenus ayant pour partie OU la découpe de la partie OU initiale à son barycentre. A chacun sera alors associée la même partie ET du monôme père. Ces deux monômes fils sont ensuite replacés en fonction de leurs nouveaux barycentres (fig. 2.9 b).

Plusieurs critères de choix des monômes à dupliquer peuvent être choisis par le concepteur, en tenant compte des caractéristiques propres à chaque monôme.

2.3.2.3. Compactage des matrices.

Cette étape consiste à placer plusieurs segments de sortie compatibles dans une même colonne, appelée "niveau" (fig. 2.9.d). Un algorithme heuristique est employé pour ne pas énumérer toutes les combinaisons possibles des segments compatibles.

	ET				OU		
(1)	0	0	0	4	1	0	0
(2)	0	2	0	0	0	2	0
(3)	1	0	3	0	1	2	0
(4)	0	2	0	4	1	0	0
(5)	1	2	0	4	0	2	0
(6)	1	2	0	0	0	0	3
(7)	1	0	3	4	0	2	3
(8)	1	0	0	4	1	0	3
(9)	0	2	3	0	1	2	3
	E1	E2	E3	E4	S1	S2	S3

FIG. **2.9a**
Représentation Entière

	ET				OU		
(1)	0	0	0	4	1	0	0
(4)	0	2	0	4	1	0	0
(3)	1	0	3	0	1	2	0
(2)	0	2	0	0	0	2	0
(5)	1	2	0	4	0	2	0
(8)	1	0	0	4	1	0	3
(9)	0	2	3	0	1	2	3
(7)	1	0	3	4	0	2	3
(6)	1	2	0	0	0	0	3
	E1	E2	E3	E4	S1	S2	S3

FIG. **2.9b**
PLA après réordonnancement des monômes.

	ET				OU		
(1)	0	0	0	4	1	0	0
(4)	0	2	0	4	1	0	0
(8)	1	0	0	4	1	0	0
(3)	1	0	3	0	1	2	0
(9)	0	2	3	0	1	2	0
(2)	0	2	0	0	0	2	0
(5)	1	2	0	4	0	2	0
(7)	1	0	3	4	0	2	3
(8)	1	0	0	4	0	0	3
(9)	0	2	3	0	0	0	3
(6)	1	2	0	0	0	0	3
	E1	E2	E3	E4	S1	S2	S3

FIG. **2.9c**
PLA après duplication des monômes 8 et 9.

	ET				OU	
(1)	0	0	0	4	1	0
(4)	0	2	0	4	1	0
(8)	1	0	0	4	1	0
(3)	1	0	3	0	1	2
(9)	0	2	3	0	1	2
(2)	0	2	0	0	0	2
(5)	1	2	0	4	0	2
(7)	1	0	3	4	3	2
(8)	1	0	0	4	3	0
(9)	0	2	3	0	3	0
(6)	1	2	0	0	3	0
	E1	E2	E3	E4	M1	M2

FIG. **2.9d**
PLA après compactage de la matrice OU.

Figure 2.9 Exemple d'application de PAOLA.

La dernière phase de PAOLA consiste alors à effectuer le dessin automatique des masques, l'organisation physique des matrices, le tracé des connexions internes, la génération du

dessin du PLA et l'évaluation des performances des PLA ainsi créés (fig. 2.10).

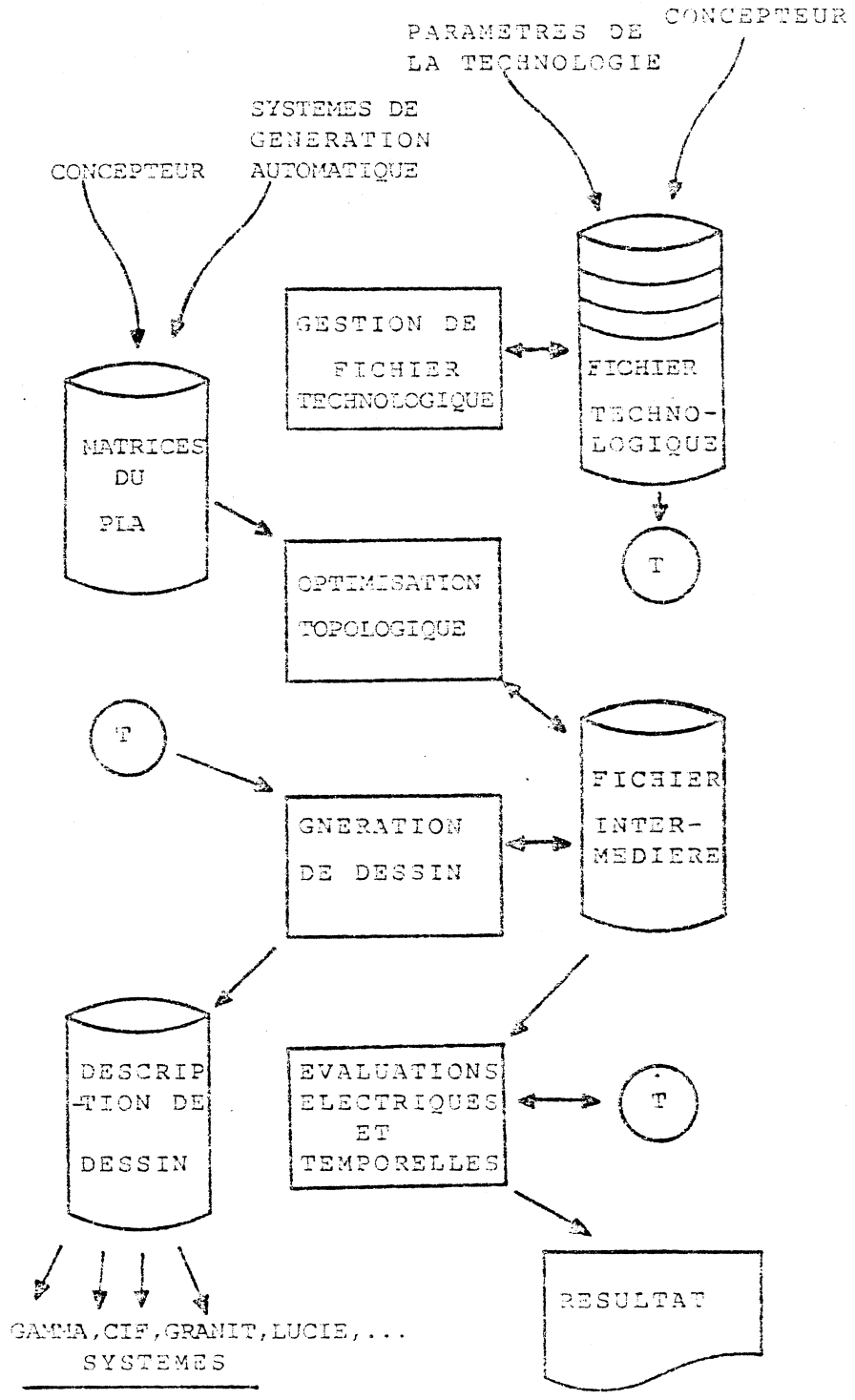


Figure 2.10 SYTEME PAOLA

Une remarque importante à faire est que le système est paramètrisé. Un fichier technologique est utilisé pour stocker toutes les informations (paramètres technologiques) indépendantes du PLA, mais fonction de la technologie utilisée. Ce fichier technologique est alors créé à l'initiative du concepteur et lui permet de faire fonctionner le système PAOLA selon ses propres contraintes technologiques.

2.3.3. Performances de PAOLA.

La figure 2.11 nous présente quelques résultats du système PAOLA.

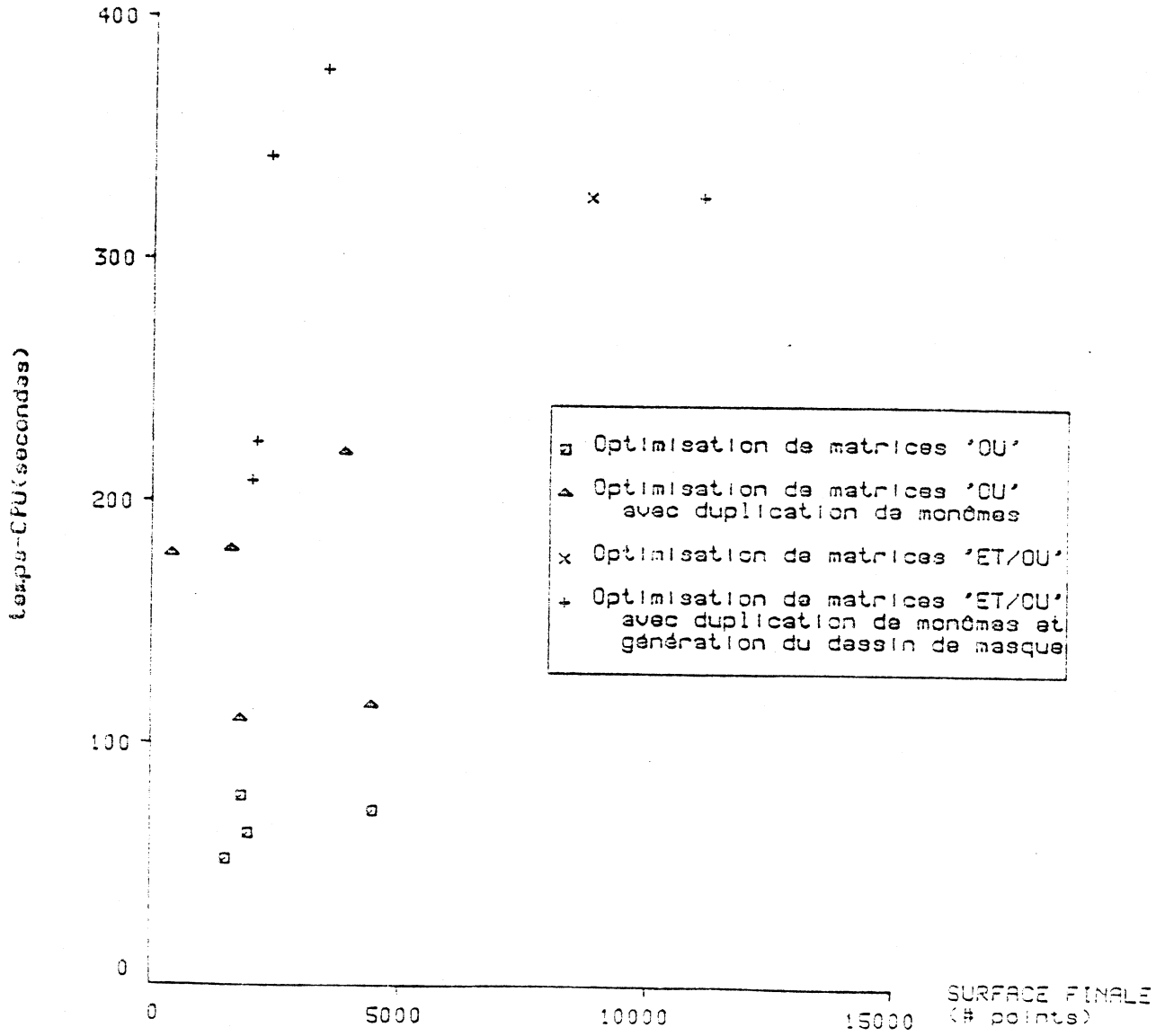


Figure 2.11 Temps CPU vs. surface finale de quelques exemples d'optimisation.

Les performances du système dépendent étroitement du degré de remplissage initial du PLA. Les PLA de séquencement, par exemple, ayant des taux de remplissage importants donnent les résultats les plus défavorables. Les PLA de décodage par contre,

donnent les meilleurs résultats car leur taux de remplissage est faible.

Les performances d'ensemble du système PAOLA, testées sur des exemples industriels ont donné des taux d'optimisation atteignant 20% à 50% (fig. 2.12).

		PLA1 (SC/MP)	PLA2 (MC2)	PLA3 (Z-80)	PLA4	PLA5
FORME CLASSIQUE	Nb. monômes	55	108	75	131	147
	Nb. sorties	46	40	45	57	38
	surface	2990	4320	3375	7467	5586
	Nb. transistors	314	139	303	520	1383
OPTIMISATION SANS DUPLICAT. DE MONOMES	Nb. monômes	65	108	75	131	147
	Nb. niveaux	28	14	26	34	30
	surface	1820	1512	1590	4454	4410
	temps-CPU (secondes)	39	25	31	36	130
OPTIMISATION AVEC DUPLIC. DE MONOMES	Nb. monômes	78	131	88	160	205
	Nb. niveaux	23	3	18	24	27
	surface	1794	393	1584	3840	5535
	temps-CPU (secondes)	55	89	90	110	250

Figure 2.12. Résultats de quelques optimisations topologiques des matrices OU.

Les temps d'exécution, dépendant de la taille du PLA (fig. 2.12), sont donc dans l'ensemble inférieurs à 5 minutes de temps CPU.

Cependant, l'optimisation topologique peut être rendue plus performante encore en la faisant précéder d'une phase d'optimisation logique, traitant des PLA complexes, afin de réduire le nombre de transistors principalement de la matrice OU. C'est cette phase d'optimisation logique, que j'ai étudiée et réalisée, qui fera l'objet des chapitres suivants.

CHAPITRE III

L'OPTIMISATION LOGIQUE DES PLA

3.1. Généralités, historique.

L'électronique a connu un développement très rapide ces vingt dernières années, mais ce développement s'est fait selon deux directions distinctes : l'électronique analogique et l'électronique digitale. L'électronique analogique a pu se développer par des méthodes de description et de calcul très proches de celles de la physique classique, telles que le calcul matriciel, le calcul différentiel, etc... L'électronique digitale, par contre, a dû avoir recours à des méthodes de calcul totalement nouvelles, basées sur l'algèbre de BOOLE, ou encore la théorie des corps de GALOIS, afin d'aboutir à des notions modernes de théorie des automates, de programmation structurée, etc...

Cette évolution s'est faite de manière très rapide et de façon un peu anarchique, ce qui a donné lieu à certaines lacunes dans le formalisme et la description des méthodes de calcul.

L'influence de ces lacunes ne se fait que peu, ou pas du tout, ressentir lorsqu'on réalise une machine digitale en peu d'exemplaires, ou avec des composants du commerce, car la mise en oeuvre de méthodes systématiques se révèle souvent trop coûteuse en regard des méthodes heuristiques. Ceci explique que les premières "machines à simplifier" développées dans les années 60

par PARKHOMENKO <PAR-60>, FLORINE <FLO-64> ou MANGE <MAN-67> sont restées peu connues des ingénieurs concepteurs. Ces machines n'avaient d'ailleurs que peu de relations avec les travaux des mathématiciens sur les systèmes logiques. Leur principe était de construire et de comparer toutes les solutions possibles d'un problème donné. Mais, comme les méthodes utilisées pour celà étaient des méthodes systématiques, leur implémentation était très coûteuse.

Aussi, de nouvelles méthodes furent développées à partir des travaux de QUINE <QUI-52>, <QUI-55> et MC CLUSKEY <MCL-56>. Ces méthodes étaient une amélioration considérable par rapport au passé, car elles permettaient de générer les impliquants premiers de fonctions booléennes, puis de créer les couvertures irrédondantes de ces fonctions, afin d'optimiser selon des critères de coût des circuits tels que les décodeurs, les PLA ou les Unités Arithmétiques et Logiques. De nombreuses méthodes d'optimisation logique ont été inspirées par les travaux de QUINE - MC CLUSKEY dont les plus connues sont <MOR-67>, <DIE-69>, <MOR-70>, <SLA-70>, <JJM-80>.

Enfin, des méthodes plus simples, ou plus efficaces dans certains cas furent également élaborées d'après ces théories: <BOW-70>, <HON-74>, <MUR-76>, <ARE-78>, <KAM-79>, <TEE-82>. L'inconvénient de ces méthodes, dont le but est de trouver la solution minimale d'un système logique est qu'elles sont adaptées à des problèmes de petite taille seulement. En effet, avec un algorithme de recherche d'une solution optimale effectué sur un calculateur, le temps de calcul et l'espace mémoire nécessaires

s'acroissent de façon quasi-exponentielle en fonction de la taille du système logique à optimiser. De ce fait, toutes ces nouvelles méthodes furent peu ou ne furent pas employées par les concepteurs de circuits car leur mise en oeuvre était lourde et malaisée.

Ce problème fut en partie résolu par l'introduction de méthodes "sous-optimales" d'optimisation de systèmes logiques. L'idée de telles méthodes est de se borner à chercher non pas LA solution optimale mais une solution, appelée "solution sous-optimale", qui donne un résultat meilleur qu'à l'origine, sans donner la meilleure solution. L'avantage de ces dernières méthodes est d'utiliser des temps de calcul et des tailles de mémoire sur ordinateur beaucoup plus petits qu'avec les solutions optimales. Les programmes d'optimisation logique développés avec ce concept par <ROT-78>, <ROT-80>, <YAC-80> ont donné d'assez bons résultats, et il semble que ce soit dans cette voie que se développent actuellement les systèmes de minimisation logique.

Dans le paragraphe qui suit, nous allons rapidement examiner quelques méthodes classiques d'optimisation logique.

3.2. Les méthodes "classiques" d'optimisation logique.

Dans toutes les méthodes développées ci-dessous, nous verrons apparaître les mêmes notions et notations. Aussi, nous allons faire un bref rappel de ces notions.

3.2.1. Rappel des notions principales utilisées en logique combinatoire.

3.2.1.1. Rappel des formules de l'Algèbre de Boole.

Nous donnerons sans démonstration un recueil de formules (axiomes et propriétés) qui seront utilisées par la suite. <FLE-67>

Définition.

On appelle Algèbre de Boole un ensemble A, muni d'une opération $X + Y$ (somme de deux éléments de A), d'une opération $X \cdot Y$ (produit de deux éléments de A), d'une opération $\overline{X}$ (complémentation d'un élément de A), d'un élément 0, d'un élément 1, tels que les axiomes suivants soient satisfaits :

1. $X + Y = Y + X$	$XY = YX$
2. $X + (Y + Z) = (X + Y) + Z$	$X(YZ) = (XY)Z$
3. $X + (YZ) = (X + Y)(X + Z)$	$X(Y + Z) = (XY) + (XZ)$
4. $X + 0 = X$	$X1 = X$
5. $X + \overline{X} = 1$	$X\overline{X} = 0$

Théorème

Les formules suivantes sont des conséquences des axiomes (1) à (5).

6. $X + X = X$	$X.X = X$
7. $X + 1 = 1$	$X.0 = 0$

8. $X + (XY) = X$ $\qquad$ $X.(X + Y) = X$

9. $X + Y = X + (\overline{X}Y)$ $\qquad$ $X.Y = X.(\overline{X} + Y)$

10. $X = (XY) + (X\overline{Y})$ $\qquad$ $X = (X + Y).(X + \overline{Y})$

11. $X + Y = Y \iff XY = X$

12. $X + Y = 0 \iff (X=0 \text{ et } Y=0)$ $\qquad$ $XY = 1 \iff (X=1 \text{ et } Y=1)$

13. $Y = \overline{X} \iff (X+Y= 1 \text{ et } XY=0)$

14. $\overline{\overline{X}} = X$

15. $X = Y \iff \overline{X} = \overline{Y}$

16. $\overline{X + Y} = \overline{X}\,\overline{Y}$ $\qquad$ $\overline{XY} = \overline{X} + \overline{Y}$

Définition

Dans toute algèbre de Boole A, la relation $X < Y$ entre deux éléments de A est définie par :

17. $X < Y \iff X + Y = Y$. $\qquad$ Par (11) il en découle :

18. $X < Y \iff X.Y = X$

Théorème

A partir de ce qui précède, on pourra démontrer les formules suivantes :

19. $X \leqslant X$

20. $(X \leqslant Y \text{ et } Y \leqslant X) \iff X = Y$

21. $(X \leqslant Y \text{ et } Y \leqslant Z) \iff X \leqslant Z$

22. $0 \leqslant Y \leqslant 1$

23. $X \leqslant Y \iff \overline{X} + Y = 1$

24. $X \leqslant Y \iff X.\overline{Y} = 0$

25. $X \leqslant Y \iff \overline{Y} \leqslant \overline{X}$

26. $(X \leqslant Y \text{ et } X' \leqslant Y') \iff X + X' \leqslant Y + Y'$

27. $(X \leqslant Y \text{ et } X' \leqslant Y') \iff X.X' \leqslant Y.Y'$

28. $(X \leqslant Y \text{ et } X' \leqslant Y) \iff X + X' \leqslant Y$

29. $(X \leqslant Y \text{ et } X \leqslant Y') \iff X \leqslant Y.Y'$

30. $X \leqslant X + Y$

31. $XY \leqslant X$

Enfin, de toutes les formules énumérées ci-dessus, on pourra déduire un autre théorème, avec (20) et (23) par exemple :

Théorème

Si X et Y sont deux éléments d'une algèbre de Boole, alors :

32. $X = Y \iff (\overline{X} + Y)(\overline{Y} + X) = 1$

33. $X = Y \iff (X\overline{Y} + Y\overline{X}) = 0$

Pour tout ce qui suivra, ces formules seront supposées familières au lecteur, et seront en général utilisées sans être mentionnées.

3.2.1.2. Notion d'impliquant.

La notion d'impliquant est liée à une réalisation "à 2 niveaux", typique des circuits logiques actuels. En effet, on peut décrire le fonctionnement d'un système par la somme des configurations dans lesquelles le système doit avoir une action donnée. Cette somme booléenne peut s'écrire par exemple :

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

On peut affirmer que le fait que $a\bar{b} = 1$ implique que $F = 1$ d'où le terme "impliquant".

Définition

Soit $F(a1,a2,\ldots a_n)$ une fonction booléenne et $A(a1,a2,\ldots a_n)$ un produit de variables, alors on dira que A est impliquant de F si :

$$A\ F = A$$

ce qui peut encore s'écrire

$$A \leqslant F \quad \text{ou encore} \quad A \Longrightarrow F$$

3.2.1.3. Notion d'impliquant premier.

Soient $F(a1,a2,\ldots a_n)$ une fonction booléenne et $A(a1,a2,\ldots a_n)$ un produit de variables, on dira que A est impliquant premier de F s'il n'existe aucun autre impliquant $B(a1,a2,\ldots a_n)$ de F tel que :

$$A \leqslant B$$

Ceci revient à dire qu'un impliquant premier est un produit composé d'un minimum de variables, "couvrant" le plus grand nombre d'états possibles.

Exemple : si $a\bar{c}$ est premier alors

$a\bar{b}\bar{c}$ n'est pas premier.

3.2.1.4. Somme de produits irrédondante.

La notion de forme "minimale" d'une fonction à 2 niveaux implique la notion de forme "irrédondante".

Définition

Une somme de produits est dite "irrédondante" si on ne peut pas enlever un terme de la somme ni une variable dans chaque produit sans changer la valeur de la somme.

Exemple : La fonction $F = a1\ a2 + \overline{a2}\ a3$ est irrédondante.

La fonction $F = a1 + a1\ a2$ n'est pas irrédondante, car on peut enlever le terme $a1\ a2$ sans changer la valeur de F.
De même, la fonction $F = a1 + \overline{a1}\ a2$ n'est pas irrédondante, car on peut écrire $F = a1 + a2$.
Ceci nous amène le théorème suivant, démontré par <JJM-80> :

Théorème :

Toute somme irrédondante n'est composée que d'impliquants premiers.

Ces principales notions sur les systèmes logiques seront utilisées par la suite. D'autres notions seront éventuellement employées, mais seront d'abord décrites.

Nous allons maintenant examiner différentes méthodes de minimisation.

3.2.2. La méthode des tables de KARNAUGH.

Cette méthode, qui a été élaborée par KARNAUGH <KAR-53>, est en fait une application "graphique" de la méthode de simplification élaborée par QUINE <QUI-52>.

3.2.2.1. Généralités.

L'analyse d'un circuit quelconque consiste à examiner le circuit, puis d'essayer de déterminer quel est le comportement du circuit pour toutes les combinaisons possibles des entrées <MCL-65>.

Un circuit de commutation combinatoire est un circuit pour lequel les sorties ne dépendent que des entrées présentes. Un circuit de commutation séquentiel est un circuit pour lequel les sorties dépendent non seulement des entrées, mais d'états antérieurs des entrées.

L'analyse d'un circuit combinatoire consiste à écrire les fonctions algébriques des sorties (une fonction par sortie). Ce sont des fonctions des variables d'entrées, à partir desquelles la condition de chaque sortie peut être déterminée, pour chacune des combinaisons des conditions d'entrée. En extrapolant cette procédure, pour chaque combinaison possible des entrées, il sera possible de former un tableau qui liste les sorties pour chaque combinaison des entrées. Ce tableau s'appelle le Tableau des combinaisons (table des combinaisons).

Exemple.

Soient x1,x2 et x3 les variables d'entrée de ce tableau T. Soit f(x1,x2,x3) la fonction remplie par le circuit. On pourra alors décrire T :

	x1	x2	x3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Dans ce cas précis, la fonction f sera vérifiée pour les configurations 1,2,3,4 et ne sera pas vérifiée pour les autres.

On pourra écrire cette fonction ainsi:

$$f(x1,x2,x3) = \Sigma(1,2,3,4)$$

D'après ce tableau, on pourrait écrire la fonction sous une forme algébrique, qui soit une somme de produits fondamentaux, appelée "somme canonique". Cette fonction peut également être écrite sous forme de produit de sommes fondamentales; cette forme est appelée "produit canonique".

En reprenant le tableau T, on pourrait alors réécrire la fonction définie sous les formes canoniques suivantes:

-somme canonique

$$f = \overline{x1}\ \overline{x2}\ x3 + \overline{x1}\ x2\ \overline{x3} + \overline{x1}\ x2\ x3 + x1\ \overline{x2}\ \overline{x3}$$

-produit canonique

$$f = (x1+x2+x3)\ (\overline{x1}+x2+\overline{x3})\ (\overline{x1}+\overline{x2}+x3)\ (\overline{x1}+\overline{x2}+\overline{x3})$$

La forme la plus simple d'une somme de produits d'une fonction sera appelée la "forme minimale".

Par exemple, la fonction $f = \overline{x}\ y\ z + x\ y\ z + x\ y\ \overline{z}$ peut s'écrire $f = y\ z + x\ y\ \overline{z}$, ou $f = \overline{x}\ y\ z + x\ y$ ou encore

$f = y\ z + x\ y$.

Seule la troisième forme d'écriture est celle de la "somme minimale", car elle contient 4 termes, les deux autres formes en contenant 5. La méthode de base pour obtenir la somme minimale est d'appliquer le théorème (10) vu précédemment :

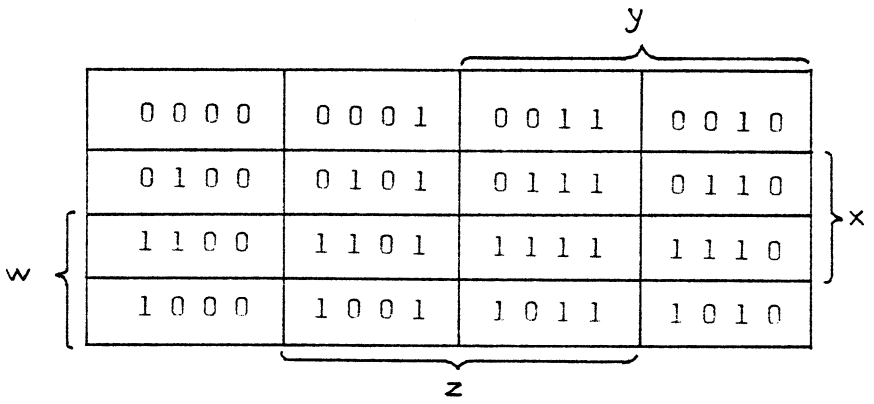
$(X\ Y + X\ \overline{Y} = X)$ autant de fois qu'il y a de termes, puis d'appliquer ensuite le théorème (8) :

$X\ Y + \overline{X}\ Z + Y\ Z = X\ Y + \overline{X}\ Z$ pour éliminer autant de termes que celà est possible.

3.2.2.2. Définition des tables de KARNAUGH <FLE-67>

Cette procédure, consistant à rapprocher 2 termes et à voir si on peut leur appliquer les théorèmes énoncés peut devenir très vite fastidieuse pour de grosses fonctions. Cette comparaison peut être simplifiée en utilisant un tableau avec n éléments.

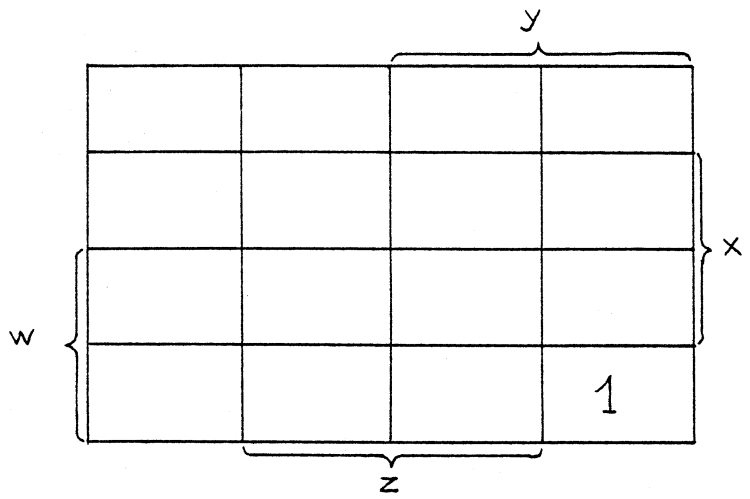
Prenons comme exemple la fonction f ayant 4 variables w,x,y,z. Chaque case de la représentation de KARNAUGH correspond à un polynôme minimal des 4 éléments w,x,y,z. La case du haut, à gauche représente w.x.y.z, celle de droite de la seconde ligne représente $\overrightarrow{w}.\overline{x}.\overline{y}.\overline{z}$ et ainsi de suite.



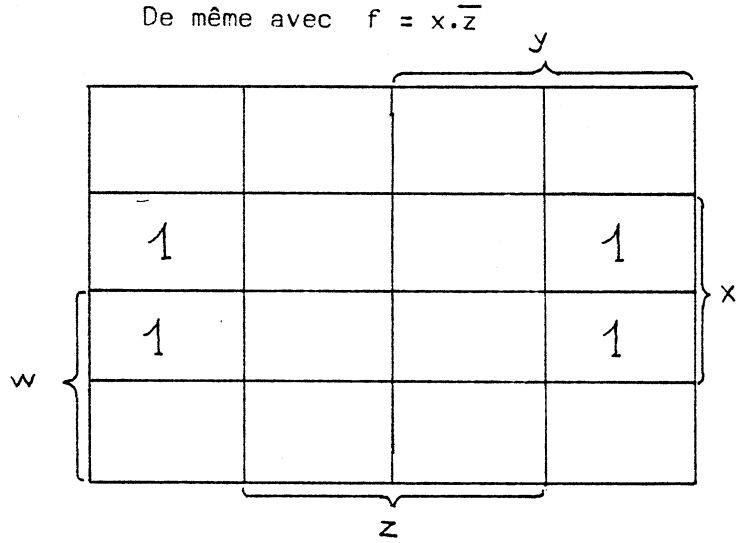
3.2.2.3. Simplification à l'aide des tables de KARNAUGH.

Pour simplifier à partir de cette représentation il faut placer le chiffre 1 dans celle des cases qui représente la fonction à simplifier. Pour chaque terme exprimé sous forme de polynôme minimal, on marquera donc une case. Les termes formés d'une intersection de 3 éléments utiliseront 2 cases, ceux formés d'une intersection de 2 éléments utiliseront 4 cases, ceux formés de un seul élément utiliseront 8 cases etc...

Si l'on considère la fonction $f = w.\overline{x}.y.\overline{z}$, nous aurons la représentation de KARNAUGH suivante :



De même avec $f = x.\overline{z}$

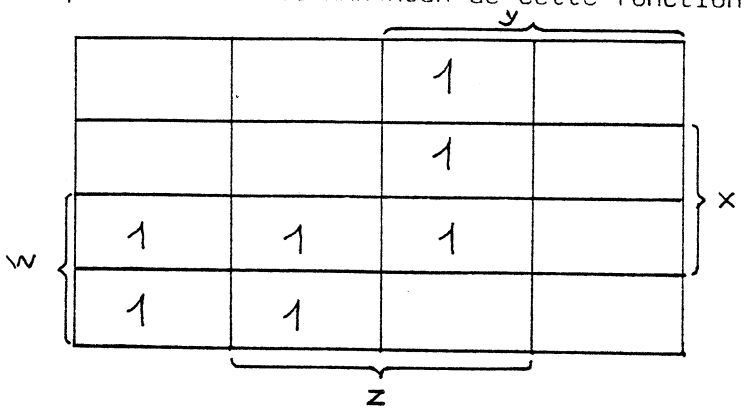


A partir de cette représentation, on peut effectuer des regroupements. Les cases marquées adjacentes seront regroupées en carrés, rectangles, lignes, colonnes ou paires, jusqu'à ce que toutes les cases marquées aient été regroupées, de toutes les manières possibles.

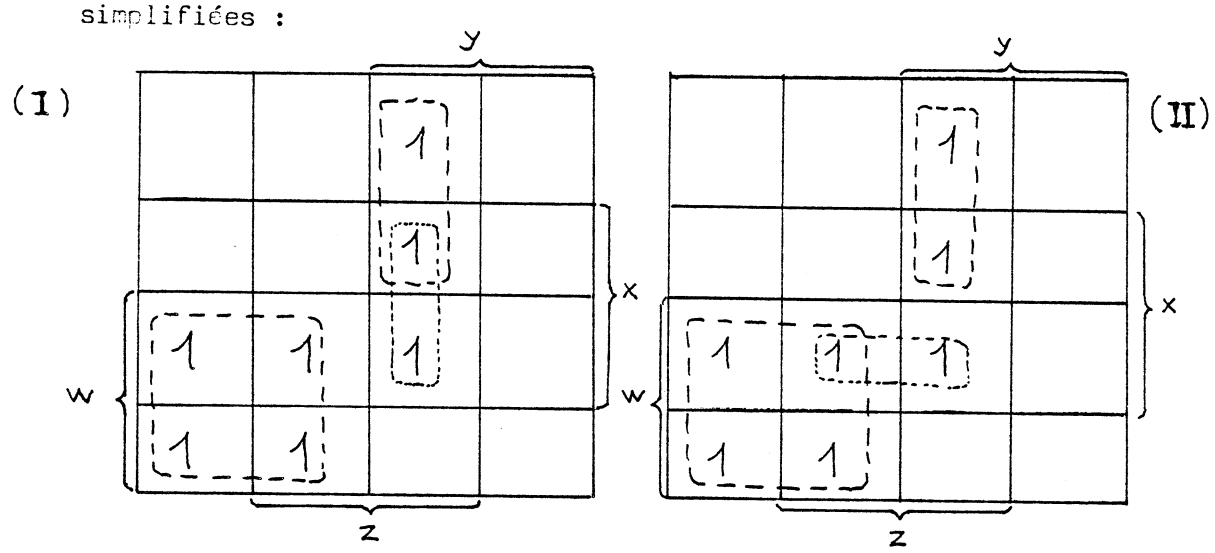
Comme exemple de simplification, nous allons considérer la fonction

$$f_0(w,x,y,z) = w\,\overline{y} + w\,x\,z + \overline{w}\,y\,z + w\,x\,\overline{y}\,z$$

La représentation de KARNAUGH de cette fonction sera :



En regroupant les cases, on aboutit aux 2 formes simplifiées :



On pourra alors écrire tout naturellement les 2 fonctions simplifiées

(I) $f_1 = w\,\overline{y} + \overline{w}\,y\,z + x\,y\,z$

(II) $f_2 = w\,\overline{y} + w\,x\,z + \overline{w}\,y\,z$

Nous constatons, en rapprochant f_0 et f_1 que le dernier terme de f_0 , qui était redondant, est éliminé.

3.2.2.4. Avantages, inconvénients de cette méthode.

L'avantage d'une telle méthode est la facilité avec laquelle on peut observer les cases adjacentes. De plus, la représentation sous forme de tableau facilite la compréhension du système et permet d'obtenir une simplification rapide et aisée.

Un autre avantage de cette méthode est qu'il est facile d'incorporer un élément libre ou une fonction qui aidera à la simplification. Si un élément ne nous intéresse pas, on peut le marquer "X" ou "Q", et l'on sera alors libre de l'utiliser ou pas, selon qu'il facilite ou non les regroupements.

Par contre, cette méthode a son revers, et présente des inconvénients notoires. Le plus important est le fait qu'au delà de 4 éléments, la représentation de la table de KARNAUGH se complique et perd ainsi beaucoup de clarté.

On a proposé d'autres méthodes pour généraliser cette représentation à des fonctions ayant jusqu'à 9 éléments. KARNAUGH, lui-même, a proposé une représentation à 3 dimensions à l'aide de 4 cartes à 4 éléments dessinées sur des plastiques transparents, en utilisant des marques de couleur.

Cependant, la gymnastique intellectuelle nécessaire pour un tel système diminue beaucoup la valeur de la suggestion.

3.2.3. Méthodes dérivées de la théorie des consensus.

La méthode précédente pour obtenir les sommes minimales suppose que les fonctions sont définies à l'origine par

des sommes canoniques, des tables de combinaisons ou par une forme équivalente, de façon à ce que chaque produit fondamental soit spécifié. Cependant, une fonction peut être spécifiée au moyen d'une expression qui est une somme de produits arbitraire et qui n'est pas une somme minimale, c'est-à-dire que cette expression peut contenir des termes qui ne sont pas des impliquants premiers.

Nous matérialiserons ceci à l'aide de l'exemple d'une réaction chimique, nécessitant 4 composants primaires pour s'effectuer. Ce processus doit être constamment surveillé, car on doit l'arrêter dans le cas où l'un des produits de base viendrait à manquer. De plus, ce processus pouvant être dangereux, un signal d'alarme doit être élaboré si des combinaisons particulières de produits apparaissent.

Ce signal d'alarme sera élaboré si:

1. Le composant z est présent et

- soit le composant x est présent et le composant w n'est pas présent,
- soit le composant w est présent et le composant y n'est pas présent,
- soit le composant y est présent et le composant x n'est pas présent,

ou

2. Les composants z et w ne sont pas présents et le composant y est présent.

A partir de ces spécifications, on peut écrire l'expression de la fonction correspondant au déclenchement du signal d'alarme:

$$f = \overline{w}\ x\ z + w\ \overline{y}\ z + \overline{x}\ y\ z + \overline{w}\ y\ \overline{z}$$

Cette expression n'est ni sous la forme d'une somme minimale ni sous la forme d'une somme canonique. Il est toujours possible de l'exprimer sous la forme d'une table de combinaisons, et d'obtenir une somme minimale au moyen de la méthode des tables de KARNAUGH. Par contre, le résultat (forme canonique) obtenu pourra contenir plus de produits fondamentaux qu'il n'en existe dans l'expression originale.

Il est possible d'éviter cette difficulté en dérivant la "somme entière" (somme de tous les impliquants premiers) directement à partir de l'expression de la somme de produits initiale, au moyen de la méthode des consensus.

L'opération consensus fut d'abord présentée par SAMSON <SAM-54> et utilisée par QUINE. Elle fut étudiée et formalisée par TISON <TIS-65> dans sa "Théorie des consensus", où il présente plusieurs algorithmes, dérivés de cette théorie, permettant la simplification de fonctions logiques.

3.2.3.1. L'opération consensus.

Les méthodes qui dérivaient une somme complète à partir d'une somme canonique faisaient appel aux théorèmes:

$X\,Y + X\,\overline{Y} = X$ (10) et

$X + X\,Y = X$ (8) vus au §3.2.1.

Les théorèmes : $X\,Y + \overline{X}\,Z = X\,Y + \overline{X}\,Z + Y\,Z$ et

$X + X\,Y = X$

formeront la base de la méthode que nous allons décrire.

Définition 1.

Soient $P = xy_1\,y_2 \ldots y_n$ et $Q = \overline{x}z_1\,z_2 \ldots z_m$, avec la possibilité qu'il y ait $y_i = z_j$ pour certains i et j.

On appellera consensus de P et Q , noté $P \not\subset Q$ l'ensemble $y_1\,y_2 \ldots y_n z_1 z_2 \ldots z_m$ (avec suppression des termes qui se répètent), à moins que $y_i = \overline{z_j}$, auquel cas le consensus n'existe pas.

Exemples

1. $(w\,x\,\overline{y}\,z) \not\subset (\overline{w}\,x\,\overline{u}\,v) = x\,\overline{y}\,z\,\overline{u}\,v$

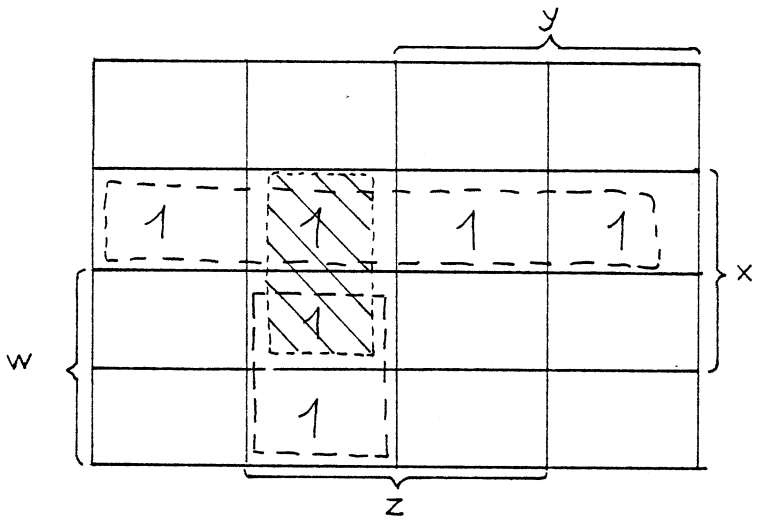
alors que

2. $(w\,x\,\overline{y}\,z) \not\subset (\overline{w}\,\overline{x}\,\overline{u}\,v)$ n'existe pas car on retrouve les termes x et $\overline{x}$ en plus de w et $\overline{w}$.

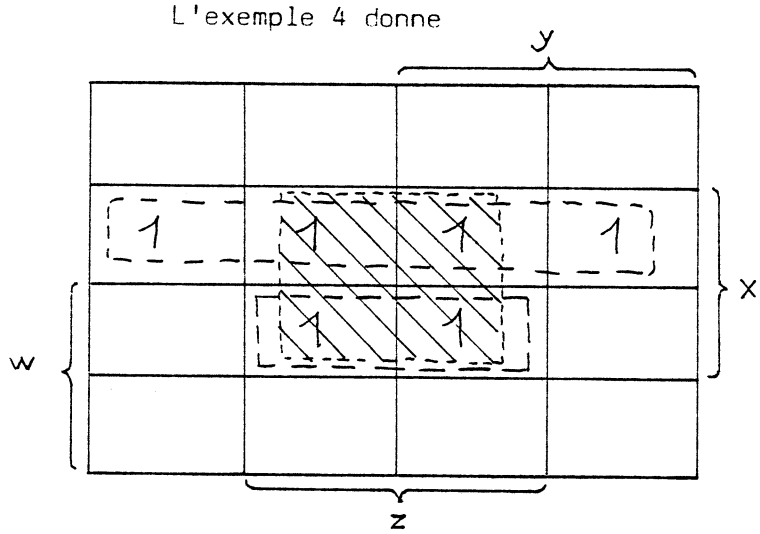
3. $\overline{w}\,x \not\subset w\,\overline{y}\,z = x\,\overline{y}\,z$

4. $\overline{w}\,x \not\subset w\,x\,z = x\,z$

Reprenons les exemples 3 et 4 pour les reporter sur une table de KARNAUGH. L'exemple 3 donne



L'exemple 4 donne



Nous constatons que dans les deux cas, le consensus est inclus dans la somme des deux termes duquel il est dérivé. De plus, il n'existe pas d'autres produits de littéraux qui contienne moins de littéraux (de produits fondamantaux) que le consensus obtenu, et qui ne soit pas inclus dans un des termes initiaux, mais inclus dans leur somme.

Le théorème suivant montre que cette propriété n'est pas spécifique aux exemples 3 et 4, mais qu'elle est vérifiée dans le cas général.

3.2.3.2. Généralisation de la notion de consensus.

Théorème 1.

Soient A,B et C des produits de littéraux. Si A + B contient C et que ni A ni B ne contiennent C, alors $A \not\subset B$ contient C. On pourra formaliser ce théorème en utilisant la relation classique, décrite par exemple dans <MCL-65>:

$$A + B \supseteq C$$

$$\text{alors } A \not\subset B \supseteq C$$

$$A \not\supseteq C \quad \text{et} \quad B \not\supseteq C$$

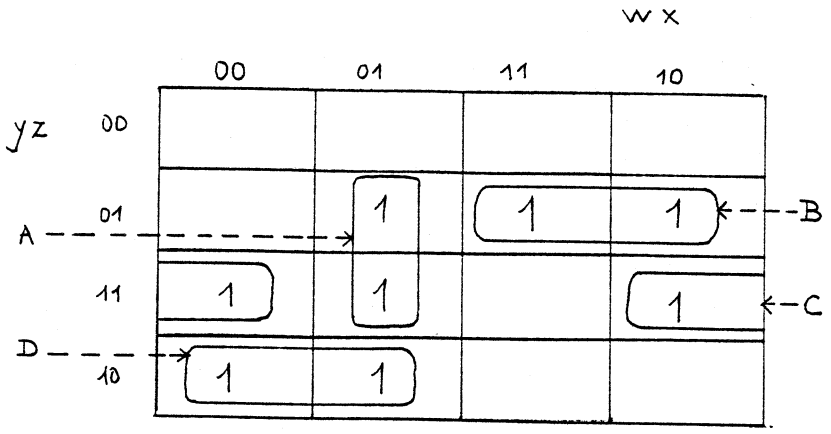
A partir de ce théorème, on peut déduire que l'addition successive de consensus avec une expression sous la forme de somme de produits, puis la suppression des termes inclus dans d'autres termes (X + X Y = X) donnera la "somme entière".

Avant de voir la formalisation précise de cette méthode permettant d'obtenir une somme entière, essayons d'illustrer ce concept par un exemple.

Exemple 5 .

Soit la fonction $f = \overline{w}\ x\ z + w\ \overline{y}\ z + \overline{x}\ y\ z + \overline{w}\ y\ \overline{z}$.

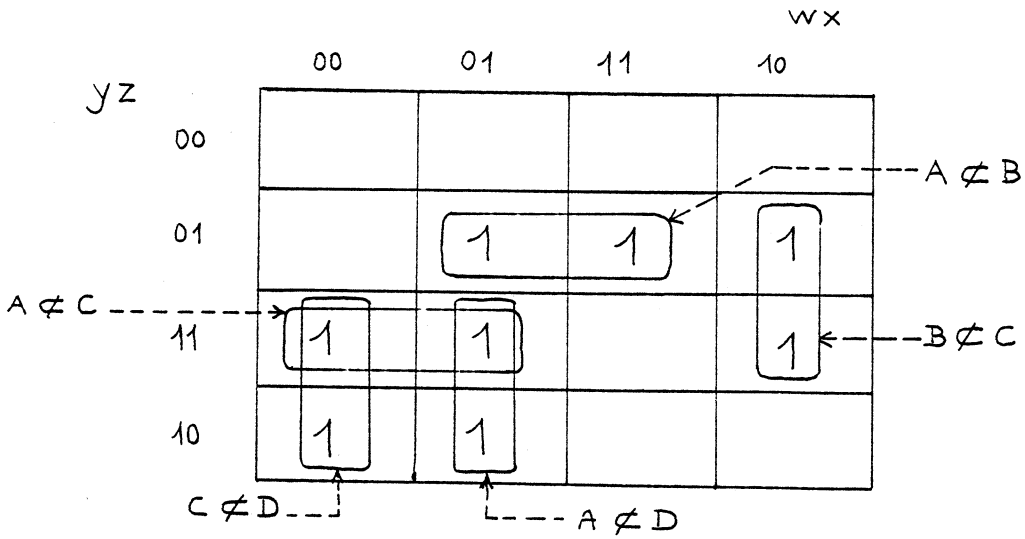
La somme exprimée est une somme irrédondante. Reportons cette fonction sur une table de KARNAUGH, pour reprendre une représentation qui nous est maintenant familière:



a) Table de KARNAUGH pour la fonction

$f = \overline{w}\ x\ z + w\ \overline{y}\ z + \overline{x}\ y\ z + \overline{w}\ y\ \overline{z}$

En appliquant l'opération consensus aux termes A,B,C,D ainsi créés, en les associant par paires, on obtient l'expression de la table suivante :

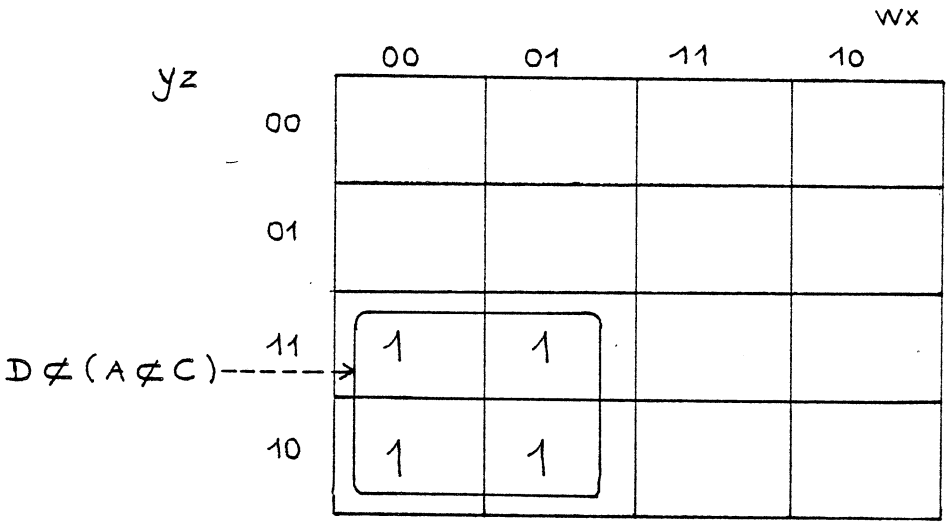


b) Table de KARNAUGH montrant les consensus formés à partir de a)

Si les termes du consensus ainsi obtenus sont additionnés à l'expression initiale, on obtient l'expression :

$$f = \overline{w}xz + w\overline{y}z + \overline{x}yz + \overline{w}y\overline{z} + x\overline{y}z + \overline{w}yz + \overline{w}xy + \overline{w}xz + \overline{w}\overline{x}y$$

La comparaison de paire de termes dans cette expression montre que le consensus $\overline{w}\ y\ \overline{z} \not\subset \overline{w}\ y\ z = \overline{w}\ y$ comme le montre la figure ci-dessous :



c) Table de Karnaugh montrant le consensus obtenu à partir de a) et b)

Ainsi $\overline{w}\ y$ peut être ajouté à l'expression, et comme $\overline{w}\ y \supseteq \overline{w}\ y\ \overline{z}$, $\overline{w}\ y \supseteq \overline{w}\ y\ z$, $\overline{w}\ y \supseteq \overline{w}\ x\ y$ et $\overline{w}\ y \supseteq \overline{w}\ \overline{x}\ y$, ces quatre termes peuvent être éliminés de l'expression. Après ces opérations, le résultat sera la fonction

$$f = \overline{w}\ y + \overline{w}\ x\ z + w\ \overline{y}\ z + \overline{x}\ y\ z + x\ \overline{y}\ z + w\ \overline{x}\ z$$

qui est la somme entière de cette fonction, sous sa forme minimale. De cet exemple, nous pouvons énoncer (sans le démontrer) un second théorème :

Théorème 2.

Une expression sous la forme de somme de produits pouvant s'écrire $E = P1 + P2 + \ldots + Pn$, représentant la fonction $f(x1,x2,\ldots,xn)$ est une somme entière pour f si et seulement si

1. aucun terme sous forme de produit ne contient un autre terme (produit), $Pi \not\supseteq Pj$ quels que soient i et j $(i \neq j)$.

2. le consensus de deux termes sous forme de produit, $Pi \not\subset Pj$ soit n'existe pas, soit est inclus dans d'autres termes (produits) $Pi \not\subset Pj \subseteq Pk$.

Ce théorème va nous servir à décrire un algorithme de minimisation d'une fonction booléenne logique.

3.2.3.3. Un algorithme de recherche d'une somme entière.

Cet algorithme tend à convertir l'expression initiale d'une fonction logique en une expression dans laquelle le consensus de 2 termes quelconques est inclus dans un autre terme, et dans laquelle aucun terme n'en inclut un autre. Pour celà,

chaque terme de l'expression est comparé successivement aux autres termes, puis

(1) chaque terme qui est contenu dans un autre terme est supprimé

(2) le consensus de 2 termes est additionné à l'expression, à moins que le consensus ne soit inclus dans un autre terme.

L'algorithme se termine quand chaque élément a été comparé à tous les autres éléments. Les éléments restants correspondent à tous les impliquants premiers. Le consensus de chaque paire d'éléments doit apparaître dans la table, s'il existe, sinon, il doit être inclus dans d'autres éléments.

Application.

Reprenons l'exemple 5 pour illustrer cet algorithme.

La fonction f vaut $f = \overline{w}\ x\ z + w\ \overline{y}\ z + \overline{x}\ y\ z + \overline{w}\ y\ \overline{z}$.

A étant le premier élément, B le deuxième,etc..., les termes initiaux de la somme de produits sont :

	W	X	Y	Z
A	0	1	-	1
B	1	-	0	1
C	-	0	1	1
D	0	-	1	0

Le consensus d'un élément avec chacun des autres donne :

	W	X	Y	Z
$B \not\subset A$	-	1	0	1
$C \not\subset B$	1	0	-	1
$C \not\subset A$	0	-	1	1
$D \not\subset C$	0	0	1	-
$D \not\subset A$	0	1	1	-

En additionnant chacun des consensus créés aux termes initiaux, celà devient :

	w	x	Y	z
$(A \not\subset C) \not\subset D$	0	-	1	-

La ligne 4 du premier tableau peût être supprimée, car elle est contenue dans le troisième tableau. De même pour les lignes 3,4 et 5 du deuxième tableau. Le résultat de cet algorithme sera la somme entière suivante :

$$f = \overline{w}\, y + w\, \overline{x}\, z + x\, \overline{y}\, z + \overline{x}\, y\, z + w\, \overline{y}\, z + \overline{w}\, x\, z.$$

Nous remarquerons que le résultat obtenu est identique au résultat de l'exemple 5, obtenu avec les tables de KARNAUGH

3.2.3.4. Généralisation de cet algorithme aux sommes minimales.

Cet algorithme, appelé "Algorithme de consensus itératif" par Mc CLUSKEY, et qui a été illustré pour une seule sortie peut être directement étendu à des fonctions à sorties multiples, moyennant quelques légères modifications. De plus, il sera possible de former, à partir de la somme entière, une table des impliquants premiers, qui donnera alors aisément la somme minimale.

3.2.3.5. Avantages, inconvénients de cette méthode.

Nous avons vu au §3.2.2.4 que la méthode dite des "tables de KARNAUGH" était difficilement implémentable sur calculateur, car elle était essentiellement graphique. La méthode utilisant les consensus, par contre, est une méthode parfaitement adaptée à un traitement informatique. De plus, ses résultats peuvent être très bons, car cette méthode nous permet d'obtenir la somme minimale, c'est-à-dire la couverture minimale de la fonction initialement définie.

Par contre, elle présente à mes yeux un inconvénient de taille : le nombre des termes (produits fondamentaux) contenus dans la somme entière est très élevé. De plus, la procédure consistant à former une table d'impliquants premiers, et par là même la somme minimale, nécessite de faire une expansion de cette somme entière. Aussi, le nombre de produits à former devient très

vite élevé, même si le problème initial est fortement indéfini. Ceci explique peut-être pourquoi une méthode générale satisfaisante n'a pu être trouvée.

Enfin, les contraintes introduites par ce grand nombre de produits créés font que, pour ce qui nous concerne, cette façon de travailler n'est guère satisfaisante car très mal adaptée à notre problème d'optimisation.

3.2.4. Une méthode de calcul des impliquants premiers et des couvertures irrédondantes.

Cette méthode a été développée par J.J. MONBARON <JJM-80>. Parmi toutes les méthodes dérivées de la même idée - calcul des impliquants premiers d'un système, puis recherche de la couverture irrédondante - MONBARON s'est attaché à trouver un algorithme simple, pouvant être programmé sur de petites machines. Sa manière de faire est, dans son principe, identique à la démarche de TISON <TIS-65> pour la recherche des consensus entre plusieurs produits. Cette recherche consiste à former tous les produits possibles des variables monoformes (c'est-à-dire non nulles) d'une implication. Chaque produit formé est alors impliquant si la somme des résidus biformes est une égalité à 1, ce qui revient à tester que le produit formé ne contient pas de zéro de la fonction.

3.2.4.1. Principe de la méthode.

Le problème consiste à trouver tous les produits possibles ne contenant pas de zéros. Pour ce faire, il suffit d'enlever à la fonction F contenant tous les produits possibles (c'est-à-dire la fonction booléenne 1) la fonction Z contenant tous les zéros du problème en calculant l'expression booléenne

$$F = 1 \cdot \overline{Z}$$

Une version générale de la méthode sera présentée, consistant simplement à calculer les solutions d'une équation F (X) = 1, représentées par les impliquants premiers de celle-ci. Un seul et même algorithme sera utilisé à la fois pour le calcul des impliquants premiers et des couvertures irrédondantes d'un système à plusieurs fonctions.

3.2.4.2. Processus de calcul des impliquants premiers.

Le principe de calcul des impliquants premiers d'une fonction F en somme de produits est simple :

il suffit de transformer $\overline{F}$ (somme de produits) en F (produit de sommes), chaque variable étant inversée, puis d'effectuer le calcul pour obtenir F en somme de produits parmi lesquels se trouvent tous les impliquants premiers.

Ceci peut se traduire par le théorème suivant :

Théorème 1.

Soient G et H , 2 fonctions exprimées par la somme de tous leurs impliquants premiers. Les impliquants premiers de F = G H s'obtiennent en effectuant le produit des impliquants premiers de G et de H.

Exemple 1.

$$G = a + b\,c \qquad H = a\,d + e\,f$$

$$F = G\,H = a\,d + a\,e\,f + a\,b\,c\,d + b\,c\,e\,f$$

3.2.4.3. Algorithme d'inversion d'une fonction, permettant le calcul de ses impliquants premiers.

L'inversion d'une fonction F, selon les lois énoncées au §3.2.1, suivie du calcul de l'expression en somme de produits fournit tous les impliquants premiers de $\overline{F}$.

Exemple 2.

Soit la fonction F donnée sous forme d'une somme de produits

$$F = A1 + A2 + A3 + \ldots + An$$

On pourra alors écrire $\overline{F}$ comme produit de sommes

$$\overline{F} = \overline{A1} \cdot \overline{A2} \cdot \overline{A3} \cdot \ldots \cdot \overline{An}$$

Si aucun Ai n'est nul, chaque $\overline{Ai}$ est une somme d'impliquants premiers.

D'où l'algorithme général :

P0 = 1 P0 = somme d'impliquants premiers

P1 = P0 . $\overline{A1}$ En développant P1 en somme de produits, on obtient P1 sous forme de somme de ses impliquants premiers (c.f. Théorème 1 vu ci-dessus)

En poursuivant le procéssus itérativement jusqu'à Pn = Pn-1 . $\overline{An}$ On obtient $\overline{F}$ = Pn sous forme de somme qui contient tous ses impliquants premiers.

Cet algorithme est aussi appelé "Sharp Algoritm" dans la littérature de langue anglaise, ou "méthode de passage à la duale" par TISON.

Exemple 3.

Soit la fonction $F = ab + \overline{b}c$

On peut aussi l'écrire :

$$\overline{F} = (\overline{a} + b)(b + \overline{c}) \quad \text{ou encore}$$

$$\overline{F} = \overline{a}\overline{b} + \overline{a}\,\overline{\overline{c}} + \overline{b}\,\overline{\overline{c}}$$

On a obtenu de cette manière les impliquants premiers de $\overline{F}$. Calculons les impliquants premiers de F de la même façon :

$$F = (a + \overline{b})(a + c)(b + c)$$

On calcule d'abord

$$(a + \overline{b})\ (a + c) \ = \ a + ac + \overline{ab} + \overline{bc}$$

On élimine les termes 2 et 3, qui ne sont pas premiers.

Il vient alors :

$$F \ = \ (a + \overline{bc})\ (b + c) \ = \ ab + ac + \overline{bc}$$

On constate que l'impliquant premier ac, qui ne figure pas dans la donnée est apparu.

Cet exemple est à rapprocher de la méthode des consensus vue au §3.2.3, car elle en est un cas particulier.

Nous avons décrit l'algorithme pour le cas de fonctions complètement définies. Cet algoritme fonctionne dans le cas général, qui est celui de fonctions incomplètement définies, le cas des fonctions complètement définies étant alors un cas particulier du précédent.

Après avoir étudié le principe de l'algorithme de calcul des impliquants premiers d'une fonction, il existe une étape finale consistant à calculer, à partir des impliquants premiers ainsi trouvés, une couverture minimale irrédondante de la fonction initialement définie.

Le détail de cet algorithme peut se retrouver dans les travaux de J.J. MONBARON <JJM-80>, où des exemples détaillés d'application sont fournis. Le but de notre étude n'était pas

d'utiliser cet algorithme, et son développement étant relativement long, nous resterons sur le principe de cet algorithme, vu plus haut, sans l'expliciter plus avant.

3.2.4.4. Efficacité de cet algorithme; avantages, inconvénients.

Cet algorithme, tel qu'il a été présenté par J.J. MONBARON semble relativement séduisant, car adapté aux petits ordinateurs, et se prêtant bien à une implémentation sous forme de programme simple et performant. MONBARON annonce que cet algorithme permet de traiter des systèmes de vingt à trente variables, dans des "temps raisonnables", et que la réduction de surface d'un PLA ainsi traité est de l'ordre de 20%. Aussi, cet algorithme a été programmé par l'équipe d'Architecture des Ordinateurs de l'IMAG, sur HB-68, système MULTICS.

Le temps CPU nécessaire au traitement d'un décodeur BCD-7 segments est de l'ordre de 40 secondes. Le taux d'optimisation est de l'ordre de 20%, comme énoncé. Par contre, l'application de cet algorithme à un gros PLA a été un échec. En effet, les données étaient rentrées sous forme de matrices dynamiques, et le fonctionnement de l'algorithme a démontré que cette structure de données n'était pas bonne (il aurait fallu utiliser des tableaux, gérés par le programme, et fonctionnant un peu comme une pile, de façon à avoir une meilleure gestion de la mémoire).

Pour le fonctionnement de cet algorithme avec de "gros" PLA, le nombre d'impliquants premiers et de produits intermédiaires formés devient vite très important, ce qui nécessite une place mémoire elle aussi très importante. Enfin, la durée de traitement (temps CPU) semble croître de façon exponantielle en fonction du nombre de termes du système initial.

Ainsi, malgré ses avantages apparents, et ses performances intéressantes, cet algorithme ne nous paraît pas adapté aux problèmes qui nous concernent. Rappelons que le but de l'optimiseur logique que nous voulions développer est de traiter de très gros PLA (des dizaines d'entrées et de sorties, des centaines de monômes), de façon à avoir une réduction de taille notable, et tout ceci avec des temps de traitement restant dans une norme acceptable (temps CPU < 1 heure).

Nous avons donc abandonné l'idée d'un algorithme de minimisation tendant à donner la couverture minimale d'un système, pour nous tourner vers les algorithmes "sous-optimaux". Nous étudierons dans le paragraphe suivant le fonctionnement d'un tel algorithme.

3.3. Les méthodes "sous-optimales" d'optimisation logique.

3.3.1. La nécessité des méthodes sous-optimales.

L'approche classique de la minimisation de fonctions logiques booléennes fait appel à un processus à 2 étapes qui génère tout d'abord tous les impliquants premiers puis recherche

ensuite la couverture minimale. Cette approche, développée par QUINE <QUI-52>, <QUI-55> et Mc CLUSKEY <MCL-56>, a été une amélioration notable des méthodes consistant à construire puis comparer toutes les solutions possibles. Cependant, cette méthode génère un nombre d'impliquants premiers assez importants. Pour un problème admettant n fonctions, le nombre d'impliquants premiers sera de l'ordre de $3^n/n$ <MIL-65>. Ceci explique que cette approche classique devienne impossible à mettre en oeuvre pour des problèmes admettant beaucoup de variables, car l'espace mémoire requis pour une implémentation sur ordinateur, ainsi que le temps de traitement nécessaire, deviennent rapidement très importants.

La taille des problèmes devant être traités a entraîné l'apparition des algorithmes "sous-optimaux". Le principe de tels algorithmes est de chercher une solution meilleure, même si cette solution n'est pas optimale.

3.3.2. L'approche heuristique des méthodes d'optimisation logique.

Cette approche fait appel à deux notions de base, quels que soient les algorithmes développés avec cette méthode.

- La première notion est que le coût associé à chaque fonction est le même, quelle que soit la fonction considérée. Le nombre d'impliquants seul est considéré, de telle sorte que beaucoup de problèmes associés au minimum ponctuel de chaque impliquant sont ainsi éliminés.

- La seconde notion est que la solution finale est obtenue à partir de la solution initiale, en améliorant de façon itérative la couverture, plutôt que de générer les impliquants premiers, puis leur couverture minimale <HON-74>.

Le principe de telles méthodes sera alors sensiblement le même :

- Chaque impliquant est réduit à sa taille minimale, tout en conservant la couverture de la fonction associée.

- Ensuite, les impliquants sont associés 2 par 2, de façon à voir si l'un d'eux peut être réduit et l'autre augmenté, de telle sorte que la couverture initiale soit augmentée.

- Enfin, chaque impliquant est amené à sa taille maximale, afin que d'autres impliquants inclus dans le premier puissent être éliminés.

A ce niveau, les différences entre différentes méthodes résideront dans l'ordre de réduction, de restructuration et d'élimination des impliquants; ceci peut amener divers algorithmes à avoir des résultats sensiblement différents, bien que leur principe soit au départ basé sur des notions identiques. Nous avons étudié et réalisé un tel algoritme, et le chapitre suivant va nous le présenter.

CHAPITRE IV

DESCRIPTION D'UN OPTIMISATEUR LOGIQUE SOUS-OPTIMAL

4.1. Définitions.

Cet algorithme faisant appel à des notions sensiblement différentes de celles que nous avons vues dans les chapitres antérieurs, nous allons introduire quelques définitions utilisées par la suite.

4.1.1. Notation cubique.

1. Cube. Si l'on considère une fonction booléenne F, vérifiant une relation entre les entrées et les sorties associées à cette fonction, on appellera cube la paire de données correspondantes en entrée et en sortie de cette fonction.

Exemple 1 Fonction = F

Entrées = a, b, c, d

Sorties = e, f, g

Si l'on assigne aux variables les valeurs a = 1,c = 0,d = 0,e = 1, f = 0, le cube ainsi formé pourra s'écrire

1	X	0	0	/	1	0	X
a	b	c	d		e	f	g
entrées					sorties		

2. Coordonnées d'un cube. Les coordonnées d'un cube sont formés par chaque élément (a,b,c...) d'un cube.

Une coordonnée c peut prendre indifféremment les valeurs 0,1 ou X.

La coordonnée X vaut 0 ou 1 : elle signifie que dans l'énoncé de la fonction, cette coordonnée n'est pas définie.

Une coordonnée ayant la valeur 0 ou 1 est appelée une coordonnée finie.

Une coordonnée ayant la valeur X est appelée une coordonnée indéfinie.

3. Sommet d'un cube. On appelle sommet le cube ayant

- En entrée, toutes ses coordonnées ayant la valeur 0 ou 1,
- En sortie, toutes ses coordonnées sauf une ayant la valeur X.

Exemple 2

0 0 1 0 / 1 X X est un sommet.

4. Coût associé à un cube.

Si f (p) est le coût associé au cube u

si f (q) est le coût associé au cube v

si u contient v alors

$f(p) \leqslant f(q)$.

Conséquence : des cubes plus importants ont un coût moindre.

Pour les PLA, on peut assimiler la notion de coût au nombre de transistors. La réduction du coût d'un PLA reviendra donc à réduire le nombre de ses transistors.

4.1.2. Opérations sur les cubes.

Après avoir défini les notions de base appliquables aux cubes, nous allons définir les opérations élémentaires que l'on peut effectuer sur les cubes.

1. Contenance.

Le cube u contient le cube v si u contient toutes les informations de v.

De cette notion, on peut déduire les règles de contenance : u contient v si on peut former v à partir de u de la façon suivante:

- Pour les entrées de u : les coordonnées qui ont pour valeur X sont changées en 0 ou 1, ou restent inchangées.
- Pour les sorties de u : les coordonnées qui ont pour valeur 0 ou 1 sont changées en X, ou restent inchangées.

Exemple 3

(u) 1 X X 1 / 1 1 X contient

(v) 1 0 X 1 / 1 X X

La 2ème entrée de u qui était X est changée en 0,

La 2ème sortie de u qui était 1 est changée en X.

Conséquence : Si u contient v, v peut être éliminé sans changer l'essence de la fonction initiale.

Cette notion de contenance est très importante, car elle nous permettra de simplifier les fonctions de PLA en supprimant de ses fonctions toutes celles qui vérifient cette règle.

2. Interface de deux cubes

On appelle encore cet interface une intersection, notée I. L'interface de 2 cubes u et v (u I v) est le cube unique, plus grand que u ou v, commun à u et à v.

L'opération INTER est effectuée coordonnée par coordonnée selon les règles suivantes :

C(v) \ C(u)	0	1	X
0	0	D	0
1	D	1	1
X	0	1	X

Le terme D indique une dégénérescence (un conflit entre les coordonnées).

Exemple 4

(1X11 / 1X) I (10X1 / 10) = 1011 / 10

u v w

Exemple 5

$$\underset{u'}{(\ 1001\ /\ X1\ )}\ I\ \underset{v'}{(\ 1100\ /\ X1\ )}\ =\ \underset{w'}{1D0D\ /\ X1}$$

Dans l'exemple 4, w dénote un interface commun à u et v.

Dans l'exemple 5, w' dénote un interface dégénéré.

3. Consistance

Deux cubes u et v sont dits consistants si un conflit D dans la partie sorties implique un conflit D dans la partie entrées.

Exemple 6.

$$\underset{u}{(\ 10X1\ /\ 1X1\ )}\ I\ \underset{v}{(\ 1XX0\ /\ 100\ )}\ =\ \underset{w}{10XD\ /\ 10D}$$

Les cubes u et v sont consistants, car leur interface contient un D dans la partie entrées et dans la partie sorties.

Exemple 7.

$$\underset{u'}{(\ 10X1\ /\ 1X1\ )}\ I\ \underset{v'}{(\ 1XX1\ /\ 100\ )}\ =\ \underset{w'}{10X1\ /\ 1XD}$$

u' et v' sont inconsistants car il n'y a de D que dans la sortie.

Cette notion de consistance servira à sélectionner les interfaces des cubes, dans l'algorithme d'optimisation.

4. Disjonction.

Deux cubes u et v sont disjoints si leur interface est dégénéré (si elle contient au moins un D).

4.1.3. Opérations sur les cubes.

Nous avons parlé, dans les chapitres précédents des couvertures de fonctions booléennes, mais une nouvelle définition sera donnée ci-dessous, plus adaptée à notre algorithme.

1. Couverture

On appellera couverture un ensemble de cubes, consistants paire par paire, et non dégénérés.

2. Couverture "ON" (matrice "ON")

L'utilisation de l'algorithme nous amène à créer une couverture, appelée couverture "ON", dont la définition est la suivante : la couverture "ON" (appelée encore matrice "ON") est la couverture formée à partir de la couverture originale C en changeant tous les 0 de la partie sorties en X.

3. Couverture "OFF" (matrice "OFF")

De même, on pourra définir la couverture "OFF" comme étant formée à partir de la couverture originale C en changeant tous les 1 de la partie sorties en X.

Exemple 8

C = 0 0 0 / 1 0
0 0 1 / 0 1
0 X 1 / 1 1

C (on) = 0 0 0 / 1 X
0 0 1 / X 1
0 X 1 / 1 1

C (off) = 0 0 0 / X 0
0 0 1 / 0 X
0 X 1 / X X

Nous avons maintenant en main toutes les données nous permettant de décrire l'algorithme d'optimisation logique.

4.2. SHRINK : un algorithme d'optimisation logique.

Dans ce paragraphe, nous allons, après avoir décrit le principe de l'algorithme, examiner les performances et la mise en oeuvre de cet algorithme.

4.2.1. Principe.

Cet algorithme est inspiré par les travaux de J.P. ROTN <ROT-78>, <ROT-80>, et de YACOUB EL-ZIQ <YAC-80>. dans les deux cas, les approches sont les mêmes, et s'inspirent des mêmes concepts de base.

L'algorithme de minimisation peut se décomposer en quatre étapes principales :

1. Elaboration des matrices ON et OFF à partir de la couverture initiale.

2. Chaque cube de la matrice ON est sélecté, et chaque élément fini (0,1) de la partie entrée de ce cube est changé en élément indéterminé (X). Le cube ainsi formé est interfacé avec chaque cube de la matrice OFF, pour essayer d'agrandir la couverture.

3. Après avoir procédé de manière itérative avec tous les cubes de ON et OFF, l'opération est recommencée, mais en sélectant un cube de ON, et en changeant un élément indéterminé de la partie sortie (X) en élément fini (0,1).

4. La dernière étape est l'élimination des redondances, effectuée grâce à la procédure détectant les contenances des cubes dans d'autres, et enfin la mise en forme des résultats sous le format d'entrée de PAOLA.

4.2.2. Description de l'algorithme.

lère étape. Cette étape consiste à lire le fichier en entrée, composé de la couverture initiale, puis de créer les matrices ON et OFF. La procédure de lecture comporte une phase

d'acquisition des paramètres du fichier lu (nom du PLA, nombre d'entrées de la partie ET, nombre de sorties de la partie OU, nombre de lignes du PLA), une phase d'acquisition des données (lecture des points du PLA), et une phase de calcul (calcul du nombre de colonnes du PLA, calcul du nombre de transistors, du nombre de cubes - monômes).

La procédure de création des matrices ON et OFF est relativement simple :

- La matrice ON est créée en changeant toutes les coordonnées 0 de la partie sortie du PLA en X.

- La matrice OFF est créée en changeant toutes les coordonnées 1 de la partie sortie du PLA en X.

2ème étape. Nous appelerons cette étape le "cofaçage" de la partie entrée de ON avec OFF.

- Chaque élément fini des entrées de chaque cube de ON est changé en X

- Chaque cube ainsi formé est interfacé avec chaque cube de OFF.

- Si le résultat de l'intersection est consistant, le nouveau cube formé remplace celui duquel il est issu dans la matrice ON.

- Si le résultat de l'intersection n'est pas consistant (c'est-à-dire si l'interface est dégénéré), le cube est restauré à sa valeur initiale.

Le processus est répété itérativement jusqu'à ce que chaque cube de chaque matrice ON et OFF ait ainsi été traité. Le résultat de cette seconde étape est une nouvelle couverture C', plus grande que la couverture initiale C, ayant donc à priori un coût moindre. Il est à noter qu'à la fin de cette étape, aucune réduction n'est encore apparue.

3ème étape. Cette étape est le cofaçage de la partie sortie de ON avec OFF.

- Chaque élément indéfini de chaque sortie de ON est changé en 1.

- Chaque cube ainsi formé est interfacé avec chaque cube de OFF.

- Si le résultat de l'intersection est consistant, le nouveau cube formé est ajouté à la matrice ON, et il est comparé avec chacun des autres cubes de ON, afin que soient éliminés les autres cubes qu'il contient.

Notons que cette phase permet d'éliminer certaines redondances : en effet si le cube formé est retenu, après avoir balayé la matrice ON, les autres cubes contenus par ce nouveau

cube sont éliminés. Par contre, si d'autres cubes de ON contiennent le cube formé, il ne pourra pas être éliminé à ce stade, mais lors de l'étape suivante. Ceci permet cependant l'élimination de certaines lignes du PLA, donc une amélioration du temps de traitement, particulièrement utile lors du traitement de gros PLA.

- Si le résultat de l'intersection n'est pas consistant, le cube formé est restauré à sa valeur initiale.

4ème étape. Cette étape peut se ramener à trois phases distinctes :

- Elimination des redondances : Dans la couverture obtenue, on compare chacun des cubes de la nouvelle matrice ON avec tous les autres cubes, de façon à éliminer les cubes contenus dans le cube sélecté (cette phase est complémentaire de la phase d'élimination des redondances de la 3ème étape).

- La partie sortie de la matrice ON est restaurée, et tous les éléments qui étaient changés en X sont mis à zéro. En effet, du point de vue logique électrique, un X dans la matrice OU d'un PLA signifie qu'il peut ne pas exister de connexion, ce qui explique que les X correspondants soient forcés à 0, ce qui donne un gain de 1 transistor. Dans le même temps s'effectue le calcul du nombre de transistors et de monômes de la couverture finale.

- Enfin, une phase de mise en forme du résultat permettra au PLA optimisé logiquement d'être scindé en deux fichiers (ET et OU), qui pourront directement être mis en entrée de l'optimiseur topologique PAOLA.

La figure 4.1 nous donne l'organigramme de l'optimisateur logique.

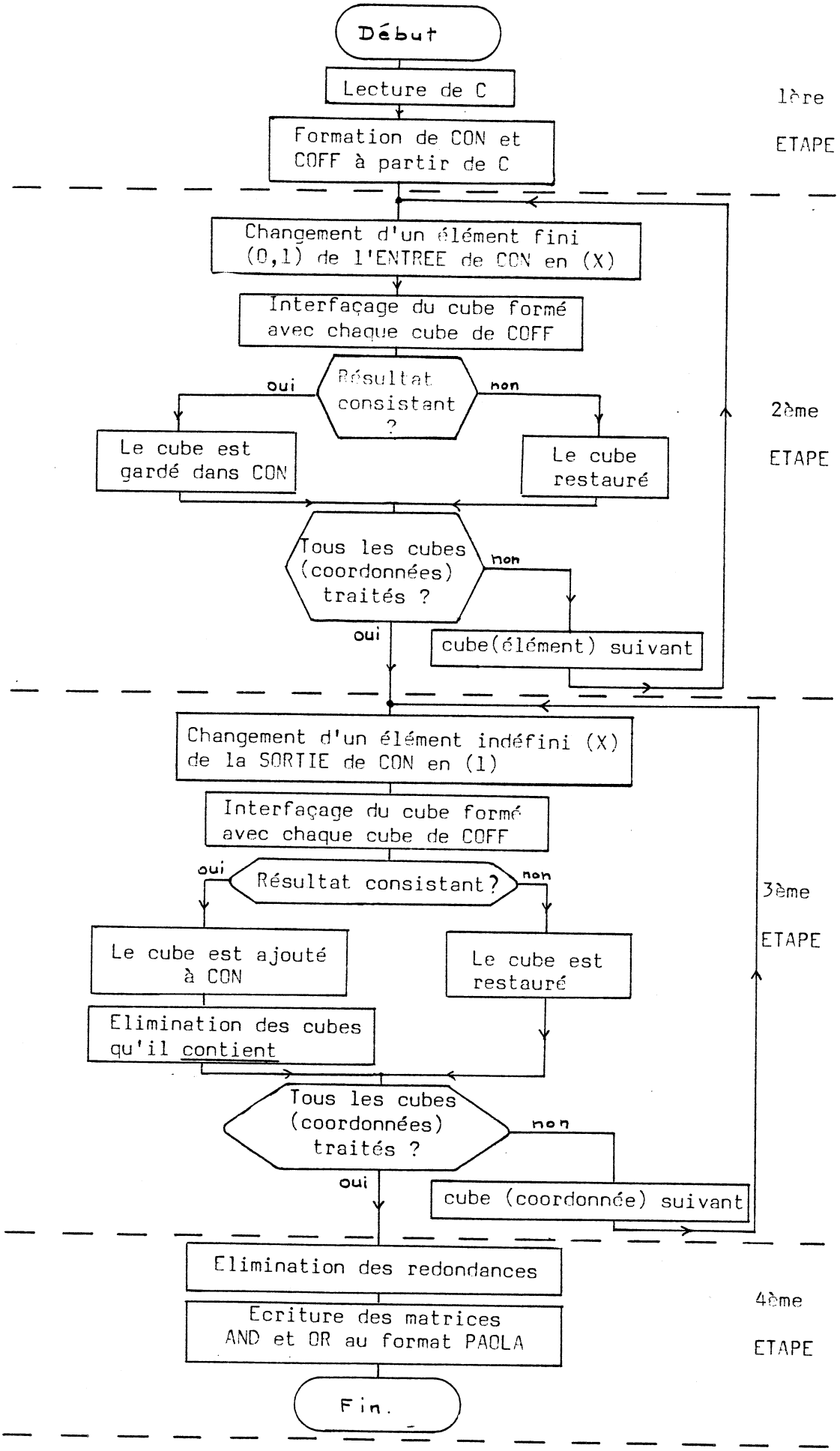


Figure 4.1 Description de l'algoritme

Nous allons illustrer le fonctionnement de cet algorithme à l'aide de l'exemple suivant.

Soit un PLA, défini avec les valeurs :

- nombre d'entrées = 3 a,b,c
- nombre de sorties = 2 x,y

La table de vérité de ce PLA sera

a	b	c	x	y
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0

1. La 1ère étape consistera à lire ce PLA et à le transformer en 2 matrices Con et Coff, ce que l'on pourra écrire, en adoptant la notation cubique :

	Couverture initiale	Matrice ON	Matrice OFF
1.	0 0 1 / 0 1	0 0 1 / X 1	0 0 1 / 0 X
2.	0 1 0 / 0 1	0 1 0 / X 1	0 1 0 / 0 X
3.	0 1 1 / 0 1	0 1 1 / X 1	0 1 1 / 0 X
4.	1 0 0 / 1 0	1 0 0 / 1 X	1 0 0 / X 0
5.	1 0 1 / 1 0	1 0 1 / 1 X	1 0 1 / X 0
	C	C1 (ON)	C (OFF)

2. La seconde étape sera le cofaçage de la partie entrées de Con avec Coff. Ceci s'effectue cube par cube.

Le premier cube est sélecté : 0 0 1 / X 1.

Le premier élément fini de la partie entrées est changé en X, on obtient m' : X 0 1 / X 1.

Le cube m' est intersecté avec le second cube de Coff (on ne fera pas l'intersection du cube de rang n de Con avec le cube de rang n de Coff). On peut écrire cette étape :

. (X 0 1 / X 1) I (0 1 0 / 0 X) = 0 D D / 0 1

Le résultat est consistant (la partie sortie ne comporte pas de dégénérescence).

On intersecte ensuite m' avec le cube 3 de Coff.

. (X 0 1 / X 1) I (0 1 1 / 0 X) = 0 D 1 / 0 1 (cube consistant)

Puis on répète l'opération avec le cube 4 de Coff.

. (X 0 1 / X 1) I (1 0 0 / X 0) = 1 0 D / X D (cube consistant car une dégénérescence de la partie sorties entraîne une dégénérescence de la partie entrées)

. (X 0 1 / X 1) I (1 0 1 / X 0) = 1 0 1 / X D (cube non consistant car dégénérescence seulement en sortie).

Le premier cube de Con obtenu ainsi n'est donc pas gardé.

Le cube m' est restauré et sauvegardé dans Con.

On passe alors au second élément fini du cube :

0 0 1 / 0 1 ==> 0 X 1 / 0 1.

Par itération de cette procédure, la fin de la seconde étape nous donne une nouvelle couverture (ON) qui est :

0 0 X / X 1

0 1 0 / X 1

C2 (ON) = 0 1 1 / X 1

1 0 0 / 1 X

1 0 1 / 1 X

Cette couverture est plus grande que la couverture initiale, mais à ce stade, aucune réduction n'est encore apparue. Nous pouvons alors passer à la troisième étape de l'algorithme.

3. Cette troisième étape sera le cofaçage de la partie sortie de Con avec Coff, cube par cube. Le premier cube de Con est sélecté, et le premier élément indéfini de la partie sorties est changé en 1. On obtient m'' :

0 0 X / X 1 ==> 0 0 X / X X.

Ensuite, comme à l'étape 2, on effectue l'intersection avec chaque cube de Coff.

(0 0 X / X X) I (0 0 1 / 0 X) = 0 0 1 / D 1

Le résultat est dégénéré : le cube n'est pas gardé dans la couverture.

Si le résultat avait été non dégénéré, le cube m'' eût été ajouté à cette couverture, puis comparé à tous les autres cubes de cette couverture afin d'éliminer les cubes que m'' contient.

Après déroulement de cette étape, nous obtenons la couverture (ON) :

$$
C3\ (ON) = \begin{array}{l} 0\ 0\ X\ /\ X\ 1 \\ 0\ 1\ 0\ /\ X\ 1 \\ 0\ 1\ 1\ /\ X\ 1 \\ 1\ 0\ 0\ /\ 1\ X \\ 1\ 0\ 1\ /\ 1\ X \end{array}
$$

Nous constatons que dans cet exemple, l'étape 3 n'est pas arrivée à agrandir la couverture obtenue à l'étape 2. Nous pouvons alors passer à l'étape 4.

4. Cette étape consiste à éliminer les redondances. Ceci est effectué à l'aide d'une procédure (CONTAIN) qui détecte tous les cubes contenus dans d'autres cubes, et qui les élimine. Cette procédure fonctionne selon le principe vu au §4.1.2.

Ainsi le cube 2 de C3 (ON) devient :

$$0\ 1\ 0\ /\ X\ 1 \quad ==> \quad 0\ 1\ X\ /\ X\ 1$$

Le cube 3 est alors contenu dans le cube 2

$$0\ 1\ X\ /\ X\ 1 \quad < \quad 0\ 1\ 1\ /\ X\ 1$$

Le cube 3 peut alors être supprimé.

La couverture Con obtenue à la fin de la quatrième étape sera alors, après changement des éléments X de la partie sortie en 0 :

$$
C4 = \begin{array}{ll} 0\ 0\ X & 0\ 1 \\ 0\ 1\ X & 0\ 1 \\ 1\ 0\ X & 1\ 0 \end{array}
$$

Il ne restera alors plus qu'à former deux fichiers de sortie (ET et OU) pour les mettre au format d'entrée de l'optimiseur topologique PAOLA.

Nous pouvons remarquer que le résultat de l'optimisation logique n'est pas optimal. En effet, les cubes 1 et 2 du résultat peuvent encore être réduits à un seul cube (0 X X / 0 1). Mais la philosophie de cet algorithme, qui est rappelons-le "sous-optimal", est d'obtenir une solution satisfaisante en un temps minimum. Aussi, cet algorithme n'effectue pas toutes les combinaisons possibles, et ceci explique que le résultat obtenu peut encore être amélioré.

Dans cet exemple, nous sommes passés de 5 cubes à 3 cubes, et de 20 transistors à 12.

Le gain obtenu est alors de 40%.

Le temps CPU nécessaire a été de 0,574 secondes.

La figure 4.2 nous montre les résultats de l'algorithme.

```
0 0 0     1 0
0 0 1     0 1
0 1 1     0 1
1 0 0     1 0
1 0 1     1 0

  Nombre de Transistors =          20

 Nombre de Monomes =          5

0 x 0     1 0
0 x 1     0 1
1 x x     1 0

  Nombre de Transistors =          12

  Nombre de Monomes =          3
```

Figure 4.2

Résultats de l'algorithme avec un exemple de PLA simple.

4.2.3. Performances, résultats.

Le programme de minimisation logique correspondant à l'algorithme a été implanté sur le HB-68 DPS de l'IMAG. Il a été écrit en PASCAL, et représente un volume de 2000 lignes environ.

Ce programme a été testé sur plusieurs exemples représentant des PLA existants de types différents. Les résultats montrent un taux d'optimisation variant de 5% à 78%. Il est à noter que le taux de 78% d'optimisation a été obtenu à partir d'un PLA de génération de code opération d'un micro-processeur, lequel PLA a été généré automatiquement par programme, et comporte donc de nombreuses redondances. Cependant, un taux moyen de 10% à 30% est à espérer, et apparaît comme logique.

Le tableau de la page suivante nous donne quelques exemples des résultats de l'optimisation logique de ce programme.

De ces résultats, nous pouvons tirer quelques observations :

- L'optimisation obtenue par programme ne donne pas la solution optimale, mais ceci n'était pas le but de l'optimiseur réalisé.

- Cette optimisation se rapproche souvent des résultats obtenus par le concepteur lorsqu'il effectue lui-même la minimisation du PLA de façon manuelle.

- Le résultat de l'optimisation est fortement dépendant du taux de remplissage du PLA initial. Ainsi, une matrice ET (une matrice OU) en entrée (sortie) très creuse et ayant beaucoup d'entrées (de sorties) ne donnera que des résultats faibles, car

Tableau comparatif de résultats d'optimisation de divers PLA.

Nom PLA	Nb ent. (ET)	Nb sor. (OU)	Nb monomes		Nb transist.		Temps CPU (secondes)	Gain %
			n.opt	opt	n.opt	opt		
PLAJPS	8	50	72	16	1596	332	102	78
PLAMON3	12	8	120	82	1724	1172	78	32
EXSIEM	8	38	161	142	2732	2497	204	12
S1	8	50	36	16	797	332	28	65
S2	10	14	21	20	266	253	6	5
S3	8	18	57	51	587	531	24	11
SIEMENS	8	38	201	164	3261	2780	292	18

la probabilité que des paires de monômes ne diffèrent que d'une entrée (une sortie) est faible.

- Le temps de traitement est dépendant du nombre d'entrées du PLA et du nombre de monômes du PLA. Mais contrairement aux méthodes d'optimisation classiques qui montrent une évolution exponentielle de ce temps de traitement (du type 2^{2^n} , n étant le nombre d'entrées), le temps de traitement de notre algorithme semble croître de façon plus linéaire.

Cependant, nous ne pouvons pas déduire de ces résultats une formule globale, car la diversité des exemples (PLA de séquencement, de génération de commandes, de génération de code opération, ...) interdit la comparaison de ces résultats.

Le résultat obtenu par cet algorithme est, nous l'avons souvent dit, sous-optimal. La manière de fonctionner de cet algorithme lui interdit d'effectuer toutes les combinaisons possibles d'éléments entre eux. Aussi, pour essayer d'améliorer les résultats obtenus, nous avons tenté de ré-introduire en entrée du programme les résultats précédents. Nous avons alors constaté que lors d'une seconde itération du programme, le résultat pouvait parfois être amélioré de façon notable. Ainsi, dans le tableau précédent, le PLA appelé PLAJPS a été optimisé en 2 étapes. La première étape nous a donné en 75 secondes une réduction de 50% (72 à 36 monômes), alors que la seconde étape nous a donné en 27 secondes une réduction de 36 à 16 monômes. Par contre, certains exemples n'ont donné aucune amélioration sérieuse lors d'un second passage.

Nous trouverons en annexe certains exemples d'optimisation obtenus avec ce programme, avec les PLA cités en exemple.

4.2.4. Mise en oeuvre de l'algorithme.

La mise en oeuvre de l'optimiseur logique nécessite l'existence d'un PLA en entrée au format adéquat. Ce format, qui pourra être celui d'un fichier saisi manuellement, ou créé par un générateur de PLA est le suivant :

- 1ère ligne : nom du PLA . << NAME - Exemple >>
- 2ème ligne : nombre de lignes (monômes du PLA).

<< ROWS - 1 >>

Le mot clé ROWS (ou LIGNE) est séparé du nombre de lignes par un espace au moins.

Le nombre de lignes est un nombre alphanumérique compris entre 1 et 999 inclus.

- 3ème ligne : nombre d'entrées du PLA << AND e >>

Le mot clé AND est séparé du nombre d'entrées par un espace au moins. Ce nombre d'entrées est un nombre alphanumérique compris entre 1 et 50 inclus.

- 4ème ligne : nombre de sorties du PLA << OR s >>

Le mot clé OR est séparé du nombre de sorties par un espace au moins.

Ce nombre de sorties, alphanumérique est compris entre 1 et 50 inclus.

- 5ème ligne : mot clé << CELLS >> annonçant que la suite se compose des cellules proprement dites du PLA.
- 6ème ligne à ligne l : cellules du PLA.

Chaque cellule est un caractère alphanumérique (0 : pas de transistor, 1 : un transistor, X : indifférent). Ces caractères peuvent être ou non séparés par un blanc.
- Dernière ligne : mot clé << END >> : fin de fichier.

Un exemple est donné de ce fichier en entrée par la figure 4.3.

```
name exemple
rows 5
and 3
or 2
cells
0 0 1 0 1
0 1 0 0 1
0 1 1 0 1
1 0 0 1 0
1 0 1 1 0
end
```

Figure 4.3. Fichier en entrée de l'optimiseur logique

Dans la partie ET du fichier, nous trouvons des 0, des 1 ou des X pour indiquer l'absence ou la présence de transistors. Or, un générateur de PLA peut générer des fichiers ayant dans la partie ET des transistors sous la forme 01 : transistor, 10 : pas de transistor ou 11 : X.

Un programme complémentaire, appelé FORME2 est capable de générer un fichier conforme aux spécifications de l'optimiseur, à partir de telles matrices ET. Dans ce cas, on affectera une matrice ET dédoublée sous le nom << ENTREE1 >>, une matrice OU normale sous le nom << ENTREE2 >>, et le programme FORME2 générera un fichier normal appelé << SORTIE >> (fig. 4.4).

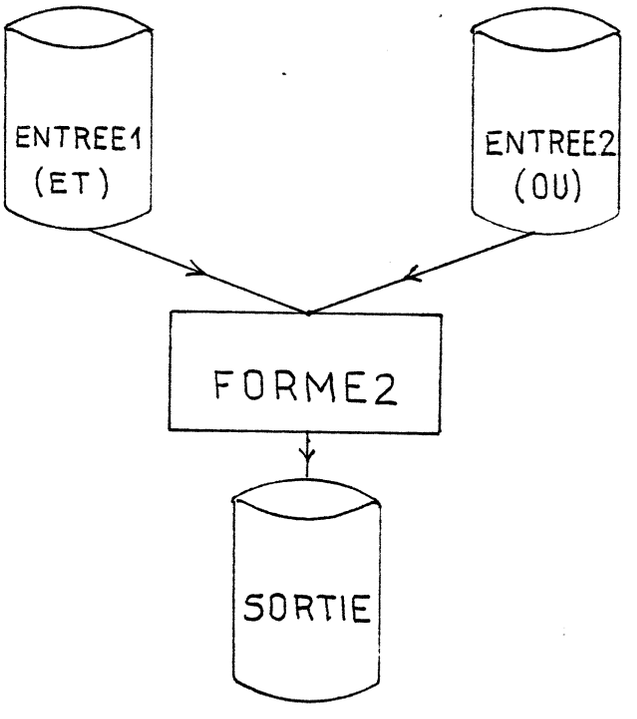


Figure 4.4. Programme de mise en forme des fichiers ET et OU

Les fichiers en sortie seront identiques au fichier d'entrée, mais ne comporteront qu'une partie ET (OU) comme le montre la figure 4.5.

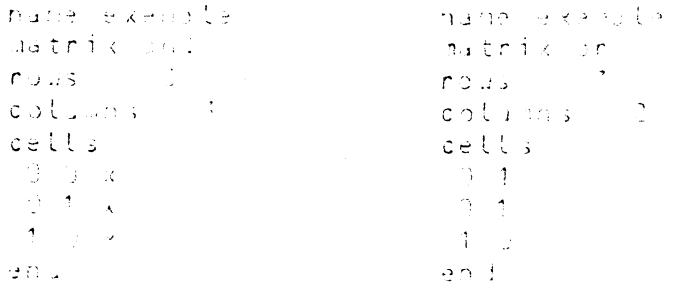


Figure 4.5. Format des fichiers ET et OU en sortie de l'optimiseur

Les fichiers en entrée du programme sont donc définis. Le programme génère en sortie un fichier appelé << SORTIE >> qui donne le résultat de l'optimisation, visualisable sur écran ou que l'on peut éditer sur imprimante (voir annexes). Il génère également deux fichiers AND et OR, qui seront au format d'entrée de l'optimiseur topologique PAOLA (voir fig. 4.6).

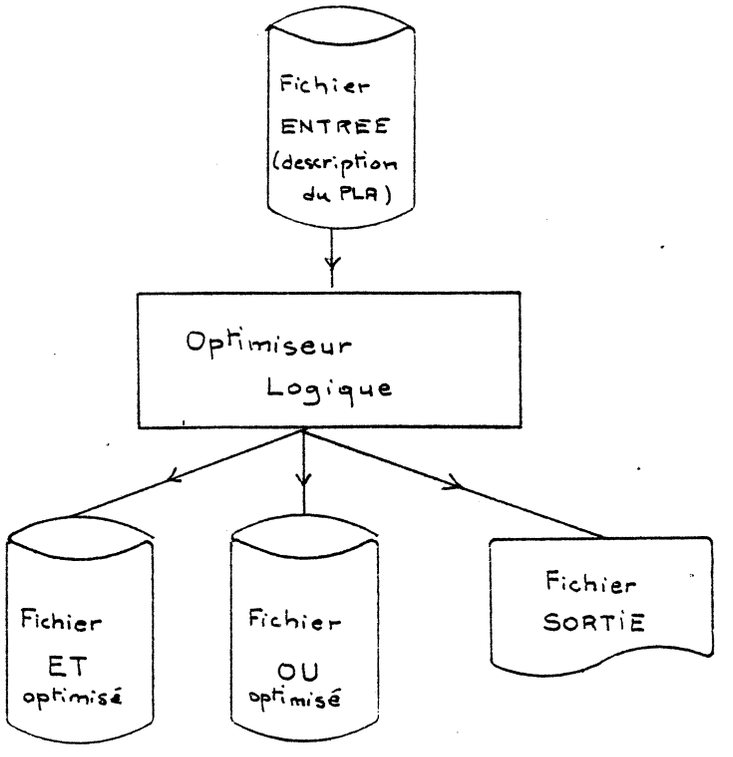


Figure 4.6. Fichiers utilisés par l'optimiseur logique.

Il est possible de reprendre les fichiers ET et OU générés et de les fusionner afin de remettre le 1er résultat de l'optimisation en entrée de l'optimiseur, afin, nous l'avons vu, d'obtenir par un second passage une meilleure optimisation. Le programme FORME, semblable au programme FORME2 permet une telle opération.

Le volume des programmes FORME et FORME2 est de l'ordre de 800 lignes PASCAL chacun.

Enfin, les résultats de l'optimisation logique sont délicats à contrôler, surtout dans le cas de gros PLA optimisés. Un programme appelé VERIF a donc été écrit, représentant moins de 1000 lignes de PASCAL (fig. 4.7). Il permet de mettre en entrée le PLA non optimisé <<ENTREE1>> et le PLA optimisé <<ENTREE2>> et de créer un fichier résultats (<< SORTIE >>) comprenant :

- Les monômes ayant disparu du fichier original, et leur numéro de ligne.

- Les monômes étant apparus dans le fichier optimisé, ainsi que leur numéro de ligne.

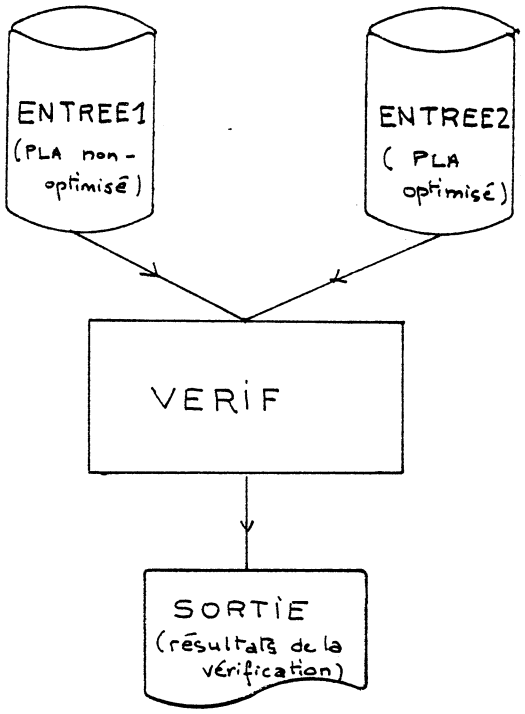


Figure 4.7. Fonctionnement du programme VERIF

Le concepteur peut ainsi vérifier manuellement le résultat de l'optimisation, et voir si toutes les fonctions du PLA initial ont été conservées.

Cependant le problème de la vérification automatique des résultats de l'optimisation logique est posé, et l'étude, voire même la réalisation d'un tel outil serait envisageable, afin de simplifier et d'automatiser au maximum la tâche du concepteur,

tout en ayant un outil de test et de vérification fiable et pratique.

CONCLUSION

Parmi tous les outils de conception et de réalisation des circuits intégrés, l'optimiseur logique semble être un outil destiné à réduire de façon notable le travail du concepteur, donc le coût de conception du PLA. En effet, l'optimisation manuelle d'un PLA est une tâche longue et fastidieuse, et propre à créer de nombreuses erreurs. Les performances de cet outil semblent être très acceptables, car nous avons vu que l'on approche de près les résultats obtenus de façon manuelle par le concepteur.

Le but qui nous était fixé semble donc atteint, car cet optimiseur permet d'obtenir un gain notable du nombre de monôme du PLA, et fonctionne avec des temps de CPU relativement faibles, ce qui permet de recommencer l'optimisation si les spécifications changent, ou encore de l'affiner si celà est possible.

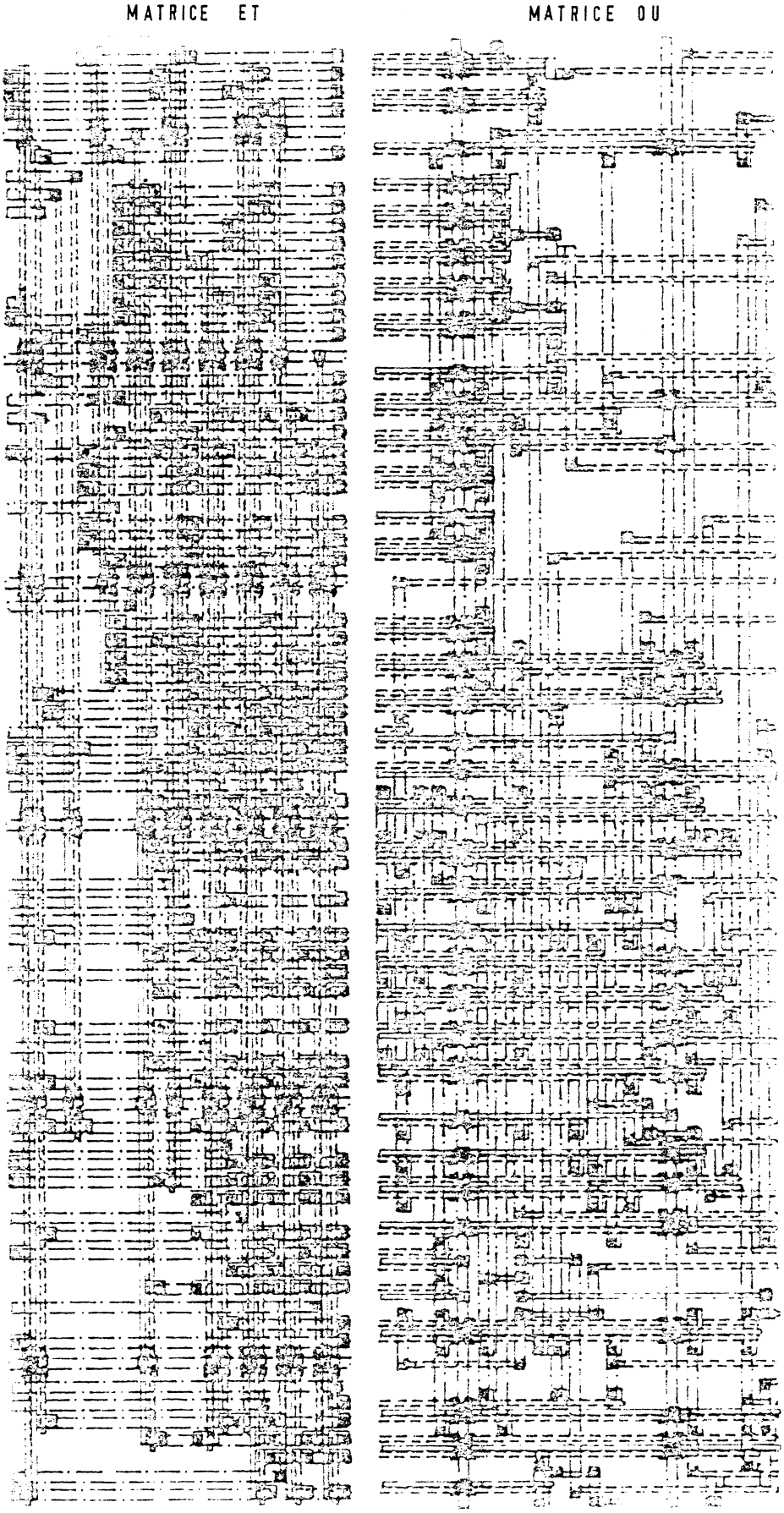
Cependant, l'idée de l'optimisation logique est étroitement liée à celle de l'optimisation topologique. Il n'est pas nécessaire, dans cette optique, d'obtenir de très bon résultats avec un optimiseur logique si ces résultats ne sont pas compatibles avec l'optimiseur topologique.

En effet, le nombre de monômes (transistors) gagnés dans une telle étape peut enlever des possibilités à l'optimiseur topologique, et le forcer ainsi à dédoubler des monômes afin de pouvoir faire son optimisation topologique ou remédier aux

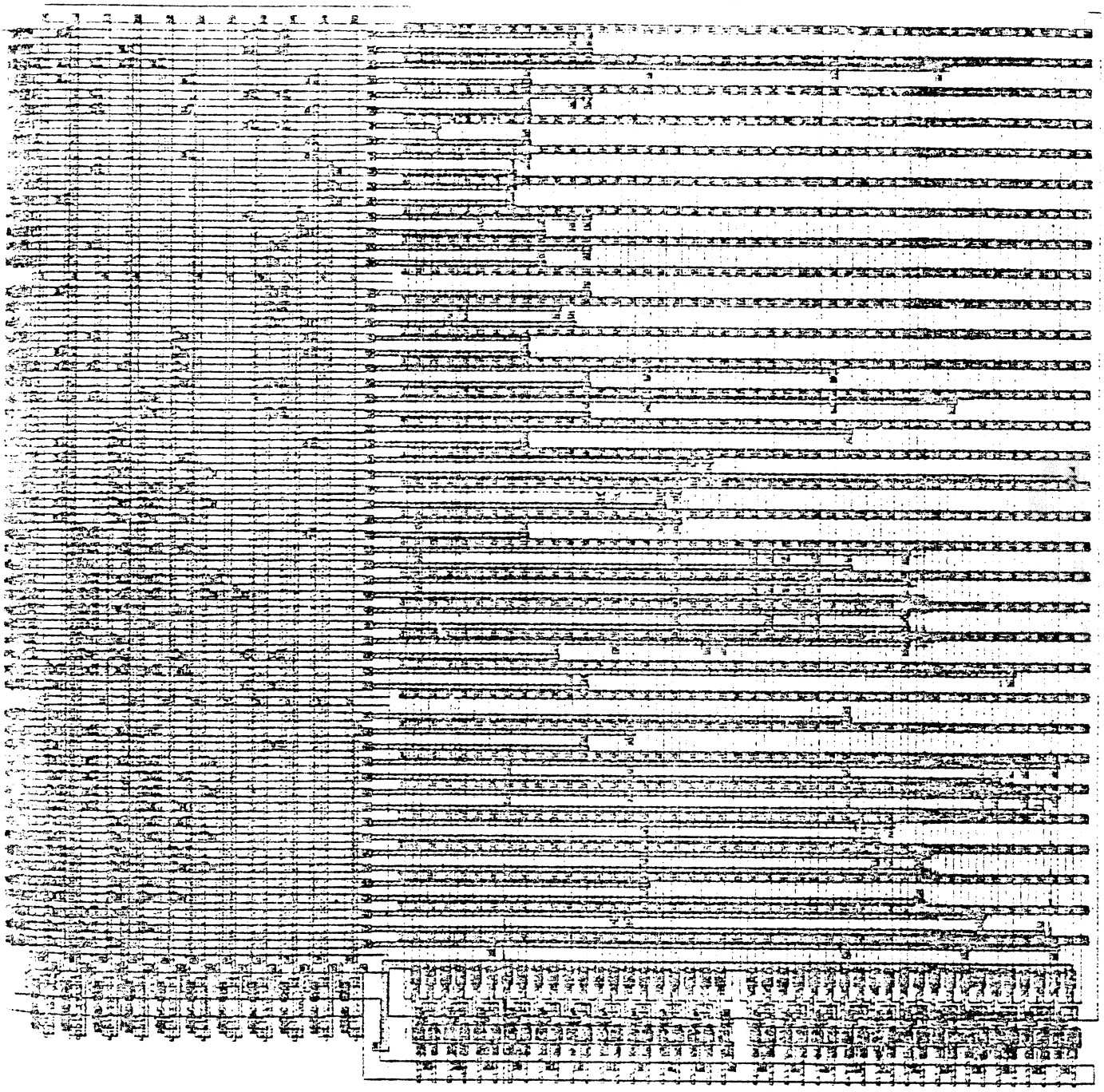
problèmes de routage interne du PLA. L'optimiseur logique sera donc une composante du système PAOLA, et non pas une entité à part entière. L'intégration de cet optimiseur logique dans le système PAOLA, et le test du système entier est à achever, afin de faire ainsi un produit homogène et performant.

Ainsi sera alors créé un nouvel outil, simple et puissant, qui marquera une étape de plus dans la réduction des coûts de conception et de production des circuits intégrés, et permettra la réalisation de circuits Ultra Haute Densité d'Intégration, tel le HP 9000 , qui existe déjà à l'heure actuelle, et qui, conçu avec de tels outils de CAO, a permis l'intégration de 500.000 composants environ sur la puce de silicium de ce micro-processeur. Alors, le million de composants sur un "chip" est-il pour demain ?

ANNEXES



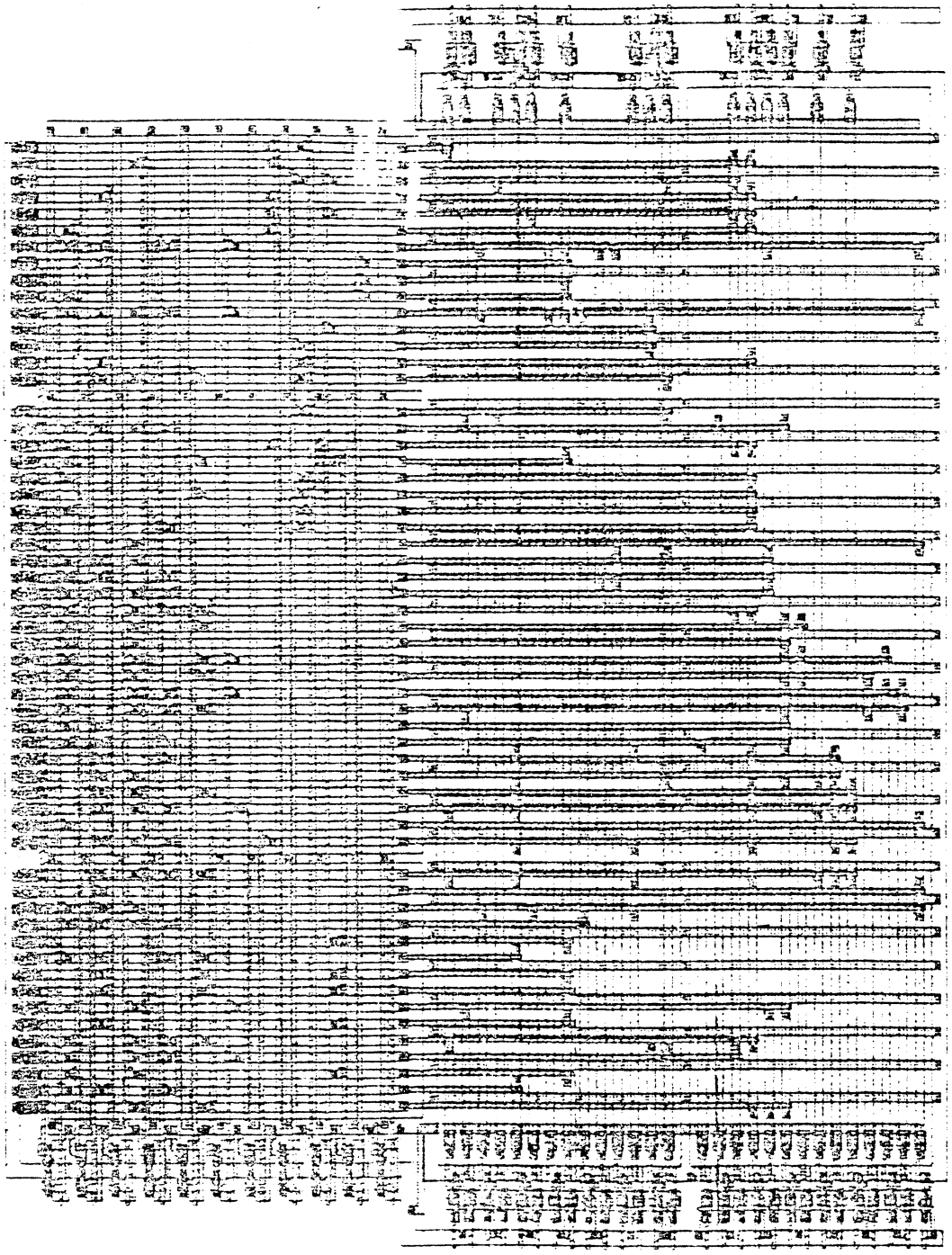
ANNEXE 1 . Dessin d'un fichier opimisé avec PAOLA.



(a)

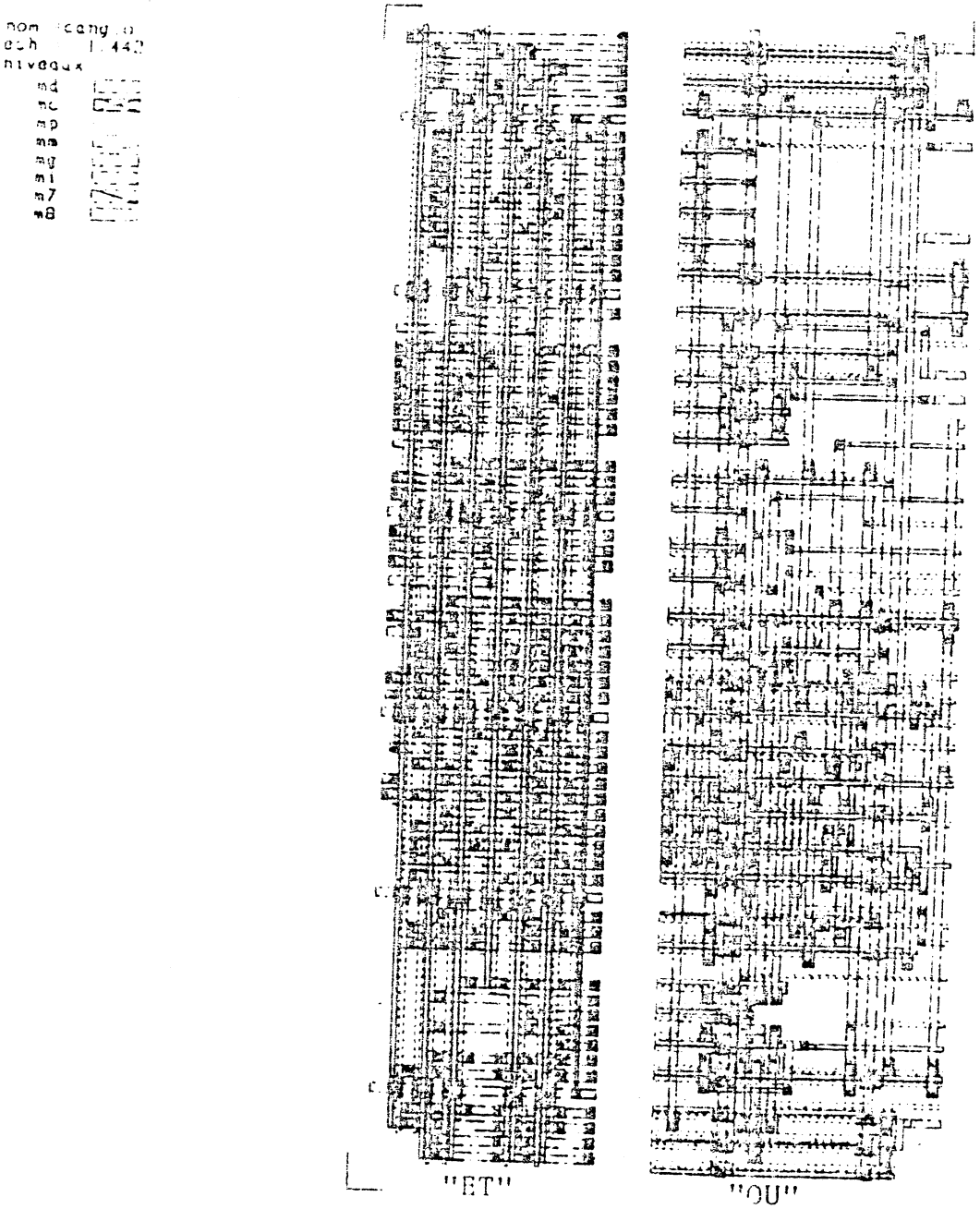
ANNEXE 2a. Optimisation topologique

PLA pris comme exemple.



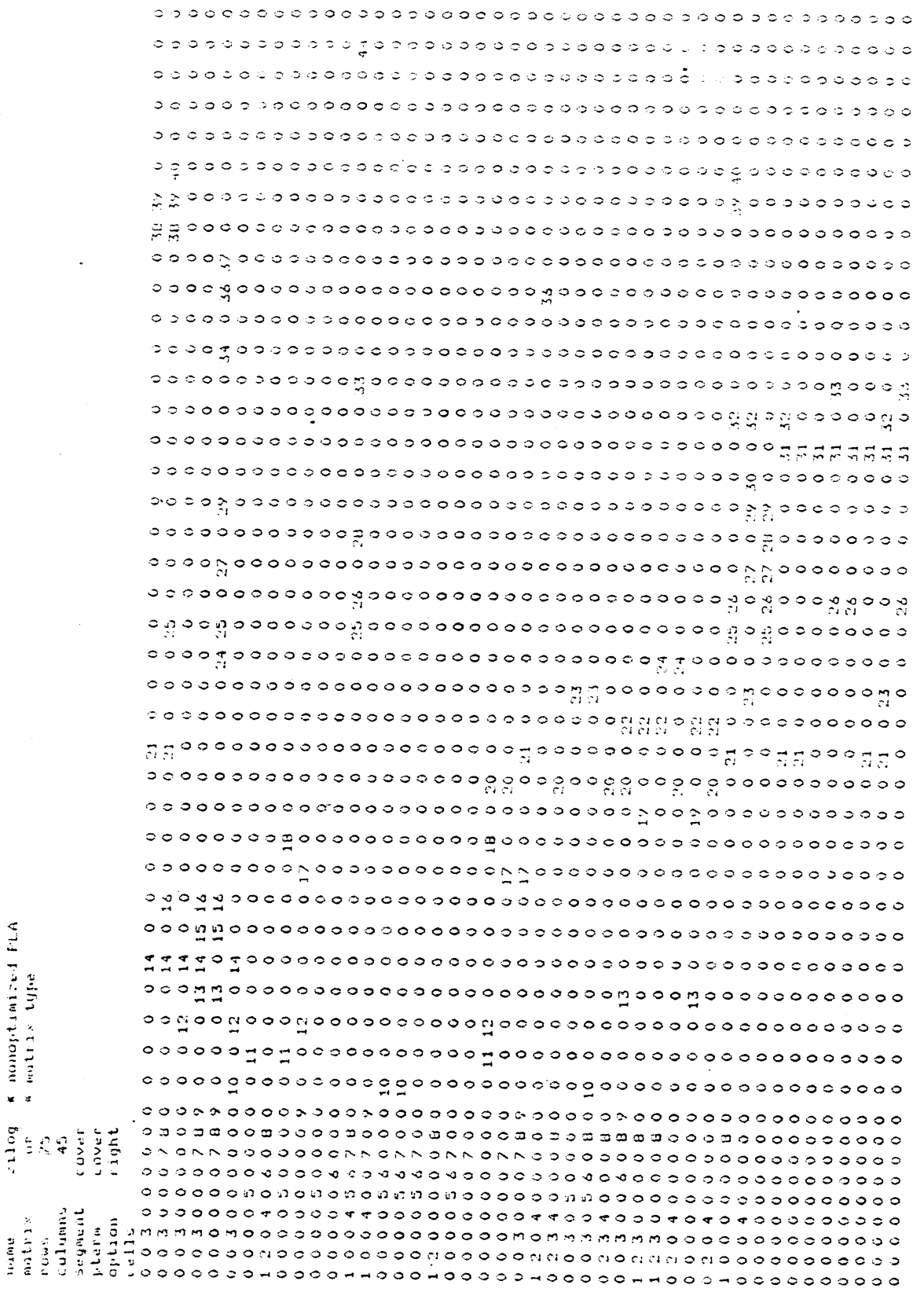
(b)

ANNEXE 2b. PLA ayant sa partie "OU" optimisée par "Folding".

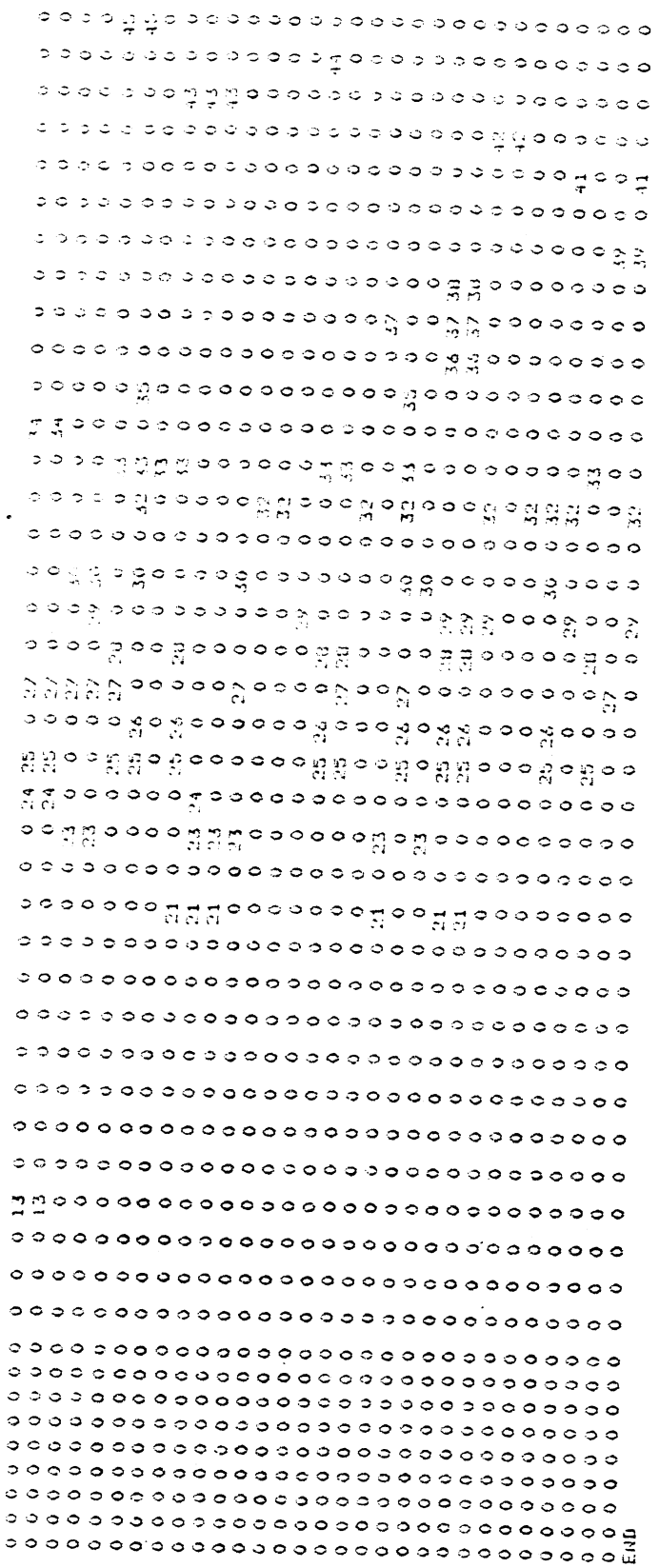


ANNEXE 2c. PLA exemple optimisé par "PAOLA".
(priorité d'optimisation matrice OU puis matrice ET)

Matrice OU non optimisée (table de bits)



Matrice OU non optimisée (table de bits (suite))



ANNEXE 3.2.

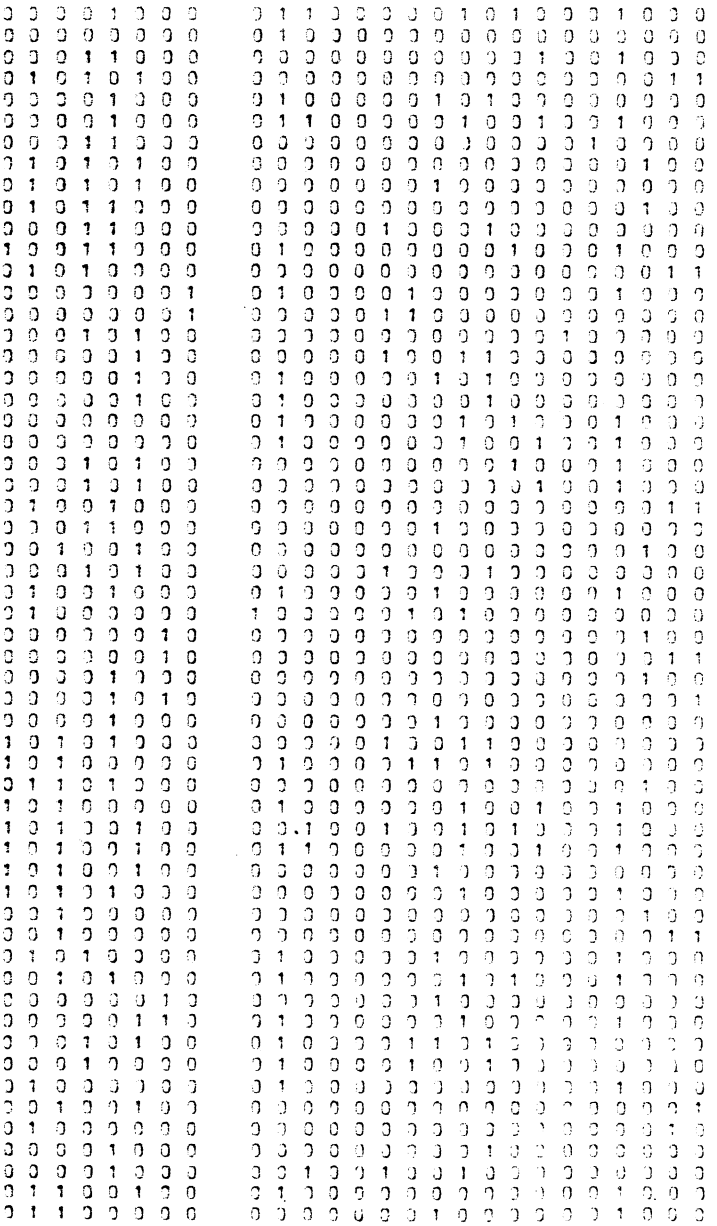
```
name       [illegible]   * optimized PLA
matrix     [illegible]   * matrix type
rows       88
columns    19
segment    narrow
pterm      left
option     right
firstcol   GNDBUS
cells
```

0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	4	0	0
0	8	0	0	0	2	3	0	0	0	0	0	0	0	1	0	0	0	0
0	8	0	0	0	2	3	0	0	0	0	0	0	0	1	0	0	0	0
0	8	0	0	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0
0	8	0	0	0	2	0	0	0	0	0	0	0	0	1	0	4	0	0
0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0	8	7	0	0	0	0	0	0	0	0	0	5	0	1	0	4	0	0
0	0	7	0	0	0	0	0	9	0	0	0	0	0	1	0	4	0	0
0	8	0	0	0	0	3	0	0	0	0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	7	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0	0	7	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	6
0	8	0	0	0	0	3	0	9	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	10	0	0	5	0	0	0	0	0	0
0	0	7	0	0	0	0	0	0	10	0	0	5	0	0	0	0	0	6
0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	8	0	20	0	2	3	0	0	0	0	0	0	0	0	0	4	0	0
0	8	0	20	0	2	3	0	0	0	0	0	0	0	0	0	4	0	0
11	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0	0	0	0	14	0	3	0	0	0	12	0	0	0	0	0	0	0	0
0	0	0	0	14	0	3	0	0	10	12	0	0	0	0	0	0	0	0
0	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	21
0	0	0	0	0	0	3	0	0	0	0	0	5	0	23	0	0	0	0
0	8	7	0	14	0	3	0	9	0	0	0	0	16	0	13	0	15	0
0	8	7	0	0	0	3	17	9	0	0	0	0	0	0	0	0	0	21
0	8	7	0	0	0	0	0	9	0	0	0	0	16	0	13	0	15	0
11	0	0	0	0	0	0	0	0	0	0	13	5	0	0	0	0	0	0
0	8	7	0	14	0	3	0	0	0	0	0	0	16	0	0	0	0	21
0	0	0	20	0	2	0	0	0	0	0	0	22	0	0	0	4	0	0
0	0	0	20	0	2	0	0	0	0	0	0	0	0	0	0	4	24	0
0	0	0	0	14	0	3	0	0	0	0	0	0	0	0	0	0	0	21
0	0	0	0	0	0	0	17	9	0	12	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	10	0	0	0	0	0	0	23	0	4	0	0
0	0	7	20	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0
11	0	0	20	0	0	0	0	0	0	12	13	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	10	0	0	0	0	23	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	22	19	0	13	0	0	0
0	0	0	20	0	0	0	0	0	0	0	0	22	0	0	13	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	22	19	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	24	0
0	0	0	0	27	0	34	0	0	0	0	0	0	25	0	13	0	24	0
0	0	0	0	27	0	34	0	0	0	0	0	0	25	0	13	0	24	0
0	0	28	26	0	0	0	0	0	0	0	0	0	25	0	0	0	0	21
0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	21
0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	21
0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	30	0	0	0
0	0	0	0	27	0	0	0	0	0	0	0	0	0	23	30	0	0	0
0	0	0	0	27	0	0	0	0	0	0	0	0	0	23	30	0	0	0
0	0	0	0	0	31	0	0	0	0	0	0	32	0	23	0	0	0	21
0	0	28	26	27	0	0	29	0	0	0	0	0	25	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	37	0	21
0	0	0	0	27	0	0	29	0	0	0	0	0	0	23	30	0	0	0
0	0	0	0	0	0	0	0	0	0	0	43	0	0	23	0	0	24	21
0	0	0	0	0	31	0	0	0	0	0	0	32	0	0	0	0	0	21
0	0	0	26	0	0	0	0	0	0	0	0	32	25	0	30	0	0	0
0	33	28	0	27	0	0	0	0	0	0	0	0	25	0	0	0	0	0
0	0	0	26	0	0	0	0	0	0	0	0	32	25	0	30	0	0	0
0	0	0	26	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0
0	33	28	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	43	0	0	23	0	0	0	21
0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	27	0	0	29	0	0	0	0	32	0	0	30	0	0	0
0	33	0	26	27	0	0	0	0	0	35	0	32	25	0	30	0	0	0
0	33	0	26	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0
0	33	0	26	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	28	26	0	0	0	29	36	38	0	0	0	25	0	0	37	0	21
0	0	28	26	0	0	0	29	36	38	0	0	0	25	0	0	37	0	21
0	0	0	0	27	0	34	29	36	0	0	0	0	25	0	0	37	24	0
0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0
0	33	28	26	0	0	0	0	0	0	0	0	0	25	0	0	0	0	44
0	33	28	26	0	0	0	0	0	0	0	0	0	25	0	0	0	0	44
0	33	28	0	27	0	0	0	0	0	0	0	0	25	0	0	45	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
0	0	0	26	0	0	0	0	0	0	0	0	32	25	0	39	0	40	0
0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	39	0	0	0
0	0	0	0	0	0	0	29	0	0	0	0	32	0	0	0	0	0	41
0	0	0	0	0	0	0	29	0	0	0	0	32	0	42	0	0	0	0
0	0	0	0	0	0	0	29	0	0	0	0	32	0	0	39	0	0	41
0	0	0	0	0	0	0	0	0	38	0	0	0	25	0	39	0	40	0
0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0	0	0	0
0	33	0	0	0	0	0	0	0	0	35	0	0	0	0	0	45	0	0
0	33	0	0	0	0	0	0	0	0	0	43	0	0	0	0	0	0	0
0	0	0	0	0	0-	0	0	0	38	0	0	0	0	0	39	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0

end

Matrice OU
optimisée par
PAOLA
(tableau de bi

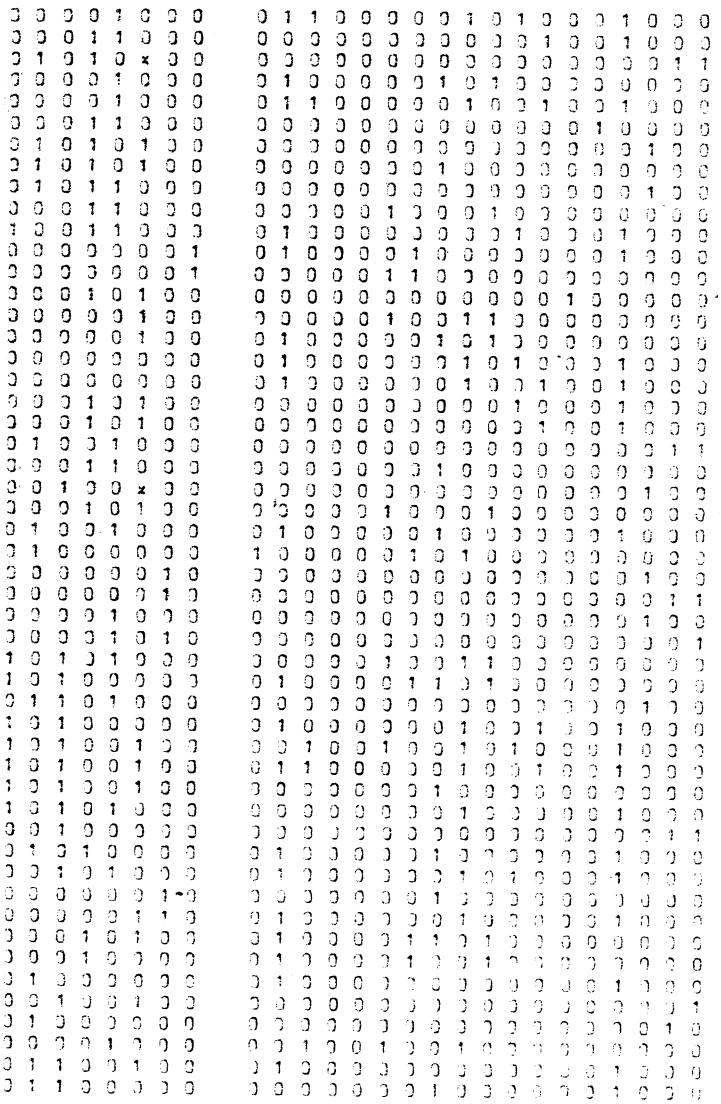
ANNEXE 3.3.



Nombre de Transistors = 537

Nombre de Monomes = 57

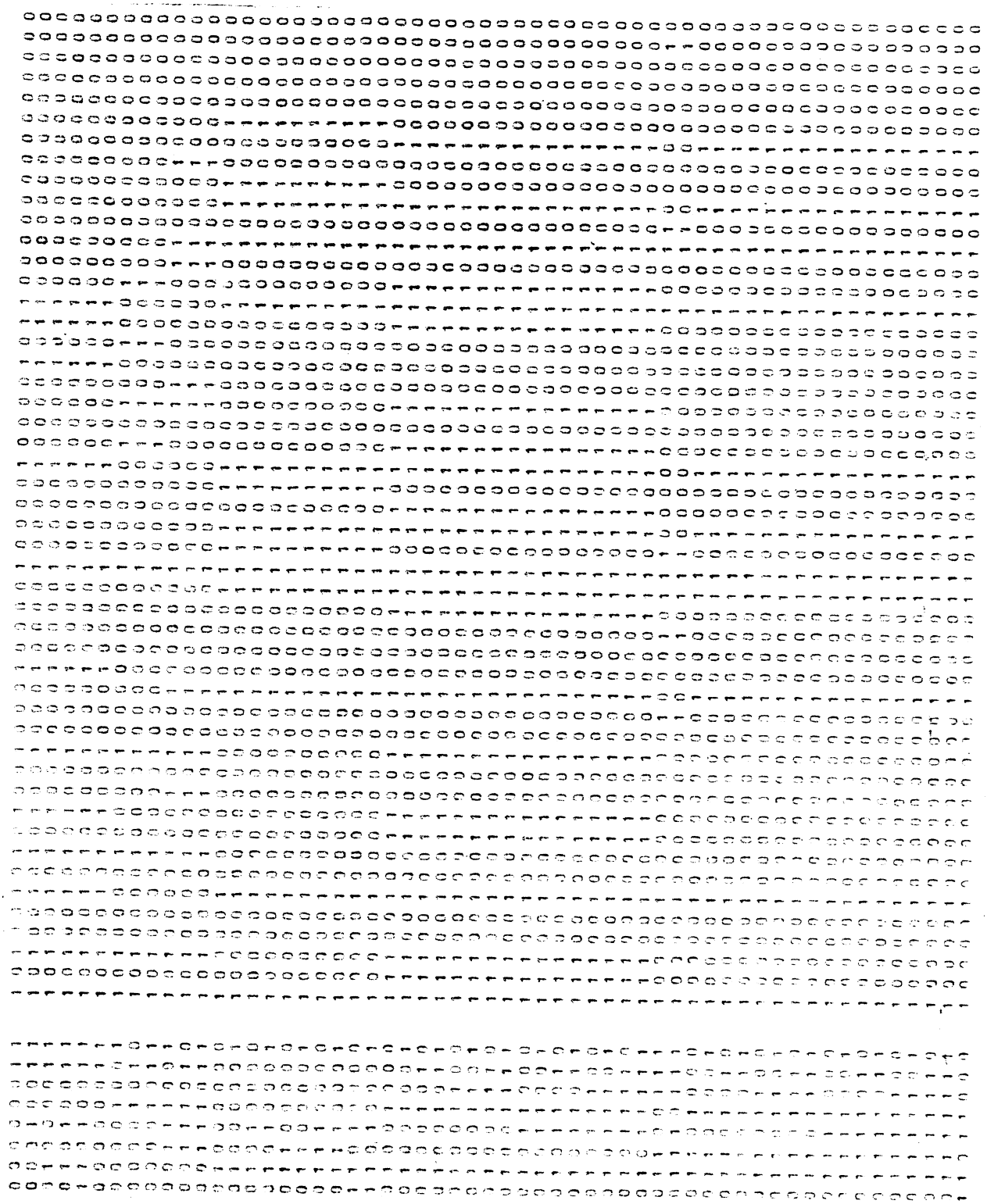
ANNEXE 4.1. PLA S3 non optimisé



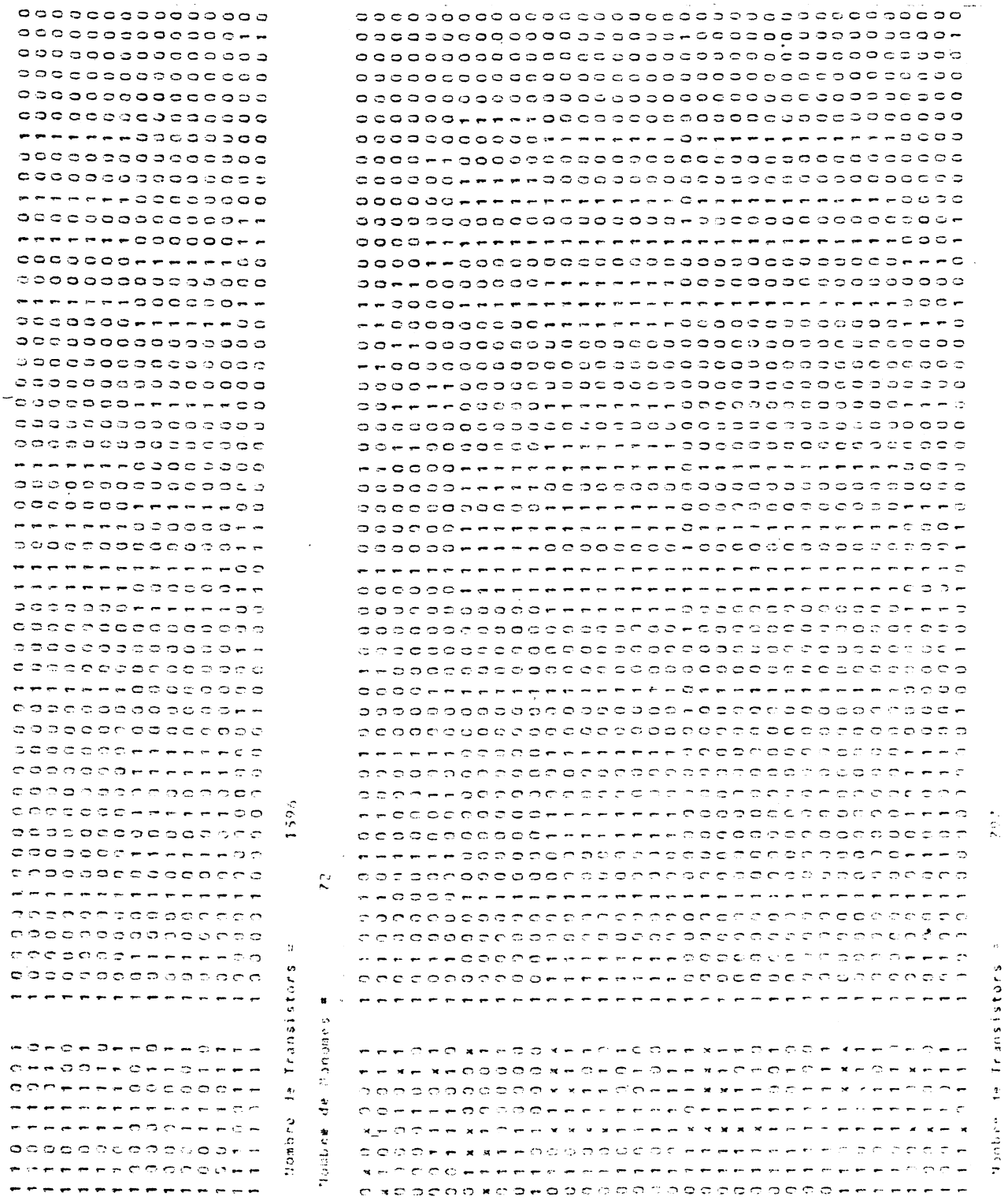
Nombre de Transistors = 531

Nombre de Monomes = 51

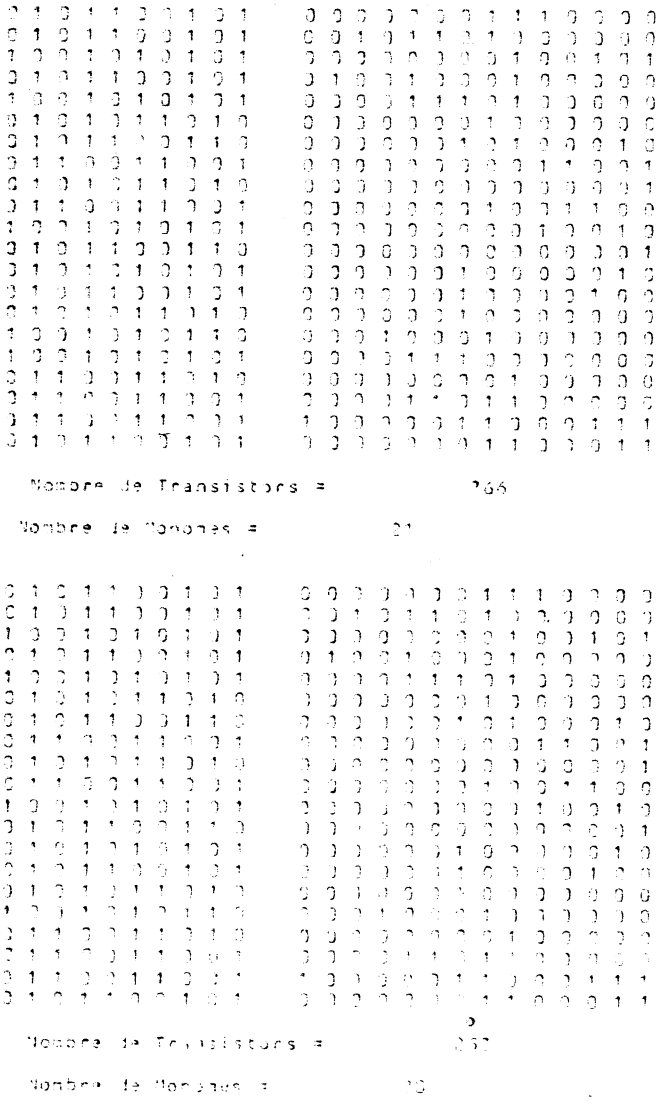
ANNEXE 4.2. PLA S3 optimisé logiquement



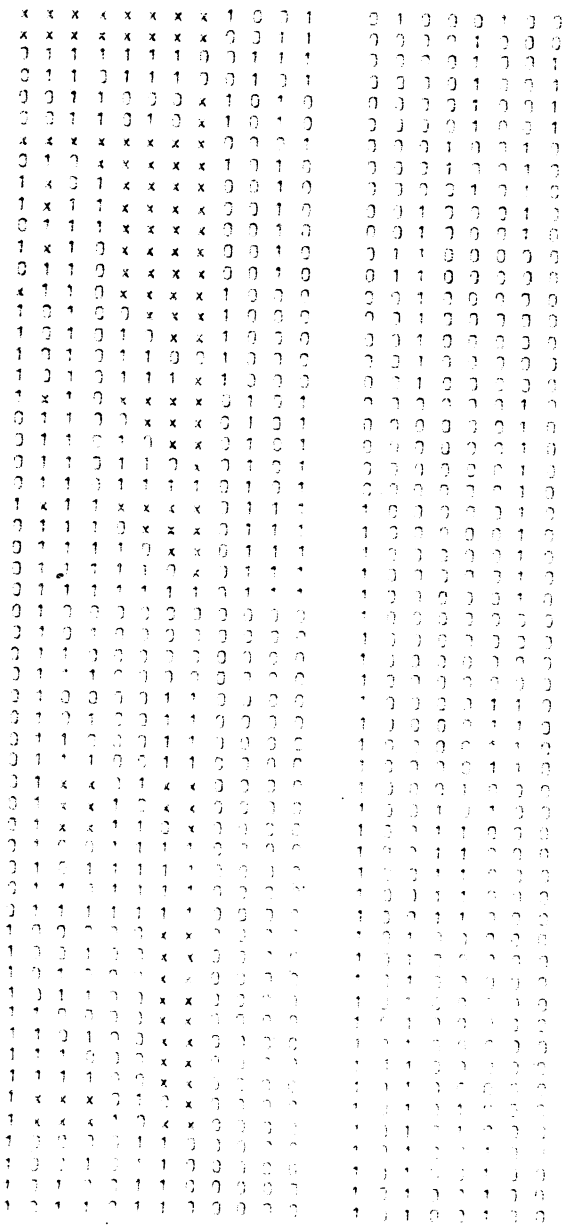
ANNEXE 5.1. PLA PLAJPS non optimisé



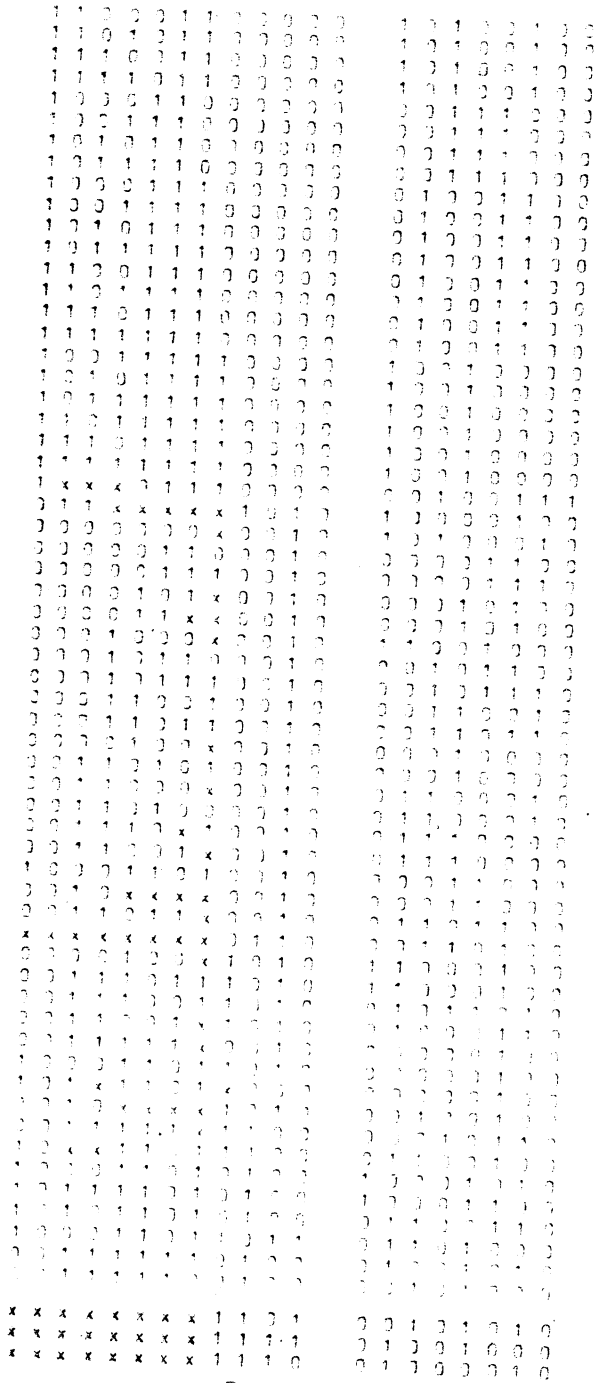
ANNEXE 5.2. PLA PLAJPS optimisé (1er passage)



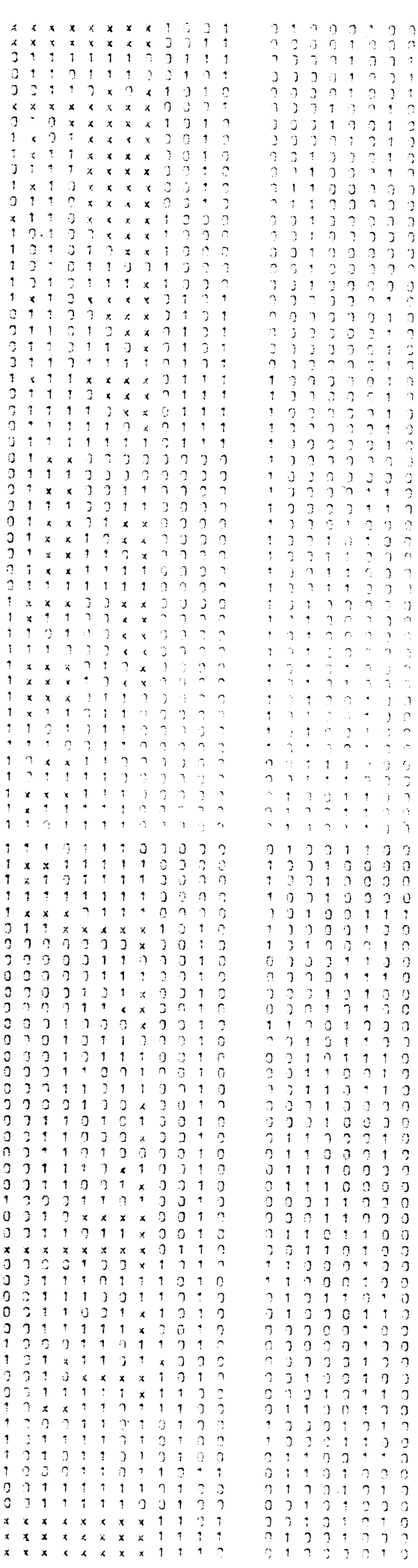
ANNEXE 6.1. Résultat d'optimisation du PLA S2.



ANNEXE 7.1. PLA PLAMON3 non optimisé (a)



ANNEXE 7.2. PLA PLAMON3 non optimisé (b)

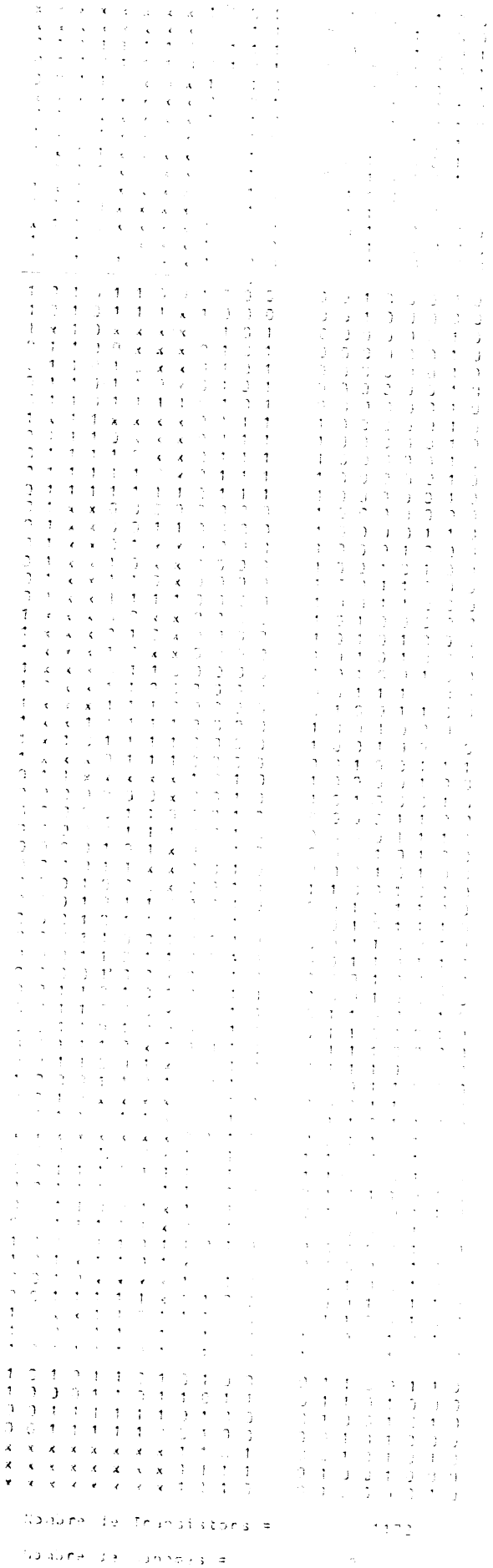


Nombre de Transistors = 1376

Nombre de Monomes = 26

PLA PLAMON3
optimisé
(1er passage)

ANNEXE 7.3.



PLA PLAMON3
optimisé
(2ème passage)

ANNEXE 7.4.

BIBLIOGRAPHIE

- <ANC-82> F. ANCEAU, R.A. REIS. "Complex Integrated Circuit Design Strategy". IEEE Journal of Solid State Circuits. Vol 17. 1982.

- <ARE-78> Z. AREVALO, J.G. BREDESON. "A method to simplify a boolean function into a near-minimal sum of products for PLA". IEEE Transactions on Computer. Vol C.27. 1978.

- <BOW-70> R.M. BOWMAN, E.S. Mc VEY. "A method for the fast approximate solution of large prime implicant charts". IEEE Transactions on Computers. Vol C.19. 1970.

- <CHU-82> S. CHUQUILLANQUI, T. PEREZ SEGOVIA. "PAOLA : a tool for topological ptimization of large PLA". Proc. of 19th DAC Las Vegas Nevada. 1982.

- <CHU-83> S. CHUQUILLANQUI, R. KRASICKI, T. PEREZ SEGOVIA. "A VLSI topological optimization strategy applied to PLA design". IEEE International Conference on C.A.D. Santa Clara California. Sept 1983.

- < DIE-69> D.L. DIETMEYER, Y.H. SU. "Computer reduction of two-level, multiple output switching circuits". IEEE Transactions on Computers. Vol C.18. 1969.

- < FLE-75> H. FLEISHER, L.I. MAISEL. "An introduction to logic array". IBM Journal on Research and Developpment. Vol. 19. 1975.

- < FLE-67> H.G. FLEGG. "L'algèbre de BOOLE et son utilisation". DUNOD. PARIS 1967.

- < FLO-64> J.F. FLORINE. "La synthèse des machines logiques". DUNOD. PARIS 1964.

- < HON-74> R.M. HONG, R.G. CAIN, D.L. OSTAPKO. "MINI : a heuristic approach for logic minimization". IBM Journal for R. & D. Sept. 1974.

- < JJM-80> J.J. MONBARON. "Description des systèmes logiques combinatoire par des fonctions booléennes caractéristiques permettant le calcul des impliquants premiers et des couvertures irrédondantes au moyen d'un algorithme unique adapté aux petits ordinateurs". Thèse de Docteur Es Sciences. Université de Neuchâtel. Suisse. 1980.

- < KAM-79> Y. KAMBAYASHI. "Logic Design of Programmable Logic Arrays". IEEE Trans. on Computers. Vol C.28. 1979.

- < KAR-53> M. KARNAUGH. "The map method for synthesis of combinational logic circuits". Trans. AIEE Vol 72. 1953.

- < MAN-67> D. MANGE. "Calculatrice spécialisée PIM 4 pour la simplification automatique des fonctions logiques". Communication du groupement pour l'étude des télécommunications de la fondation HASLER. N° 7. BERNE 1967.

- < MCL-56> E.J. Mc CLUSKEY Jr. " Minimization of Boolean functions". Bell Syst. Tech. Journal N° 35. 1956.

- < MCL-65> E.J. Mc CLUSKEY Jr. "Introduction to the theory of switching circuits". Mc GRAW-HILL Book Company. NEW-YORK. 1965.

- < MOR-67> E. MORREALE. "Partitioned-list algoritms for prime implicant determination from canonical forms". IEEE Trans. on Computers. Vol C.19. 1967.

- < MOR-70> E. MORREALE. "Recursive operators for prime implicant and irredundant normal form determination". IEEE Trans. on Computers. Vol C.19. 1970.

- < MUR-76> S. MUROGA, H. CHI-LAY. "minimization of logic networks under a generalized cost function". IEEE Trans. on Computers. Vol C.25. 1976.

- < MIL-65> R.E. MILLER. "Switching Theory". Vol 1; "Combinational Circuits". J. WILEY and sons Incorporated. NEW-YORK. 1965.

- < PAR-60> P.P. PARKHOMENKO. "Machine analysis of switching circuits". Automation and Remote Control. Vol 20. 1960.

- < PER-79> T. PEREZ SEGOVIA. "Etude de la conception et de la minimisation du PLA". Rapport de DEA. 1979.

- < PER-80> T. PEREZ SEGOVIA. "Optimisation topologique des PLA". Rapport de Recherche N° 216. IMAG GRENOBLE 1980.

- < QUI-52> W.V. QUINE. "The problem of simplifying truth functions". American Mathematics monthly N° 59. 1952.

- < QUI-55> W.V. QUINE. "A way to simplify truth functions". American Mathematics monthly N° 62. 1955.

- < ROT-78> J.P. ROTH. "Programmable Logic Array Optimization". IEEE Trans. on Computers. Vol C.27. 1978.

- < ROT-80> J.P. ROTH. "Computer Logic, Testing and Verification". Computer S cience Press Inc. POTOMAC. MARYLAND. 1980.

- < SAM-54> E.W. SAMSON, B.E. MILLS. "Circuit minimization: Algebra and Algorithms for new boolean canonical expressions". AFRC Technical Report N° 5421. 1954.

- < SLA-70> J.R. SLAGLE, C.L. CHANG, R.C. LEE. "A new algoritm for generating prime implicants". IEEE Trans. on Computers. Vol C.19. 1970.

- < TEE-82> B. TEEL, D. WILDE. "A logic minimizer for VLSI PLA design". Proc. of 19th D.A.C. Report. Las Vegas. NEVADA. 1982.

- < TIS-65> P. TISON. "Théorie des Consensus". Thèse de Docteur-Ingénieur. Faculté des Sciences. Université de GRENOBLE. 1965.

- < YAC-80> YACOUB EL-ZIQ, PRAKASH RAO. Honeywell Alds Research Report N° 1. 1980.