Synthèse logique à base de règles pour les compilateurs de silicium

Stéphane Hanriat

THESE

Présentée à

L'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR de l'INPG
spécialité "Informatique"

par

Stéphane HANRIAT

♠

SYNTHESE LOGIQUE A BASE DE REGLES POUR LES COMPILATEURS DE SILICIUM

♣

Thèse soutenue le 29 septembre 1986 devant la commission d'examen.

D. ETIEMBLE Président

V. C. HAMACHER
P. BAYLE Examinateurs
J. FREHEL

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année universitaire 1985-1986

Vice-Présidents: B. BAUDELET
R. CARRE
H. CHERADAME
J.M PIERRARD

Professeurs des Universités

BARIBAUD	Michel	ENSERG
BARRAUD	Alain	ENSIEG
BAUDELET	Bernard	ENSIEG
BEAUFILS	Jean-Claude	ENSEEG
BESSON	Jean	ENSEEG
BLIMAN	Samuel	ENSERG
BLOCH	Daniel	ENSIEG
BOIS	Philippe	ENSHG
BONNETAIN	Lucien	ENSEEG
BONNIER	Etienne	ENSEEG
BOUVARD	Maurice	ENSHG
BRISSONNEAU	Pierre	ENSIEG
BRUNET	Yves	ENSIEG
BUYLE-BODIN	Maurice	ENSERG
CAILLERIE	Denis	ENSHG
CAVAIGNAC	Jean-François	ENSIEG
CHARTIER	Germain	ENSIEG
CHENEVIER	Pierre	ENSERG
CHERADAME	Hervé	UERMCPP
CHERUY	Arlette	ENSIEG
CHIAVERINA	Jean	UERMCPP
COHEN	Joseph	ENSERG
COUMES	André	ENSERG
DURAND	Francis	ENSEEG
DURAND	Jean-Louis	ENSIEG
FONLUPT	Jean	ENSIMAG
FOULARD	Claude	ENSIEG
GANDINI	Alessandro	UERMCPP
GAUBERT	Claude	ENSIEG
GENTIL	Pierre	ENSERG
GUERIN	Bernard	ENSERG
GUYOT	Pierre	ENSEEG
IVANES	Marcel	ENSIEG
JAUSSAUD	Pierre	ENSIEG
JOUBERT	Jean-Claude	ENSIEG
JOURDAIN	Geneviève	ENSIEG
LACOUME	Jean-Louis	ENSIEG
LESIEUR	Marcel	ENSHG
LESPINARD	Georges	ENSHG
LONGEQUEUE	Jean-Pierre	ENSIEG
LOUCHET	François	ENSEEG
MASSELOT	Christian	ENSIEG
MAZARE	Guy	ENSIMAG
MOREAU	René	ENSHG
MORET	Roger	ENSIEG
MOSSIERE	Jacques	ENSIMAG
OBLED	Charles	ENSHG
PARIAUD	Jean-Charles	ENSEEG
PAUTHENET	René	ENSIEG
PERRET	René	ENSIEG
PERRET	Robert	ENSIEG
PIAU	Jean-Michel	ENSHG
POLOUJADOFF	Michel	ENSIEG
POUPOT	Christian	ENSERG
RAMEAU	Jean-Jacques	ENSEEG
RENAUD	Maurice	UERMCPP
ROBERT	André	UERMCPP
ROBERT	François	ENSIMAG
SABONNADIERE	Jean-Claude	ENSIEG
SAUCIER	Gabrielle	ENSIMAG
SCHLENKER	Claire	ENSIEG
SCHLENKER	Michel	ENSIEG
SERMET	Pierre	ENSERG
SILVY	Jacques	UERMCPP
SOHM	Jean-Claude	ENSEEG
SOUQUET	Jean-Louis	ENSEEG
TROMPETTE	Philippe	ENSHG
VEILLON	Gérard	ENSIMAG

Professeurs Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S

CARRE	René	Directeur de recherche
CAILLET	Marcel	"
FRUCHART	Robert	"
JORRAND	Philippe	"
LANDAU	Ioan	"
ALLIBERT	Colette	Maître de recherche
ALLIBERT	Michel	"
ANSARA	Ibrahim	"
ARMAND	Michel	"
BINDER	Gilbert	"
BONNET	Roland	"
BORNARD	Guy	"
CALMET	Jacques	"
DAVID	René	Maître de recherche
DEPORTES	Jacques	"
DRIOLE	Jean	"
EUSTATHOPOULOS	Nicolas	"
GIVORD	Dominique	"
JOUD	Jean-Charles	"
KAMARINOS	Georges	"
KLEITZ	Michel	"
LEJEUNE	Gérard	"
MERMET	Jean	"
MUNIER	Jacques	"
SENATEUR	Jean-Pierre	"
SUERY	Michel	"
WACK	Bernard	"

Personnalités agrées à titre permanent à diriger des travaux de recherche (Décision du conseil scientifique)

E.N.S.E.E.G

BERNARD	Claude	FOSTER	Panayotis	RAVAINE	Denis
CAILLET	Marcel	GALERIE	Alain	SAINFORT	Paul
CHATILLON	Catherine	HAMMOU	Abdelkader	SARRAZIN	Pierre
CHATILLON	Christian	MALMEJAC	Yves	SIMON	Jean-Paul
COULON	Michel	MARTIN GARIN	Régina	TOUZAIN	Philippe
DIARD	Jean-Paul	NGUYEN TRUONG	Bernadette	URBAIN	Georges

E.N.S.E.R.G

BOREL	Joseph	DOLMAZON	Jean-Marc
CHOVET	Alain	HERAULT	Jeanny

E.N.S.I.E.G

BORNARD	Guy	KOFMAN	Walter	MAZUER	Jean
DESCHIZEAUX	Pierre	LEJEUNE	Gérard	PERARD	Jacques
GLANGEAUD	François			REINISCH	Raymond

E.N.S.H.G

ALEMANY	Antoine	DARVE	Félix	ROWE	Alain
BOIS	Daniel	MICHEL	Jean-Marie	VAUCLIN	Michel

E.N.S.I.M.A.G

BERT	Didier	COURTIN	Jacques	FONLUPT	Jean
CALMET	Jacques	COURTOIS	Bernard	SIFAKIS	Joseph
		DELLA DORA	Jean		

U.E.R.M.C.P.P

CHARUEL	Robert

C.E.N.G

CADET	Jean	JOUVE	Hubert	PERROUD	Paul
COEURE	Philippe	NICOLAU	Yvan	PEUZIN	Jean-Claude
DELHAYE	Jean-Marc	NIFENECKER	Hervé	TAIEB	Maurice
DUPUY	Michel			VINCENDON	Marc

Laboratoires extérieurs :

C.N.E.T

DEMOULIN	Eric	GERBER	Roland	MERCKEL	Gérard
DEVINE				PAULEAU	Yves

.....

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

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

Professeurs de 1ère Catégorie

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

Professeurs de 2ème catégorie

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

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

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

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

.....

A qui le veut...

Remerciements

Je tiens à exprimer toute ma reconnaissance à Madame **Gabrièle Saucier**, Professeur à l'ENSIMAG, pour m'avoir accueilli dans son laboratoire de recherche, et pour avoir encadré mon travail et diriger mes recherches avec compétence tout au long de ces deux dernières années.

Je tiens à remercier

Monsieur **Daniel ETIEMBLE**, Professeur à l'Université Pierre et Marie Curie, d'avoir accepter d'être rapporteur de cette thèse et de me faire l'honneur d'en présider le jury,

Monsieur **Pierre Bayle**, Responsable du calcul scientifique à l'Electronique Serge Dassault, ainsi que,

Monsieur **Jean Frehel**, Responsable de la CAO THOMSON CSF à Saint Egrève,
d'avoir accepté de faire partie de ce jury.

Je remercie tous particulièrement Monsieur V. **Carl Hamacher**, Professeur à l'Université de Toronto, pour avoir étudier le texte de cette thèse en français, et accepter d'en être également le rapporteur.

Que tous mes collègues du Laboratoire **Circuits et Systèmes** trouvent ici l'expression de ma profonde sympathie pour les remarques constructives dont ils m'ont fait profiter, et pour le climat agréable qu'ils ont su faire régner au sein de l'équipe.

Je remercie également Mac Intosh pour les nombreuses heures qu'il m'a permis de passer en sa compagnie, ainsi que le service de reprographie de l'IMAG pour la prise en charge du tirage de cette thèse.

SYNTHESE LOGIQUE

A BASE DE REGLES

POUR LES COMPILATEURS DE SILICIUM

SOMMAIRE

INTRODUCTION :

La conception assistée par ordinateur de circuits intégrés complexes a pris un essor magistral depuis quelques années. Ceci n'est pas dû au hasard, mais correspond bien à la nécessité de rabaisser les coûts de production en concevant vite et bien, de façon sûre et systématique. L'augmentation de la fiabilité dans la conception d'un circuit intégré passe forcément par la réduction de l'intervention humaine, et donc par l'utilisation d'outils logiciels offrant aux concepteurs une assistance efficace. Le facteur de rapidité est également primordial. Certains secteurs industriels font de plus en plus usage de circuits spécialisés pour des applications spécifiques (ASIC), et il devient impératif de pouvoir concevoir ces circuits rapidement et au moindre coût.

Dans cette étude, nous abordons un domaine très actuel de la conception assistée par ordinateur de circuits intégrés, celui de la compilation de silicium. La compilation de silicium est un environnement d'outils logiciels permettant d'obtenir de façon quasi automatique le dessin des masques d'un circuit à partir de descriptions structurelles ou fonctionnelles. Dans cet environnement, nous distinguons deux types d'outils : les assembleurs et les compilateurs de silicium.

Les assembleurs de silicium :

Les assembleurs de silicium sont des outils de génération automatique de dessin de circuits décrits comme un assemblage hiérarchique de blocs. Un bloc est soit un assemblage d'autres blocs, soit une structure régulière (PLA, RAM, ROM, registre, etc...), soit encore une cellule de base localement optimisée appartenant à une bibliothèque de cellules propre à une technologie donnée. L'assemblage hiérarchique de blocs est décrit de façon structurelle (figure 1) dans un langage adapté [LIM82] [MON85]. Les avantages liés à l'utilisation des assembleurs de silicium sont nombreux : le concepteur décrit et manipule les blocs de son circuit à l'aide de procédures dont l'exécution engendre la géométrie et le dessin du circuit. Il bénéficie alors des possibilités des langages de programmation de haut niveau :

- Manipulation de descriptions condensées donc facilement modifiables.
- Paramétrisation des procédures permettant d'engendrer des représentations différentes d'un circuit à partir d'une même description paramétrée différemment.

L'utilisation des assembleurs de silicium permet d'engendrer des circuits complexes à structure régulière, pour lesquels le facteur de régularité (nombre total de transistors divisé par le nombre de transistors effectivement dessinés) est très élevé.

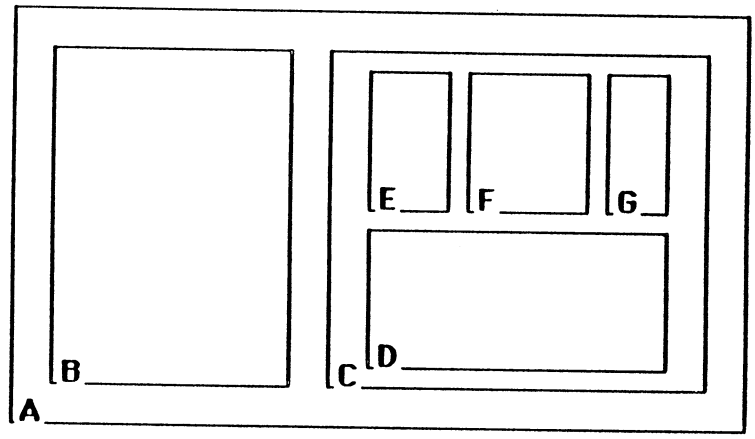


Figure 1 : Description structurelle hiérarchique

Dans un langage d'assemblage, un circuit est décrit en spécifiant sa structure, c'est à dire ses sous-circuits et leurs interconnexions. Il est donc nécessaire de disposer d'un ensemble d'outils qui s'articulent autour du langage de description structurelle, assurant les fonctions de génération du plan de masse, placement et routage sur la structure hôte (structure sans contraintes -full custom-, réseaux prédiffusés).

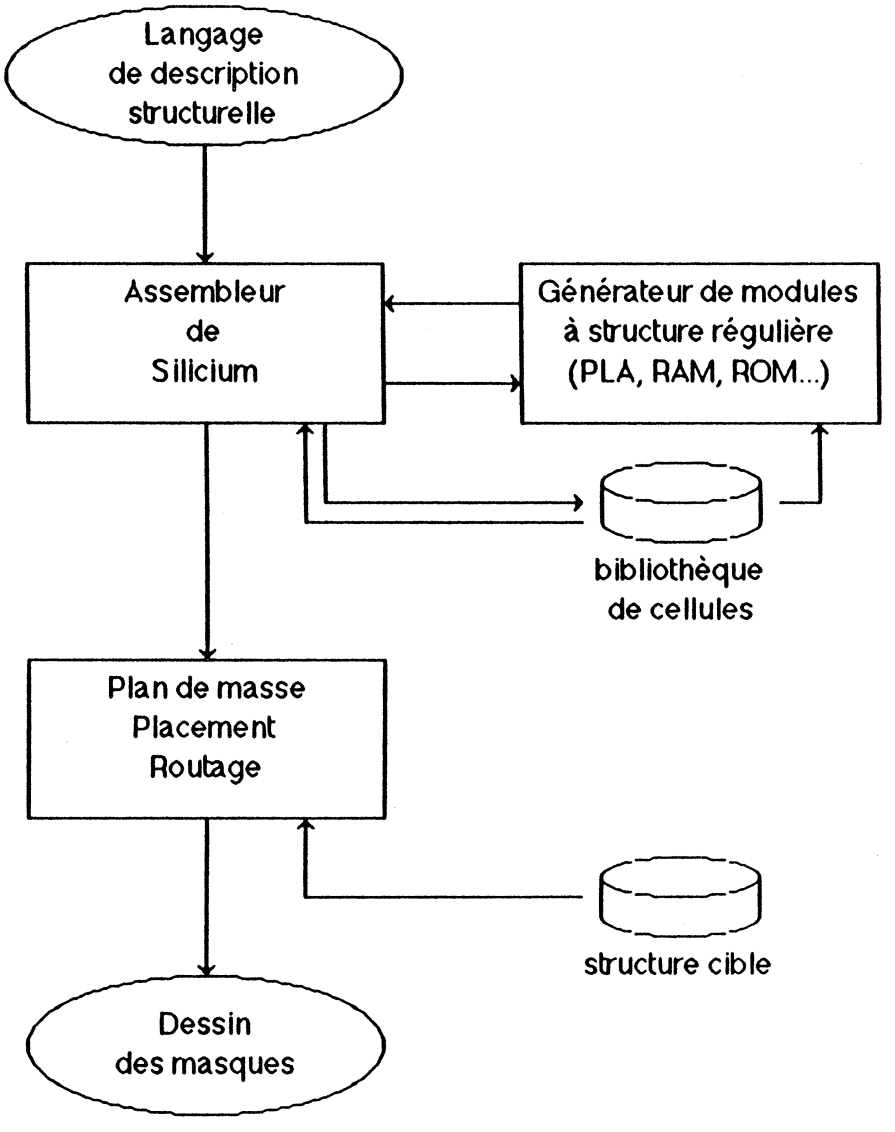


Figure 2 : environnement d'un assembleur de silicium.

Les compilateurs de silicium :

La description d'un circuit dans le contexte des assembleurs de silicium est constituée de procédures d'assemblage, chaque procédure correspondant à un bloc structurel du circuit. L'aspect fonctionnel du circuit n'est donc jamais pris en compte dans les descriptions, et les assembleurs de silicium n'ont donc aucun effort de synthèse à faire pour engendrer le dessin d'un circuit.

Les compilateurs de silicium sont des outils logiciels qui opèrent sur des descriptions fonctionnelles d'un circuit [SOU83] [BLA85]. Le compilateur de silicium engendre la structure d'un circuit à partir de son comportement, selon des modèles d'architecture donnés. Il applique des méthodes de synthèse à partir de descriptions comportementales. L'environnement de la compilation de silicium convient aux diverses méthodologies de conception (conception ascendante ou descendante) en offrant au concepteur un contrôle permanent à chacune des étapes de la synthèse. La sûreté de conception est également accrue, le concepteur étant libéré du risque d'erreur dû au traitement manuel d'un grand nombre de tâches fastidieuses. L'intérêt primordial des techniques de compilation de silicium réside dans l'utilisation de langages de description comportementale pour spécifier les circuits. On peut ainsi bénéficier, en plus des outils de synthèse, d'outils qui s'articulent autour de ces langages de haut niveau (figure 3) tels que les générateurs de tests fonctionnels, les outils de simulation comportementale, d'aide au diagnostic, etc... Le concepteur peut donc valider le comportement de son circuit avant d'en effectuer la synthèse.

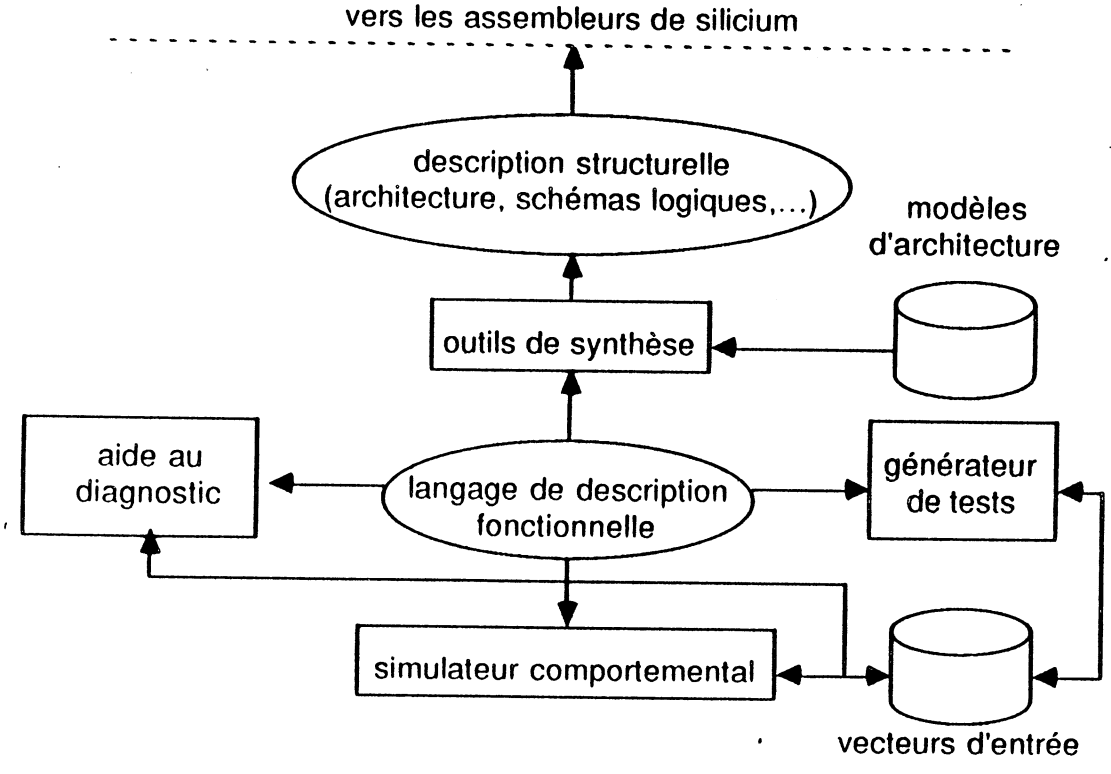


Figure 3 : Environnement d'un compilateur de silicium.

Les outils de synthèse dans la compilation de silicium :

Les principaux outils de synthèse peuvent se classer de la manière suivante :

- Les outils de synthèse combinatoire.
- Les outils de synthèse séquentielle (synthèse de partie contrôle).
- Les outils de synthèse de partie opérative.

Les outils de synthèse combinatoire :

Ces outils effectuent la synthèse logique d'un ensemble de fonctions Booléennes. Le principal problème que doivent résoudre les outils de synthèse combinatoire est celui de la minimisation multi-critères, c'est à dire la minimisation dont les critères d'optimalité sont déterminés par la structure cible (logique aléatoire, portes complexes, réseaux prédiffusés, PLA, ROM, ...).

Les outils de synthèse de contrôleur :

Ces outils synthétisent un contrôleur (partie contrôle de microprocesseur ou de circuit spécifique) à partir de sa description fonctionnelle qui est :

- soit une description du jeu d'instruction de la machine à contrôler exprimée à un niveau de transfert de registres. Ce type de description convient plutôt à une synthèse en microprogrammation, pour laquelle chaque instruction est interprétable par une séquence de microinstructions [NAG82].

-soit une description du graphe de contrôle en vue d'une synthèse dite cablée, pour laquelle chaque état du graphe est physiquement représenté au niveau de la machine implantée [BRO81] [PAP85].

Les outils de synthèse de partie opérative :

Partant de la description algorithmique (opérations arithmétiques et de transfert entre registres) qui caractérise le fonctionnement d'un circuit [HAF82] [KOW85], ces outils élaborent un schéma logique de la partie opérative selon certains modèles architecturaux. La synthèse est réalisée en extrayant dans la description les chemins de données, ainsi que le degré de parallélisme, c'est à dire les actions qui peuvent être exécutées pendant un même cycle de la machine [THO83] [JAM85].

L'objet de cette thèse est l'étude des outils de synthèse intégrables dans l'environnement des compilateurs de silicium. Nous nous intéressons aux outils de synthèse de circuits combinatoires et de contrôleurs. Notre approche s'inspire des récents progrès dans le domaine d'application de l'intelligence artificielle, en particulier celui de la conception de circuits VLSI [KOW85]. Cette approche se caractérise par la prise en compte de l'expérience des concepteurs au sens large, des "styles" de conception, des contraintes réalistes imposées par la technologie. On ne recherche plus la solution "optimale" dans un contexte donné, mais une solution viable selon les vœux d'un concepteur.

Les recherches sur la compilation de silicium au sein du laboratoire *Circuits et Systèmes* s'articulent autour du langage de description et de simulation fonctionnelles du système CADOC [CRA85]. Les outils de synthèse sont vus comme des serveurs du système CADOC, apportant une assistance "intelligente" aux concepteurs, aux différentes étapes de la conception. Les deux serveurs présentés dans cette thèse réalisent la synthèse de circuits combinatoires et de contrôleurs sur des cibles variées (PLA, ROM, logique aléatoire, portes complexes). L'accent étant mis sur la diversité des cibles et donc des critères d'optimisation, une approche à base de règles a été recherchée. Ceci permet d'associer à chaque structure cible un jeu de règles, prenant en compte l'enrichissement de l'expérience des concepteurs.

La première partie de la thèse concerne la synthèse combinatoire. Les principales structures cibles sont présentées ainsi que les critères d'optimisation qui en dépendent. Le problème de la minimisation logique d'un ensemble de fonctions Booléennes est traité en détail, et un système de règles pour la minimisation relative à une implantation sur PLA est proposé.

La seconde partie est consacrée à la synthèse de contrôleur. Les problèmes de modélisation et de conformité sont abordés, et une méthode de synthèse paramétrable par le type des points de mémorisation est décrite.

Un logiciel d'**A**ssistance à la **SY**nthèse **L**ogique (ASYL) a été écrit en PASCAL (15000 lignes) et est opérationnel sur les VAX et microVAX sous le système d'exploitation VMS. Ce système constitue une application pratique des principes de synthèse combinatoire et séquentielle exposés dans cette thèse.

PARTIE I

SYNTHESE AUTOMATIQUE
A BASE DE REGLES
POUR LA LOGIQUE COMBINATOIRE

I.1. RAPPELS SUR LA LOGIQUE COMBINATOIRE

Un circuit combinatoire se caractérise par la connaissance de sa fonction d'entrées/sorties, c'est à dire l'ensemble des fonctions Booléennes [KUN65] qui définissent les valeurs des sorties du circuit pour toutes les combinaisons possibles de ses entrées (figure 4). Une fonction Booléenne à n entrées doit donc être définie pour les 2^n états possibles de ses entrées.

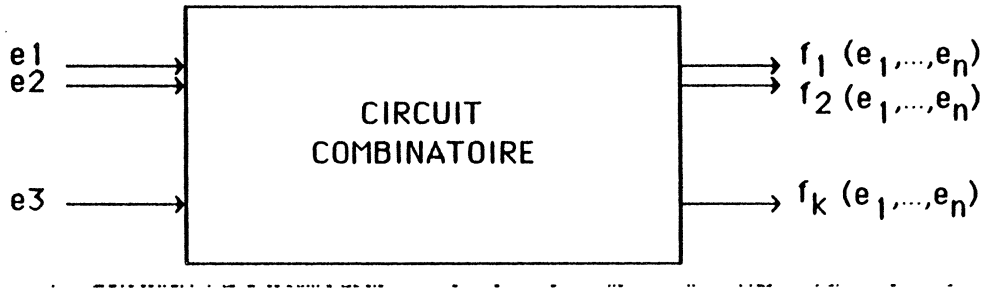


Figure 4 : Circuit combinatoire.

En raison des différences de terminologie qui apparaissent dans la littérature, nous allons rappeler les principes et les définitions qui sont à la base de tout ce qui va suivre.

Une fonction Booléenne peut prendre ses valeurs dans l'ensemble (0,1,ø). La pseudo valeur ø signifie 1 ou 0 indifféremment.

Définition 1.
Une fonction Booléenne qui prend ses valeurs dans (0,1) est dite complète ou complètement spécifiée.
Définition 2.
Une fonction Booléenne qui prend ses valeurs dans (0,1,ø) est dite phi-booléenne, ou incomplète ou encore incomplètement spécifiée.

Il est à noter que les circuits combinatoires ne réalisent physiquement que des

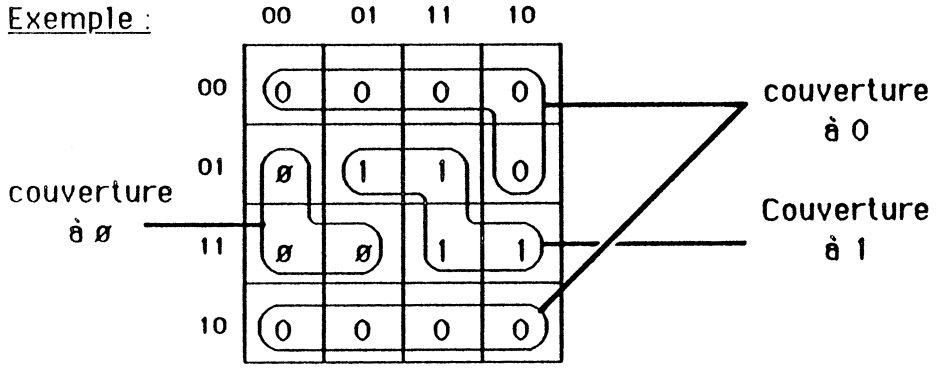
fonctions complètement spécifiées. L'usage de la pseudo valeur ø ne peut intervenir qu'au niveau de la spécification.

Définition 3.

La **couverture** à 0, à 1, ou à ø d'une fonction Booléenne est l'ensemble des combinaisons des variables d'entrée pour lesquelles la fonction vaut 0, 1, ou ø. La couverture à ø est également appelée couverture indifférente.

Une fonction Booléenne complète peut être décrite par l'une de ses couvertures à 1 ou à 0, l'autre étant comprise comme le complément de l'une. Une fonction incomplète peut être décrite par deux parmi ses trois couvertures à 0, à 1, et à ø, la troisième couverture est alors le complément de l'union des deux autres.

Exemple :



Toute couverture d'une fonction Booléenne peut être exprimée par une expression Booléenne construite à partir des variables de la fonction et d'un certains nombre d'opérateurs appelés opérateurs logiques. Les opérateurs binaires les plus fréquemment utilisés sont l'union (ou, +), l'intersection (et, .). L'unique opérateur logique unaire est la négation (non, ¬).

Définition 4.

Un **monôme** est une expression Booléenne composée du produit de une ou plusieurs variables apparaissant sous forme normale (x) ou complémentée (¬x).

ex : $x_1.\neg x_2.x_3$ est un monôme.

Définition 5.

Un monôme est dit **canonique** dans une fonction si toutes les variables apparaissant dans la fonction apparaissent dans sa constitution.

ex : $x_1.\neg x_2.x_3$ est un monôme canonique dans $f(x_1, x_2, x_3)$.

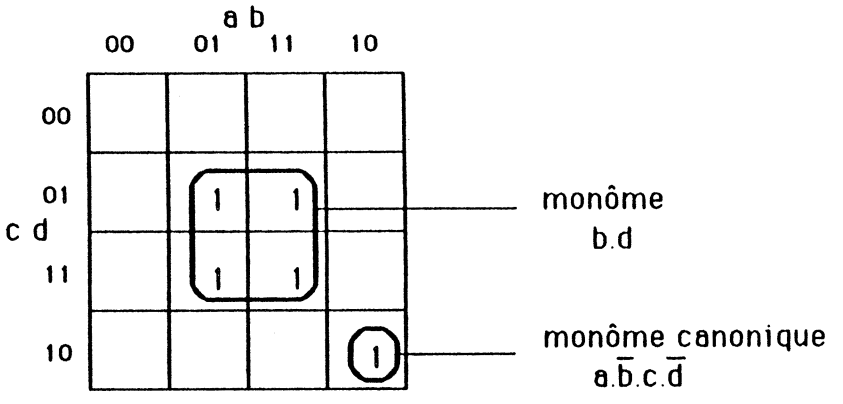


Figure 5 : Illustration des définitions 4 et 5.

Définition 6.

Une expression Booléenne exprimée sous la forme d'une somme de monômes est appelée **polynôme**. On dit aussi que l'expression est sous sa forme normale.

ex : E (a, b, c, d) = b.d + a.¬b.c.¬d (figure 5).

Définition 7.

Une expression Booléenne E_1 est incluse dans une expression Booléenne E_2 si et seulement si il existe une expression Booléenne E_3 telle que : $E_2 = E_1 + E_3$.

Remarque 1 : Un monôme M_1 est inclus dans un monôme M_2 si les variables de M_2 apparaissent sous la même forme dans M_1.

Remarque 2 : si M_1 est inclus dans M_2 , alors $M_1 + M_2 = M_2$.

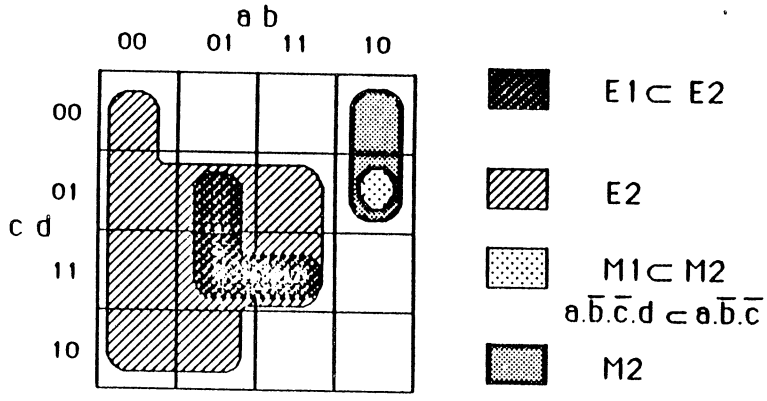


Figure 6 : Inclusion d'une expression dans une autre.

Définition 8.

La taille d'un monôme M de p variables dans une fonction de n variables est égale au nombre de monômes canoniques inclus dans M, soit 2^{n-p}.

ex : dans la figure 5, le monôme b.d a une taille égale à $2^{4-2} = 4$.

Définition 9.

Un monôme M est dit **premier** dans une fonction f s'il n'existe pas de monôme P≠M tel que P soit inclus dans f et M soit inclus dans P (figure 23).

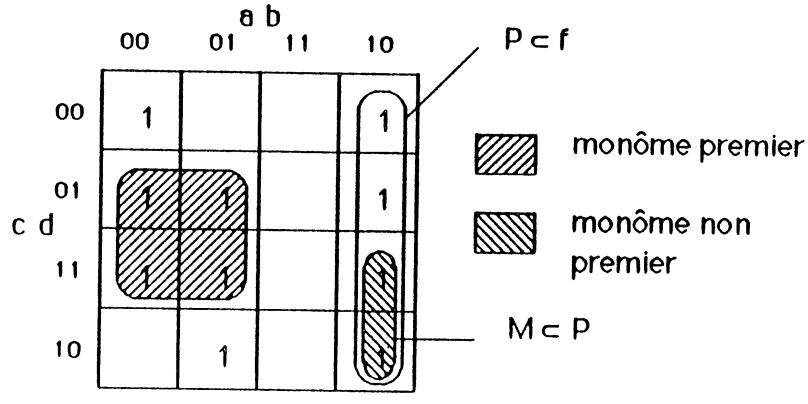


Figure 7 : Illustration de la définition 9.

Le lecteur peut se rapporter à [KUN65] pour de plus amples détails et des définitions plus formelles concernant l'algèbre de Boole.

I.2. LES STRUCTURES "CIBLE" POUR LA LOGIQUE COMBINATOIRE ET LES CRITERES D'OPTIMISATION

Nous allons présenter les différentes structures pour l'implémentation des circuits combinatoires. Les méthodes de synthèse logique étant dépendantes de la technologie, nous exposons les critères d'optimisation relatifs à chacune des structures "cible".

I.2.1. La synthèse à base de réseaux programmables.

On désigne par réseau logique programmable une structure logique constituée de deux matrices, permettant d'implémenter un ensemble de fonctions Booléennes des mêmes variables et exprimées par des polynômes. La première matrice (étage ET) réalise les monômes, c'est à dire les produits de variables complémentées ou non, la seconde matrice (étage OU) réalise les fonctions comme des sommes de monômes. Le PLA (figure 8) est un réseau logique dont les deux matrices sont programmables. Le PAL (figure 9) est un réseau logique dont seul l'étage ET est programmable. Dans un PAL, l'étage OU est préprogrammé de telle sorte qu'un monôme n'est connecté qu'à une seule fonction. Contrairement au PAL, la ROM (read only memory) est un réseau logique dont seul l'étage OU est programmable. L'étage ET d'une ROM réalise un décodeur, c'est à dire un ensemble de monômes canoniques (figure 10).

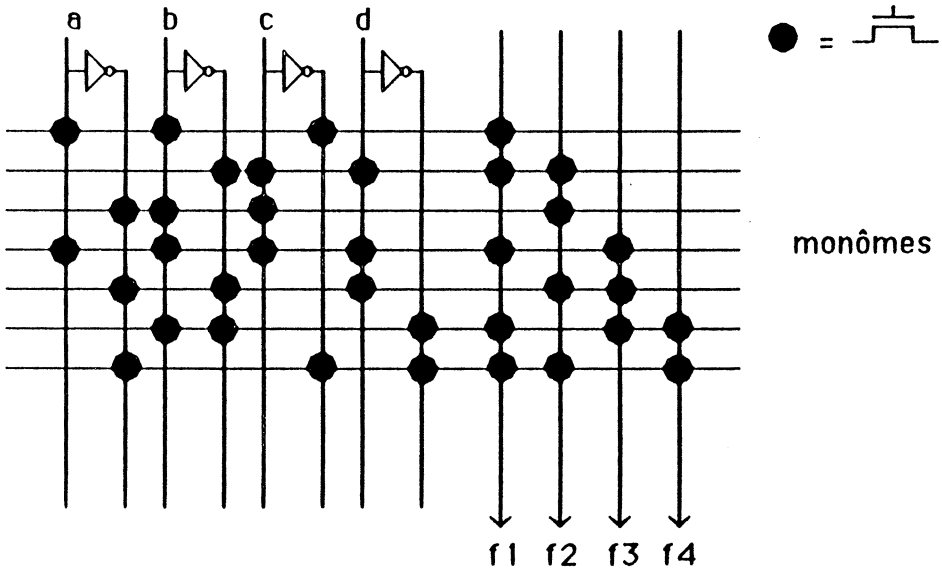


Figure 8 : Un PLA.

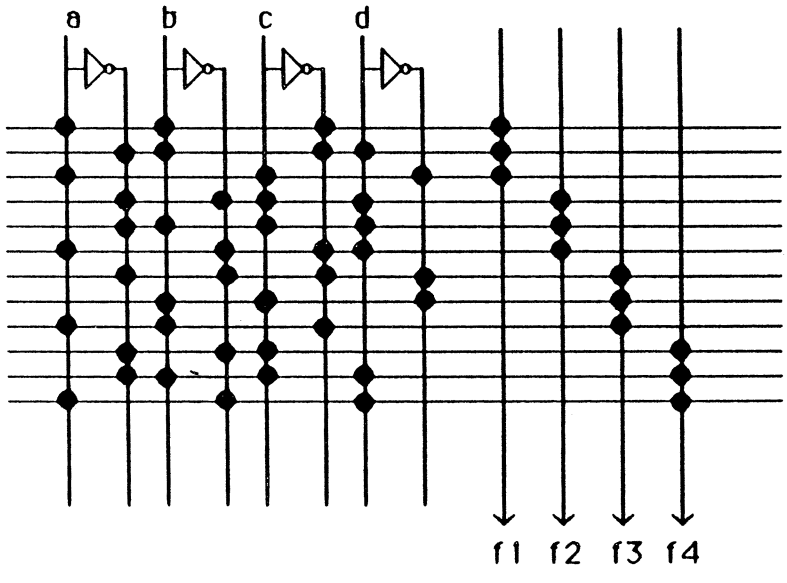


Figure 9 : Un PAL.

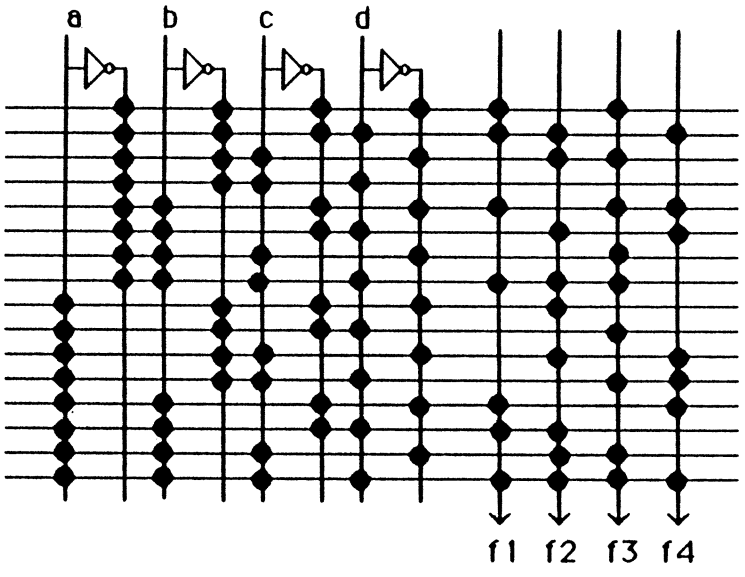


Figure 10 : Une ROM.

Les critères d'optimisation pour la synthèse à base de réseaux logiques :

Dans tous les cas, l'optimalité d'un circuit combinatoire est liée à la surface occupée sur le silicium ainsi qu'aux performances du circuit, c'est à dire ses temps de réponse et sa dissipation thermique (consommation). Dans le cas des réseaux logiques programmables, les performances dépendent pour une technologie donnée, de trois paramètres qui sont :

- la charge des entrées,
- la charge des monômes,
- la charge des sorties.

La charge d'une entrée est le nombre de transistors qui lui sont connectés. La charge d'un monôme est le nombre de transistors qui le connectent aux sorties. La charge d'une sortie est le nombre de transistors qui lui sont connectés. D'une façon générale, on admet que les performances d'un réseau logique programmable sont meilleures, d'autant que le nombre de transistors qu'il contient est petit.

En ce qui concerne la surface, elle dépend du nombre d'entrées **n**, du nombre de sorties **k**, et du nombre de monômes **p**. En première approximation, on peut la calculer ainsi :

$$S = p \cdot (2n+k)$$

Si l'on admet que la minimisation ne porte pas sur le nombre d'entrées et de sorties du circuit, on déduit que la surface du réseau est directement liée au nombre de monômes.

* *Cas de la ROM.*

Une ROM réalise des fonctions exprimées comme une somme de monômes canoniques (1^{ere} forme de Lagrange). Pour une combinaison des variables d'entrées, il n'existe qu'un seul monôme canonique vrai. Ainsi, pour une fonction donnée, un monôme canonique est soit obligatoire, soit interdit. Du fait de la non existence de redondance dans les formes canoniques, on ne peut pas parler de minimisation logique dans une ROM. La seule minimisation possible dans une ROM est d'ordre topologique, et consiste à supprimer dans l'étage de décodage les monômes qui n'apparaissent dans aucune des fonctions.

* *Cas du PAL.*

Dans un PAL, chaque fonction est réalisée indépendamment des autres. Si un monôme doit apparaître dans **q** fonctions, il sera dupliquer **q** fois dans le PAL. Il est évident que le PAL ne présente aucun intérêt en tant que partie d'un circuit intégré, mais il existe en tant que circuit programmable pour la réalisation de cartes. La capacité d'accueil d'un PAL est déterminée par le nombre de fonctions, le nombre total de monômes, et le nombre de monômes par fonctions. Ainsi, la minimisation logique pour une synthèse sur PAL est une réduction locale du nombre de monômes dans chacune des fonctions à implémenter. La minimisation locale d'une fonction Booléenne est exposée en II.3.

* *Cas du PLA.*

Le PLA est très largement utilisé dans la conception de circuits intégrés du fait qu'il implémente de façon régulière un ensemble de fonctions combinatoires sur une surface généralement inférieure à celle de la ROM équivalente [GRA83]. La minimisation logique pour une synthèse sur PLA est une réduction globale du nombre

total de monômes, dans laquelle toutes les fonctions à implanter sont considérées simultanément. La minimisation globale d'un ensemble de fonctions Booléenne est exposée en II.3. Indépendamment de la minimisation logique, les PLA peuvent être minimisés topologiquement afin de gagner en surface. En effet, la grande majorité des PLA est constituée de matrices "creuses" entraînant des pertes considérables de surface. Aussi, des techniques d'optimisation topologique [PAI84] [PER82] ont été développées pour récupérer les zones "creuses" (sans transistors) par des méthodes de compactage, de pliage, ou bien de triangularisation du PLA initial.

I.2.2. La synthèse combinatoire en logique aléatoire.

La synthèse en logique aléatoire consiste à implanter un circuit combinatoire à l'aide d'un réseau d'opérateurs interconnectés. Les opérateurs peuvent réaliser des fonctions logiques simples (portes AND, OR, NAND, NOR, INV...) ou bien des fonctions plus complexes (décodeur, multiplexeur...). Pour réaliser un circuit combinatoire en logique aléatoire, le concepteur dispose d'une bibliothèque d'opérateurs qui est liée à la structure d'accueil (circuits TTL pour la conception de cartes, bibliothèque de cellules pour les réseaux prédiffusés...).

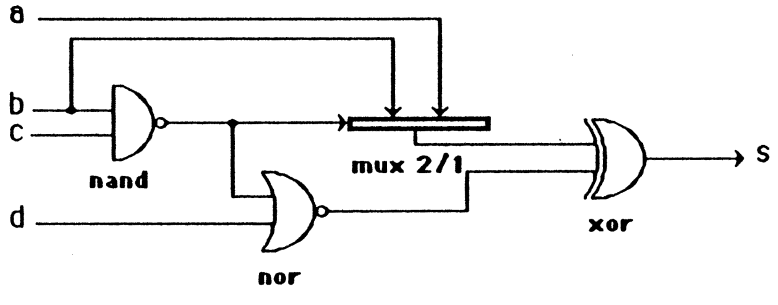


Figure 11 : circuit combinatoire en logique aléatoire.

Les critères d'optimisation pour la synthèse en logique aléatoire :

Si le nombre de couches logiques dans un réseau logique programmable est toujours égal à 2, ce n'est plus le cas ici. En logique aléatoire, le nombre de couches

logiques est égal au nombre maximum d'opérateurs qu'un chemin reliant une entrée à une sortie peut traverser. Le temps de réponse du circuit est donc directement lié au temps de propagation dans les opérateurs, et la minimisation du nombre de couches logiques est donc un premier critère d'optimalité.

Le second critère d'optimalité est lié à la surface du circuit. La surface occupée dépend du nombre d'opérateurs, mais également du nombre d'interconnexions entre ces opérateurs. Réduire le nombre de connexions facilite toujours l'implantation, qu'elle soit manuelle (dessin au micron) ou bien automatisée (placement et routage sur des structures à contraintes telles que les réseaux prédiffusés).

Minimiser le nombre d'opérateurs n'a de sens que relativement à une bibliothèque donnée. Il serait dommage de réaliser un multiplexeur avec des portes si l'opérateur multiplexeur existe déjà dans la bibliothèque. Ainsi, il est nécessaire que les algorithmes de minimisation pour la synthèse en logique aléatoire soient paramétrables par un ensemble d'opérateurs disponibles.

* *Les méthodes de synthèse en logique aléatoire :*

Dés le début des années 60, la C.A.O s'est intéressée à la synthèse en logique aléatoire. Les premières méthodes proposées se limitaient à la synthèse en portes NAND ou NOR. Dietmeyer et Schneider [DIE63] proposent une méthode de factorisation pour la synthèse en portes NAND ou NOR, respectant les limitations sur le nombre d'entrées des opérateurs et basée sur l'équivalence suivante :

NOR (a, b, c, d) = NOR (NOR (NOR (a, b)), c, d).

Cette technique consiste à réduire le nombre d'opérateurs par la génération de facteurs communs.

exemple :

Considérons la fonction de 5 variables f = NOR (NOR (a, b, c, d), NOR (a, b, c, e)), devant être réalisée avec des portes NOR à 3 entrées maximum. La décomposition de cette fonction se fait à l'aide du facteur commun FC = NOR (NOR (a, b, c)).

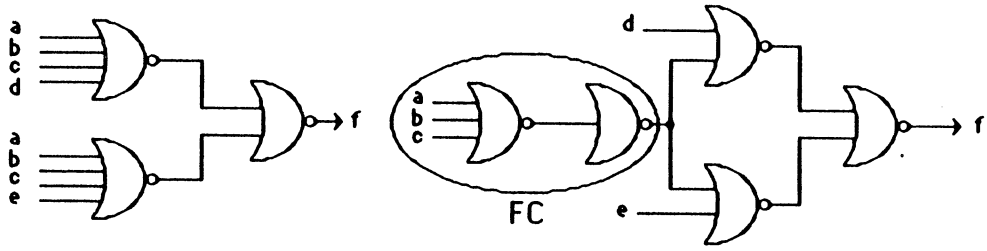


Figure 12 : synthèse en NOR par factorisation.

Ces principes de factorisation ont été étendus à la synthèse multi-fonctions à partir d'une représentation en deux couches ET/OU minimisée (figure 13) [SU71] [HAN85]. La figure 14 représente le circuit de la figure 13 synthétisé par factorisation pour des opérateurs à deux entrées et pour une sortance (nombre d'entrées connectées à une sortie) maximale égale à 2.

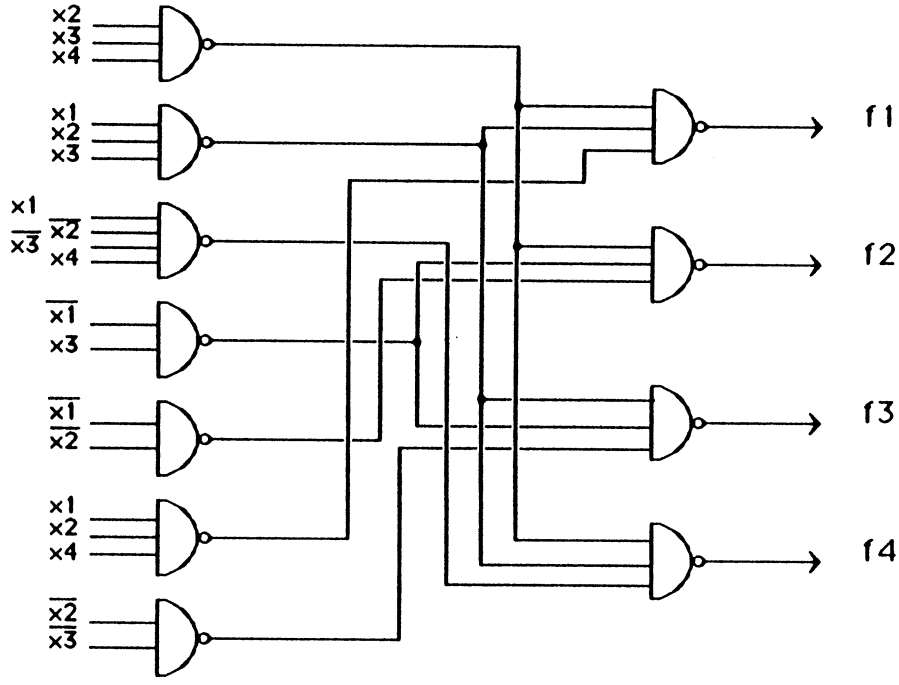


Figure 13 : Circuit sur 2 couches sans contraintes.

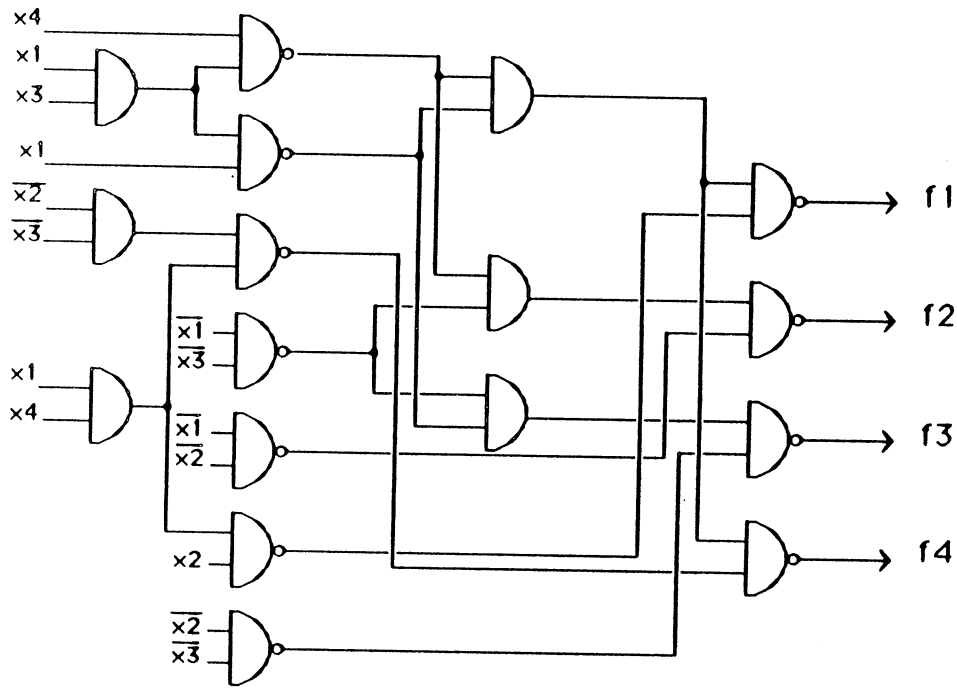


Figure 14 : le même circuit synthétisé sous contraintes.

Dans [SCH68] et [CER74], les auteurs présentent des techniques de synthèse paramétrées par un ensemble de modules fonctionnels. Les méthodes procèdent par décomposition des fonctions à réaliser en termes d'assemblage de modules fonctionnels.

Les techniques de recherche opérationnelle, notamment la programmation en nombre entier ont également permis de résoudre les problèmes de la synthèse logique sous contraintes. Cette approche est celle présentée par Muroga dans [MUR72] et [MUR76]. La formulation en un problème de programmation en nombre entier permet de prendre en compte tous les choix du concepteur, quant aux types d'opérateurs à employer et aux contraintes à respecter. Une fois le problème formulé, sa résolution est éffectuée par une méthode de recherche opérationnelle appelée "méthode de l'énumération implicite", à travers une fonction de coût, définie par le concepteur en fonction des critères d'optimisation souhaités. Si le problème admet une solution, alors cette solution est un optimum global , relativement à la fonction de coût.

Si une telle approche est d'un intérêt évident sur le plan théorique, elle l'est

beaucoup moins sur le plan pratique en raison de performances médiocres quant au temps d'exécution nécessaire à la résolution du problème. A titre indicatif, le temps de calcul sur un IBM 360 nécessaire pour obtenir d'une fonction de 4 variables un réseau optimal avec 5 portes NOR et 16 connexions, est de 3500 secondes [MUR76]. La figure 14 représente une solution optimale pour un additionneur 1 bit en portes NOR avec les critères d'optimisation suivants :

- la minimisation du nombre de portes,
- la minimisation du nombre de connexions.

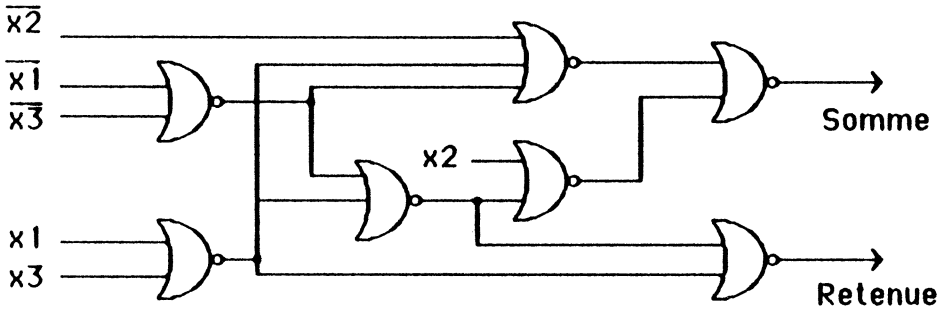


Figure 15 : cellule optimale en NOR d'un additionneur 1 bit.

Depuis le début des années 80, de nouvelles techniques pour la synthèse en logique aléatoire apparaissent, qui s'écartent des méthodes algorithmiques classiques, en optant pour des approches orientées vers les systèmes experts et les systèmes de règles. Les méthodes les plus significatives à retenir sont construites sur le principe de transformations locales. Le circuit optimisé est obtenu à partir des fonctions minimisées, auxquelles on applique une série de transformations locales formulées par des règles [DAR80] [GEU85].

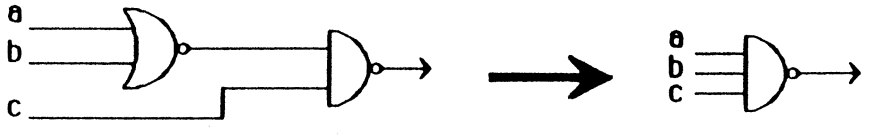


Figure 16 : règle de transformation locale.

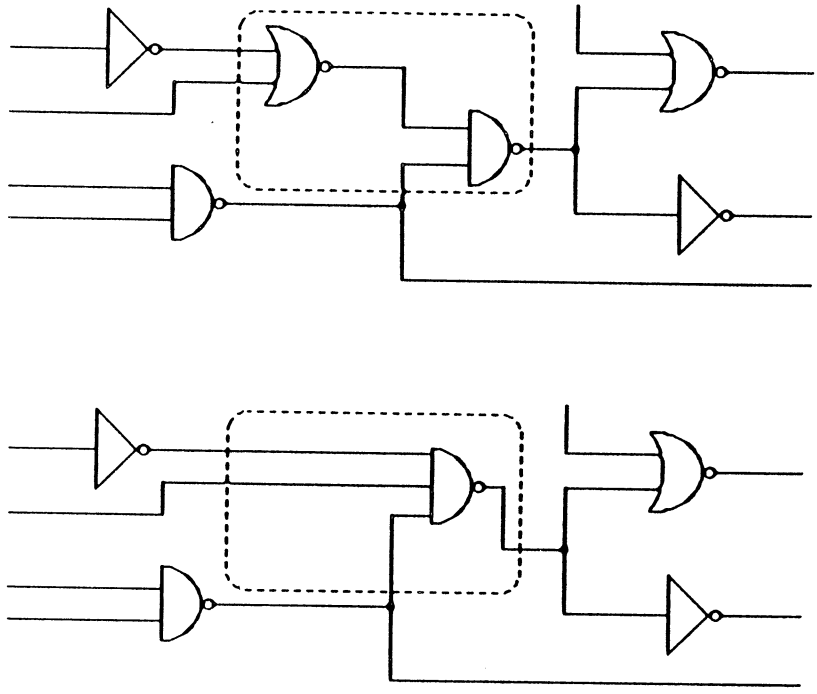


Figure 17 : transformation locale appliquée à un circuit.

Les méthodes basées sur des optimisations locales sont efficaces et conviennent à la synthèse de gros circuits. De plus, un concepteur expert peut aisément formuler les contraintes à respecter, et affiner la synthèse par l'adjonction de nouvelles règles caractérisant son savoir faire.

I.2.3. La synthèse combinatoire en portes complexes.

La synthèse en portes complexes est une synthèse combinatoire de bas niveau, pour laquelle les fonctions logiques sont réalisées à l'aide de réseaux d'interrupteurs (transistors MOS) [CAL62].

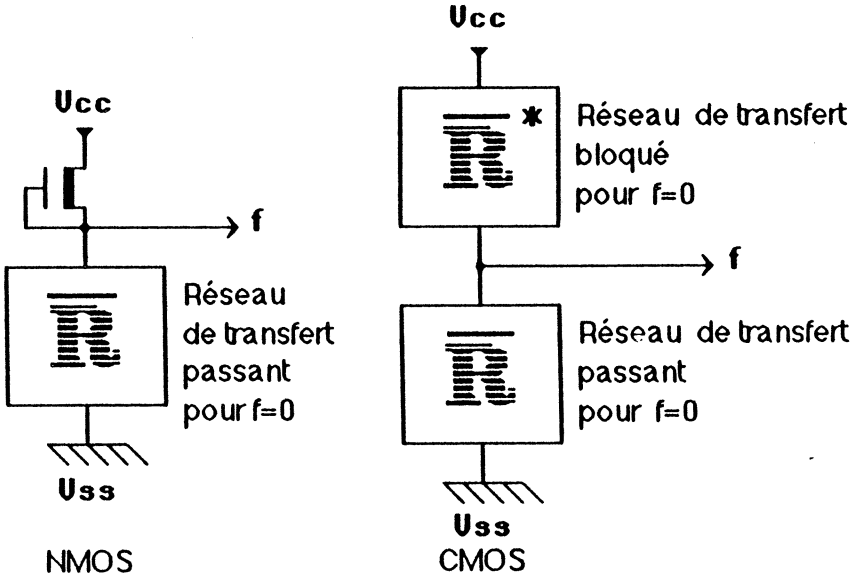


Figure 18 : Portes complexes en technologie NMOS et CMOS.

Les critères d'optimisation pour la synthèse en portes complexes :

L'optimalité d'une porte complexe dépend de la surface qu'elle occupe sur le silicium et de ses performances. Les performances sont principalement liées au nombre de transistors en série et en parallèle dans le réseau. La surface dépend du nombre total de transistors, mais aussi du dimensionnement des transistors. Plus il y a de transistors en série ou en parallèle dans un réseau, plus grand doivent ils être dimensionnés, rendant ainsi l'implantation sur silicium plus difficile. Pour faire de la synthèse en portes complexes, il est donc nécessaire de minimiser le nombre total de transistors, mais aussi le nombre de transistors en série ou en parallèle.

Une méthode de minimisation d'un réseau de transfert est présentée dans [SAU85], qui procède de la façon suivante :

1/ l'expression de la fonction de transfert étant sous la forme d'un polynôme minimisé logiquement, un réseau initial série-parallèle est construit, dans lequel chaque monôme est représenté par un chemin (figure 19) ;

2/ Un algorithme de partitionnement engendre ensuite un ordonnancement optimisé des variables et des chemins série-parallèles du réseau, pour le transformer

en un réseau arborescent, par le fusionnement de certains transistors sur une extrémité du réseau (figure 20) ;

3/ le réseau est ensuite transformé en un réseau non série-parallèle, par le fusionnement de transistors sur l'autre extrémité du réseau, en vérifiant que les chemins créés ne changent pas la fonction de transfert (figure 21).

exemple :

Considérons la fonction $f=\neg x_1.\neg x_3 + x_2.\neg x_4 + x_1.\neg x_2.x_3$.

* *étape 1.*

Le polynôme minimal de la fonction de transfert de **f** est obtenu par minimisation logique de $\neg$**f**.

$\neg f = x_1.x_2.x_4 + \neg x_1.x_3.x_4 + x_1.\neg x_2.\neg x_3 + \neg x_1.\neg x_2.x_3$

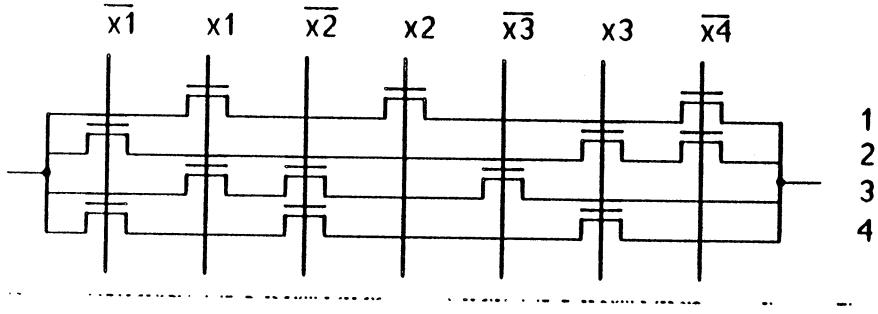


Figure 19 : Réseau initial série-parallèle.

* *étape 2.*

L'ordonnancement optimal (qui permet un gain maximum en nombre de transistors) est le suivant :

- pour les variables : x_4, $\neg x_2$, x_2, $\neg x_3$, x_3, $\neg x_1$, x_1
- pour les lignes : 1, 3, 2, 4

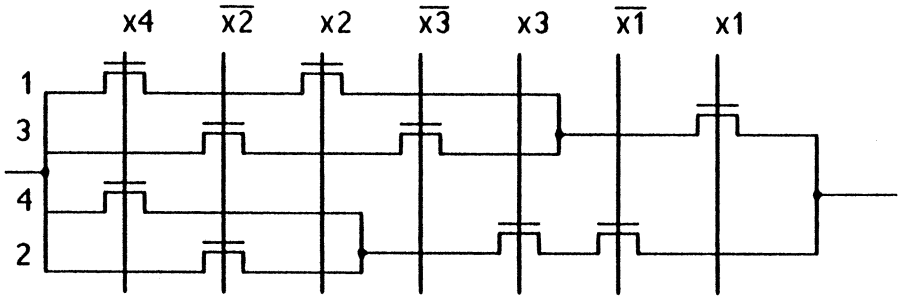


Figure 20 : Réseau série-parallèle optimal.

Le gain obtenu après le fusionnement des transistors est de **25 %.**

* *étape 3.*

On peut fusionner les transistors contrôlés par la variable x_4 sur l'extrémité gauche du réseau. Les nouveaux chemins introduits par ce fusionnement ne modifie pas la fonction de transfert car ils sont impossibles (caractérisés par des monômes toujours faux).

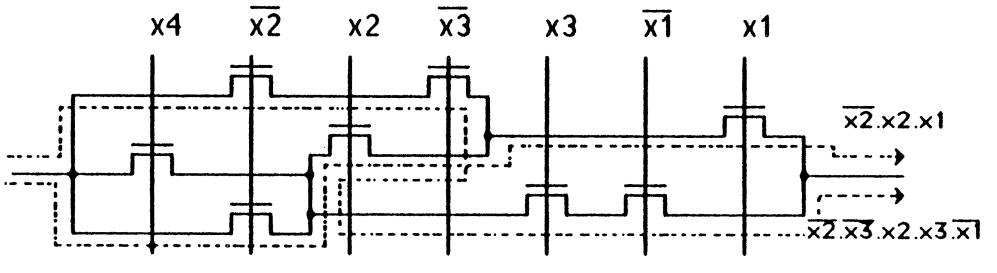


Figure 21 : Réseau final non série-parallèle.

Le gain finalement obtenu après les trois étapes est de 8/12 = **33 %.**

La synthèse en portes complexes s'avère intéressante pour les petits circuits combinatoires, car si la surface d'implantation n'est pas toujours inférieure à celle du PLA équivalent après minimisation topologique, les performances sont meilleures (fonction de 5 variables : porte complexe NMOS : 7,5 ns ; PLA NMOS : 16 ns). De plus, l'implantation des réseaux de transistors bien structurés est automatisable en

utilisant des techniques d'implantation en lignes ou en blocs de diffusion [SAU85]. Ces méthodes s'adaptent parfaitement à la synthèse multi-fonctions en conservant un même ordonnancement des entrées à travers les différentes fonctions.

Les techniques d'implantation, de minimisation logique et topologique de portes complexes ont fait l'objet de nombreuses études ; le lecteur intéressé peut consulter [THU83] pour de plus amples informations.

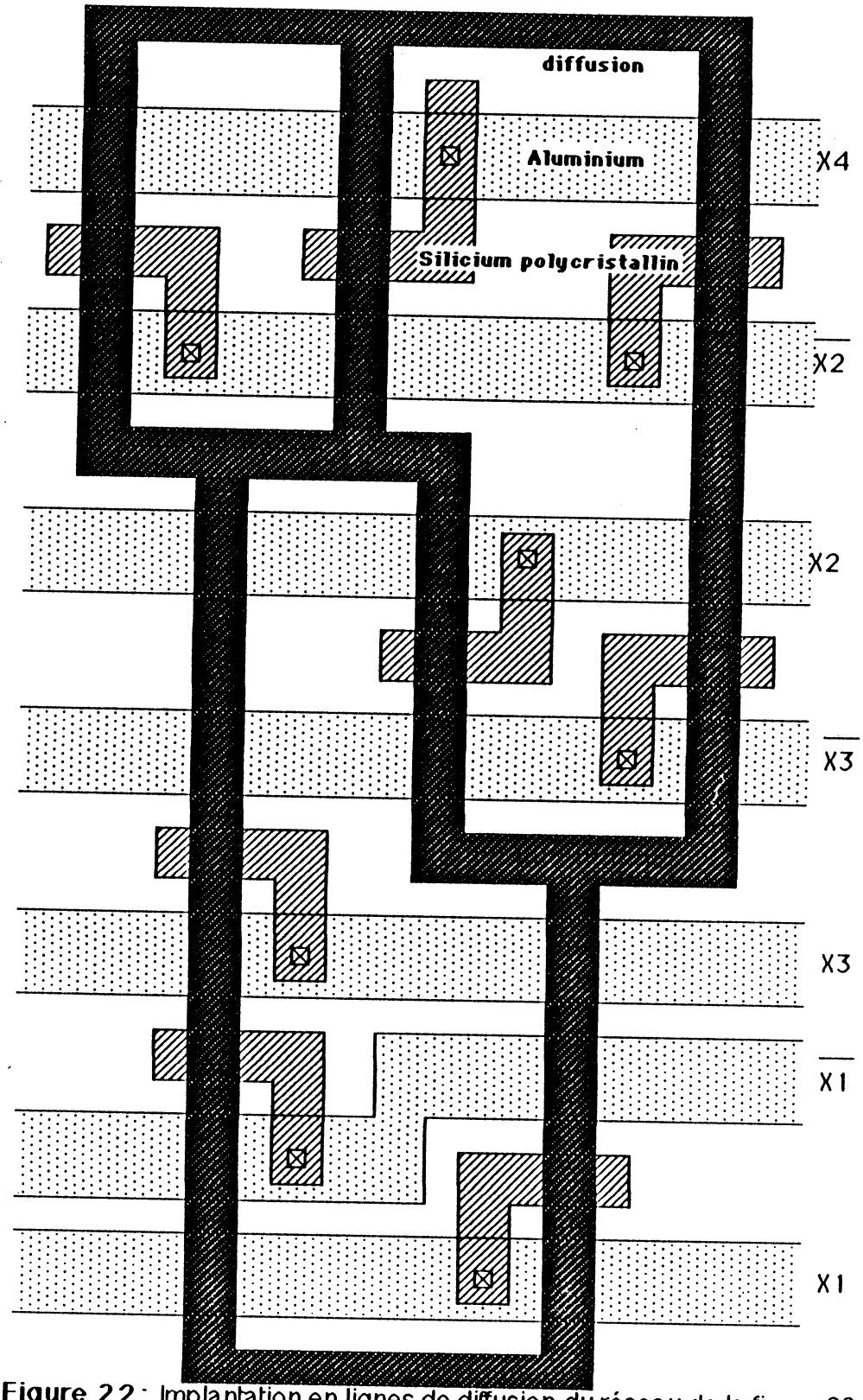


Figure 22 : Implantation en lignes de diffusion du réseau de la figure 20.
(transistors non dimensionnés)

I.3. LA MINIMISATION LOGIQUE A BASE DE REGLES

Ce chapitre est consacré à l'étude d'une nouvelle approche au problème de la minimisation logique. Le premier paragraphe positionne le problème et donne l'état de l'art dans le domaine. Les paragraphes suivants concernent notre manière de résoudre le problème de la minimisation logique en s'appuyant sur un système de règles. Le logiciel ASYL a mis en application les règles de minimisation, et a permis d'en vérifier l'efficacité relativement aux résultats obtenus et aux temps de calcul consommés.

I.3.1. Position du problème et état de l'art

Le problème de la minimisation logique s'est posé au début des années 50, à une époque où le matériel était coûteux. Il était alors indispensable de développer des techniques permettant d'implémenter une ou plusieurs fonctions Booléennes avec un nombre minimum de ressource (diodes, relais, transistors, portes,...). La minimisation logique devenait à partir de cette époque un sujet de recherche de toute première importance.

La première technique manuelle de minimisation logique qui est apparue à cette époque est celle des tableaux de Karnaugh (figure 23), qui permet de minimiser des fonctions de moins de 5 variables. Des techniques automatisables et plus sophistiquées ont été introduites par Quine et Mac Cluskey [MCC65], puis par Tison qui mit au point la théorie des consensus et son application à la minimisation logique [TIS65] [TIS67]. Ces dernières méthodes procèdent en deux étapes qui sont :

- la génération de tous les monômes premiers (définition 9) de la fonction,
- l'extraction d'une couverture première irrédondante.

Le principe d'extraction d'une couverture première irrédondante permet d'obtenir une solution optimale au sens du nombre total de monômes. Elle s'obtient en sélectionnant parmi tous les monômes premiers de la fonction, un ensemble de taille minimale dans lequel aucun monôme n'est redondant.

Les méthodes de génération de tous les monômes premiers ont été rendues plus efficaces au cours des années suivantes [SLA70], mais malgré cela, le caractère

combinatoire du problème rend ces techniques totalement impraticables pour des fonctions dépassant une taille moyenne (moins de 12 variables).

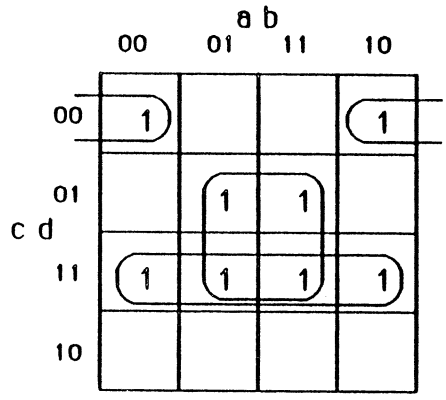


Figure 23 : Minimisation à partir d'un tableau de Karnaugh.

Dans le début des années 70, la minimisation logique n'est plus aussi essentielle qu'auparavant pour de nombreux concepteurs, en raison d'une forte diminution du coût des composants. Néanmoins, elle retrouve un intérêt primordial avec le développement de circuits compacts à forte densité d'intégration du type microprocesseurs ou bien circuits spécialisés (ASIC). La taille des problèmes combinatoires évoluant avec la complexité des circuits et les capacité d'intégration, de nouvelles méthodes sont développées, qui conduisent à des solutions presque optimales.

Les programmes les plus classiques pour la minimisation logique multi-fonctions sont **MINI** [HON74], **PRESTO** [BRO81], **ESPRESSO-IIC** [BRA85], **McBOOLE** [DAG85], **MOM** [AGR85] et **PHIPLA** [LAA85]. Ces programmes sont tous de type algorithmique, basés sur des considérations heuristiques pour limiter les problèmes dus à l'explosion combinatoire (croissance exponentielle du temps CPU avec le nombre de variables). Malgré ces heuristiques, certains programmes ne peuvent encore traiter des problèmes dépassant 12 à 16 variables. C'est le cas des programmes **PHIPLA** et **MOM** qui opèrent sur la forme canonique des fonctions (2^n monômes pour n variables).

Notre approche du problème de la minimisation logique est complètement différente dans sa formulation à base de règles et dans ses objectifs. Dans le but de pouvoir minimiser des systèmes combinatoires de grande taille (50 variables d'entrée

et 100 fonctions), nous avons défini des règles de transformation qui garantissent l'obtention d'un résultat final sans redondance, mais pas nécessairement optimal.

I.3.2. Un système de règles pour la minimisation locale d'une fonction complète.

La minimisation locale concerne la réduction d'une seule fonction Booléenne, indépendamment de toute autre fonction. Les règles que nous avons définies pour la minimisation locale transforment l'expression de la couverture à 1 de la fonction à minimiser en une somme quasi minimale de monômes premiers. Ces règles ont été étudiées de façon à respecter les deux conditions suivantes :

- 1/ On ne doit pas évaluer la forme canonique (2^n monômes pour n variables) de la fonction à minimiser, contrairement aux méthodes de Quine-Mc Cluskey.
- 2/ On ne doit pas engendrer tous les monômes premiers ($\leq 3^n/n$ pour n variables) de la fonction à minimiser, contrairement à la méthode classique des consensus.

I.3.2.1. Règles pour la mise sous forme normale d'une expression:

Le premier ensemble de règles intervient pour la réécriture d'une expression Booléenne quelconque en un polynôme logiquement équivalent. Ceci est obtenu en réduisant la portée des opérateurs de négation à une variable simple, et en développant les factorisations.

Soient E_i, E_j, E_k, E_l des expressions Booléennes quelconques.

Règles pour la réduction de la portée des opérateurs de négation :

Règle 1 : non (non (E_i)) --> E_i

Règle 2 : non (E_i.E_j) --> non (E_i) + non (E_j)

Règle 3 : non (E_i + E_j) --> non (E_i) . non (E_j)

Règles pour le développement des factorisations :

Règle 4 : $E_i . (E_j + E_k) \dashrightarrow E_i.E_j + E_i.E_k$

Règle 5 : $(E_i + E_j) . (E_k + E_l) \dashrightarrow E_i .(E_k + E_l) + E_j .(E_k + E_l)$

Ces règles sont appliquées sur l'expression à transformer jusqu'à ce qu'aucune d'elles ne puisse l'être.

I.3.2.2. Règles pour la minimisation locale:

Ces règles transforment une somme de monômes quelconque en une somme quasi minimale de monômes premiers.

Considérons une fonction Booléenne complètement spécifiée **f** définie par sa couverture à 1 comme une somme de **p** monômes $M_1, M_2,\ldots,M_p$; et posons **EM** = $(M_1, M_2,\ldots,M_p)$.

Soient $\mathbf{M_i}$, $\mathbf{M_j}$ deux monômes de **EM**, et **x** une variable Booléenne. La négation de **x** est notée $\neg$**x**.

Règles primaires de transformation :

Les règles primaires de transformation sont dérivées des règles élémentaires du calcul formel Booléen. Ce sont des règles de faible complexité, qui engendrent des transformations locales à fort gain.

Règle 6 : règle de tautologie.

si $M_i = x$ et si $M_j = \neg x$ **alors** f est une **tautologie** ($a + \neg a = 1$).

Losque cette règle est appliquée, le processus de minimisation est interrompu, la fonction f étant formellement toujours vraie (figure 24).

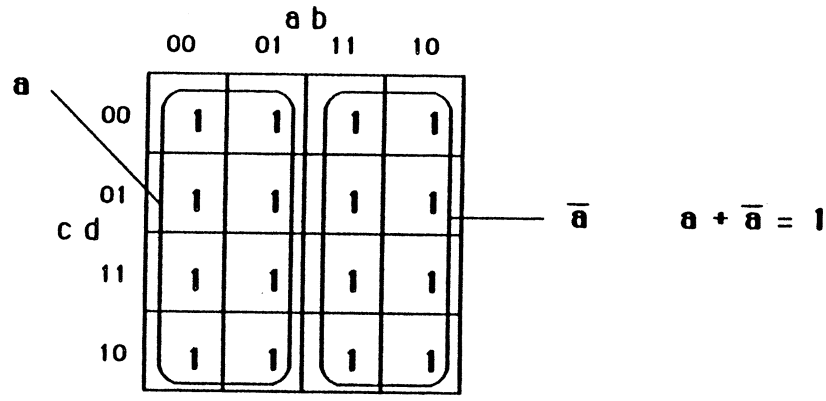


Figure 24 : Règle de tautologie.

Règle 7 : règle d'exclusion d'un terme nul.

si il existe un monôme M_i de Em tel que dans sa constitution, apparaissent la variable x et son complément $\neg x$, **alors** M_i est un terme toujours nul ($x.\neg x=0$) et est supprimé de EM.

De tels monômes peuvent apparaître après le développement des factorisations dans l'application des règles 1 à 5.

Règle 8 : règle d'identité.

si $M_i = M_j$, **alors** M_j est supprimé de EM ($a + a = a$).

Le gain associé à l'application de cette règle est de 1 monôme.

Règle 9 : règle d'adjacence.

si il existe une variable x telle que $M_i = x.N$ et $M_j = \neg x.N$, où N est un monôme, **alors** M_i et M_j sont **adjacents** ($N = M_i + M_j$) et

- M_j est supprimé de EM,
- M_i est remplacé dans EM par N. (figure 25)

Le gain associé à l'application de la règle d'adjacence est de 1 monôme et de 1 variable dans le monôme M_i.

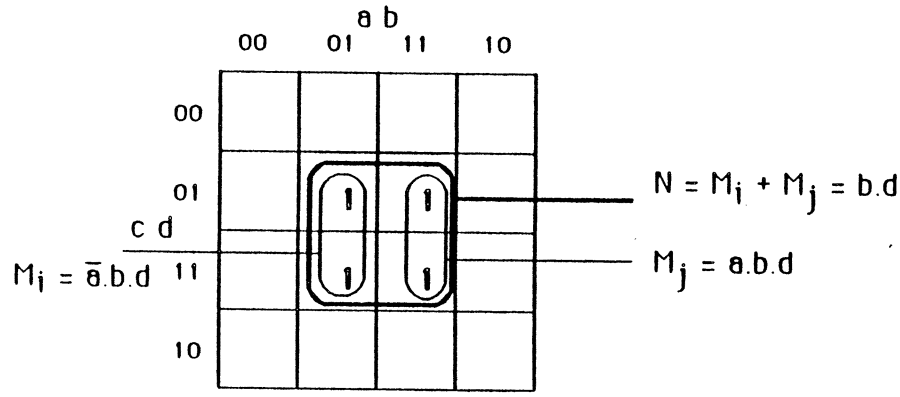


Figure 25 : Règle d'adjacence.

Règle 10 : règle d'inclusion.

si il existe un monôme N tel que $M_j = M_i.N$, **alors** M_j est **inclus** dans M_i ($M_i+M_j = M_i$) et

- M_j est supprimé de EM. (figure 26).

Le gain associé à l'application de cette règle est de 1 monôme.

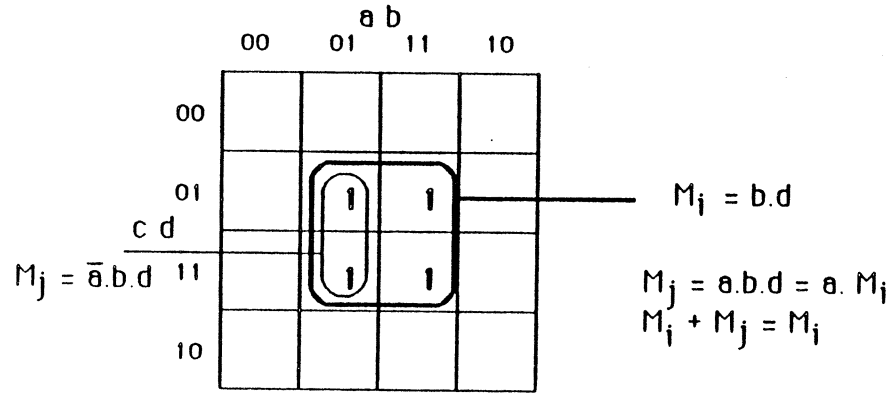


Figure 26 : Règle d'inclusion.

Règle 11 : Règle d'expansion locale.

si il existe une variable x et deux monômes N et P tels que $M_i = \neg x.N.P$ et $M_j = x.P$, alors la variable $\neg x$ peut être supprimée dans M_i ($\neg a.b.c + a.c = b.c + a.c$). (figure 27).

Le gain associé à l'application de cette règle est de 1 variable dans le monôme M_i.

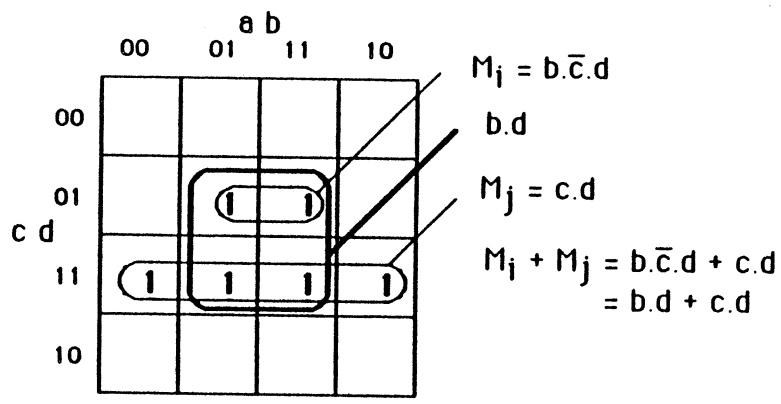


Figure 27 : Règle d'expansion locale.

L'application des règles 6 à 11 ne résoud que partiellement le problème de la minimisation locale d'une fonction complète en engendrant des monômes qui ne sont pas toujours premiers. D'une part, la qualité du résultat peut s'avérer insuffisante et d'autre part, il n'est pas toujours possible de prouver formellement qu'une fonction est une tautologie.

exemple 1 :

$f = \neg a.\neg c.\neg d + \neg a.\neg c.d + \neg a.b.d + a.b.c.d$

L'application des règles 6 à 11 va engendrer la solution minimisée suivante :

$f = \neg a.\neg c + \neg a.b.d + b.c.d.$ (figure 28).

On n'a pu détecter la présence du monôme redondant **$\neg$a.b.d**.

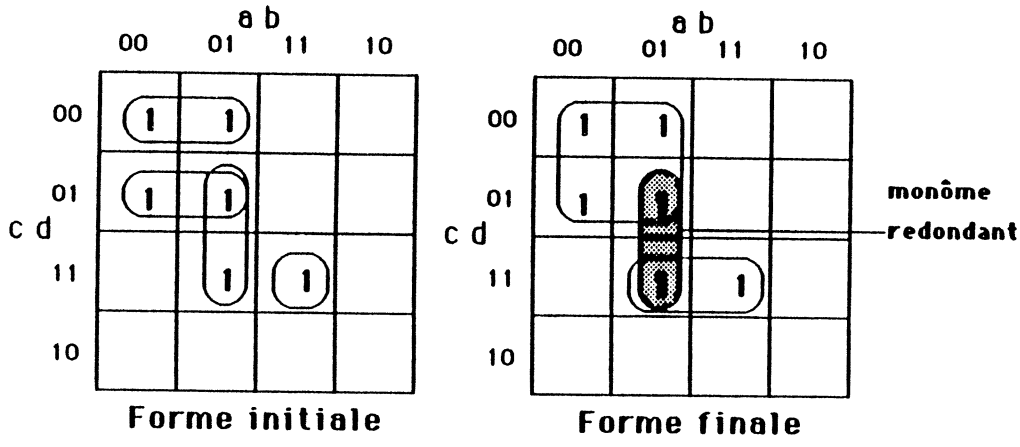


Figure 28

Exemple 2:

$$f = \neg a.b.\neg c + a.\neg c.d + b.c.d$$

Aucune des règles 6 à 11 n'est applicable et on ne peut détecter la redondance de la variable **c** dans le monôme **b.c.d** (figure 29).

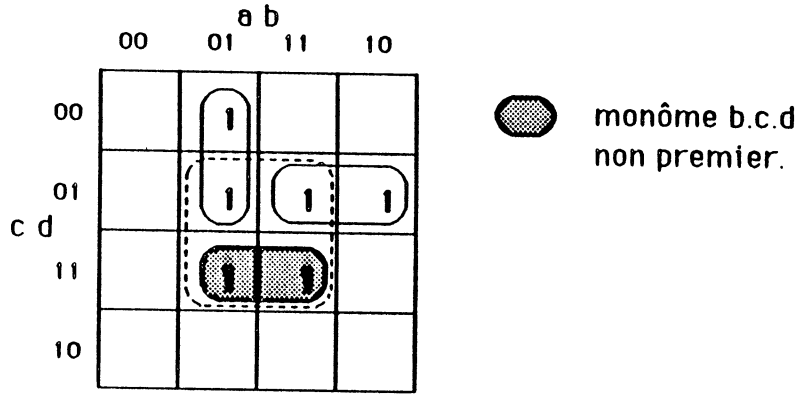


Figure 29.

Exemple 3:

$$f=\neg b.\neg c.\neg d+\neg a.\neg c.d+b.c.d+a.c.\neg d+\neg a.\neg b.c+\neg a.b.\neg d+a.b.\neg c+a.\neg b.d$$

Là encore, aucune des règles 6 à 11 n'est applicable, et on ne peut pas prouver formellement que f est une tautologie (figure 30).

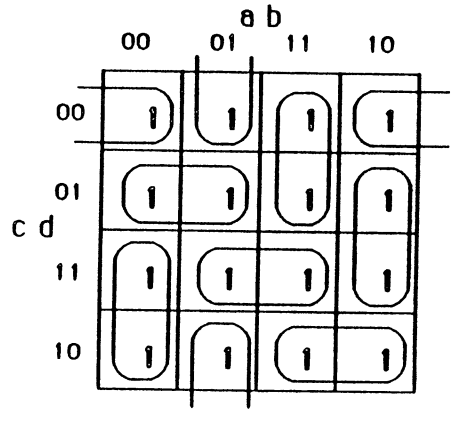


Figure 30.

Règles secondaires de transformation :

Les règles secondaires ont été définies dans le but d'améliorer la qualité de la minimisation locale et de pouvoir faire de la preuve formelle de tautologie. Ces règles sont complémentaires des règles primaires décrites précédemment, et doivent forcément leur être associées pour être opérationnelles. L'association des règles primaires et secondaires engendre pour toute fonction complète, une somme de monômes premiers quasi minimale. Les règles secondaires de transformation s'appuient sur la théorie des consensus du premier ordre [TIS65].

Définition 10:

Soient deux monômes $\mathbf{M_i}$ et $\mathbf{M_j}$. On appelle **consensus** du premier ordre entre M_i et M_j, le monôme de plus grande taille M_{ij} tel que :

- M_{ij} est inclus dans $M_i + M_j$,
- M_{ij} n'est pas inclus dans M_i,
- M_{ij} n'est pas inclus dans M_j (figure 31).

Remarque : Par propriété de l'inclusion, si M_{ij} est le consensus entre M_i et M_j, alors

$M_{ij} + M_i + M_j = M_i + M_j$

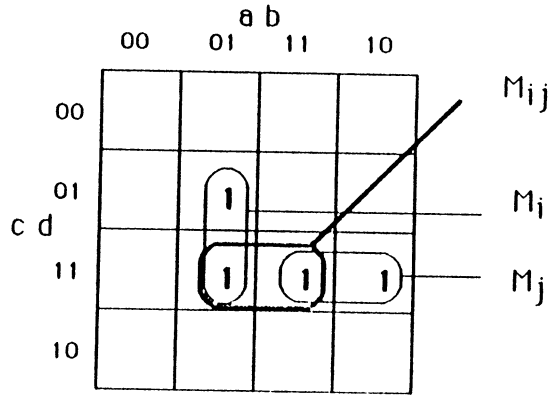


Figure 31 : Consensus du premier ordre entre deux monômes.

Les règles 12 à 14 sont des règles génératrices de nouveaux monômes, consensus entre des monômes existant dans l'ensemble EM. Les consensus ainsi

engendrés ne doivent en aucun cas être ajoutés à l'ensemble EM ; ils ne servent qu'à faire d'éventuelles simplifications.

Règle 12 : Règle d'identité par consensus.

Si il existe un consensus M_{ij} entre deux monômes M_i et M_j de EM, et **si** M_{ij} appartient à EM, alors M_{ij} est redondant et est supprimé de EM (figure 32).

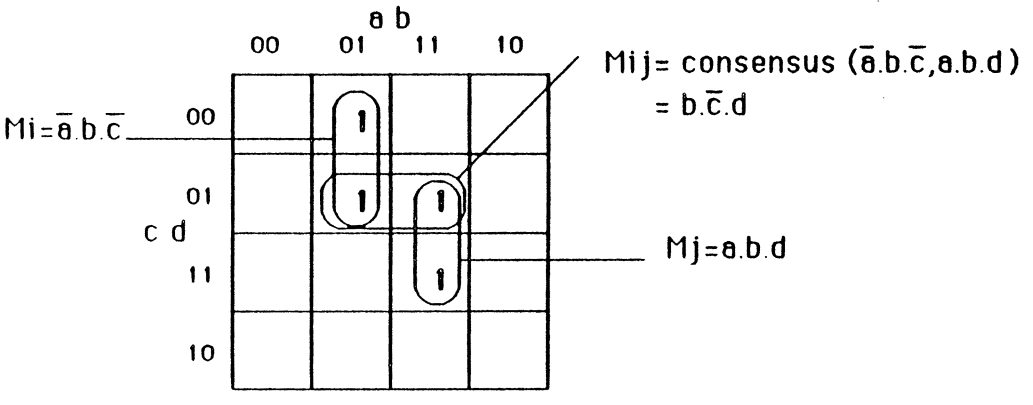


Figure 32 : Règle d'identité par consensus.

Règle 13 : Règle d'absorption par consensus.

Si il existe un consensus M_{ij} entre deux monômes M_i et M_j de EM, et **si** il existe M_k dans EM tel que M_{ij} soit inclus dans M_k, alors M_k est redondant et est supprimé de EM (figure 33).

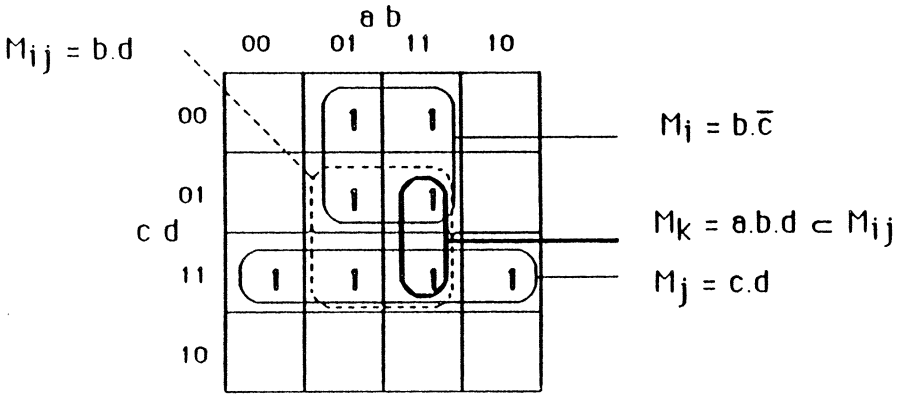


Figure 33 : Règle d'absorption par consensus.

Remarque :

Par définition du consensus, M_{ij} est inclus dans $M_i + M_j$

donc $M_i + M_j + M_k = M_i + M_j + M_{ij} + M_k$

M_k est inclus dans M_{ij} donc $M_k + M_{ij} = M_{ij}$

donc $M_i + M_j + M_k = M_i + M_j + M_{ij}$

$= M_i + M_j$

Règle 14 : Règle d'adjacence par consensus.

Si il existe un consensus M_{ij} entre deux monômes M_i et M_j de EM, et **si** il existe M_k dans EM tel que M_{ij} et M_k ($M_{ij} + M_k = M_{ijk}$) sont adjacents, alors M_k peut être remplacé dans EM par M_{ijk} (figure 34).

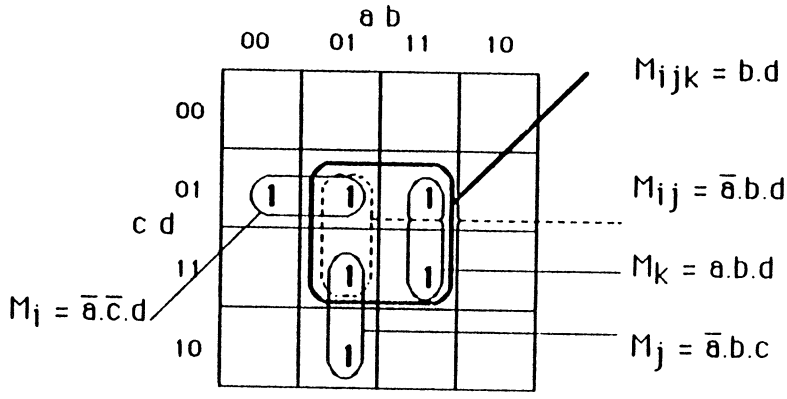


Figure 34 : Règle d'adjacence par consensus.

Remarque :

Par définition du consensus, M_{ij} est inclus dans $M_i + M_j$

donc $M_i + M_j + M_k = M_i + M_j + M_{ij} + M_k$

M_k et M_{ij} sont adjacents, donc $M_k + M_{ij} = M_{ijk}$

donc $M_i + M_j + M_k = M_i + M_j + M_{ijk}$

I.3.2.3 Propriété invariante dans l'application des règles.

Les règles 6 à 14 ont été établies dans le but d'effectuer la minimisation logique d'une fonction Booléenne complètement spécifiée d'une façon efficace en temps de calcul consommé et en occupation de place mémoire. La solution finale est obtenue dans EM lorsqu'il n'y a plus aucune règle applicable. Tout au long du processus de minimisation, l'assertion suivante est invariante :

- *Chaque fois qu'une règle de transformation est appliquée, la taille du problème diminue, soit par l'exclusion d'un monôme de l'ensemble EM, soit par l'exclusion d'une variable dans un monôme de EM.*

On peut donc affirmer que les place mémoire nécessaire à la réduction d'une expression est bornée par le nombre de monômes et par le nombre de variables dans les monômes du polynôme initial qui caractérise l'expression. Cette propriété nous permet d'installer le jeu de règles et le système qui contrôle leur application sur des ordinateurs dont la capacité mémoire est restreinte (micro ordinateurs).

I.3.2.4. Contrôle de l'application des règles:

Le coût de l'évaluation des parties gauches des règles n'est pas le même pour toutes les règles. Il est évident que les règles de consensus (12 à 14) sont beaucoup plus coûteuses à reconnaître que les règles primaires (6 à 11). Pour cette raison, le lecteur peut remarquer que les règles primaires d'adjacence (9) et d'expansion locale (11) sont des cas particuliers des règles de consensus ; nous les avons volontairement différenciées afin d'accroître l'efficacité du système. Ainsi, dans l'application des règles, les règles primaires sont prioritaires sur les règles secondaires car leur rapport **coût de l'évaluation** sur **gain apporté par la transformation** est meilleur.

L'efficacité de la minimisation peut encore être améliorée par l'introduction d'une **heuristique** dans le contrôle de l'application des règles. Cette heuristique consiste à faire en sorte que les premières règles appliquées entraînent les plus gros gains, c'est à dire des exclusions de monômes par adjacence ou bien par inclusion.

I.3.2.5. Exemple d'application de règles:

Nous donnons ici une manière possible de résoudre le problème de la preuve de tautologie pour la fonction de la figure 30. L'ordre d'application des règles dans cet exemple est tout à fait arbitraire et indifférent, le but recherché étant d'aboutir à l'application de la règle 6 ($a + \neg a \rightarrow 1$).

$f=\neg a.\neg b.\neg c+\neg a.b.\neg d+a.b.c+a.\neg b.d+a.\neg c.\neg d+b.\neg c.d+\neg a.c.d+\neg b.c.\neg d$

$f = \sum_{(i=1 \text{ à } 8)} M_i$.(figure 35)

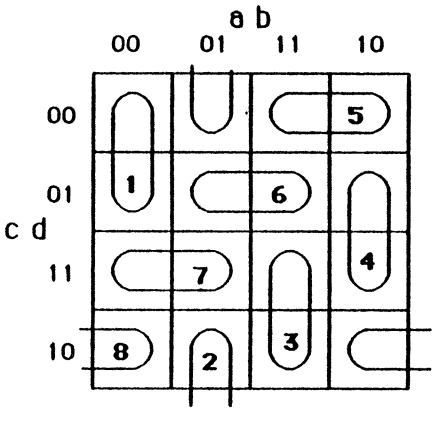


Figure 35 : forme initiale.

(M_1, M_2) → règle 14 : consensus (M_1, M_2)=$\neg a.\neg c.\neg d$ adjacent à M_5

M_5 expansé en $\neg c.\neg d$

(M_1, M_6) → règle 14 : consensus (M_1, M_6)=$\neg a.\neg c.d$ adjacent à M_7

M_7 est expansé en $\neg a.d$

(M_2, M_3) → règle 14 : consensus (M_2, M_3)=$b.c.\neg d$ adjacent à M_8

M_8 est expansé en $c.\neg d$

(M_3, M_6) → règle 14 : consensus (M_3, M_6)=$a.b.d$ adjacent à M_4

M_4 est expansé en $a.d$ (figure 36)

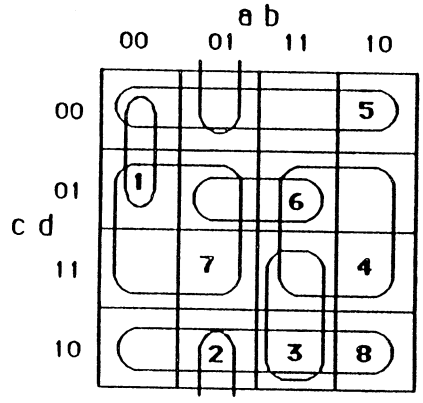


Figure 36 : forme obtenue après 4 transformations.

(M_2, M_7) → règle 11 : (expansion locale) M_2 est expansé en a.d

(M_4, M_5) → règle 13 : consensus (M_4, M_5)=a.c absorbe M_3=a.b.c

(M_5, M_7) → règle 13 : consensus (M_5, M_7)=¬a.¬c absorbe M_1=¬a.¬b.¬c

(M_4, M_7) → règle 9 : M_4 adjacent à M_7, M_4 est expansé en d, M_7 est supprimé.

(M_4, M_6) → règle 10 : M_6 inclus dans M_4, M_6 est supprimé.

(M_5, M_8) → règle 9 : M_5 adjacent à M_8

M_5 est expansé en ¬d, M_8 est supprimé. (figure 37)

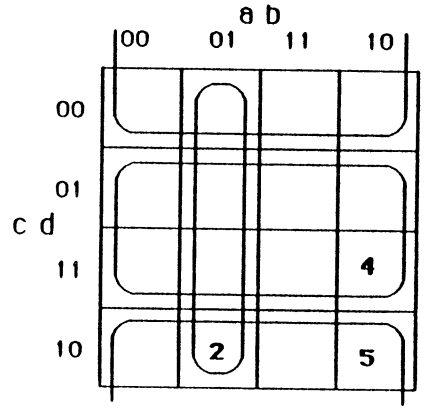


Figure 37 : forme obtenue après 6 transformations.

(M_4, M_5) → règle 6 : $M_4+M_5=1$, f est une **tautologie**.

I.3.3. La minimisation locale des fonctions partiellement spécifiées.

Les règles qui vont suivre sont des règles additionnelles aux règles 6 à 14, qui permettent la minimisation locale des fonctions phi-booléennes. Elles s'appliquent sur les monômes des couvertures à 1 et à phi de la fonction, mais ne modifient que la couverture à 1.

Soient EM = (M_1, M_2,...,M_p) un ensemble de monômes caractérisant la couverture à 1 de la fonction f à minimiser, et EMØ = ($MØ_1$,$MØ_2$,...,$MØ_q$) un ensemble de monômes caractérisant la couverture indifférente de f.

Règle 15 : règle d'adjacence phi-booléenne.

Si il existe une variable x et un monôme n tels que M_i=x.N et $MØ_j$=¬x.N, **alors** M_i est adjacent à $MØ_j$ et est remplacé dans EM par N (figure 38).

Le gain associé à l'application de cette règle est de 1 variable dans le monôme M_i.

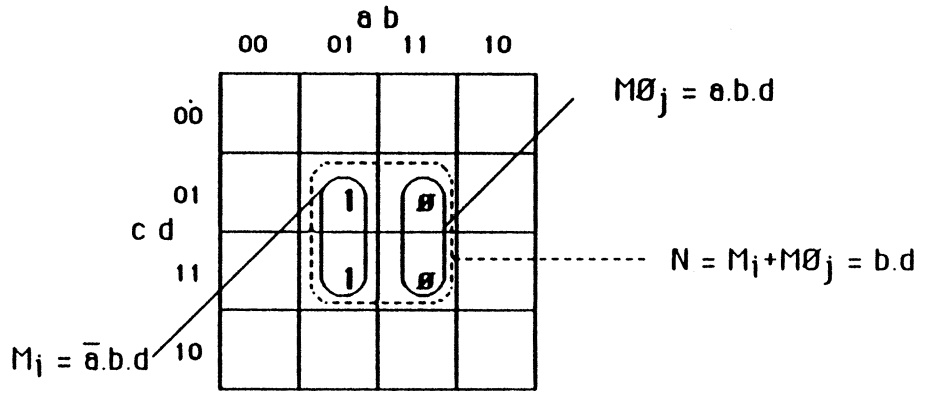


Figure 38 : règle d'adjacence phi-booléenne.

Règle 16 : règle d'expansion locale phi-booléenne.

si il existe une variable x et deux monômes N et P tels que M_i = ¬x.N.P et $MØ_j$ = x.P, alors la variable ¬x peut être supprimée dans M_i (¬a.b.c + a.c = b.c + a.c). (figure 39).

Le gain associé à l'application de cette règle est de 1 variable dans le monôme M_i.

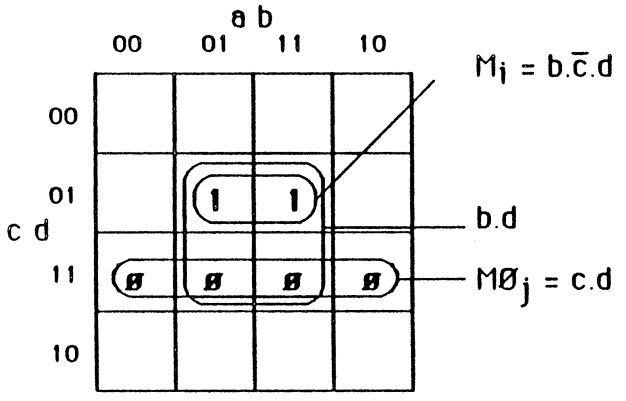


Figure 39 : règle d'expansion locale phi-booléenne.

Règle 17 : règle d'assignation à une tautologie.

Si $M_i = x$ et **si** $M\emptyset_j = \neg x$, **alors** f peut être assignée à une **tautologie** (figure 40).

L'application de cette règle stoppe le processus de minimisation phi-booléenne.

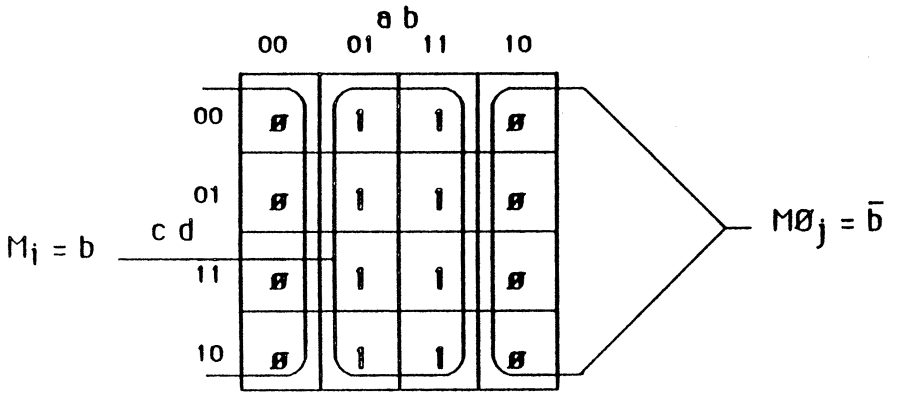


Figure 40 : Règle d'assignation à une tautologie.

Règle 18 : règle d'adjacence phi-booléenne par consensus.

Si il existe un consensus $M\emptyset_{ij}$ entre $M\emptyset_i$ et $M\emptyset_j$, et si il existe M_k dans EM tel que M_k et $M\emptyset_{ij}$ sont adjacents ($M\emptyset_{ij}+M_k=M_{ijk}$), alors M_k est remplacé dans EM par M_{ijk}.

Le gain associé à l'application de cette règle est de 1 variable dans M_k.

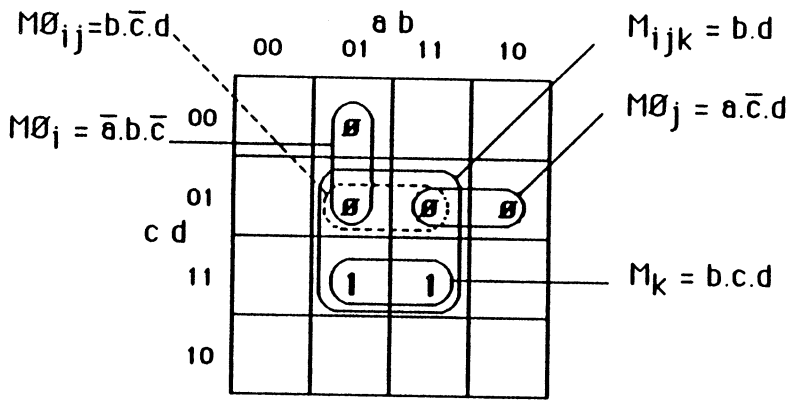


Figure 41 : règle phi-booléenne d'adjacence par consensus.

Remarques :

- Losqu'aucune des règles 6 à 18 n'est applicable sur les ensembles EM et EMØ d'une fonction phi-booléenne, la solution minimisée est dans EM et EMØ n'est pas modifié.

- Les règles phi-booléennes 15 à 18 respectent l'assertion définie en II.3.2.3.

- Les règles phi-booléennes 15 à 18 sont des règles complémentaires des règles 6 à 14, et doivent leur être associées pour être opérationnelles. En effet, les transformations provoquées par l'application des règles phi-booléennes peuvent rendre possible de nouvelles simplifications dans l'ensemble EM par les règles 6 à 14.

I.3.4. Un système de règles pour la minimisation globale.

La minimisation globale d'un système combinatoire (minimisation multi-fonction) est basée sur une observation simultanée de toutes les fonctions à implémenter. Le but recherché est de couvrir l'ensemble des fonctions avec un nombre minimal de monômes.

Classiquement, on procède par la génération de tous les monômes premiers pour chacune des fonctions, et pour toutes les combinaisons possibles de produits de fonctions. On sélectionne ensuite parmi les monômes premiers générés, une couverture première irrédondante, tout comme pour la minimisation locale. Pour minimiser un ensemble de p fonctions, il faut calculer les monômes premiers de :

$$\sum_{i=1}^{p} C_p^i \quad \text{fonctions.}$$

La génération des monômes premiers qui croît de façon exponentielle ($3^n/n$) avec le nombre de variables, croît aussi de façon exponentielle avec le nombre de fonctions, car :

$$\sum_{i=1}^{p} C_p^i = 2^p - 1$$

Ainsi, ces méthodes ne peuvent pas traiter des problèmes dont la taille dépasse une huitaine de fonctions [ARE78]. Notre objectif étant de minimiser des systèmes combinatoires de 50 variables et de plus de 100 fonctions, nous avons défini un système de règles qui constitue une approche efficace pour la minimisation globale, en garantissant l'obtention d'une solution sans redondance mais pas nécessairement optimale au sens du nombre total de monômes [HAN86].

Les règles pour la minimisation globale d'un ensemble de fonctions Booléennes ont été définies de façon à engendrer une solution globalement optimisée à partir des formes localement minimales de chacune des fonctions du système. Cette manière de procéder est particulièrement efficace lorsque l'on peut faire apparaître dans plusieurs fonctions des mêmes sous expressions (polynômes) localement optimisées. Dans le cas contraire, on est amené à expanser l'ensemble EM des monômes obtenus par

minimisation locale des fonctions, en lui ajoutant des monômes générés par intersections généralisées, rendant ainsi plus complexe et coûteuse la sélection d'une couverture minimale irrédondante. Ainsi, le système de règles que nous proposons est peu adapté au traitement des fonctions dont la forme globalement optimale est presque canonique (monômes de petite taille et presque tous disjoints) et dont les formes localement minimales ne le sont pas (monômes de grosse taille et peu disjoints). En effet, dans ce type de problème, les monômes intéressants pour la forme globalement optimale n'apparaissent pas dans les formes localement minimales, et doivent être retrouvés par des intersections généralisées.

L'intérêt d'une approche à base de règles pour la minimisation Booléenne est que l'on peut définir des jeux de règles différents pour les divers types de systèmes combinatoires à minimiser. Il serait tout à fait envisageable de définir un jeu de règles adapté à la minimisation des systèmes dont la forme optimale est presque canonique. Il suffirait alors d'analyser le système combinatoire initial pour déterminer le jeu de règles le plus adapté à sa minimisation.

Soit un système combinatoire $\mathbf{F}=(f_1, f_2,\ldots,f_m)$ à minimiser globalement. On note $\mathbf{C1}(f_k)$ l'expression de la couverture à "1" de la fonction f_k et $\mathbf{C\emptyset}(f_k)$ celle de sa couverture indifférente. Contrairement à la minimisation locale, les règles proposées ne font plus la distinction entre fonctions complètes et fonctions incomplètes.

Soit $\mathbf{EM}=(M_1, M_2,\ldots,M_p)$ l'ensemble initial des **p** monômes obtenus en minimisant localement les expressions des couvertures à "1" et à "ø" de chacune des fonctions de F.

Soit $\mathbf{EA}=(A_1, A_2,\ldots,A_p)$ l'ensemble associé à **EM** des apparitions d'un monôme dans F, définies par :

- $A_i = (A1_i, A\emptyset_i)$
- $A1_i$ est l'ensemble des fonctions f_k de F telles que M_i est inclus dans $C1(f_k)$.
- $A\emptyset_i$ est l'ensemble des fonctions f_k de F telles que M_i est inclus dans $C1(f_k) \cup C\emptyset(f_k)$.

I.3.4.1. Règles d'expansion globale:

Les règles d'expansion globale sont les règles génératrices des ensembles $A1_i$ et $A\emptyset_i$ pour chaque monôme M_i de EM. L'application de ces règles à un monôme M_i permet de déterminer l'ensemble des fonctions f_k de F dans lesquelles M_i peut apparaître, soit en tant que contribution à la couverture à "1" (M_i inclus dans $C1(f_k)$), soit en tant que contribution à l'union des couvertures à "1" et à "ø" (M_i inclus dans $C1(f_k) \cup C\emptyset(f_k)$).

Règle 19 : règle d'expansion globale.

Si il existe M_i dans EM et f_k dans F tels que

M_i est inclus dans $C1(f_k)$,

alors f_k est rajoutée dans $A1_i$.

Règle 20 : règle d'expansion globale phi-booléenne.

Si il existe M_i dans EM et f_k dans F tels que

M_i n'est pas inclus dans $C1(f_k)$,

et M_i est inclus dans $C1(f_k) \cup C\emptyset(f_k)$,

alors f_k est rajoutée dans $A\emptyset_i$.

Exemple:

Soit **F**=(f_1, f_2, f_3) un ensemble de trois fonctions des variables **a**, **b**, **c** et **d**. Les formes localement minimales obtenues par applications des règles de minimisation locale sont représentées en figure 42.

$C1(f_1) = b.\neg d + a.b + \neg a.c.d$ $\qquad C\emptyset(f_1) = 0$

$C1(f_2) = b.\neg c.\neg d + \neg a.b.c + a.\neg b.d + \neg a.c.d$ $\qquad C\emptyset(f_2) = 0$

$C1(f_3) = b.c + a.d + c.d$ $\qquad C\emptyset(f_3) = 0$

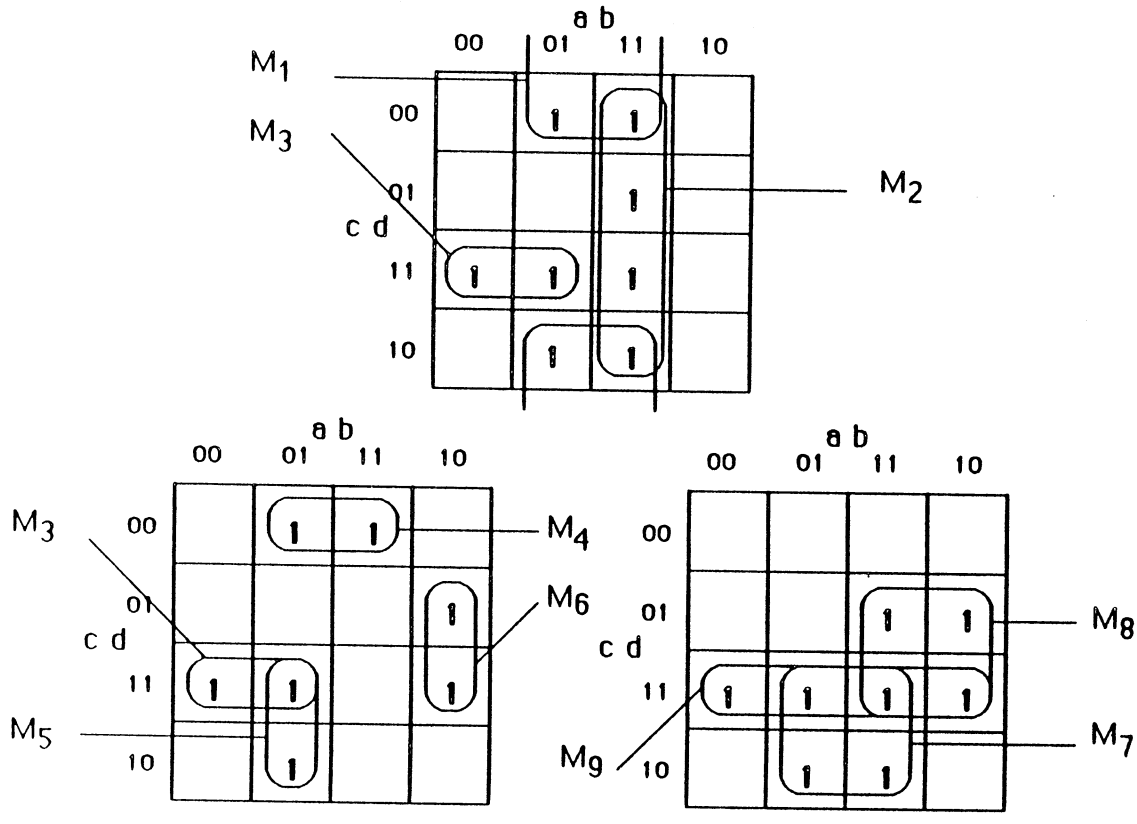


Figure 42 : Formes localement minimales de f_1, f_2, f_3.

L' ensemble initial de monômes EM est donc :

EM = (b.¬d, a.b, ¬a.c.d, b.¬c.¬d, ¬a.b.c, a.¬b.d, b.c, a.d, c.d)

L'application des règles d'expansion globale va constituer l'ensemble EA de la façon suivante (le symbole ≤ représente l'inclusion) :

$M_1 \leq C1\ (f_1)$	$\mathbf{A1_1} = (f_1)$
$M_2 \leq C1\ (f_1)$	$\mathbf{A1_2} = (f_1)$
$M_3 \leq C1\ (f_1), M_3 \leq C1\ (f_2), M_3 \leq C1\ (f_3)$	$\mathbf{A1_3} = (f_1, f_2, f_3)$
$M_4 \leq C1\ (f_1), M_4 \leq C1\ (f_2)$	$\mathbf{A1_4} = (f_1, f_2)$
$M_5 \leq C1\ (f_1), M_5 \leq C1\ (f_2), M_5 \leq C1\ (f_3)$	$\mathbf{A1_5} = (f_1, f_2, f_3)$
$M_6 \leq C1\ (f_2), M_6 \leq C1\ (f_3)$	$\mathbf{A1_6} = (f_2, f_3)$

$M_7 \leq C1\ (f_1),\ M_7 \leq C1\ (f_3)$ $\quad$ $\mathbf{A1_7} = (f_1, f_3)$

$M_8 \leq C1\ (f_3)$ $\quad$ $\mathbf{A1_8} = (f_3)$

$M_9 \leq C1\ (f_3)$ $\quad$ $\mathbf{A1_9} = (f_3)$

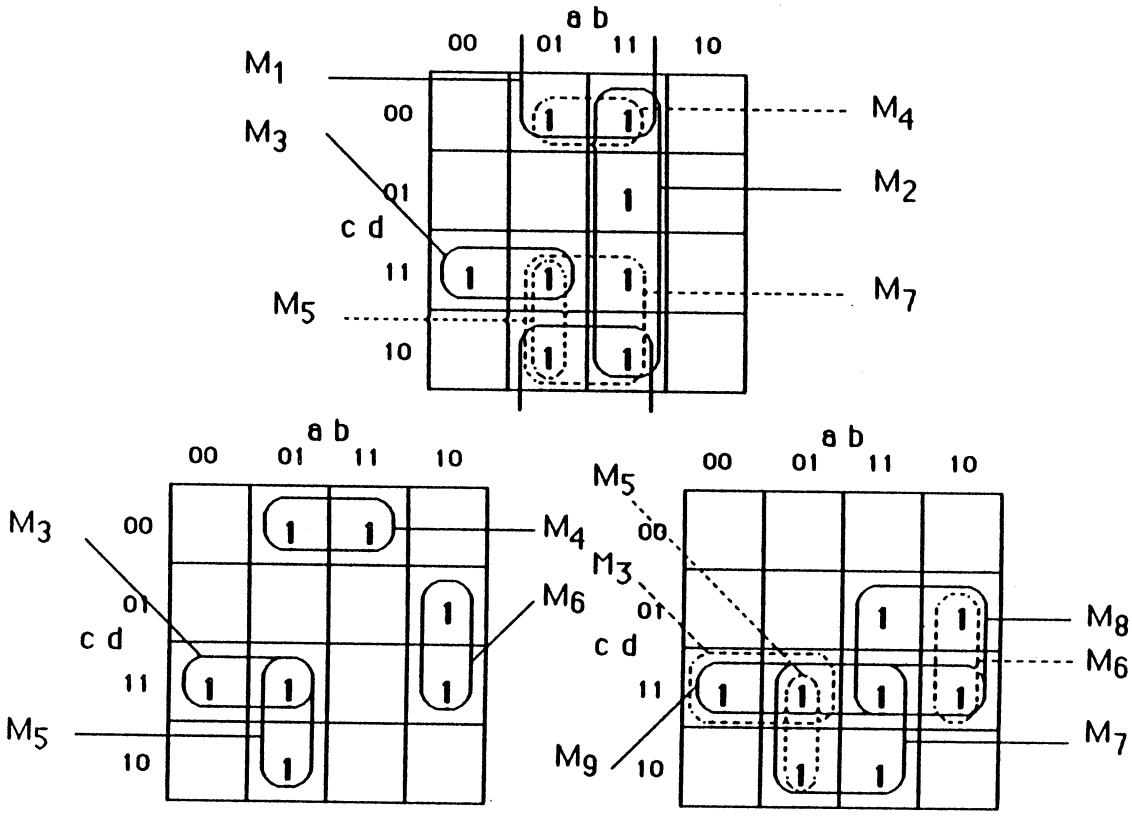


Figure 43 : formes globalement expansées de f_1, f_2, f_3.

I.3.4.2. Règle d'intersection généralisée:

La règle d'intersections généralisées est une règle de production dont le rôle est d'engendrer de nouveaux monômes susceptibles d'être plus intéressants (monômes pouvant apparaître dans un plus grand nombre de fonctions) dans la synthèse que ceux contenus initialement dans l'ensemble EM. Cette règle est expansive dans le sens où elle est la seule règle dont l'application fait croître la taille de l'ensemble EM, rendant ainsi l'extraction d'une couverture minimale plus complexe. De ce fait, elle doit être guidée par une heuristique de manière à limiter l'expansion de EM. La règle

d'intersections généralisées s'appuie sur la propriété suivante :

* Propriété :

Soit M_i et M_j deux monômes de EM, tels que : $M_i.M_j \neq 0$.

Si on note M_k le produit de M_i par M_j, alors

$A1k \subset A1i \cup A1j$.
$A\emptyset k \subset A\emptyset_i \cup A\emptyset k$. (figure 44)

Preuve :

Soit $f_k \in A1_i \cup A1_j$

on a: $M_i \leq C1\ (f_k)$ et $M_j \leq C1\ (f_k)$.

donc $M_i.M_j \leq C1\ (f_k)$

donc $f_k \in A1_k$.

Soit $f_k \in A\emptyset_i \cup A\emptyset_j$

on a: $M_i \leq C\emptyset\ (f_k)$ et $M_j \leq C\emptyset\ (f_k)$.

donc $M_i.M_j \leq C\emptyset\ (f_k)$

donc $f_k \in A\emptyset_k$.

Règle 21 : règle d'intersections généralisées.

Si il existe $M_k = \prod_{M_i \in P(EM)} M_i \neq 0$

alors $A1_k \subset \bigcup_{M_i \in P(EM)} A1_i$ et $A\emptyset_k \subset \bigcup_{M_i \in P(EM)} A\emptyset_i$

et après expansion des ensembles $\mathbf{A1_k}$ et $\mathbf{A\emptyset_k}$ par applications des règles 19 et 20,

$\mathbf{M_k}$ peut être ajouté à **EM**.

et $\mathbf{A_k} = (\mathbf{A1_k}, \mathbf{A\emptyset_k})$ peut être ajouté à **EA**.

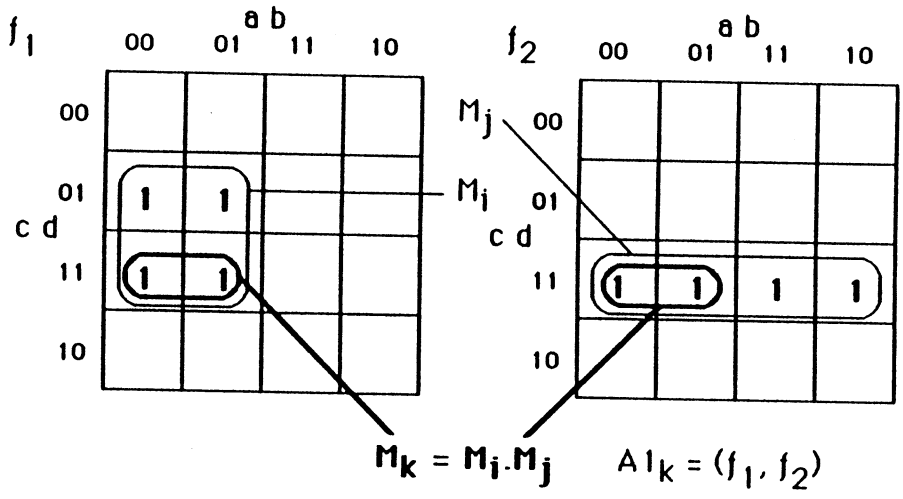


Figure 44 : Illustration de la propriété d'intersection

Dans notre exemple (figures 42 et 43), la règle d'intersections généralisées va engendrer les monômes suivants :

$M_{10} = M_1.M_7$ $\quad A1_{10} = (f_1, f_3)$

$M_{11} = M_2.M_8$ $\quad A1_{11} = (f_1, f_3)$

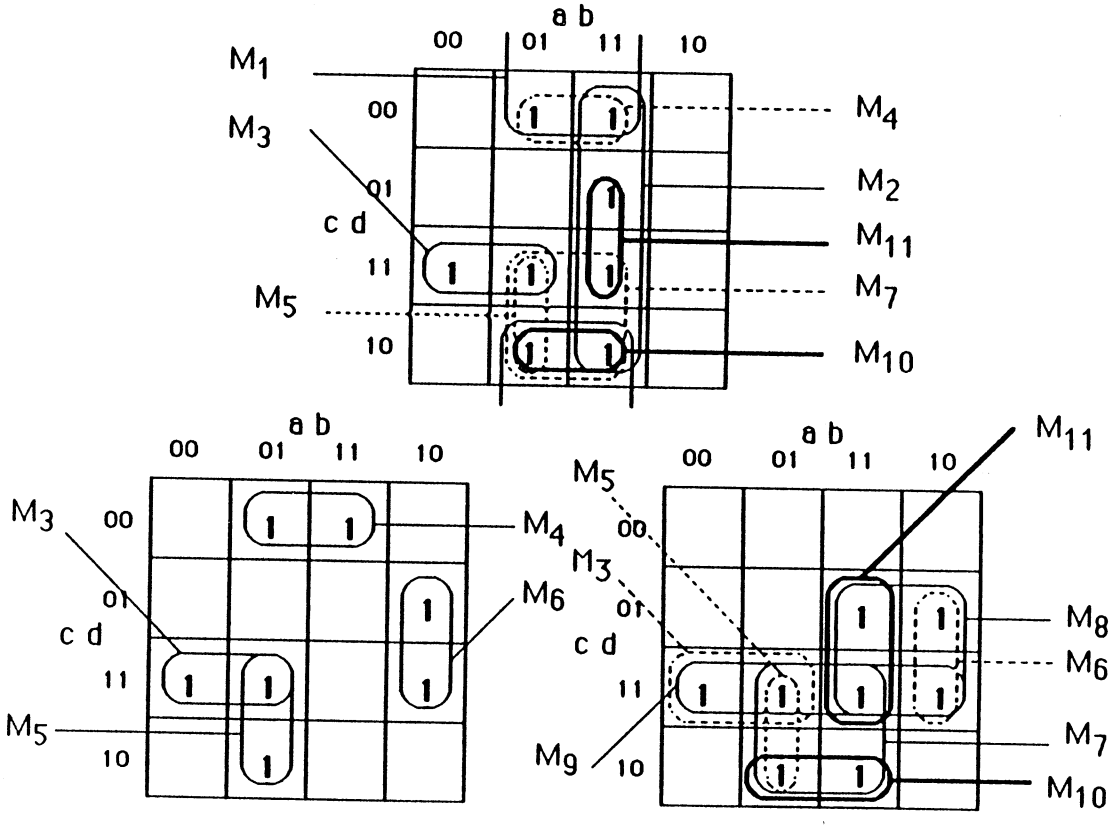


Figure 45 : Forme de F après intersections généralisées.

I.3.4.3. Règles d'élimination de la redondance :

Les règles d'élimination de redondance sont celles qu'on applique pour la sélection d'une couverture irrédondante à partir des monômes de l'ensemble EM. Ces règles sont au nombre de trois : les deux premières interviennent pour tester la redondance locale d'un monôme dans une fonction f_k, et la dernière intervient pour tester la redondance globale d'un monôme dans l'ensemble de toutes les fonctions $F=(f_1,f_2,\ldots,f_m)$.

Règle 22 : règle de redondance locale.

si il existe M_i dans EM et f_k dans F tels que :

$$f_k \in A1_i, \text{ et}$$

$$M_i \subset \sum_{\substack{j=1 \\ j\neq i}}^{p} M_j \,/\, f_k \in A1_j$$

alors M_i est localement redondant dans f_k, et peut être supprimé de $A1_i$.

Exemple :

Dans notre exemple $F = (f_1,f_2,f_3)$, la règle 22 pourrait s'appliquer sur le terme M_7 dans la fonction f_1.

$f_1 \in A1_7$ et $M_7 \leq M_1+M_2+M_3+M_4+M_5$. (figure 46)

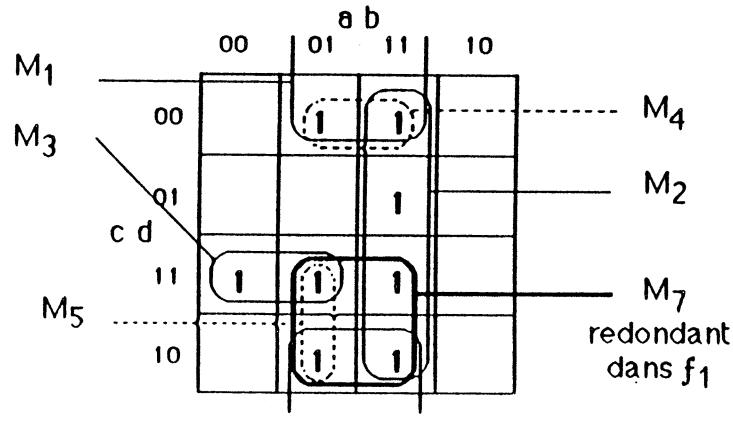


Figure 46 : Redondance locale d'un monôme dans une fonction.

Règle 23 : règle de redondance locale phi-booléenne.

Si il existe M_i dans EM et f_k dans F tels que :

$$f_k \in A1_i$$

$$M_i \not\subset \sum_{\substack{j=1 \\ j\neq i}}^{p} M_j \,/\, f_k \in A1_j \quad \text{et} \quad M_i \subset \sum_{\substack{j=1 \\ j\neq i}}^{p} M_j \,/\, f_k \in A1_j \cup A\emptyset_j$$

alors M_i peut être rendu localement redondant dans F à condition d'assigner à "1" les monômes indifférents qui intersectent avec M_i, c'est à dire à condition de faire :

pour tout *Mj tel que* ($f_k \in A\emptyset_j$) *et* ($M_j . M_i \neq 0$) ***faire***

début

$A\emptyset_j := A\emptyset_j - \{f_k\}$;

$A1_j := A1_j + \{f_k\}$;

fin ;

Exemple :

Considérons une fonction f et un ensemble de 5 monômes $M_1,\ldots,M_5$ tels que :

$M_1 \leq C1\ (f)$

$M_2 \leq C1\ (f)$

$M_3 \leq C1\ (f)$

$M_4 \leq C1\ (f) \cup C\emptyset\ (f)$

$M_5 \leq C1\ (f) \cup C\emptyset\ (f)$ (figure 47)

La règle 23 peut s'appliquer sur le monôme M_2 à condition d'inclure le monôme M_5 dans la couverture à "1" de f. (figure 48)

En effet,

$M_2 \subset M_1 + M_3$ et $M_2 \not\subset M_1 + M_3 + M_4 + M_5$

et $M_2 . M_5 \neq 0$.

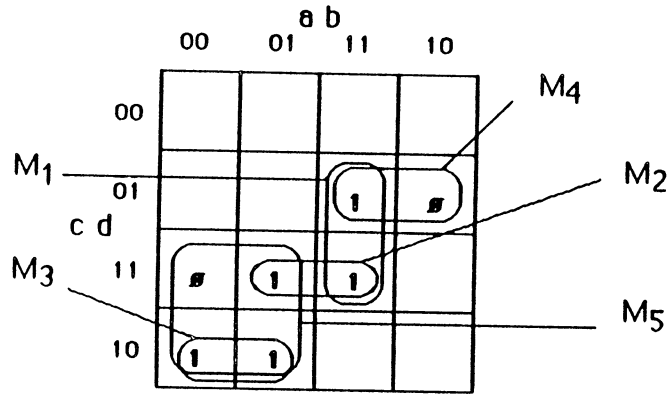


Figure 47.

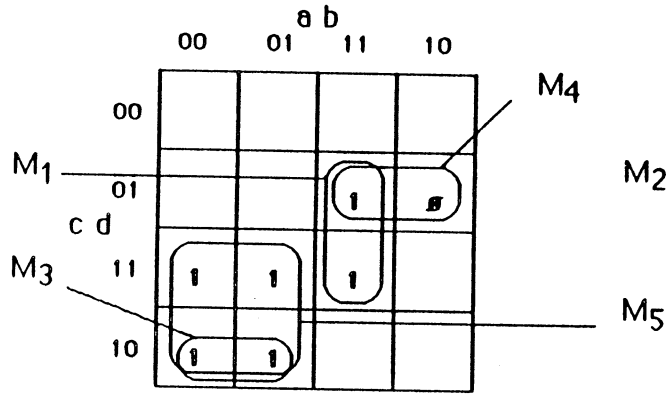


Figure 48.

Règle 24 : <u>règle de redondance globale.</u>

Si il existe M_i dans EM tel que

$\forall f_k \in A1_i$, M_i est ou peut être rendu localement redondant dans f_k,

alors M_i est globalement redondant dans F, et peut être supprimé de EM.

I.3.4.4. Contrôle de l'application des règles :

La procédure de minimisation globale par les règles qui viennent d'être énoncées, se décompose en quatre étapes consécutives :

1/ Expansion globale :

Partant d'un ensemble initial de monômes EM (obtenu par minimisation locale des fonctions ou non), on applique les règles d'expansion globale de façon à déterminer les ensembles $A1_i$ et $A\emptyset_i$ pour chaque monôme M_i de EM.

2/ Intersection généralisée :

Cette étape est optionnelle, mais nécessaire pour approcher un réel optimum global. Le fait d'appliquer la règle d'intersections généralisées fait croître l'ensemble EM et rend ainsi plus complexe la sélection d'une couverture minimale irrédondante. De toutes les façons, il est indispensable de définir une heuristique permettant d'estimer de manière prédictive, si un monôme généré par intersections multiples d'autres monômes de EM va être intéressant ou non. L'heuristique que nous proposons consiste à rejeter un monôme lorsque celui ci ne peut apparaître dans un nombre suffisant de fonctions relativement aux apparitions des monômes de EM dont il est le produit. On limite ainsi l'expansion en taille de l'ensemble EM par le rejet des monômes qui ne sont pas globalement intéressants.

3/ Elimination de la redondance globale :

Cette étape procède à la sélection d'une couverture minimale par la réduction du nombre total de monômes nécessaires pour assurer la couverture de toutes les fonctions de F. Cette étape est assumée par applications de la règle 24 sur chaque monôme de EM. L'ordre dans lequel on va tenter d'exclure les monômes est primordial : en effet, lorsqu'un monôme M_i de EM est déclaré globalement redondant dans F, il est supprimé de EM. Ainsi, certains monômes de EM peuvent devenir essentiels à la suite de l'exclusion d'autres monômes de EM. Il est donc nécessaire de ne pas tenter d'exclure les monômes dans n'importe quel ordre. La réduction du nombre total de monômes est implicitement liée à la maximalisation du nombre de monômes communs à plusieurs des fonctions à réaliser. Partant de ce principe, nous avons choisi une heuristique qui consiste à tenter d'exclure de EM, les monômes M_i selon un ordre décroissant d'une fonction probabiliste $\mathbf{P(M_i)}$ définie par :

$P(M_i)$ est la probabilité estimée que M_i soit globalement redondant dans F,

$$P(M_i) = \prod_{f_k \in A1_i} P(M_i / f_k)$$

où $P(M_i / f_k)$ est la probabilité estimée que M_i soit localement redondant dans f_k. Dans une première approche, on peut estimer les $P(M_i / f_k)$ par une constante non nulle et strictement inférieure à 1. On se ramène alors à l'exclusion des M_i de EM selon un ordre croissant de leur fréquence d'apparition dans les couvertures à "1" des fonctions, c'est à dire du cardinal des $A1_i$.

4/ Elimination de la redondance locale :

Le critère de réduction lors de cette dernière étape n'est plus la réduction du nombre total de monômes, mais la réduction du nombre de fonctions dans lesquelles un monôme apparaît. Ceci revient à minimiser le nombre de connexions d'un monôme à une sortie du circuit, ou encore le nombre de transistors dans l'étage OU du PLA qui réalise F. L'élimination de la redondance locale se fait par applications de la règles 22. A l'issue de cette étape, on obtient une forme irrédondante de F.

Exemple de la figure 45 :

* *classement des monômes de EM selon leur fréquence d'apparition.*

M_1 $A1_1 = (f_1)$

M_2 $A1_2 = (f_1)$

M_8 $A1_8 = (f_3)$

M_9 $A1_9 = (f_3)$

M_4 $A1_4 = (f_1, f_2)$

M_6 $A1_6 = (f_2, f_3)$

M_7 $A1_7 = (f_1, f_3)$

M_{10} $A1_{10} = (f_1, f_3)$

M_{11} $A1_{11} = (f_1, f_3)$

M_3 $A1_3 = (f_1, f_2, f_3)$

M_5 $A1_5 = (f_1, f_2, f_3)$

* *élimination de la redondance globale.*

M_1 dans f_1 : $M_1 \leq M_4+M_{10}$

M_1 est globalement redondant dans F et supprimé de EM.

M_2 dans f_1 : $M_2 \leq M_4+M_{10}+M_{11}$

M_2 est globalement redondant dans F et supprimé de EM.

M_8 dans f_3 : $M_8 \leq M_9+M_6+M_7+M_{10}+M_{11}+M_3+M_5$

M_8 est globalement redondant dans F et supprimé de EM.

M_9 dans f_3 : $M_9 \leq M_6+M_7+M_{10}+M_{11}+M_3+M_5$

M_8 est globalement redondant dans F et supprimé de EM.

M_4 dans f_1 : $M_4 > M_7+M_{10}+M_{11}+M_3+M_5$

M_4 est essentiel car obligatoire dans f_1.

M_6 dans f_2 : $M_6 > M_4+M_3+M_5$

M_6 est essentiel car obligatoire dans f_1.

M_7 dans f_1 : $M_7 \leq M_4+M_{10}+M_{11}+M_3+M_5$

M_7 dans f_3 : $M_7 \leq M_6+M_{10}+M_{11}+M_3+M_5$

M_7 est globalement redondant dans F et supprimé de EM.

M_{10} dans f_1 : $M_{10} > M_4+M_{11}+M_3+M_5$

M_{10} est essentiel car obligatoire dans f_1.

M_{11} dans f_1 : $M_{11} > M_4+M_{10}+M_3+M_5$

M_{11} est essentiel car obligatoire dans f_1.

M_5 dans f_1 : $M_5 \leq M_4+M_{10}+M_{11}+M_3$

M_5 dans f_2 : $M_5 > M_4+M_6+M_3$

M_5 est essentiel car obligatoire dans f_2.

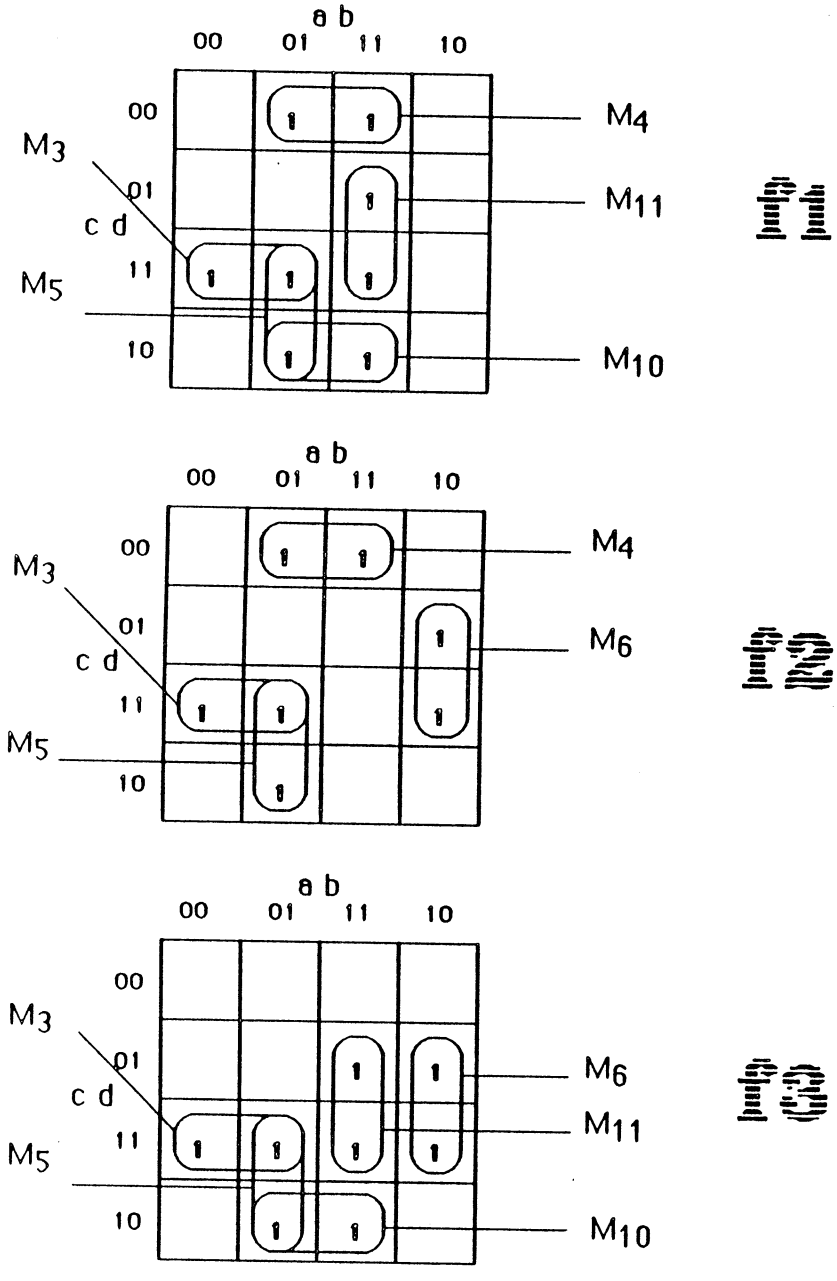


Figure 49 : forme de F après élimination de la redondance globale.

* *élimination de la redondance locale.*

M_5 dans f_1 : $M_5 \leq M_3+M_4+M_{10}+M_{11}$

M_5 est localement redondant dans f_1.

M_5 dans f_3 : $M_5 \leq M_3+M_6+M_{10}+M_{11}$

M_5 est localement redondant dans f_3.

La solution finale comprend donc les 6 monômes M_3, M_4, M_5, M_6, M_{10}, M_{11}.

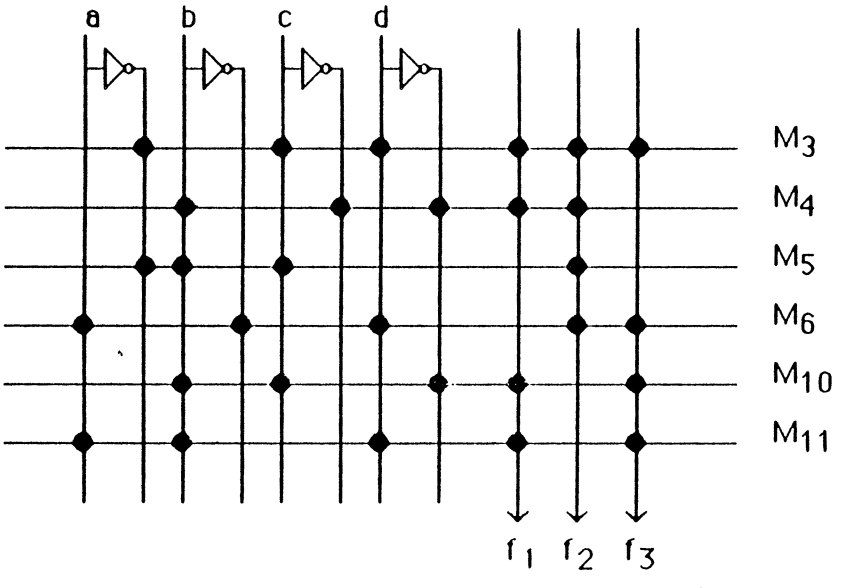


Figure 50 : PLA implémentant la forme minimale de F.

I.3.5. Le test d'inclusion : une approche formelle.

I.3.5.1. Approche classique pour le test d'inclusion:

Pour évaluer les parties gauches des règles énoncées en II.3.3., on a besoin de procéder au test d'inclusion d'un monôme dans une expression Booléenne. Classiquement, on procède au test d'inclusion d'un monôme M dans une expression E en vérifiant que tous les monômes canoniques inclus dans M apparaissent dans la première forme de Lagrange de E. Cette manière de tester l'inclusion impose que l'expression E soit transformée en sa forme canonique. Si l'expression E est donnée sous la forme d'un polynôme minimisé, on peut tester l'inclusion de M dans E en vérifiant que tous les monômes canoniques inclus dans M le sont aussi dans au moins un monôme de E. Cette méthode exige l'évaluation de 2^{n-p} tests élémentaires si **p** est le nombre de variables dans M et **n** le nombre total de variables dans E.

exemple :

Soit le polynôme E = ¬a.¬b + b.d + a.¬b.c, et le monôme M = c.d

On a p=2 et n=4, soit 4 tests élémentaires à faire :

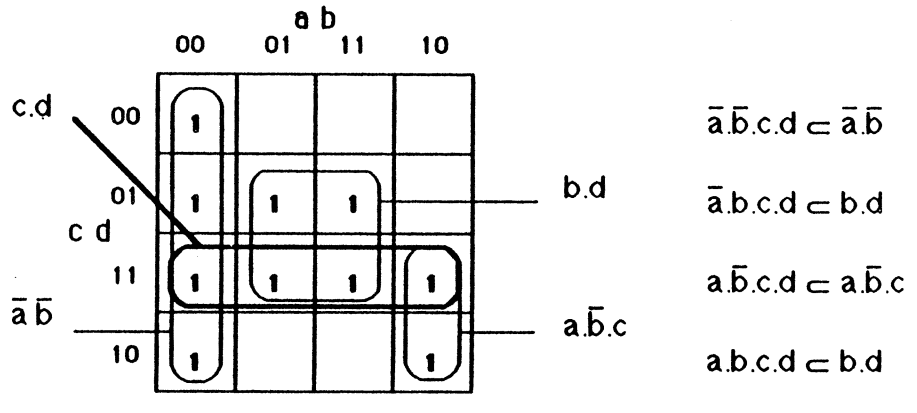


figure 51 : inclusion d'un monôme dans un polynôme.

Une extension de ces méthodes est proposée dans [BRO81] pour le programme PRESTO, qui consiste à couper le monôme M en deux monômes adjacents M_g et M_d, de façon récursive sur un arbre, jusqu'à ce que tous les monômes des feuilles de l'arbre soient inclus dans un monôme de E. Des critères heuristiques sont introduits

pour déterminer les lignes de coupe les plus favorables afin d'éviter de descendre dans l'arbre jusqu'au niveau canonique.

Pour l'exemple de la figure 51, on peut engendrer deux arbres différents suivant que l'on coupe le monôme **c.d** par les variables **a** puis **b** (figure 52) ou bien par les variables **b** puis **a** (figure 53).

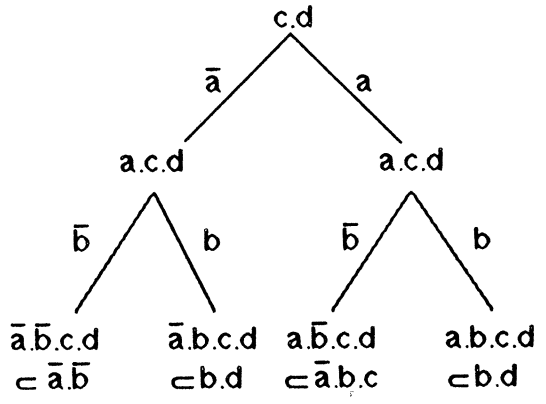


Figure 52 : arbre de test coupé par **a** puis **b**.

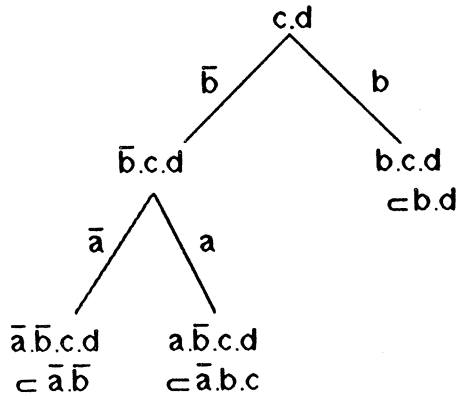


Figure 53 : arbre de test coupé par **b** puis **a**.

I.3.5.2. Le test d'inclusion par la preuve formelle de tautologie:

Nous présentons dans ce paragraphe une méthode basée sur la preuve formelle de tautologie, pour tester l'inclusion d'un monôme dans une expression. La preuve formelle de tautologie est faite ici à l'aide du système de règles pour la minimisation locale.

Définition.

Soient E une expression Booléenne et M un monôme. On appelle **expression unifiée de E sachant M**, notée E/M, l'expression E dans laquelle les variables de M sont assignées aux valeurs qui rendent M égal à "1".

Exemple :

$E = \neg a.b.\neg c + a.c.d$

$M = b.d$

$E/M = \neg a.b.\neg c + a.c.d \,/\, b = 1, d = 1$

$= \neg a.1.\neg c + a.c.1 = \mathbf{\neg a.\neg c + a.c}$

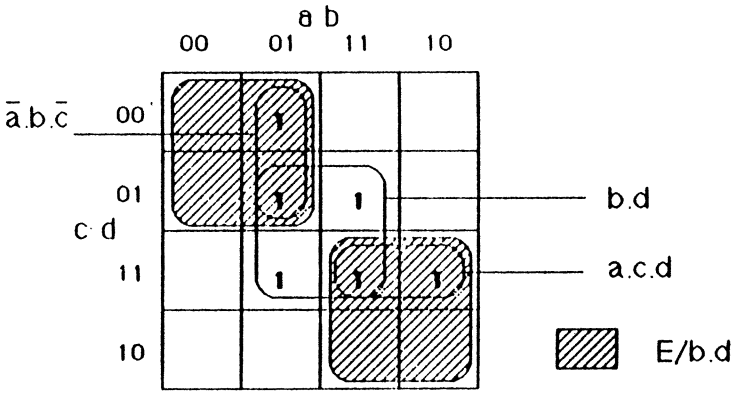


Figure 54 : expression unifiée de E sachant M.

Théorème :

Soient un monôme **M** et une expression Booléenne **E** ; pour que le test d'inclusion de **M** dans **E** soit positif, il suffit que l'expression unifiée **E/M** soit une **tautologie**.

* preuve:

$E/M = 1 \Longrightarrow M \subset E$

$\Longleftrightarrow M \not\subset E \Longrightarrow E/M \neq 1$

$\Longleftrightarrow$ il existe un monôme canonique

$MC \subset M$ et $MC \not\subset E \Longrightarrow E/MC \neq 1$

Soit **MC** un monôme canonique inclus dans **M** et non inclus dans **E**.
Puisque **MC** est canonique (taille (MC) = 1), on a **MC.E = 0**.

D'autre part (E1.E2)/M = E1/M . E2/M donc (E.M)/M = E/M . M/M = E/M

Ainsi E/MC = (E.MC)/MC
Or E.MC = 0
donc (E.MC)/MC = 0
donc E/MC = 0, E/MC ≠ 1

$$MC \subset M \text{ et } MC \not\subset E \Longrightarrow E/MC \neq 1$$

Exemple:

E = (a + b + ¬c).(¬a + b + c), M = b.d (figure 55)

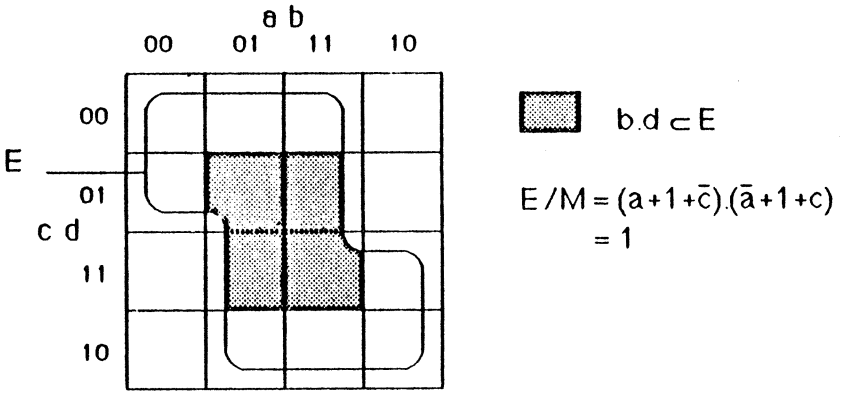


Figure 55.

Dans le système ASYL, on procède au test d'inclusion de M dans E en minimisant localement la forme polynômiale de l'expression unifiée E/M jusqu'à preuve formelle de tautologie.

$$E/M = 1 \equiv \left(\sum_{i=1}^{p} Mi \right) / M = 1$$

$$\equiv \sum_{i=1}^{p} (Mi/M) = 1$$

Exemple:

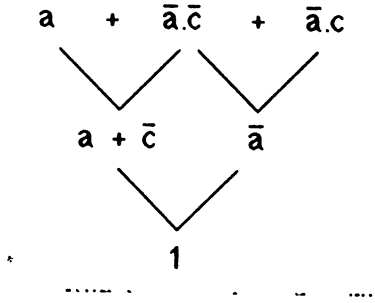
De l'exemple précédent, on peut extraire un polynôme décrivant E.

$E = a.b + a.\neg b.c + \neg a.\neg c + \neg a.b.c$

$M = b.d$

$E/M = a.1 + a.0.c + \neg a.\neg c + \neg a.1.c$

$= a + a.c + \neg a.\neg c + \neg a.c$



PARTIE II

LA SYNTHESE AUTOMATIQUE

DE CONTROLEUR

II. LA SYNTHESE AUTOMATIQUE DE CONTROLEUR.

Nous allons exposer les principes de synthèse automatique d'un contrôleur à partir d'une description de son comportement par un graphe de contrôle. Les problèmes de modélisation du comportement d'un contrôleur sont abordés en II.1., et les règles de conformité sont exposées en II.2. Le paragraphe II.3. est consacré à la génération des équations et à l'étude de deux types de synthèse : la synthèse en codage 1 parmi n, et la synthèse en codage compact. Le problème de l'assignation d'un codage optimisant les équations d'un contrôleur est présenté en II.4. Une démarche à base de règles est suggérée pour la production d'un codage optimisant la surface du contrôleur implanté à l'aide de PLA.

II.1. Modélisation d'un contrôleur.

Pour modéliser un contrôleur, on s'appuie sur la sémantique des graphes interprétés et temporisés [SIF82]. Un contrôleur est ainsi décrit comme un automate à états finis par son graphe d'évolution appelé graphe de contrôle. La description du graphe de contrôle est donné dans un langage adéquat ; pour notre part, il s'agit d'un langage issu du langage de description et de simulation fonctionnelles du système CADOC [CRA86] [BEL85]. L'intégration de l'outil de synthèse dans l'environnement du système CADOC offre au concepteur la possibilité de valider la description d'un contrôleur par simulation comportementale. Il peut ainsi suivre le comportement temporel du contrôleur dans le contexte des opérateurs à gérer. Ainsi, le processus de synthèse peut être appliquer sur un contrôleur à partir de sa description validée dans le contexte du circuit dans lequel il sera implanté.

Dans tout ce qui va suivre, nous considérons la synthèse automatique des systèmes séquentiels synchrones, dont l'évolution périodique est synchronisé sur une horloge externe. Les problèmes de synchronisation des contrôleurs dans leur contexte ont été largement étudiés dans [THU83]. Une approche formelle pour caractériser le fonctionnement temporel des automates ainsi que les hypothèses temporelles nécessaires au bon fonctionnement des contrôleurs synchronisés sont présentées dans [AMB84].

II.1.1. Description d'un contrôleur par un graphe de contrôle:

Soient $P = (P_1, P_2, ..., P_n)$ un ensemble de **n** places,

et $T = (t_1, t_2, ..., t_m)$ un ensemble de **m** transitions.

Soient α l'ensemble des arcs de **P x T** reliant une place à une transition,

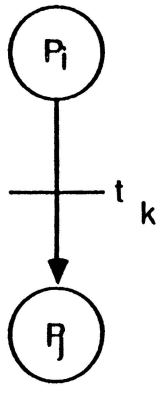
et β l'ensemble des arcs de **T X P** reliant une transition à une place.

A chaque transition t_i, est associé un prédicat de franchissement caractérisé par une expression Booléenne C_i.

$$C = (C_1, C_2, ..., C_m)$$

Pour construire un graphe de contrôle, on dispose de cinq sous graphes élémentaires.

** La séquence:*



$P = (P_i, P_j)$

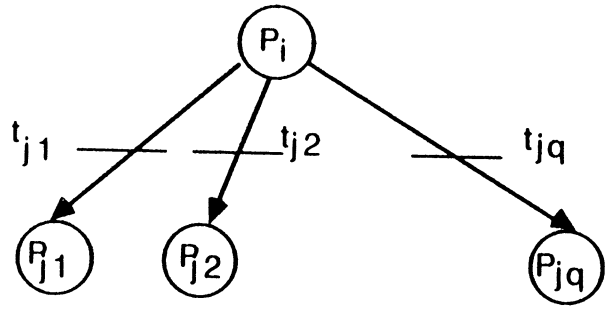
$T = (t_k)$

$C = (C_k)$

$\alpha = ((P_i, t_k))$

$\beta = ((t_k, P_j))$

* *La sélection:*



P = (P_i, P_{j1},...,P_{jq})

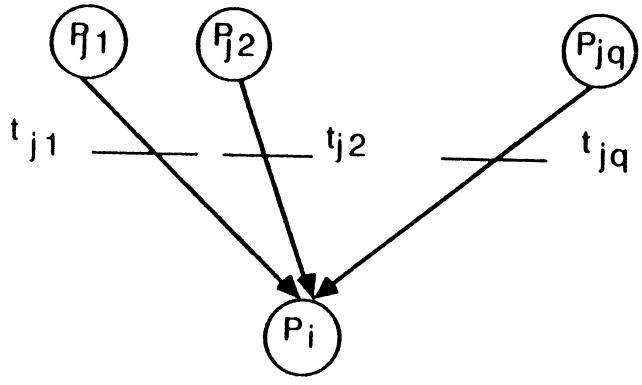
T = (t_{j1}, ..., t_{jq})

C = (C_{j1},...,C_{jq})

α = ((P_i, t_{j1}), (P_i, t_{j2}),..., (P_i, t_{jq}))

β = ((t_{j1}, P_{j1}), (t_{j2}, P_{j2}),..., (t_{jq}, P_{jq}))

* *L'attribution:*



P = (P_i, P_{j1},...,P_{jq})

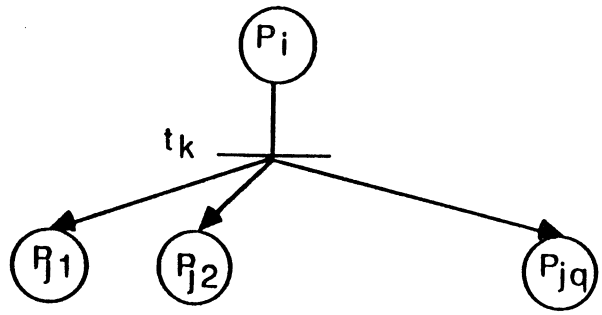
T = (t_{j1}, ..., t_{jq})

C = (C_{j1},...,C_{jq})

α = ((P_{j1}, t_{j1}), (P_{j2}, t_{j2}),..., (P_{jq}, t_{jq}))

β = ((t_{j1}, P_i), (t_{j2}, P_i),..., (t_{jq}, P_i))

** La distribution:*



P = (P_i, P_{j1},...,P_{jq})

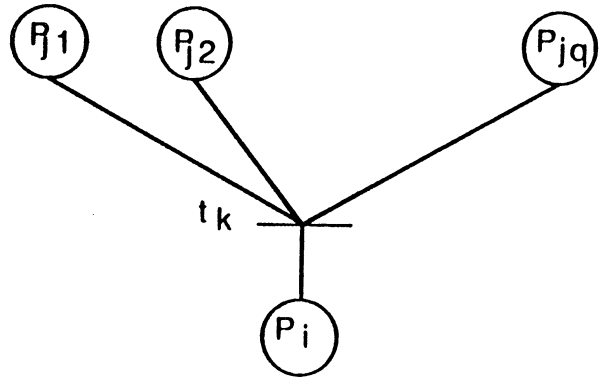
T = (t_k)

C = (C_k)

α = ((P_{j1}, t_k))

β = ((t_k, P_{j1}), (t_k, P_{j2}),..., (t_k, P_{jq}))

** La jonction:*



P = (P_i, P_{j1},...,P_{jq})

T = (t_k)

C = (C_k)

α = ((P_{j1}, t_k), (P_{j2}, t_k),..., (P_{jq}, t_k))

β = ((t_k, P_i))

Les sous graphes **distribution** et **jonction** servent à décrire des graphes de contrôle à évolution parallèle (plusieurs places actives à un même instant lors de l'interprétation). Un graphe de contrôle à évolution non parallèle (une seule place active à un instant donné de l'interprétation) est appelé graphe d'états d'un contrôleur. Dans un graphe d'états, il y a une équivalence directe entre les places du graphes et les états physiques du contrôleur.

II.1.2. Séquencement et émission des commandes:

Un graphe de contrôle tel qu'il a été introduit dans le paragraphe précédent ne caractérise que le séquencement du contrôleur, c'est à dire son évolution. La génération des commandes que doit émettre un contrôleur peut se définir sur un graphe de contrôle de plusieurs manières. Lorsque l'on associe les commandes aux places du graphe, on définit un contrôleur de **Moore**. Lorsque l'on associe les commandes à émettre aux transitions du graphe, on définit un contrôleur de **Mealey**. Dans un automate de Moore, les commandes ne dépendent pas des entrées externes du contrôleur, mais seulement de l'état de celui-ci.

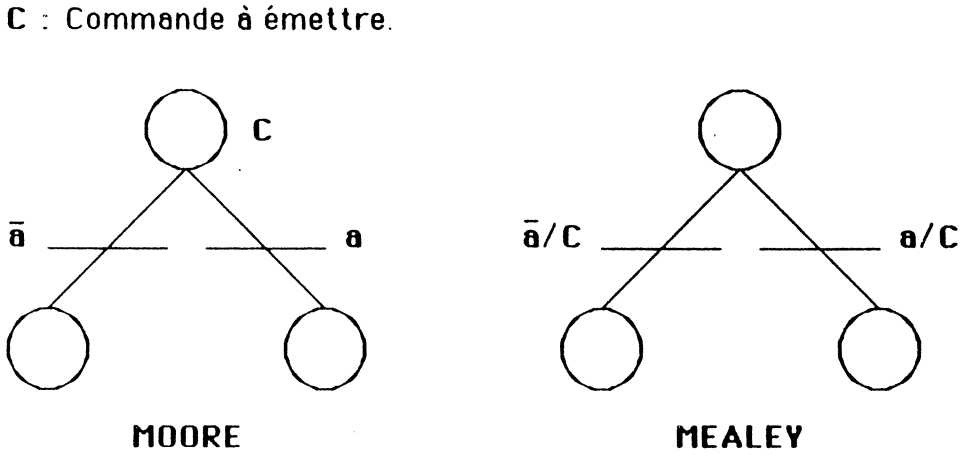


Figure 57 : émission des commandes dans un graphe de contrôle.

Pour faciliter la description des contrôleurs complexes, il peut être intéressant de

conditionner les commandes associées aux états dans un automate de Moore. On définit ainsi un troisième type de contrôleur que l'on appelle contrôleur de Moore à **affichage conditionnel**. Cette possibilité de décrire des contrôleurs de Moore à commandes conditionelles est offerte dans le système ASYL.

Exemple de description de contrôleur:

Nous prenons ici un exemple de contrôleur tiré de [BRO81]. Il s'agit d'un contrôleur de Mealey à deux entrées **x** et **y** et qui émet trois commandes **phase1**, **phase2**, et **errorout**. La forme graphique du contrôleur est donnée en figure 58 et sa description textuelle dans le langage de description fonctionnelle du système ASYL est la suivante :

```
graphe dac81 moore

entrées     x, y ;
sorties     phase1, phase2, errorout ;

état alpha ;
      ¬x : : gamma ;
      x  : : beta ;
état beta ;
      ¬y : phase1 : beta ;
      y : phase2 : alpha ;
état gamma ;
      x.y : : alpha ;
      non (x.y) : : error ;
état error ;
      vrai : errorout : error ;
fin
```

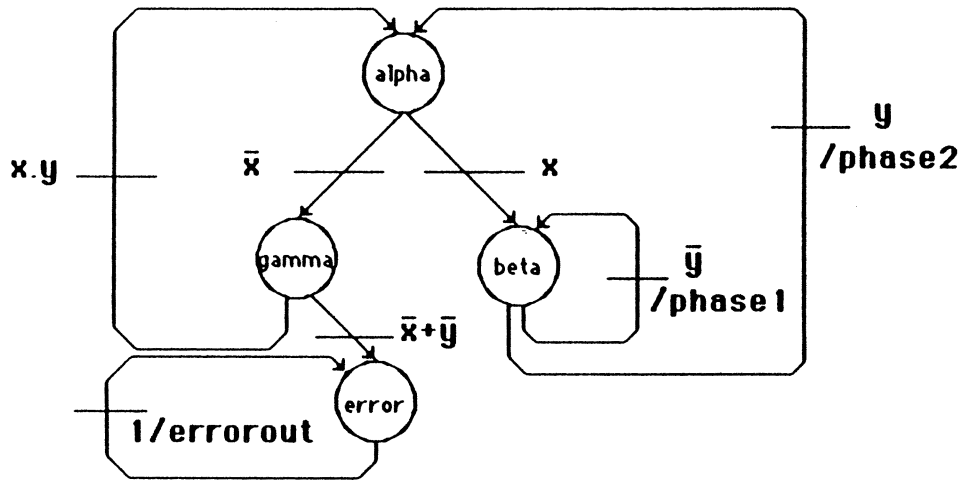


Figure 58 : description graphique d'un contrôleur de Mealey.

II.2. Conformité d'un contrôleur.

Pour qu'un graphe de contrôle soit synthétisable de façon déterministe et non ambiguë, indépendamment des méthodes de synthèse employées, il faut qu'il respecte certaines règles de conformité. Pour dire qu'un contrôleur est conforme et synthétisable, on vérifie qu'il ne peut présenter de conditions de blocage dans son évolution temporisée. Cette vérification peut se faire de manière formelle, comme elle peut se faire de façon empirique par simulation du contrôleur dans son environnement. La deuxième vérification à faire est un test de non parallélisme dans l'interprétation d'un graphe d'état, et doit se faire de manière formelle.

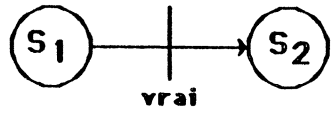
II.2.1. Blocage dans l'évolution d'un contrôleur:

Un blocage peut survenir dans l'évolution d'un contrôleur lorsqu'aucune transition en aval d'un état n'est franchissable à l'instant où les prédicats de franchissement associés aux transitions sont évalués. Dans le cas d'une synthèse de contrôleur synchrone, un blocage peut entraîner un comportement ambigu du contrôleur, qui peut entraîner des conséquences fâcheuses dans le comportement du circuit qu'il est sensé contrôler.

Nous pensons qu'il est indispensable de vérifier formellement l'absence de blocage dans un graphe de contrôle en faisant usage des règles de conformité suivantes :

1ere règle de conformité.

La condition de franchissement dans un sous graphe de **séquence** doit être inconditionnelle.



2ème règle de conformité.

Dans un sous graphe de **sélection**, la somme des conditions de franchissement des transitions en aval de l'état source doit être toujours vraie dans le contexte où elles sont évaluées (figure 59).

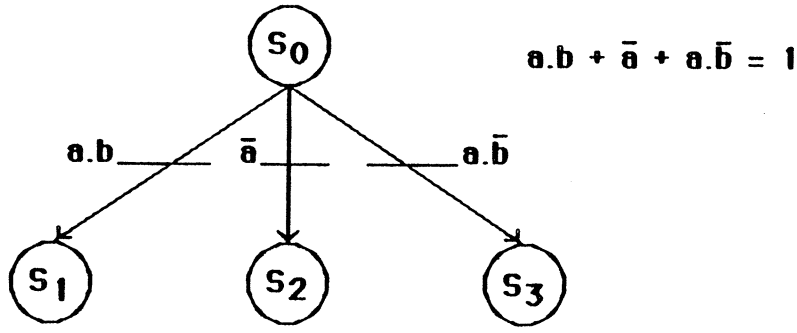


Figure 59 : condition d'absence de blocage dans un sous graphe de sélection.

Dans le système ASYL, on procède à une vérification formelle non contextuelle de conformité en vérifiant pour tous les motifs de sélection dans le graphe de contrôle, que la somme des conditions associées aux transitions est une tautologie. Dans le cas où cette règle de conformité n'est pas vérifiée, le concepteur en est avisé. Il doit alors s'assurer que les conditions de non déterminicité ne surviennent jamais dans le contexte où les prédicats de sélection sont évalués, ou bien, dans le cas contraire, il doit rajouter une ou plusieurs transitions dans la description du contrôleur afin de lever les ambiguités.

Exemple :

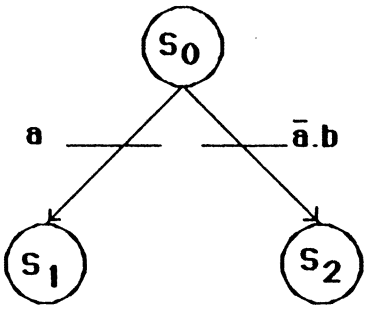


Figure 60 : motif de sélection ambiguë.

Si **a=b=0** lors de l'évaluation des conditions de franchissement, l'évolution du contrôleur est indéterminée. Si le concepteur souhaite spécifier une attente dans S_0 dans le cas où les deux conditions **a** et $\neg$**a.b** sont fausses, il doit l'indiquer explicitement dans la description (figure 62).

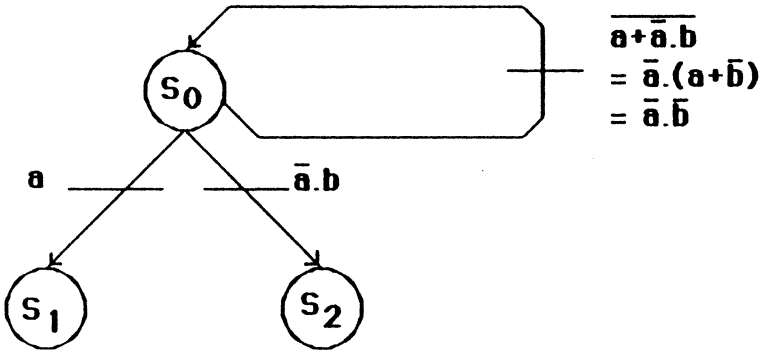


Figure 61 : motif de sélection non ambiguë.

II.2.2. Evolution parallèle dans un graphe d'états:

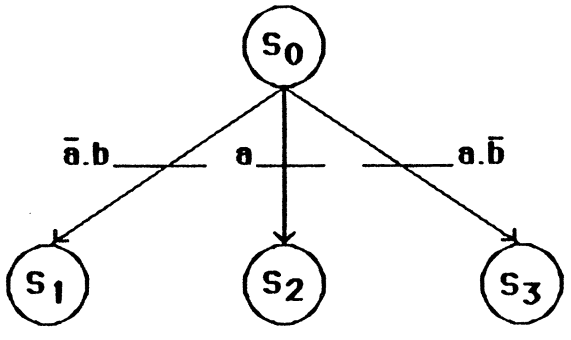
Lorsque l'on synthétise un contrôleur décrit par son graphe d'états, on fait une équivalence directe entre les places du graphe et les états de la machine physique qui implémente le contrôleur. Pour un état actif du contrôleur, il existe un unique état suivant, qui doit être calculé de façon déterministe par l'organe de séquencement. On a vu précédemment que les graphes d'états ne comportaient pas de sous graphes **distribution** et **jonction**. Néanmoins, les sous graphes **sélection** peuvent

engendrer un comportement parallèle dans l'interprétation du graphe, entraînant une évolution indéterminée du contrôleur implanté.

3ème règle de conformité.

Dans un graphe d'états, les conditions de franchissement associées aux transitions d'un motif de **sélection** doivent être mutuellement exclusives.

Exemple:



Les conditions de franchissement en amont des états S_2 et S_3 ne sont pas mutuellement exclusives car $a.(a.\neg b) \neq 0$.

Dans le système ASYL, on teste formellement qu'un graphe d'états n'évolue pas de façon parallèle en vérifiant pour chaque couple de transitions (t_i, t_j) d'un motif de sélection que $C_i.C_j=0$, ou bien que l'expression $\neg C_i+\neg C_j$ est une tautologie.

Lorsqu'un graphe d'états ne comporte pas de conditions de blocage et lorsqu'il a été prouvé formellement que son évolution ne peut être parallèle, alors on dit que le graphe est conforme et synthétisable.

II.3. La synthèse automatique d'un contrôleur décrit par un graphe d'états

Pour réaliser physiquement un contrôleur, il est nécessaire d'assigner un code binaire de caractérisation des états du graphe. Dans un contrôleur synchrone, l'état courant est maintenu pendant un cycle de l'horloge externe de synchronisation à

l'aide de points de mémorisation de différents types (bascules D, J/K, R/S, T, latch,...). A partir du graphe d'états et de la connaissance des codes attribués à chacun des états du graphe, on peut calculer de façon systématique les équations de génération des commandes (interface de commande) et les équations de calcul de l'état suivant (organe de séquencement).

II.3.1. La génération des équations:

Soit $S = (s_1, s_2, \ldots, s_n)$ l'ensemble des états du contrôleur. On admet qu'il existe une transition $t_{ij} = (s_i, s_j)$ dans le graphe d'états du contrôleur si et seulement si il existe une condition non nulle C_{ij} de franchissement depuis l'état s_i vers l'état s_j.

On associe à chaque état du graphe un code binaire sur **p** bits et on note $Y_1, \ldots, Y_p$ les variables d'état.

On dénote par **code** (s_i) le monôme de caractérisation du code binaire de l'état s_i, défini comme suit :

$$\text{code}(s_i) = \prod_{Y_j \in \{Y_i\}} Y_j \cdot \prod_{Y_k \notin \{Y_i\}} \bar{Y}_k$$

où $\{Y_i\}$ est l'ensemble des variables valant 1 dans le code de s_i .

II.3.1.1. Les équations des commandes:

Les équations des commandes diffèrent selon que l'on synthétise un contrôleur de Moore ou de Mealey.

Soit $O = (o_1, \ldots, o_q)$ l'ensemble des commandes que peut émettre le contrôleur.

* Contrôleur de Moore :

A chaque état s_i, est associé l'ensemble O_i inclus dans **O** des commandes émises lorsque s_i est actif.

Les équations des commandes se calculent de la manière suivante :

pour tout k ∈ [1,q]

$$o_k = \sum_{i \,/\, o_k \in O_i} code\,(s_i)$$

Remarque :

Dans le cas d'un automate de Moore à affichage conditionnel, il faut prendre en compte la condition de validation d'une commande en plus du code de l'état pour lequel la commande peut être émise.

* Contrôleur de Mealey :

A chaque transition (s_i,s_j) conditionnée par l'expression C_{ij}, est associé l'ensemble O_i inclus dans O des commandes émises lorsque (s_i,s_j) est franchie, c'est à dire lorque s_i est actif et C_{ij} est vraie.

Les équations des commandes se calculent de la manière suivante :

pour tout k ∈ [1,q]

$$o_k = \sum_{i \,/\, o_k \in O_i} code\,(s_i) \,.\, C_{ij}$$

II.3.1.2. Les équations de séquencement:

Les équations de séquencement ou des variables d'état ne dépendent par du modèle d'automate, mais seulement du type de registre utilisé pour la mémorisation de l'état.

Soient les ensembles suivants :

$A_{k\,(0\to0)}$: l'ensemble des transitions t_{ij} telles que la variable d'état Y_k vaut **0**

dans le code de s_i et vaut **0** dans le code de s_j.

A_k ($1 \rightarrow 1$) : l'ensemble des transitions t_{ij} telles que la variable d'état Y_k vaut **1** dans le code de s_i et vaut **0** dans le code de s_j.

A_k ($0 \rightarrow 1$) : l'ensemble des transitions t_{ij} telles que la variable d'état Y_k vaut **0** dans le code de s_i et vaut **1** dans le code de s_j.

A_k ($1 \rightarrow 0$) : l'ensemble des transitions t_{ij} telles que la variable d'état Y_k vaut **1** dans le code de s_i et vaut **0** dans le code de s_j.

Nous donnons les formules de génération des équations de séquencement pour quatre types de bascules : **D, J/K, R/S, et T.**

* Cas des bascules D :

Le fonctionnement logique de la bascule D est donné par le tableau suivant :

D	Q_t	Q_{t+1}
1	0	1
0	1	0
1	1	1
0	0	0

Q_t : état de la bascule avant échantillonnage de ses entrées.

Q_{t+1} : état de la bascule après échantillonnage de ses entrées.

L' équation de la variable Y_k se calcule de la manière suivante :

$$D\_Y_k = \sum_{(s_i,s_j) \in A_{k(0\rightarrow1)} \cup A_{k(1\rightarrow1)}} code\,(s_i) \cdot C_{ij}$$

* Cas de la bascule J/K:

Le fonctionnement logique d'une bascule J/K est donné dans le tableau suivant :

J	K	Q_t	Q_{t+1}
1	φ	0	1
φ	1	1	0
φ	0	1	1
0	φ	0	0

Comme on le remarque sur le tableau, les équations de séquencement pour des bascules J/K font apparaître des couvertures indifférentes contrairement aux bascules D. La prise en compte des couvertures indifférentes dans les équations générées peut permettre des simplifications supplémentaires lors de la minimisation logique. Il est important de remarquer que la bascule J/K (ou R/S) possède deux entrées contre une pour la bascule D (ou T). Ainsi, le PLA de séquencement pour des bascules J/K comprend deux fois plus de colonnes de sortie que celui pour des bascules D.

Les deux équations de la variable d'état Y_k se calculent de la manière suivante :

$$C1\,(J\_Y_k) = \sum_{(s_i,s_j) \in A_{k(0\to1)}} code\,(s_i) \cdot C_{ij}$$

$$C1\,(K\_Y_k) = \sum_{(s_i,s_j) \in A_{k(1\to0)}} code\,(s_i) \cdot C_{ij}$$

$$C\emptyset\,(J\_Y_k) = \sum_{(s_i,s_j) \in A_{k(1\to0)} \cup A_{k(1\to1)}} code\,(s_i) \cdot C_{ij}$$

$$C\emptyset\,(K\_Y_k) = \sum_{(s_i,s_j) \in A_{k(0\to1)} \cup A_{k(0\to0)}} code\,(s_i) \cdot C_{ij}$$

* Cas des bascules T:

Une bascule T est un cas particulier de la bascule J/K avec **J=K.** On en déduit donc le tableau suivant :

J	=	K	Q_t	Q_{t+1}
1		1	0	1
1		1	1	0
0		0	1	1
0		0	0	0

Les équations de séquencement pour des bascules T sont donc complètement spécifiées. L'équation de la variable d'état Y_k se calcule de la manière suivante :

$$T\_Y_k = \sum_{\substack{(s_i,s_j) \in A_{k(0\to1)} \\ \cup A_{k(1\to0)}}} code\,(s_i) \cdot C_{ij}$$

L'avantage à utiliser des bascules J/K ou T dans l'organe de séquencement d'un contrôleur réside dans le fait que les boucles d'attente sur un état n'apparaissent pas dans les équations des variables d'état. En effet, une boucle d'attente se traduit par des transitions de 1 vers 1 ou de 0 vers 0 pour toutes les variables d'état, lesquelles transitions n'interviennent pas dans les équations des bascules T ou interviennent de façon indifférentes dans les équations des bascules J/K. Il est donc intéressant d'utiliser des bascules T ou J/K pour synthétiser des contrôleurs dans lesquels il y a beaucoup de boucles d'attente. La figure 63 représente un graphe d'états comportant quatre boucles d'attente. Indépendamment de toute optimisation, on a besoin de 14 monômes pour caractériser toutes les transitions possibles dans les équations de séquencement pour des bascules D. Si l'on choisit des bascules T, les quatre transitions de rebouclage sont sans effet sur les équations, et il suffit donc de 7 monômes soit deux fois moins, pour réaliser le contrôleur.

De plus, les problèmes de blocage dans l'évolution d'un contrôleur évoqués en II.2.1. se traduisent par un comportement à caractère asynchrone, c'est à dire par une

attente dans l'état bloquant jusqu'à ce qu'une des conditions de franchissement devienne vraie. Les risques de comportement indéterminé du contrôleur sont donc réduits.

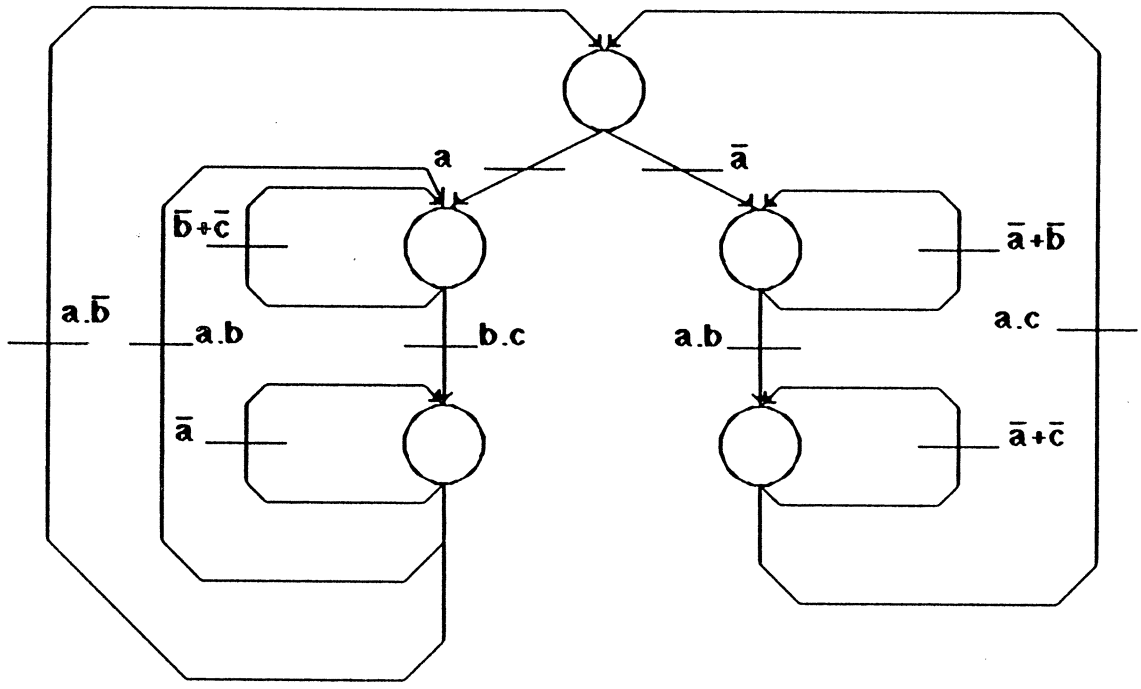


Figure 63 : graphe avec boucles d'attente.

* Cas des bascules R/S (Reset/Set):

Le fonctionnement logique de la bascule R/S est donné dans le tableau suivant :

R	S	Q_t	Q_{t+1}
0	1	0	1
1	0	1	0
0	φ	1	1
φ	0	0	0

On observe sur le tableau que le fonctionnement de la bascule R/S n'est pas défini pour **R=S=1**.

Les équations de la variable d'état Y_k se calculent de la manière suivante :

$$C1\,(R\_Y_k) = \sum_{(s_i,s_j) \in A_{k(1\rightarrow 0)}} code\,(s_i) \cdot C_{ij}$$

$$C1\,(S\_Y_k) = \sum_{(s_i,s_j) \in A_{k(0\rightarrow 1)}} code\,(s_i) \cdot C_{ij}$$

$$C\emptyset\,(R\_Y_k) = \sum_{(s_i,s_j) \in A_{k(0\rightarrow 0)}} code\,(s_i) \cdot C_{ij}$$

$$C\emptyset\,(S\_Y_k) = \sum_{(s_i,s_j) \in A_{k(1\rightarrow 1)}} code\,(s_i) \cdot C_{ij}$$

II.3.2. Implémentation des équations:

Les équations des commandes ainsi que celles des variables d'état sont implémentées par un ou plusieurs circuits combinatoires, et l'état du contrôleur est mémorisé à l'aide d'un registre de variables d'état. Dans le cas d'une implémentation sur PLA, on peut procéder à un découpage fonctionnel en séparant l'interface de commande et l'organe de séquencement. Il est aussi possible de synthétiser le contrôleur sur un seul PLA, puis d'appliquer des techniques de partitionnement [DAN86], créant ainsi plusieurs PLA, découpés de façon structurelle en limitant le nombre de monômes à dupliquer.

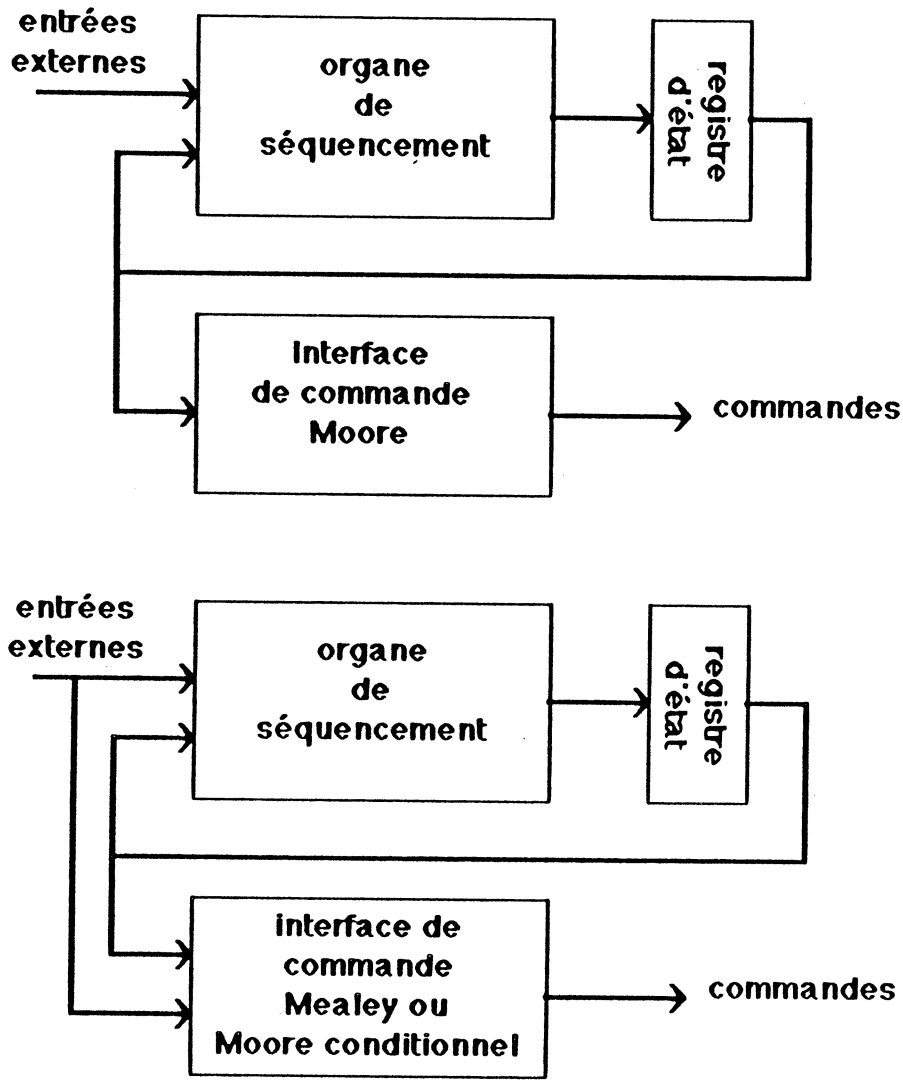


Figure 64 : schémas de contrôleurs de Moore et de Mealey.

II.3.3. La synthèse en codage 1 parmi n:

Cette méthode de synthèse s'appuie sur la représentation du code courant par une variable **n-valuée**, où **n** est le nombre total d'états dans le graphe. Une variable n-valuée V^n est caractérisée par un ensemble de **n** variables Booléennes simples $v_0, v_1, \ldots, v_{n-1}$ physiquement représentées par **n** points de mémorisation. Elle prend **n** valeurs possibles de **0** à **n-1** de la manière suivante :

$$V^n = i \Leftrightarrow v_i = 1 \text{ et } \forall j \neq i \; v_j = 0$$

Exemple:

On prend ici l'exemple classique du contrôleur de gestion de feu issu de [MEA80]. Nous avons choisi de décrire ce contrôleur par son graphe des états (figure 65) comme un automate de Moore à affichage conditionnel.

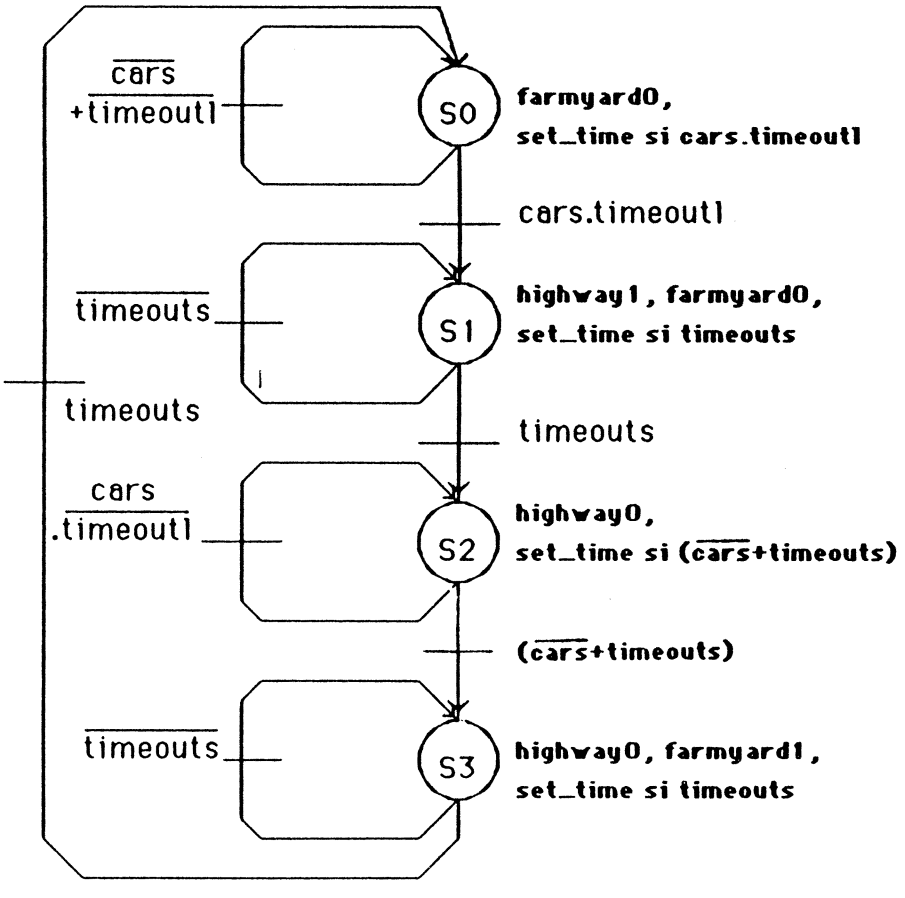


Figure 65 : contrôleur du système de gestion de feu.

Le graphe comprend 4 états codés à l'aide d'une variable multi-valuée (v_1,v_2,v_3,v_4) comme suit :

état	v^4	v_3	v_2	v_1	v_0
S0	0	0	0	0	1
S1	1	0	0	1	0
S2	2	0	1	0	0
S3	3	1	0	0	0

Dans le cas où le registre d'état est réalisé à l'aide de bascules D, les équations des variables d'état sont :

$$v_3 = \neg\text{timeouts}.\mathbf{v_3}.\neg v_2.\neg v_1.\neg v_0 + \neg\text{cars}.\ \neg v_3.\mathbf{v_2}.\neg v_1.\neg v_0$$
$$+ \text{timeoutl}.\neg v_3.\mathbf{v_2}.\neg v_1.\neg v_0$$

$$v_2 = \text{cars}.\neg\text{timeoutl}.\neg v_3.\mathbf{v_2}.\neg v_1.\neg v_0 + \text{timeouts}.\neg v_3.\neg v_2.\mathbf{v_1}.\neg v_0$$

$$v_1 = \neg\text{timeouts}.\neg v_3.\neg v_2.\mathbf{v_1}.\neg v_0 + \text{cars}.\text{timeoutl}.\neg v_3.\neg v_2.\neg v_1.\mathbf{v_0}$$

$$v_0 = \neg\text{cars}.\neg v_3.\neg v_2.\neg v_1.\mathbf{v_0} + \neg\text{timeoutl}.\neg v_3.\neg v_2.\neg v_1.\mathbf{v_0}$$
$$+ \text{timeouts}.\mathbf{v_3}.\neg v_2.\neg v_1.\neg v_0.$$

Les équations ci dessus ont été calculées selon les formules données en II.3.1. Elles sont justes mais redondantes. En effet, le fait d'assigner un codage **1 parmi n** fait qu'un état est caractérisé de façon unique par une et une seule variable simple $\mathbf{v_i}$. Si $\{V_i\}$ est l'ensemble des variables valant **1** dans le code de $\mathbf{s_i}$, on a la relation :

$$\forall\ \mathbf{i} \in\ \mathbf{[1,n],\ card\ (\{V_i\})\ =\ 1}$$

Ainsi, le monôme de caractérisation d'un code binaire défini précédemment par :

$$\mathbf{code\ (s_i)} = \prod_{V_j \in \{V_i\}} \mathbf{v_j} \cdot \prod_{V_k \notin \{V_i\}} \mathbf{\bar{v}_k}$$

peut être ramené à la définition suivante :

$$\mathbf{code}\ (s_i) = V_i$$

Dans notre exemple, les équations deviennent :

$$v_3 = \neg timeouts.\mathbf{v_3} + \neg cars.\ \mathbf{v_2} + timeoutl.\mathbf{v_2}$$

$$v_2 = cars.\neg timeoutl.\mathbf{v_2}.\neg v_1.\neg v_0 + timeouts.\mathbf{v_1}$$

$$v_1 = \neg timeouts.\mathbf{v_1} + cars.timeoutl.\mathbf{v_0}$$

$$v_0 = \neg cars.\mathbf{v_0} + \neg timeoutl.\mathbf{v_0} + timeouts.\mathbf{v_3}$$

Les équations des commandes sont les suivantes :

$$farmyard0 = \mathbf{v_0} + \mathbf{v_1}$$

$$farmyard1 = \mathbf{v_3}$$

$$highway0 = \mathbf{v_2} + \mathbf{v_3}$$

$$highway1 = \mathbf{v_1}$$

$$set\_time = cars.timeoutl.\mathbf{v_0} + timeouts.\mathbf{v_1} + (\neg cars + timeoutl).\mathbf{v_2} + timeouts.\mathbf{v_3}$$

Remarque sur la synthèse en codage **1 parmi n**:

De par l'assignation du codage, il n'est pas possible de minimiser logiquement les équations en vue d'une réalisation sur PLA. Ce type de synthèse ne convient donc qu'à une implantation en logique anarchique (logique aléatoire et points de mémorisation) soit manuelle (dessin au micron) soit automatisée sur réseaux prédiffusés par exemple (placement et routage du schéma logique). La synthèse en codage 1 parmi n engendre souvent des réalisations compactes lorsque le dessin du

schéma logique est fait manuellement (cas du microprocesseur Z8000 de ZILOG). Elle est néanmoins très coûteuse car difficile à mettre en œuvre par manque de régularité dans les schémas logiques, et par l'impossibilité de modifier les spécifications du contrôleur sans en refaire l'implantation complète.

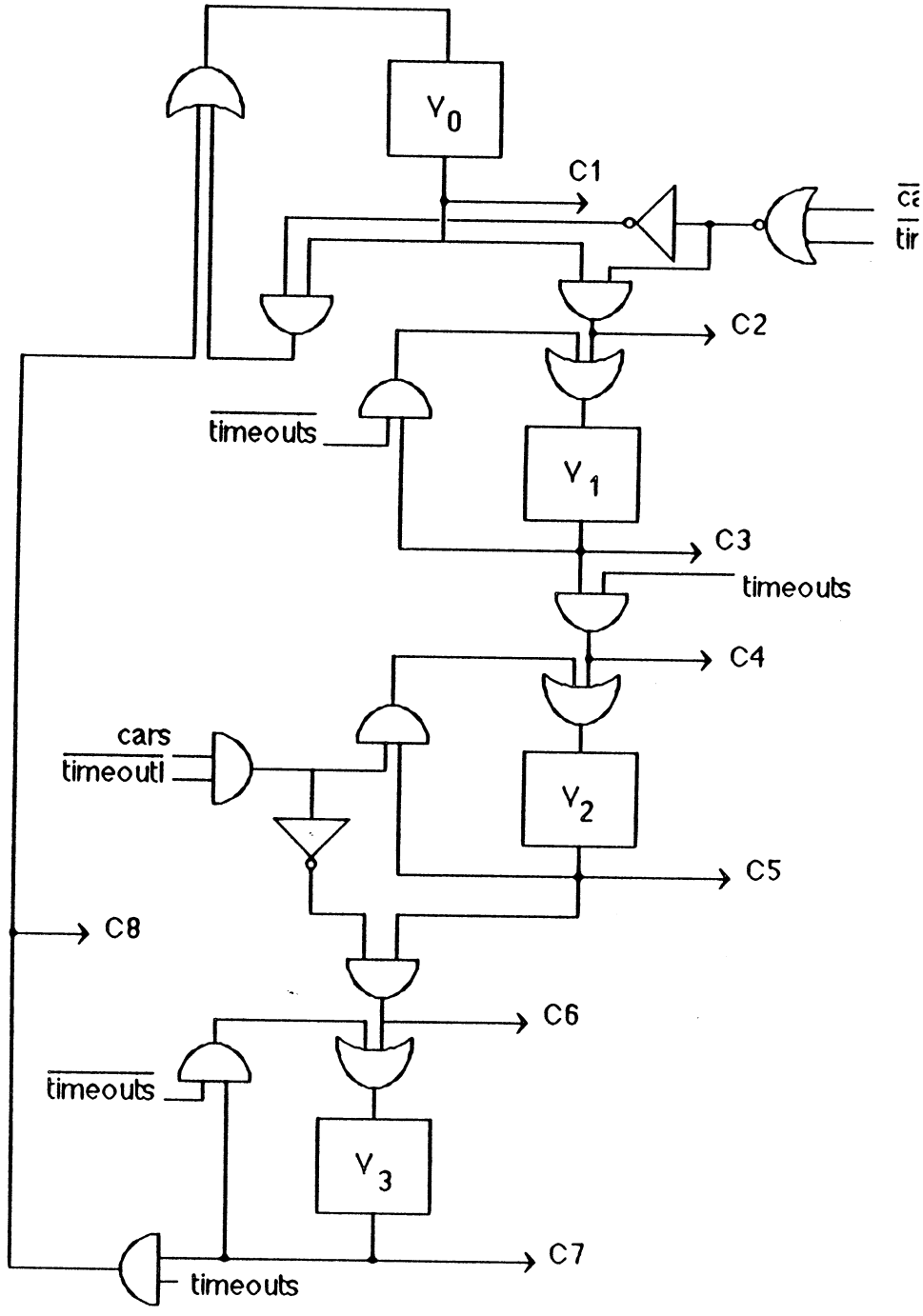


Figure 66 : organe de séquencement du contrôleur de feu en logique anarchique.

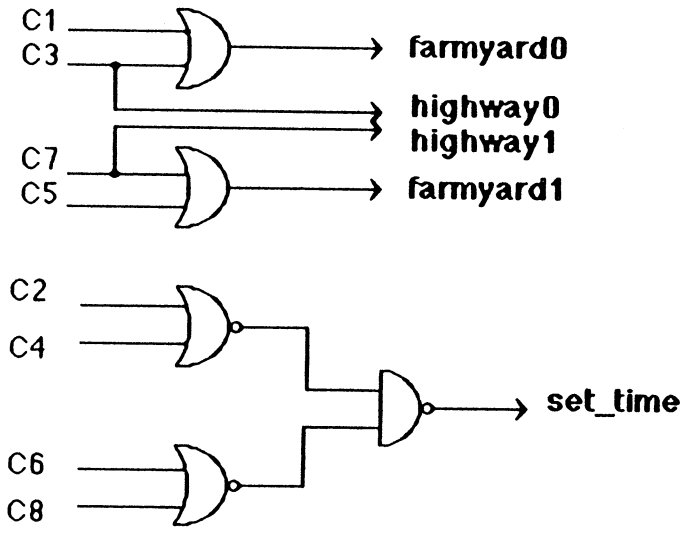


Figure 67 : Interface de commande du contrôleur de feu.

II.3.4. La synthèse en codage compact:

Dans une synthèse en codage compact, on va caractériser les **n** états du contrôleur par des codes binaires représentés sur **p** variables d'état $Y_1, Y_2, \ldots, Y_p$ où :

$$p \geq \lceil \log_2 (n) \rceil$$

Les équations de génération des commandes et les équations de séquencement sont calculées selon les principes exposés en II.3.1., en fonction du type de points de mémorisation choisi.

Exemple :

On reprend le contrôleur de gestion de feu dont le graphe est donné en figure 65, et on va coder les quatre états à l'aide de deux variables d'états Y_1 et Y_2 selon le tableau suivant :

état	Y_1	Y_2
S0	0	0
S1	1	0
S2	1	1
S3	0	1

Nous donnons les équations pour une synthèse à base de bascules de type T.

* Equations de séquencement :

$$T\_Y_1 = cars.timeoutl.\neg Y_1.\neg Y_2 + \neg cars.Y_1.Y_2 + timeoutl.Y_1.Y_2$$

$$T\_Y_2 = timeouts.Y_1.\neg Y_2 + timeouts.\neg Y_1.Y_2$$

* Equations des commandes :

$$farmyard0 = \neg Y_1.\neg Y_2 + Y_1.\neg Y_2$$

$$farmyard1 = \neg Y_1.Y_2$$

$$highway0 = Y_1.Y_2 + \neg Y_1.Y_2$$

$$highway1 = Y_1.\neg Y_2$$

$$set\_time = cars.timeoutl.\neg Y_1.\neg Y_2 + timeouts.Y_1.\neg Y_2 + \neg cars.Y_1.Y_2 + timeoutl.Y_1.Y_2 + timeouts.\neg Y_1.Y_2$$

La figure 68 représente un schéma synchronisé du contrôleur de gestion de feu, réalisé avec des registres de bascules D pour le maintien des entrées et des commandes, et un registre de deux bascules J/K montées en bascules T pour le maintien du code d'état de l'automate. La partie combinatoire pour l'organe de séquencement et pour l'interface de commande est implantée sur un unique PLA.

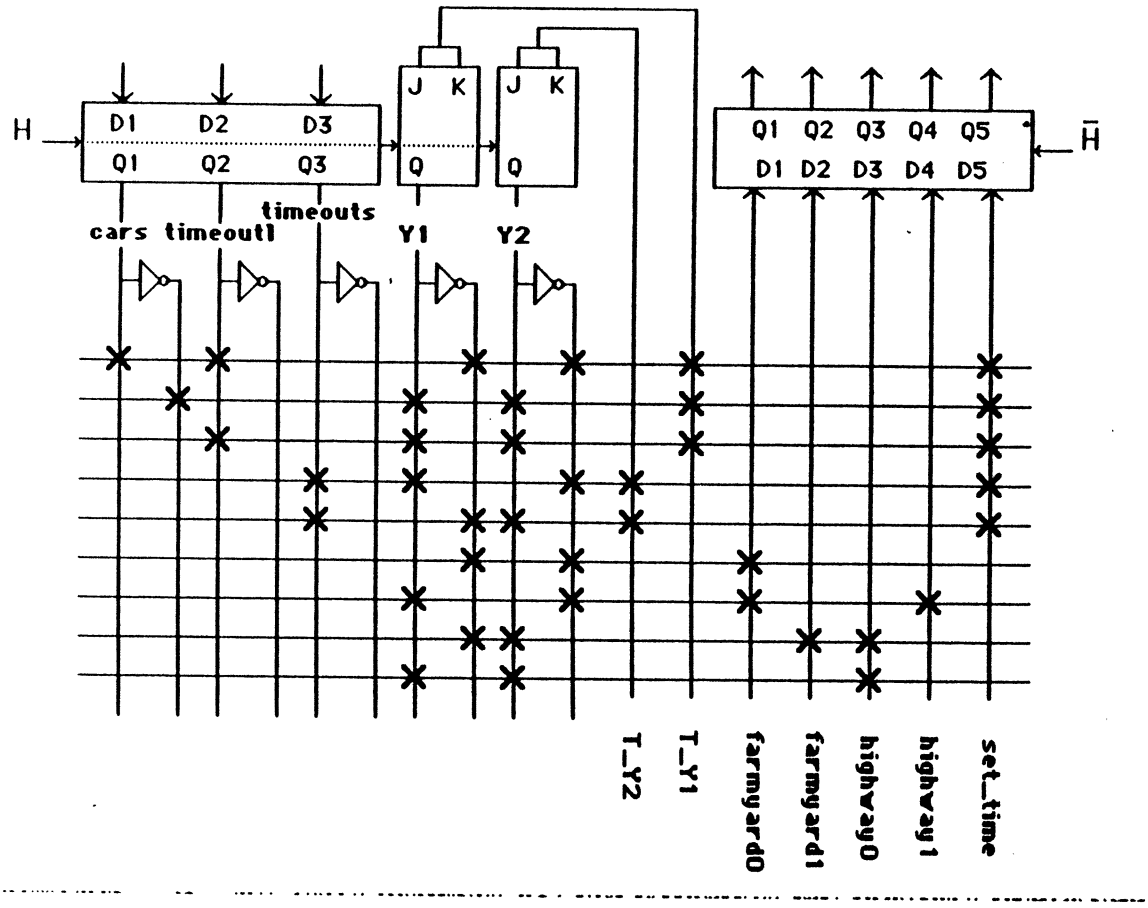


Figure 68 : réalisation mono-PLA et synchronisation à front d'horloge.

II.4. Optimisation du codage par un système de règles.

Le problème crucial posé par la synthèse en codage compact est celui de l'encodage. La taille des équations et donc la surface implantée du contrôleur dépendent directement du choix des codes attribués aux états du graphe. Tout comme la minimisation Booléenne, le problème du choix du codage est de nature combinatoire, et les techniques mises en œuvre dans les années 60 [ARM62] [KAR64] [HAR66] sont limitées à des exemples didactiques qui n'ont aucune correspondance avec des contrôleurs réalistes. Des méthodes heuristiques ont donc été développées afin de résoudre partiellement le problème de l'assignation du codage [DEM84a] [DEM84b] [SAU86] [DUP86] dans le but de traiter dans des temps raisonnables des

contrôleurs de taille importante tels que les parties contrôles de microprocesseurs.

Les techniques d'optimisation du codage étudiées au sein du *Laboratoire Circuits et Systèmes* [DUP86] reposent sur la définition de règles énoncées à partir d'un ensemble de mécanismes élémentaires, afin d'engendrer des équations simplifiables suivant des critères multiples. On souhaite ainsi définir des jeux de règles correspondant à diverses architectures de contrôleurs et diverses techniques de conception. Par exemple, on peut envisager la définition d'un jeu de règles pour engendrer un codage facilitant le partitionnement ultérieur d'un contrôleur mono-PLA en plusieurs PLA.

L'assignation d'un codage optimisant les équations d'un contrôleur se décompose en deux processus. Le premier processus cherche à appliquer les règles d'optimisation au vu du graphe d'état du contrôleur à synthétiser. L'application de ces règles engendre un système de contraintes entre les codes (contraintes d'adjacence de deux codes, d'adjacence généralisées de 2^n codes, dominance d'un code sur un autre,...). Le rôle du second processus est donc de résoudre le système de contraintes établi lors de l'application des règles. La résolution doit essayer de satisfaire le maximum de contraintes, et lorsque des contraintes sont incompatibles, on doit procéder à une résolution des conflits en rejetant la ou les contraintes qui engendrent les gains les plus faibles. Le problème de la résolution des contraintes est un problème classique de l'Intelligence Artificielle, mettant en œuvre des techniques de retour arrière (backtracking).

Notations :

Soit **n**, le nombre de variables d'état choisi pour coder les **m** états du contrôleur

$n \geq \lceil \log_2 (m) \rceil$. On note $Y_1,...,Y_n$ les variables.

Le monôme de caractérisation du code de l'état s_i est noté **code** (s_i).

Soit $\{Y_i\}$ l'ensemble des variables d'état qui valent un dans le code de s_i.

Les deux contraintes sur les codes qui interviennent dans les règles sont la **dominance** et **l'adjacence**.

* Dominance :

Le code de l'état s_i est dominant sur le code de l'état s_j si $\{Y_j\}$ est inclus dans $\{Y_i\}$.

ex : **0111** est dominant sur **0101**

* Adjacence :

Le code de l'état s_i est adjacent au code de l'état s_j si **code** (s_i) et **code** (s_j) sont adjacents.

ex : **0111** est adjacent à **0110**

Exemples de règles d'optimisation du codage.

Règle 1 : Règle d'optimisation pour les équations des commandes dans un contrôleur de Moore:

Si il existe dans le graphe un ensemble de **p** états $(2^{k-1} < p \leq 2^k)$ qui émettent un même ensemble de commandes **O**, alors on peut engendrer les contraintes suivantes :

- coder k parmi les n variables dans le code des p états avec p combinaisons,
- laisser les (n-k) autres variables invariantes dans les codes des p états,
- ne plus utiliser les (2^k-p) combinaisons restantes.

Les contraintes produites par l'application de cette règle entraînent un gain de **p/1=p,** c'est à dire **1** monôme contre **p** monômes dans les équations des commandes de l'ensemble **O**.

Exemple:

n = 5, p = 6

k = 3 car $(4 < 6 \leq 8)$

Les p états pourront être codés de la manière suivante :

	k			n-k	
état	Y1	Y2	Y3	Y4	Y5
1	0	0	0	1	0
2	0	0	1	1	0
3	0	1	0	1	0
4	0	1	1	1	0
5	1	0	0	1	0

p

codes non utilisés :

Y1	Y2	Y3	Y4	Y5
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

2^k-p

Règle 2 : règle d'optimisation pour les équations de séquencement dans un sous graphe d'attribution:

Si il existe dans un graphe un motif d'attribution à **p** antécédents ($2^{k-1} < p \leq 2^k$), dont les **p** transitions sont conditionnées par la même condition **C**, alors on peut établir les contraintes suivantes :

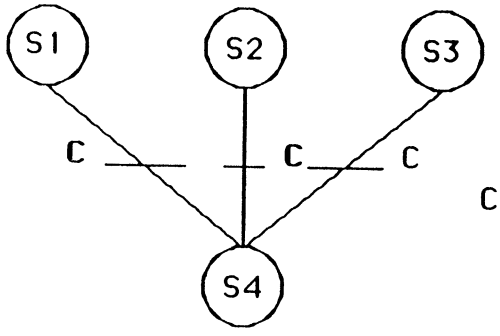
- coder k parmi les n variables dans le code des p états avec p combinaisons,
- laisser les (n-k) autres variables invariantes dans les codes des p états,
- ne plus utiliser les (2^k-p) combinaisons restantes.

Les contraintes produites par l'application de cette règle entraînent un gain de **p*t/1*t = p,** c'est à dire **t** monôme contre **p*t** monômes dans les équations des variables d'état, **t** étant le nombre de monômes dans la forme minimale de **C**.

Exemple:

k = 2, p = 3, n = 4.

Les **p** états pouront être codés de la manière suivante :



	k		n-k	
état	Y1	Y2	Y3	Y4
S1	0	0	1	1
S2	0	1	1	1
S3	1	0	1	1

Codes non utilisés :

	Y1	Y2	Y3	Y4
2^k-p	1	1	1	1

Le monôme intervenant dans les équations est $Y_3.Y_4.C$ au lieu de 4 monômes.

Règle 3 : règle d'optimisation pour les équations de séquencement dans un sous graphe de sélection:

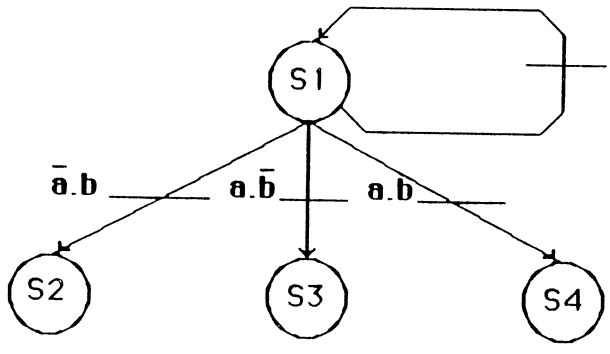
Si il existe dans un graphe un motif de sélection à **p** successeurs ($p = 2^k$), dont les **p** transitions sont conditionnées par des conditions canoniques C_i dont la somme est une tautologie, alors on peut établir les contraintes suivantes :

- coder k parmi les n variables dans le code des p états avec p combinaisons,
- laisser les (n-k) autres variables invariantes dans les codes des p états,

Les contraintes produites par l'application de cette règle entraînent un gain de **p/k+1,** c'est à dire **k+1** monômes contre 2^k monômes dans les équations des variables d'état.

Exemple :

p = 4, n = 4, k = 2.



	k		n-k	
état	Y1	Y2	Y3	Y4
S1	0	0	1	1
S2	0	1	1	1
S3	1	0	1	1
S4	1	1	1	1

Les monômes générés sont :

- pour Y_1 : $a.\neg b.code\ (S1) + a.b.code\ (S1) =$ **a.code (S1)**
- pour Y_2 : $\neg a.b.code\ (S1) + a.b.code\ (S1) =$ **b.code (S1)**
- pour Y_3 et Y_4 : $\neg a.\neg b.code\ (S1) + \neg a.b.code\ (S1)$
  $+ a.\neg b.code\ (S1) + a.b.code\ (S1) =$ **code (S1)**

soit 3 monômes au lieu de 4.

Il existe beaucoup d'autres règles mettant en jeu des motifs plus complexes, dont la reconnaissance peut s'avérer difficile. La règle 4 décrite ci après est un exemple de ces règles complexes basées sur la reconnaissance de plusieurs motifs ressemblants dans le graphe.

Règle 4 : Règle d'optimisation pour les équations de séquencement dans 2 motifs simples identiques de sélection.

Si il existe dans le graphe d'états, deux motifs identiques de sélection à 2 états (figure 69), alors on peut établir les contraintes suivantes :

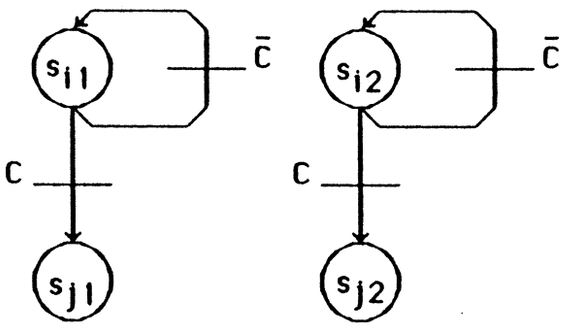


Figure 69 : deux sélections identiques à 2 états.

- rendre s_{i1} et s_{i2} adjacents,
- rendre s_{j1} et s_{j2} adjacents,
- rendre s_{j1} dominant sur s_{i1},
- rendre s_{j2} dominant sur s_{i2}.

Les contraintes produites par l'application de cette règle entraînent un gain de **4/3**, c'est à dire **3** monômes contre **4** monômes dans les équations des variables d'état.

Sur l'exemple de la figure 69, et pour un codage sur 4 bits, on peut assigner les codes suivants :

états	Y1	Y2	Y3	Y4
si1	0	1	0	0
sj1	0	1	1	0
si2	1	1	0	0
sj2	1	1	1	0

Les monômes obtenus sont :

- pour Y_1 : $\neg$b. code (s_{i2}) + b. code (s_{i2}) = $\mathbf{Y_1.Y_2.\neg Y_3.\neg Y_4}$
- pour Y_2 : $\neg$b. code (s_{i1}) + b. code (s_{i1}) = code (s_{i1}) = $\mathbf{\neg Y_1.Y_2.\neg Y_3.\neg Y_4}$

- pour Y_3 : b. code (s_{i1}) + b. code (s_{i2}) = b.$Y_2.\neg Y_3.\neg Y_4$

- pour Y_4 : aucun monôme.

Règle 5 : règle d'optimisation pour les équations de séquencement dans un motif complexe.

Si il existe dans le graphe un motif complexe comme celui de la figure 70, alors on peut établir les contrainte suivantes :

- rendre s_i et s_j adjacents,
- rendre s_j et s_k adjacents,
- ne pas poser de contraintes sur s_u et sur s_v.

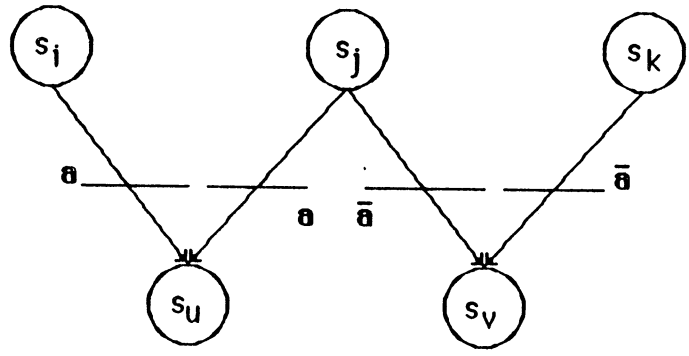


Figure 70 : motif complexe.

Les contraintes produites par l'application de cette règle entraînent un gain de 4/2, c'est à dire 2 monômes contre 4 monômes dans les équations des variables d'état.

Exemple :

On code à l'aide des 4 variables d'état Y_1, Y_2, Y_3, Y_4 de la manière suivante :

état	Y1	Y2	Y3	Y4
si	0	0	1	0
sj	1	0	1	0
sk	1	0	1	1
su	0	1	0	0
sv	0	1	1	0

Les monômes nécessaires sont les suivants :

- pour Y_1 : aucun.
- pour Y_2 : b. [code (s_i) + code (s_j)] = $\mathbf{b.\neg Y_2.Y_3.\neg Y_4}$
- pour Y_3 : $\neg$b. [code (s_j) + code (s_k)] = $\mathbf{\neg b.Y_1.\neg Y_2.Y_3}$
- pour Y_4 : aucun.

L'application des règles et la résolution des contraintes sont prises en charge par un moteur d'inférence guidé par des règles de contrôle qu'il reste à définir pour une meilleure efficacité du système. Un premier prototype a néanmoins été construit en LISP, et a donné des résultats prometteurs. Le lecteur est invité à consulter les exemples de synthèse de contrôleurs dont les codes ont été assignés automatiquement par le prototype d'encodage.

Dans un proche avenir, un ensemble plus complet de règles va être défini, produisant des simplifications sur les équations des variables d'état et celles des commandes. Des jeux de règles différents vont être établis pour des synthèses à base de points de mémorisation divers (J/K, R/S, T) et pour diverses architectures de contrôleur (architecture mono-PLA, bi-PLA par découpage fonctionnel, multi-PLA par partitionnement, structures microprogrammées,...).

*

PARTIE III

ASYL : UN OUTIL D'ASSISTANCE

POUR LA SYNTHESE LOGIQUE

III. ASYL : UN OUTIL D'ASSISTANCE A LA SYNTHESE LOGIQUE.

III.1. Description générale du système ASYL.

Le système ASYL est un outil logiciel pour la synthèse automatique de circuits combinatoires et séquentiels. Il élabore automatiquement la synthèse des circuits combinatoires et séquentiels à partir de descriptions de haut niveau qui sont :

* un graphe de contrôle pour les circuits séquentiels,
* un système de fonctions Booléennes éventuellement incomplètes pour les circuits combinatoires.

La synthèse séquentielle :

Le système ASYL procède à la compilation de la description textuelle du graphe de contrôle du contrôleur à synthétiser. Les vérifications sémantiques de conformité exposées en II.2. sont faites après compilation ; le concepteur est avisé lorsqu'elles ne sont pas vérifiées.

A partir d'un choix de points de mémorisation et de l'assignation d'un codage, le système établit automatiquement les équations qui caractérisent le fonctionnement du contrôleur. L'assignation du codage peut être faite manuellement par le concepteur ou bien automatiquement par le système, de façon à engendrer des équations globalement simplifiables.

Plusieurs structures "cible" sont proposées pour implémenter les équations du contrôleur :

- synthèse en logique anarchique (logique aléatoire temporisée) à partir d'un codage 1 parmi n,
- synthèse mono-PLA à partir d'un codage compact,
- synthèse bi-PLA à partir d'un codage compact et d'une décomposition fonctionnelle du contrôleur (organe de séquencement et interface de commandes),

- synthèse multi-PLA à partir d'un codage compact, et par l'utilisation de techniques de partitionnement [DAN86].

La synthèse combinatoire :

Le système ASYL met en pratique les règles de minimisation logique exposées en I.3. Il produit donc des solutions minimisées pour des implémentations sur PLA, mais également en logique aléatoire (techniques de factorisation I.2.2.) et en portes complexes (optimisation logique et topologique en technologie NMOS et CMOS I.2.3.).

Les points d'entrées du système ASYL sont des descriptions textuelles d'un circuit combinatoire ou bien d'un contrôleur. Les contrôleurs sont décrits par leur graphe des états et les circuits combinatoires par un ensemble de fonctions Booléennes ou bien par une forme matricielle (PLA ou table de vérité).

Nous donnons ci après les cartes syntaxiques des langages de description du système ASYL sous forme BNF.

1/ Langage de saisie d'un circuit combinatoire sous forme d'expressions logiques :

Fonction ::= *fonction* **nom**
entrées **listenoms** " ;"
sorties **listenoms** ";"
(**nom** "=" ((*vrai* / *indif* / *faux*) **expressionsimple**) + ";") +
fin

listenoms ::= **nom** ("," **nom**) *

expressionsimple ::= **terme** (("+" / *ou*) **terme**) *

terme ::= **primaire** (("." / *et*) **primaire**) *

primaire ::= (**negation**) (**nom** / "(" **expressionsimple** ")")

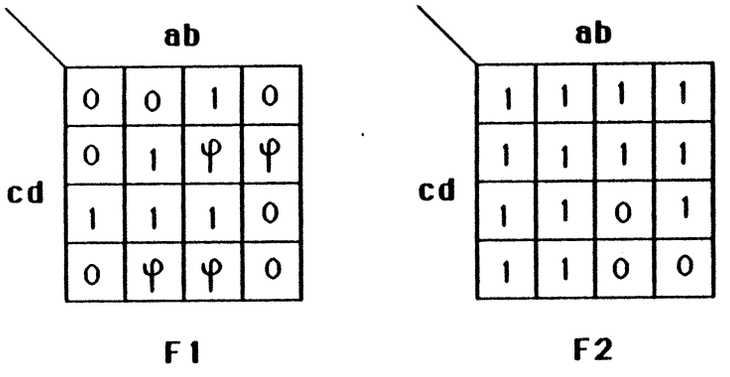
negation ::= *non* / "^"

nom ::= "a".."z" / "A" .. "Z" ("a".."z" / "A".."Z" / "_" / "0".."9") *

2/ Langage de saisie d'un circuit combinatoire sous forme matricielle :

Fonction ::= *fonction* **nom**
entrées **listenoms** ";"
sorties **listenoms** ";"
matrice **matrice** "fin_de_fichier"
matrice ::= (**ligne_entrées** **ligne_sorties**) +
ligne_entrées ::= ("0" / "1" / "X" / "." / "-") +
ligne_sorties ::= ("0" / "1" / "D" / "." / "-") +

Exemple:



Dans cet exemple, les fonctions F1 et F2 sont représentées par des tableaux de Karnaugh.

Représentation matricielle:

La représentation suivante définit la table de vérité des fonctions F1 et F2.

fonction F1_F2

entrées a, b, c, d ;
sorties f1, f2 ;

matrice

0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	-	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	-	1
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	-	1
1	1	1	0	-	0
1	1	1	1	1	0

La représentation précédente est une table de vérité (tous les états possibles des entrées sont spécifiés), une représentation matricielle condensée de type PLA pourrait être :

fonction F1_F2

entrées a, b, c, d ;

sorties f1, f2 ;

matrice

0	.	1	1	1	1
0	1	.	1	1	1
.	1	1	1	1	0
1	1	0	0	1	1
1	.	0	1	-	1
.	1	1	0	-	0
0	.	.	.	0	1
.	.	0	.	0	1
1	0	1	1	0	1

Représentation par expressions logiques:

Si l'on souhaite représenter ces fonctions par des expressions logiques, on pourra écrire par exemple :

```
fonction F1_F2
entrées  a, b, c, d ;
sorties  f1, f2 ;
        f1 = vrai ^a.c.d + ^a.b.d + b.c.d + a.b.^c.^d
             indif a.^c.d + b.c.^d ;
        f2 = vrai non (a.b.c + a.c.^d) ;
fin
```

3/ Langage de description d'un contrôleur :

graphe ::= *graphe* **nom** (*mealy / moore*)
declaration_entrées
declaration_sorties
declaration_états^{+} *fin*

declaration_états ::= *état* **nom** (" *:* " **commandes** $_{\text{moore}}$)" ; "
(**prédicat** " *:* " (**commandes** $_{\text{mealy}}$ " :") **nom**$_{\text{état aval}}$ " ;") $^{+}$

declaration_entrées ::= *entrées* **liste_var** " ; "

declaration_sorties ::= *sorties* **liste_var** " ; "

liste_var ::= **variable1** (" , " **variable1**)*

variable1 ::= **nom** (" *(* " **constante**$_{\text{taille du champ}}$ " *)* ")

commandes ::= (**décodage** /
(*indif* / " *?*") **variable2** (*si* **prédicat**))
(" , " **commandes**)

décodage ::= *decodage* " *(* " **variable3** " , " **liste_variable3** " *)* "

liste_variable3 ::= **variable3** (" , " **liste_variable3**)

variable2 ::= **variable3** / (" = " **valeur**)

variable3 ::= **nom** (" *(* " (**constante** / **intervalle**) " *)* ")

intervalle ::= **<constante>** " *:* " **<constante>**

predicat ::= **predicat1** (" + " / *ou* **predicat**)

predicat1 ::= **predicat2** (" . " / *et* **predicat1**)

predicat2 ::= (*non*) (**terme** / " *(* " **predicat** " *)* ")
terme ::= **variable3** (" =" (**valeur** / " *(* " **liste_constantes** *")*"))
liste_constantes ::= (**constante** / (" % " **valeur_binaire**)
/ (" *$* " **valeur_hexa**)) (" , " **liste_constantes**)
valeur ::= (**variable3** / **constante** / (" % " **valeur_binaire**)
/ (" *$* " **valeur_hexa**))
valeur_binaire ::= (0 / 1) +
valeur_hexa ::= ("0".."9" / "A".."F")

exemple:

Voici un exemple de déclaration d'un état pour une machine de Moore.

état E1 : C1, **indif** C2 **si** A.^B, C3 (1:2)=IN (4:5), C4 (0:2)=%101 ;
A.(IN1 (2:3)=1 + IN2 (0:2)=%010) : E2 ;
A.**non** (IN1 (2:3)=1 + IN2 (0:2)=%010) : E3 ;
^A : E1 ;

où C1, C2, C3 (3), C4 (3) sont déclarés comme sorties,
et A, B, IN (6), IN1 (4), IN2 (3) sont déclarés comme entrées.

La fonction ***decodage:***

La fonction ***decodage*** est une fonction prédéfinie dans le langage. La sémantique de cette fonction est celle du décodeur, la valeur à décoder étant le premier argument de la fonction, et les commandes à envoyer étant les arguments 2 à n de la fonction.

Exemple :

decodage (IN (0:1), OUT (3:4), C1, C2)

Cette fonction est interprétée de la façon suivante :

OUT_3 si ^IN_0.^IN_1, OUT_4 si ^IN_0.IN_1,
C1 si IN_1.^IN_2, C2 si IN_1.IN_2 ;

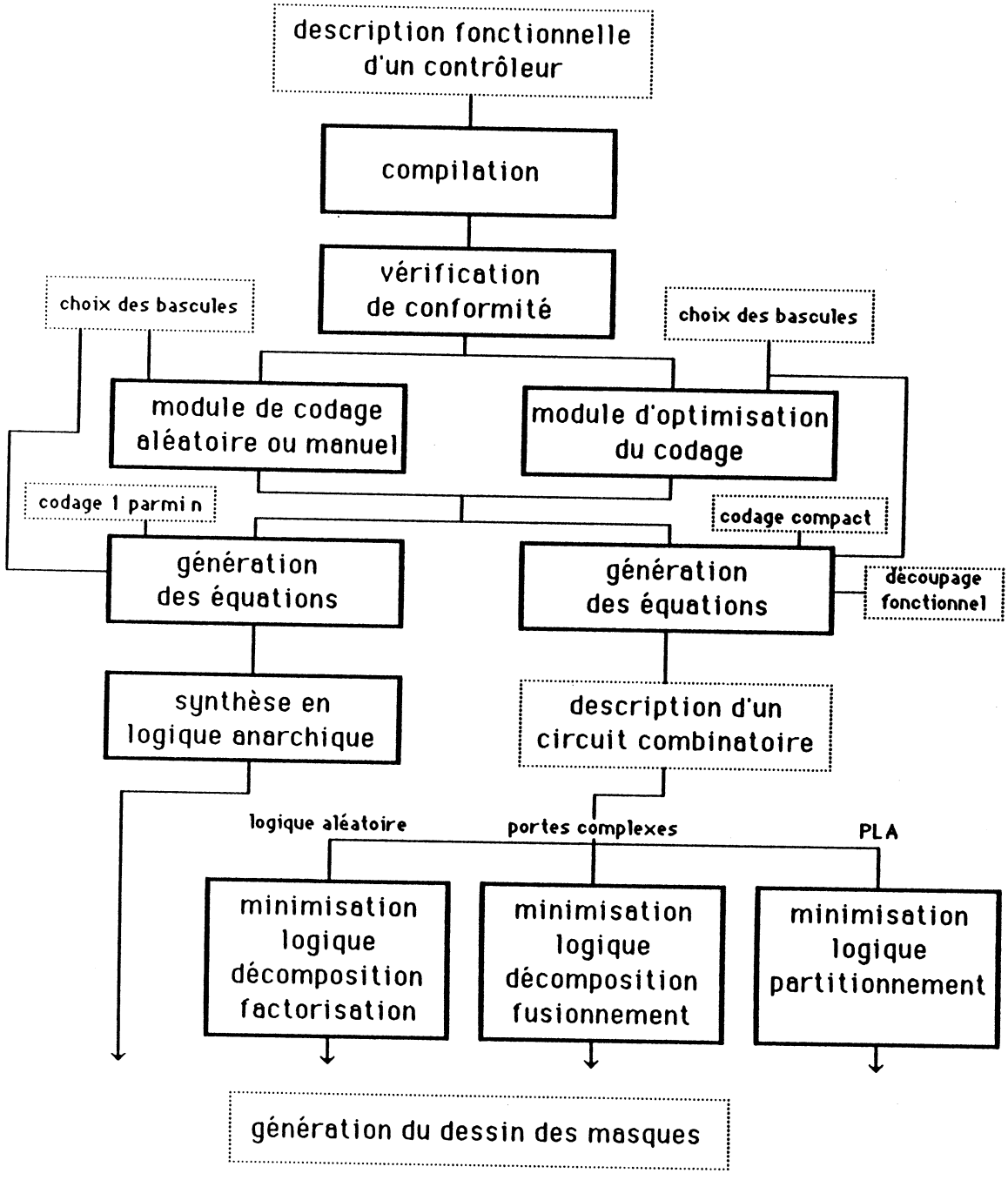


Figure 56 : Synoptique du système ASYL.

La version actuelle du système ASYL est écrite en PASCAL et la taille des programmes avoisine les 15000 lignes. Il est opérationnel sur les machines DIGITAL équipées du système d'exploitation VMS.

III.2. Exemples de contrôleurs synthétisés par le système ASYL.

III.2.1. Exemple 1 : contrôleur de gestion d'un protocole d'échange.

Ce contrôleur est issu de [FLA84]. Il s'agit d'un contrôleur à 19 états scrutant 9 entrées externes et émettant 9 commandes. Sa description textuelle dans le langage ASYL est donnée ci après.

```
graphe  protocole_échange moore

entrées
    fannion_nom, trame_stockée, busack,
    reveil, nmr, nmx, fannion_données,
    mode_appareil, wait ;
sorties
    set_busrq, gen_nle, load_port,
    arret_timer, gen_wr, reset_busrq,
    signal_transmission, gen_rd, lance_timer ;

état P0 ; /pas d'émission de commandes /
    vrai : P1 ; / franchissement inconditionnel /

état P1 ;
    mode_appareil : P2 ;
    non mode_appareil et nmx : P15 ;

état P2 ;
    non fannion_nom : P2 ;
    fannion_nom : P3 ;

état P3 ;
    non trame_stockée : P3 ; / attente active de trame_stockée /
```

trame_stockée : P4 ;

état P4 : set_busrq ; / émission de set_busrq /
non busack : P4 ; / attente de libération du bus /
busack : P5 ;

état P5 : gen_nle, load_por ;
vrai : P19 ;

état P6 : arrêt_timer ;
vrai : P2 ;

état P7 : arrêt_timer ;
non fannion_données : P7 ;
fannion_données : P9 ;

état P8 : arrêt_timer, set_busrq ;
non busack : P8 ;
busack : P13 ;

état P9 ;
non trame_stockée : P9 ;
trame_stockée : P10 ;

état P10 : set_busrq ;
non busack : P10 ;
busack : P11 ;

état P11 : gen_wr, load_por ;
vrai : P12 ;

état P12 : reset_busrq ;

vrai : P2 ;

état P13 : gen_rd, load_por ;
vrai : P14 ;

état P14 : reset_busrq, signal_transmission ;
vrai : P2 ;

état P15 : set_busrq ;
non busack : P15 ;
busack : P16 ;

état P16 : gen_rd, load_por ;
vrai : P17 ;

état P17 : reset_busrq, signal_transmission ;
vrai : P18 ;

état P18 ;
non wait : P18 ;
wait : P15 ;

état P19 : reset_busrq, lance_timer ;
réveil : P6 ;
non réveil **et** nmr **et non** nmx : P7 ;
nmx **et non** réveil **et non** nmr : P8 ;

fin / de la description du contrôleur protocole_échange /

La vérification formelle de conformité sur la description du contrôleur a détectée deux états pour lesquels il peut exister un risque de blocage.

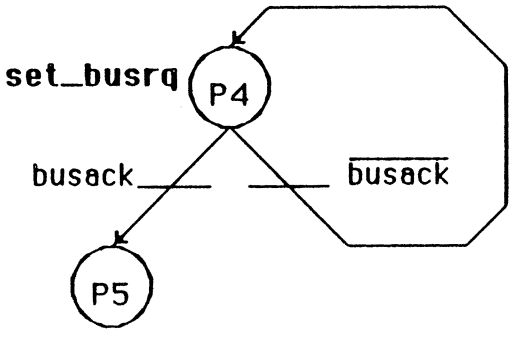
- état P1 : ¬mode_appareil.nmx + mode_appareil n'est pas une **tautologie**.
- état P19 : ¬reveil.nmr.¬nmx + reveil + nmx.¬reveil.¬nmr
  n'est pas une **tautologie**.
- Toutes les conditions de franchissement dans les motifs de sélection sont mutuellement exclusives, il n'y a donc pas d'évolution parallèle du graphe d'états.

On admettra donc que la description est correcte et que les conditions de blocages ne surviennent jamais dans l'environnement du contrôleur.

Synthèse mono-PLA en codage compact aléatoire:

Nous avons engendré les équations de l'organe de séquencement ainsi que celles de l'interface de commande pour des bascules D, et pour un codage généré aléatoirement sur 5 bits, entre 0 (%00000) et 31 (%11111).

Un premier PLA non optimisé a été produit par ASYL, contenant 36 monômes. L'application des règles d'élimination de redondance globale sur ce PLA initial a supprimé 4 monômes dans la génération des commandes, en les remplaçant par des sommes de monômes obligatoires dans les équations des variables d'états (figure 71).



sans optimisation :
3 monômes
code (P4)
busack. code (P4)
et busack. code (P4)

avec optimisation :
2 monômes
busack. code (P4)
et busack. code (P4)

Figure 71 : Optimisation locale sur un codage quelconque.

Ce motif étant présent 4 fois dans le graphe du contrôleur, on obtient finalement un PLA de 32 monômes dont la représentation après génération automatique du dessin des masques en technologie CMOS est donnée en figure 72.

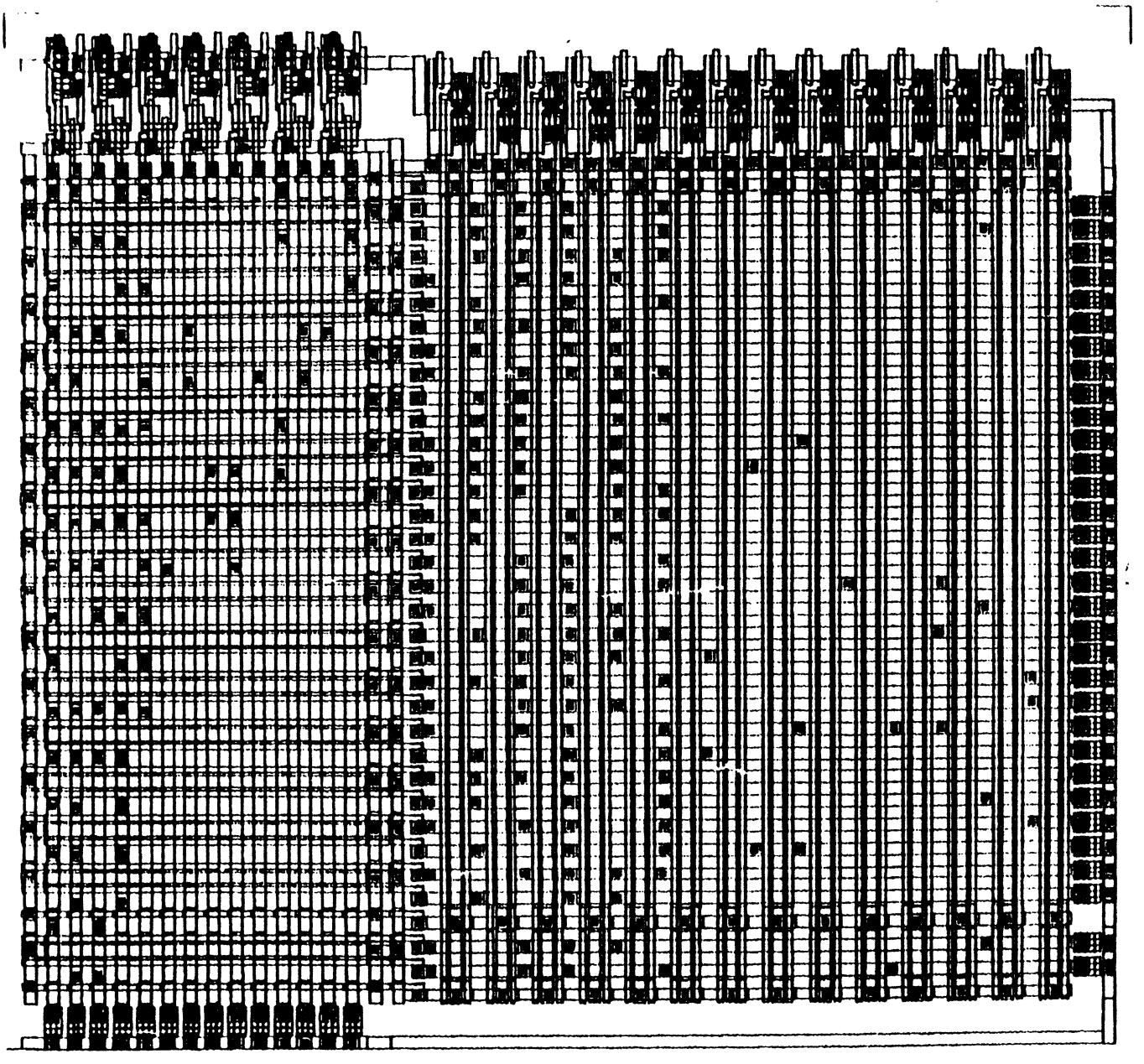


Figure 72 : PLA de 32 monômes pour le contrôleur **protocole_échange**.

Synthèse mono-PLA en codage compact avec optimisation du codage.

Un codage optimisant simultanément les équations des variables d'état et les équations des commandes à été obtenu à l'aide du prototype d'encodage automatique. Le programme d'encodage a détecté dans le graphe, un ensemble de 4 motifs identiques de sélection et en a déduit un ensemble de contraintes.

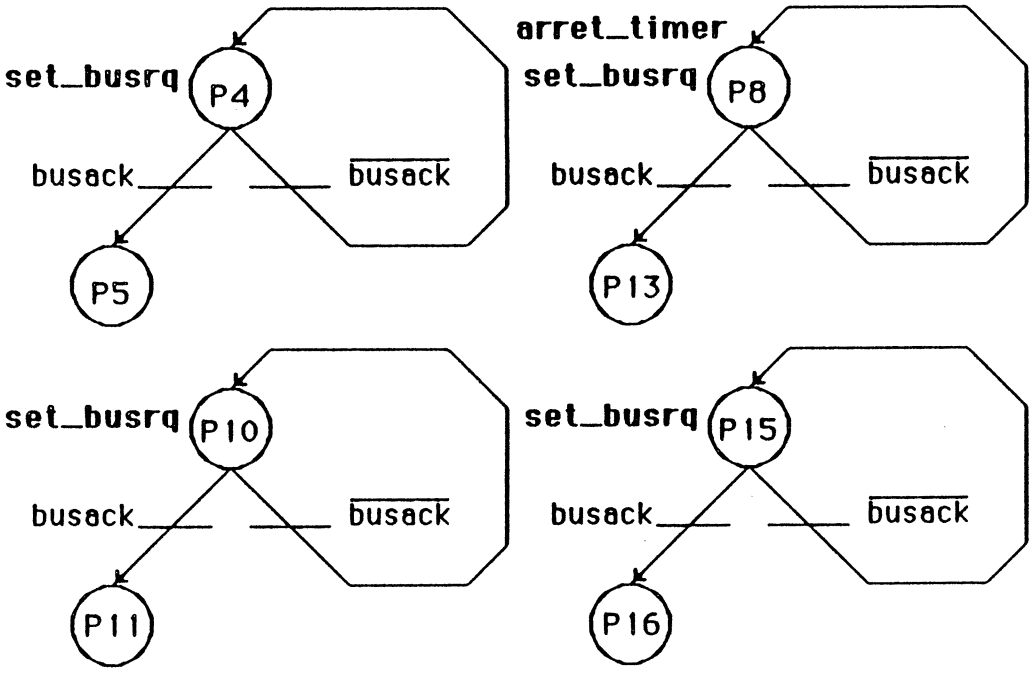


Figure 73 : Motifs identiques dans le contrôleur **protocole_échange**.

Les contraintes associées à l'identité de ces motifs sont :

- P4, P8, P10, P15 en adjacence généralisée sur 2 bits.
- P5, P13, P11, P16 en adjacence généralisée sur 2 bits.
- P5 dominant sur P4,
- P13 dominant sur P8,
- P11 dominant sur P10,
- P16 dominant sur P16.

Après résolution des contraintes établies, on a finalement obtenu le codage suivant :

Etat	Y_1	Y_2	Y_3	Y_4	Y_5	Code décimal
P0	1	0	0	1	0	18
P1	1	1	1	1	1	31
P2	1	1	1	1	0	30
P3	0	1	0	1	1	11
P4	0	1	0	0	0	8
P5	0	1	1	0	0	12
P6	1	1	1	0	1	29
P7	1	1	0	0	1	25
P8	1	0	0	0	0	16
P9	1	1	0	1	1	27
P10	1	1	0	0	0	24
P11	1	1	1	0	0	28
P12	1	1	0	1	0	26
P13	1	0	1	0	0	20
P14	1	0	1	1	1	23
P15	0	0	0	0	0	0
P16	0	0	1	0	0	4
P17	1	0	1	1	0	22
P18	1	0	1	0	1	21
P19	1	0	0	1	1	19

Le système ASYL a construit les équations de séquencement et celles de génération des commandes à partir du codage précédent et pour des bascules D. La minimisation globale de ces équations a fourni un PLA optimisé comprenant 28 monômes (figure 74).

Evaluation de la surface du PLA :

28*(2*14+14) = **1176**

Remarque:

Une simple observation du dessin final du PLA (figure 74) nous permet de constater la structure triangulaire de l'étage OU. Il est donc aisé de minimiser topologiquement ce PLA en récupérant la surface inoccupée (à peu près 50 %) dans le matrice OU. L'étage ET est également propice aux optimisations topologiques en raison de sa structure creuse.

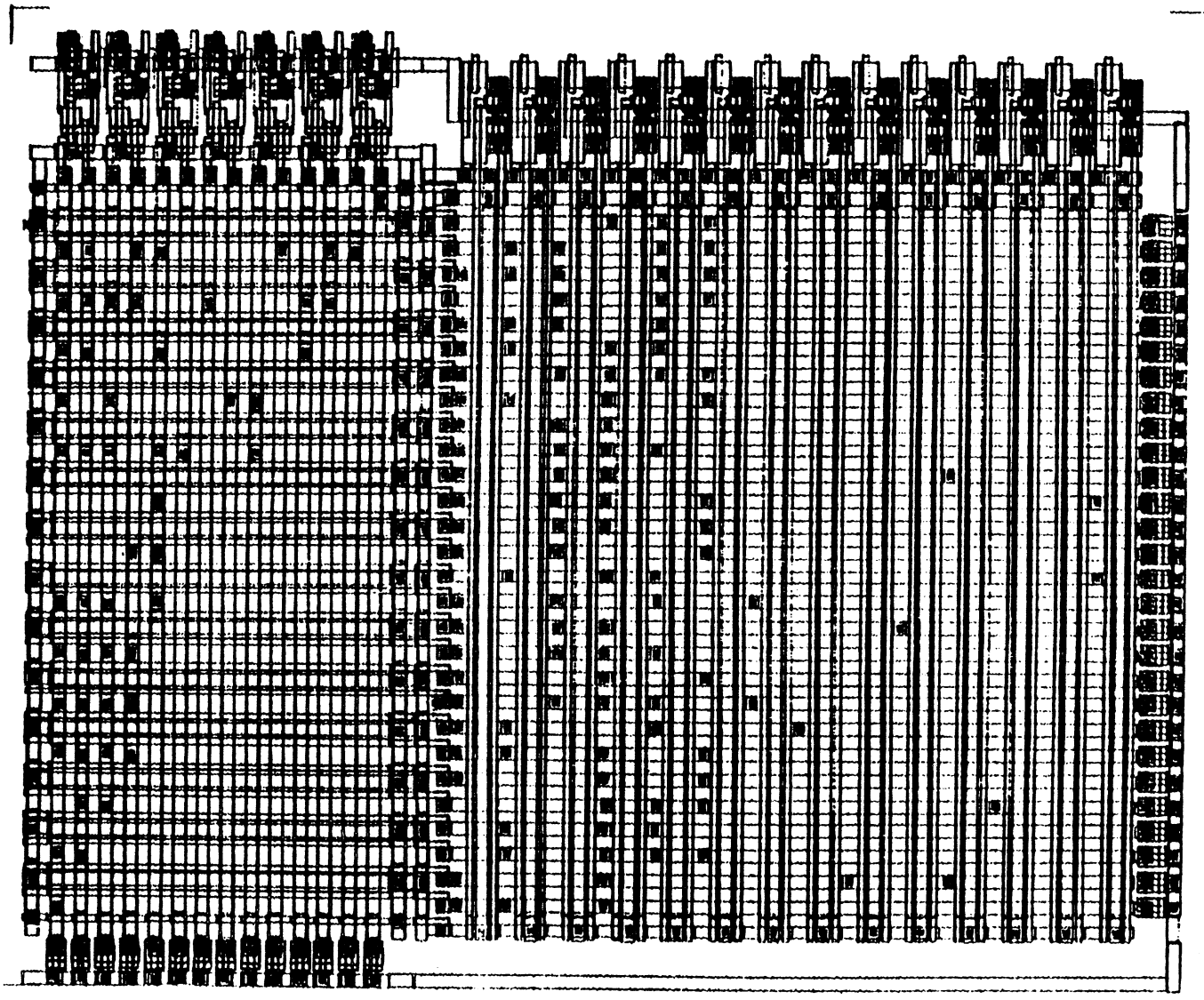


Figure 74 : PLA minimisé pour le contrôleur **protocole_échange**.

Evaluation des temps de calcul : ***(micro VAX II - VMS)***

compilation du graphe :	8 secondes
vérification formelle de conformité :	2 secondes
génération des équations :	5 secondes
compilation du système d'équations :	7 secondes
minimisation locale et globale :	1 minute 10 secondes

Les temps de calcul tiennent compte des entrées/sorties sur console et sur fichiers, et des sauvegardes intermédiaires efféctuées par le système.

Synthèse bi-PLA en codage compact avec le codage optimisé.

Partant du même codage que celui qui engendre le PLA de 28 monômes, nous avons fait un découpage fonctionnel du contrôleur en deux parties :

- un organe de séquencement qui assure le calcul de l'état suivant,
- un interface de commande qui assure la génération des commandes.

Les deux blocs fonctionnels ont donc été minimisés globalement mais indépendamment l'un de l'autre. Comme il s'agit d'un contrôleur de Moore (les commandes ne sont pas conditionnées par les entrées), l'interface de commande ne reçoit donc en entrée, que les 5 variables d'état. Le nombre de monômes obtenu après minimisation est de :

24	pour l'organe de séquencement,	(figure 75)
10	pour l'interface de commande.	(figure 76)

Evaluation des surfaces :

$24*(2*14+5) + 9*(2*5+9) =$ **963**

On constate que la synthèse sur deux PLA, par découpage fonctionnel donne un gain de surface de 963/1176 = **18 %**

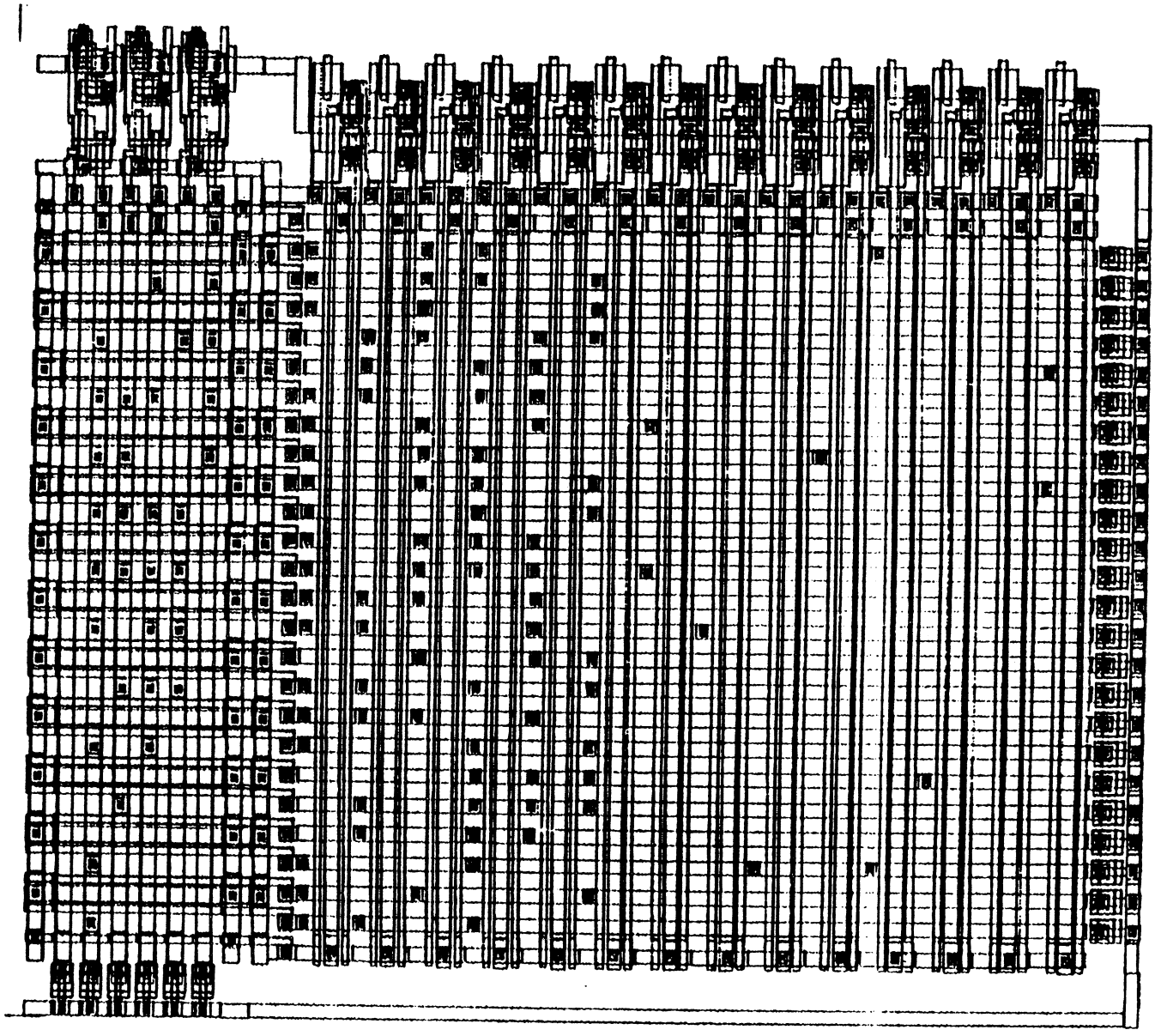


Figure 75 : organe de séquencement du contrôleur **protocole_échange**.

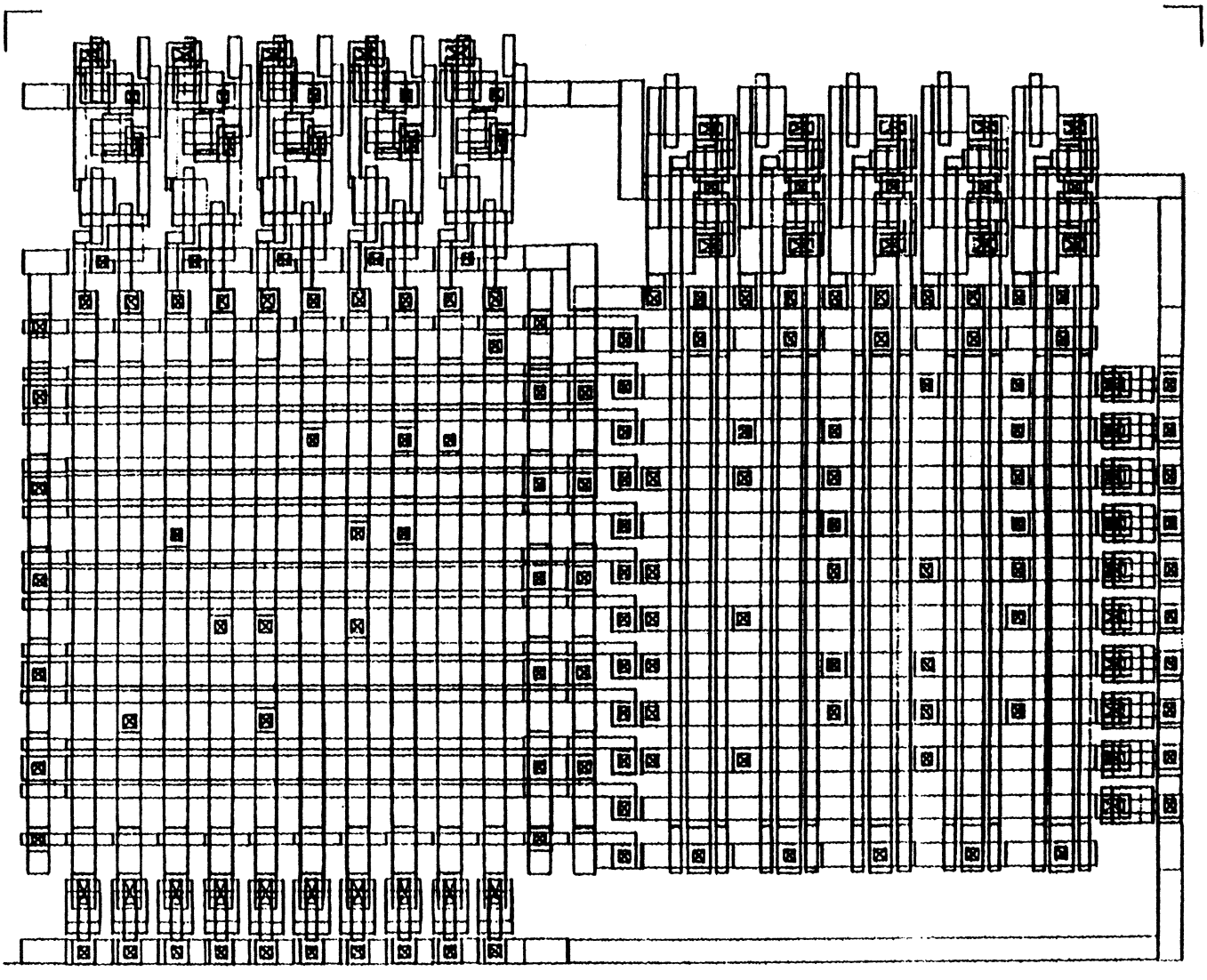


Figure 76 : interface de commande du contrôleur **protocole_échange**.

III.2.2. Exemple 2 : contrôleur de trafic.

Description textuelle dans ASYL:

graphe traffic **moore**

entrées

sena, senb ;

sorties

yela, grna, grnb, yelb ;

état s0 : grna ;

sena. **non** senb : s0 ;

sena.senb **ou non** sena. **non** senb : s1 ;

non sena.senb : s4 ;

état s1 : grna ;

vrai : s2 ;

état s2 : grna ;

vrai : s3 ;

état s3 : grna ;

vrai : s4 ;

état s4 : grna ;

vrai : s5 ;

état s5 : grna ;

vrai : s6 ;

```
état s6 : grnb ;
        sena. non senb : s10 ;
        sena.senb ou non sena. non senb : s7 ;
        non sena.senb : s6 ;

état s7 : grnb ;
        vrai : s8 ;

état s8 : grnb ;
        vrai : s9 ;

état s9 : grnb ;
        vrai : s10 ;

état s10 : grnb ;
        vrai : s11 ;

état s11 : yelb ;
        vrai : s0 ;
fin
```

Le codage sur 4 bits suggéré par le système d'encodage a permis d'obtenir une forme minimisée des équations avec 14 monômes (figure 77) au lieu de 18 dans la forme brute. Pour le même codage, nous avons effectué la synthèse du contrôleur sur 2 PLA, et obtenu 11 monômes pour le séquencement (figure 78) et 6 monômes pour la génération des commandes (figure 79).

Evaluation des surfaces :

mono-PLA :

- 14*(8+2*6) = **280**

bi-PLA :

- 11*(2*6+4) + 6*(2*4+4) = **248**, soit un gain en surface de **12 %**

Evaluation des temps de calcul : ***(micro VAX II - VMS)***

compilation du graphe :	5 secondes
vérification formelle de conformité :	2 secondes
génération des équations :	3 secondes
compilation du système d'équations :	4 secondes
minimisation locale et globale :	15 secondes

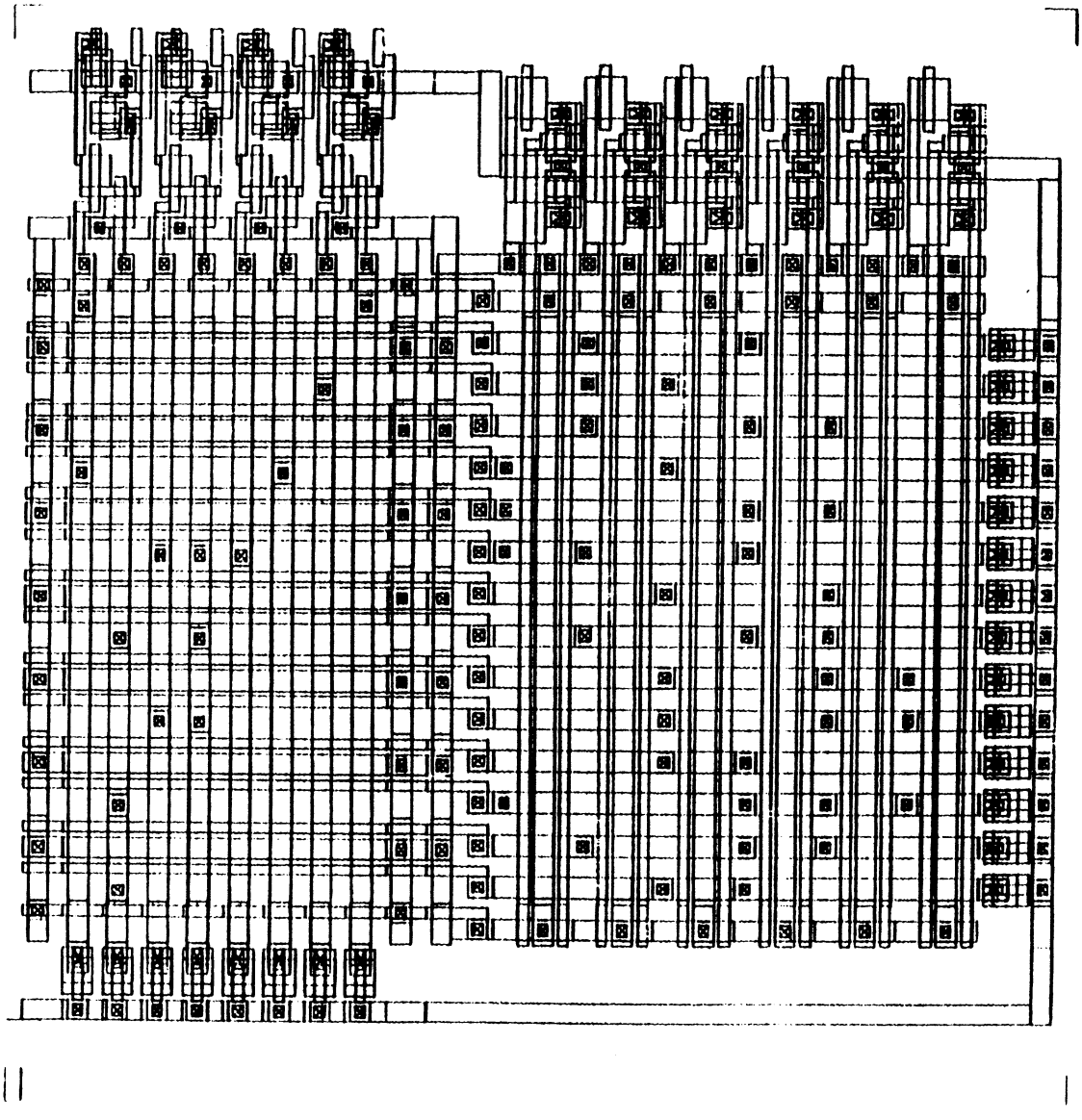


Figure 77 : forme minimisée mono-PLA du contrôleur **traffic**.

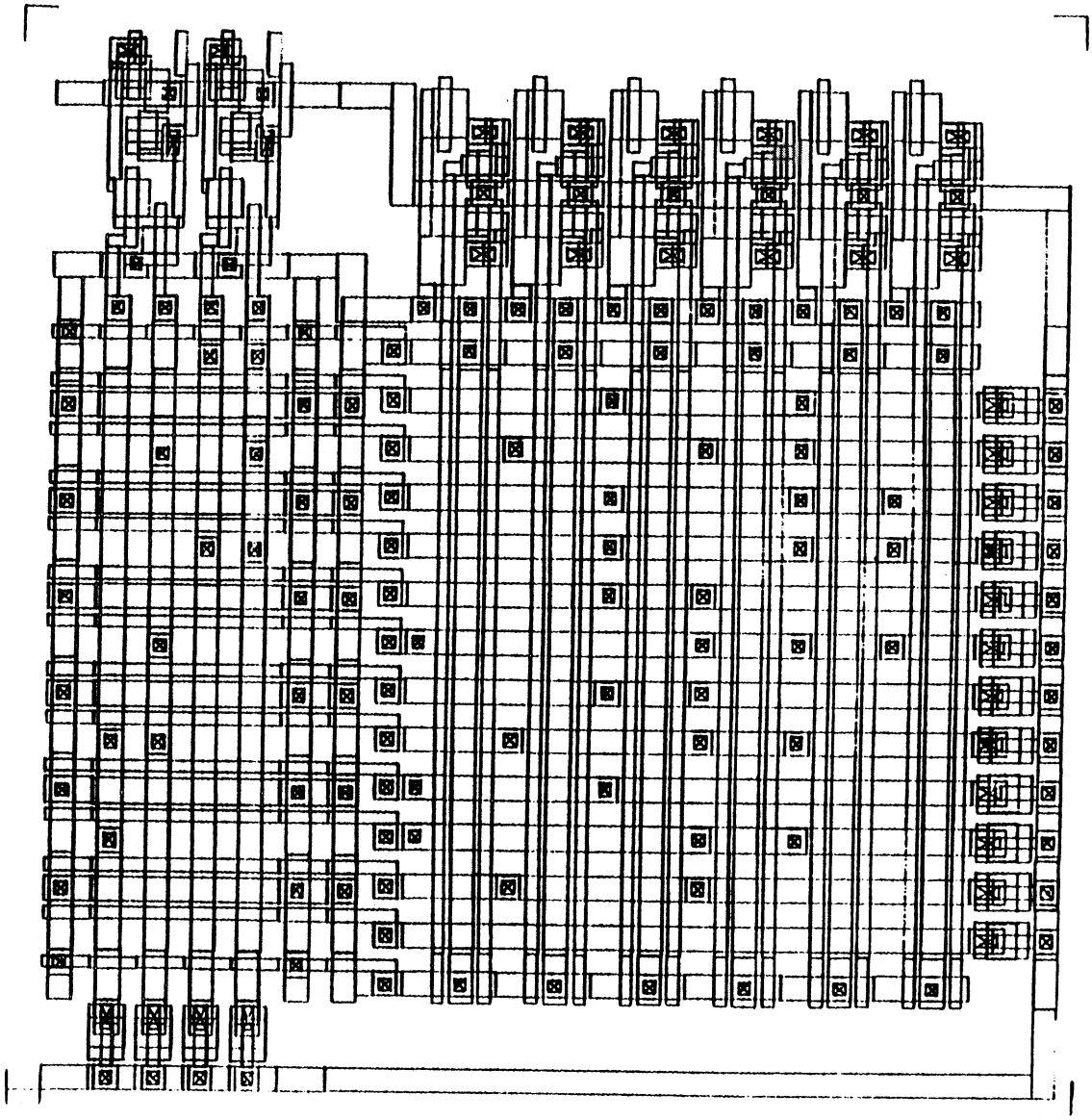


Figure 78 : organe de séquencement minimisé du contrôleur **traffic**.

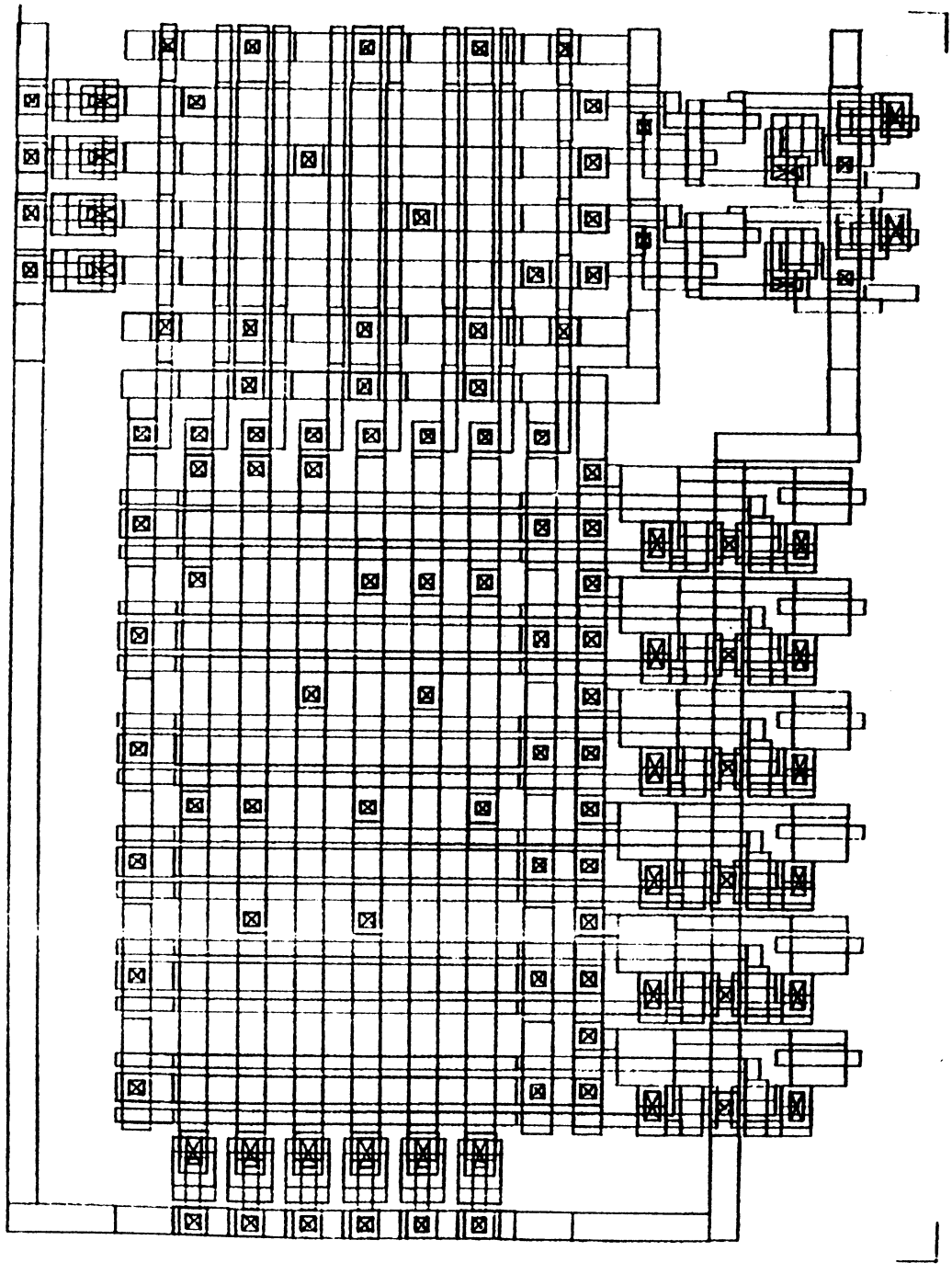


Figure 79 : interface de commande minimisé du contrôleur **traffic**.

III.3. Conclusion sur le système ASYL.

Comme nous l'avons vu précédemment, le système ASYL peut synthétiser rapidement et efficacement des contrôleurs et des circuits combinatoires de moyenne taille. Pour des raisons de place, nous ne pouvons présenter ici de plus gros exemples, mais nous tenons à signaler que le système ASYL a été abondamment employé pour la conception d'un microprocesseur 16 bits à haute sûreté de fonctionnement développé dans le laboratoire *Circuits et Systèmes*. Le langage de description de contrôleur s'est montré apte à décrire aisément la partie contrôle de ce microprocesseur. Cette description comporte 181 états, une quarantaine d'entrées, et plus de 120 commandes. A titre indicatif, voici les temps de calcul consommés pour la synthèse de ce contrôleur.

Evaluation des temps de calcul : ***(micro VAX II - VMS)***

- compilation du graphe : **4 minutes**
(la description fait 1400 lignes)

- vérification formelle de conformité : **1 minute 20 secondes**
- génération des équations : **6 minutes**
(à partir d'un codage manuel)

- compilation du système d'équations : **17 minutes**
(le fichier de description des équations dépasse 6000 lignes)

- minimisation locale des équations de séquencement :
1 heure 10 minutes
(chaque équation comprend initialement 200 monômes, il y a 8 équations)

- minimisation locale des équations des commandes :
15 minutes
(plus d'une centaine d'équations)

- minimisation globale de l'organe de séquencement :

3 heures

(environ 800 monômes à manipuler pour 8 équations, soit environ **6400 tests d'inclusion**)

- minimisation globale de l'interface de commande :

4 heures

(environ 400 monômes à manipuler pour une centaine d'équations, soit environ **40000 tests d'inclusion**)

- minimisation globale de toutes les équations : **15 heures**

(environ 1200 monômes à manipuler pour 120 équations, soit environ **144000 tests d'inclusion**)

Au vu des performances obtenues (et recherchées), nous pensons que le système ASYL est un outil d'aide à la conception adapté à la synthèse des circuits à très haute complexité de la nouvelle génération. Il reste néanmoins encore beaucoup de travail pour en faire l'outil espéré. Dans un premier temps, il devrait être envisagé de s'affranchir des contraintes des langages procéduriels (PASCAL) en optant pour les systèmes experts. Ensuite, il faudrait enrichir les systèmes de règles pour atteindre réellement les objectifs fixés : synthèses multi-cibles, et optimisations multi-critères. Le but de cette thèse était simplement de montrer qu'il est possible de s'attaquer aux problèmes combinatoires complexes en les résolvant partiellement par des systèmes de règles. Nous estimons que le but est atteint, constituant un premier pas vers les outils de synthèse et d'aide à la conception du siècle prochain.

Références bibliographiques

[NAG82] A. W. Nagle, R. Cloutier, A. C. Parker,
Synthesis of hardware for the control of digital system
IEEE Trans. on C.A.D. of I.C. & Systems, Oct. 82, pp : 201-212.

[HAF82] L. J. Hafer, A. C. Parker,
Automated Synthesis of Digital Hardware.
IEEE Trans. on Comp., Vol C31, No 2, Feb. 82, pp : 93-109.

[KOW85] T. J. Kowalski, D. J. Geiger, W. H. Wolf,
The VLSI Design Automation Assistant : from Algorithms to Silicon.
IEEE Design & Test, Aug. 85, pp : 33-42.

[THO83] D. E. Thomas, C. Y. Hitchcock III, T. J. Kowalski, J. V. Rajan,
Automatic Data Path Synthesis.
IEEE Computer, Dec. 83, pp : 59-70.

[JAM85] R. Jamier, A. A. Jerraya,
Apollon : A Data Path Silicon Compiler.
IEEE Circuits & Devices magazine, 1985, pp : 6-14.

[DIE63] D. L. Dietmeyer, P. R. Schneider,
A Computer-Oriented Factoring Algorithm for NOR Logic Design.
IEEE Trans. on Elec. Comp., Vol EC14, Dec. 63, pp : 868-874.

[SCH68] P. R. Schneider, D. L. Dietmeyer,
An Algorithm for Synthesis of Multiple-Output Combinational Logic.
IEEE Trans. on Comp., Vol C17, Feb. 68, pp :

[MUR72] S. Muroga, T. Ibaraki,
Design of Optimal Switching Networks by Integer Programming.
IEEE Trans. on Comp., Vol. C21, Jun 72, pp : 573-582.

[CER74] E. Cerny, M. A. Marin,
A Computer Algorithm for the Synthesis of Memoryless Logic Circuits.
IEEE Trans. on Comp., Vol. C23, May 74, pp : 455-465.

[SU71] S. Y. H. Su, C-W. Nam,
Computer Aided Synthesis of Multiple-Output Multi-level Networks with Fan-In and Fan-Out Constraints.
IEEE Trans. on Comp., Vol. C20, Dec. 71, pp : 1445-1454.

[DAR80] J. A. Darringer, W. H. Joyner Jr,
A new look at Logic Synthesis.
17th D.A.C., Jun 80, pp : 543-548.

[GEU85] A. J. de Geus, W. Cohen,
A rule based System for Optimizing Combinational Logic.
IEEE Design & Test, Aug. 85, pp : 22-32.

[CAL62] S. H. Caldwell,
Switching circuits and Logical Design.
John Wiley & Sons, inc, 1962.

[THU83] G. Thuau,
Conception logique et topologique en technologie MOS.
Thèse de 3eme cycle, INP Grenoble, France, Dec. 83.

[SAU85] G. Saucier, G. Thuau,
Systematic and optimized layout of MOS cells.
22nd D.A.C., Jul 1985, pp : 53-61.

[MCC65] E. J. McCluskey,
Introduction to the theory of switching circuits.
Mc Graw-Hill, 1965.

[KUN68] J. Kuntzmann
Algèbre de Boole
édition DUNOD, Paris, 1968.

[MON85] L. Monier, J. Vuillemin,
Using Silicon Assemblers.
VLSI. Int. Conf., Tokyo, Aug. 85, pp : 309-318.

[BLA85] T. Blackman, J. Fox, C. Rosebrugh,
The SILC Silicon Compiler : Language and Features.
22nd D.A.C., 1985, pp : 232-237.

[SOU83] J. R. Southard,
Mac Pitts : An Approach to Silicon Compilation.
IEEE Computer, Dec. 83, pp : 74-82.

[BRO81] D. W. Brown
A State Machine Synthesizer.
18th D.A.D., 1981, pp : 301-305.

[GRA83] W. Grass,
A Synthesis System for PLA-based Programmable Hardware.
Microprocessing & Microprogramming, Vol. 12, 1983, pp : 15-31.

[DAN86] A. Dandache,
Conception de PLA CMOS.
Thèse de Docteur de L'INP Grenoble, France, Juillet 86.

[PAI84] J. F. Paillotin,
Implantation automatique de logique en bande.
Thèse de 3eme cycle, INP Grenoble, France, Décembre 84.

[PER82] T. Perez Segovia, S. Chuquillanqui,
PAOLA : A tool for topological optimization of large PLAs.
19th D.A.C., Las vegas, Jun. 1982, pp : 300-306.

[LAA85] P. J. M. van Laarhoven, E. H. L. Aarts, M. Davio,
PHIPLA : A new Algorithm for Logic Minimization.
22nd D.A.C., Jul. 1985, pp : 739-743.

[ARE78] Z. Arevalo, J. G. Bredeson,
A Method to simplify a Boolean function into a near minimal sum-of-products for Programmable Logic Arrays.
IEEE Trans. on Comp., Vol. C27, Nov. 78, pp : 1028-1039.

[HAN86] S. Hanriat, E. Dupont, J. Idt, G. Saucier,
A Computer Aided System for Logic Synthesis using Artificial Intelligence.
Silicon Design Conference, Wembley, Jul. 86, pp : 315-324.

[SIF82] J. Sifakis,
Notes et compléments de cours sur les Réseaux de Pétri.
Polycopié ENSIMAG, Grenoble, 1982.

[FOR83] J. Forrest, Md. Edwards,
The automatic generation of programmable logic array from algorithmic state machine description.
VLSI. IFIP., 1983, pp : 183-193.

[HAN85] S. Hanriat, J. Idt,
Compilateur de fonctions Booléennes et de contrôleurs sur réseaux prédiffusés.
Colloque national sur la conception de circuits à la demande sur réseaux prédiffusés et précaractérisés, INP Grenoble, France, Mai 85,
pp : 493-517.

[TIS65] P. Tison,
Théorie des consensus.
Thèse de l'université des sciences de Grenoble, 1965.

[TIS67] P. Tison,
Generalization of consensus theory and application to the minimization of Boolean functions.
IEEE Trans. on Elec. Comp., Vol. EC16, 1967, pp : 446-456.

[SLA70] J. R. Slagle, C. L Chang, R. C. T. Lee,
A New Algorithm for Generating Prime Implicants.
IEEE Trans. on Comp., Vol. C19, 1970, pp : 304.

[HON74] S. J. Hong, R. G. Cain, D. L. Ostapko,
MINI : A heuristic approach for Logic Minimization.
IBM Journal of Research & Developpment, Vol. 18, Sept. 74,
pp = 443-458.

[BRA85] R. K. Brayton, G. D. Hachtel, C. T. McMullen,
A. L. Sangiovanni Vincentelli,
Logic Minimization Algorithms for VLSI Synthesis.
Kluwer Academic Publishers, 1985.

[DAG85] M. R. Dagenais, V. K. Agarwal, N. C. Rumin,
The McBOOLE Logic Minimizer.
22nd D.A.C., 1985, pp : 667-673.

[AGR85] P. Agrawal, V. D. Agrawal, N. N. Biswas,
Multiple Output Minimizatin.
22nd. D.A.C., 1985, pp : 674-680.

[AMB84] P. Amblard,
Conception temporellement sûre de circuits intégrés complexes.
Thèse de 3eme cycle, INP Grenoble, France, Juin 84.

[FLA84] E. Flamand,
A Complete and Automatic System for Sequencer Design.
IEEE ICCD., New York, Oct. 84, pp : 324-330.

[MEA80] C. Mead, L. Conway,
Introduction to VLSI systems.
Addison Welsey, 1980.

[ARM62] D. B. Armstrong,
A Programmed Algorithm for Assigning internal codes to Sequential Machines.
IRE Trans. on Elec. Comp., Vol. EC11, Aug. 62, pp : 466-472.

[KAR64] R. Karp,
Some Techniques for state Assignment for Synchronous Sequential Machines.
IEEE Trans. on Elec. Comp., Vol. EC13, Oct. 64, pp : 507-518.

[HAR66] J. Hartmanis, R. E. Stearns,
Algebraic Structure Theory of Sequential Machines.
Prentice Hall, 1866.

[CRA85] M. Crastes de Paulet,
Spécification et simulation fonctionnelles de circuits complexes : Le système CADOC.
Thèse de Docteur Ingénieur, INP Grenoble, France, Nov. 85.

[DEM84a] G. De Micheli, A. sangiovanni Vincentelli, R. Brayton,
KISS : A program for Optimal State Assignment of Finite State Machines.
ICCAD, Santa Clara, Nov. 84.

[DEM84b] G. De Micheli,
Optimal Encoding of Control Logic.
IEEE ICCD., New York, Oct. 84, pp : 16-22.

[SAU86] G. Saucier, S. Hanriat,
Rule based logical synthesis for Silicon Compilers.
ICCAD, Nov. 86, à paraître.

[DUP86] E. Dupont, J. Idt, G. Saucier,
A Rule based System for the Optimal State Assignment of Controllers.
FJCC, Nov. 86, à paraître.

[BEL86] C. Bellon, M. Crastes de Paulet, S. Hanriat, J. Rarivomanana, G. Saucier,
CADOC System : A tool for multilevel description and test generation for VLSI circuits.
7th Int. Conf. on C.H.D.L., 85, pp : 364-380.

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs
. D. ETIEMBLE, Professeur
. C.V HAMACHER, Professeur

Monsieur HANRIAT Stéphane

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, Spécialité "Informatique".

Fait à Grenoble, le 22 septembre 1986

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

Résumé:

Cette thèse est consacrée à la synthèse logique pour les compilateurs de silicium. Les problèmes de synthèse combinatoire (fonctions Booléennes) et de synthèse séquentielle (contrôleurs) sont discutés, et abordés de façon nouvelle comme application de l'intelligence artificielle, et notamment des systèmes de règles.

Une méthode efficace est proposée pour la minimisation logique sur PLA, et une approche du test d'inclusion par la preuve formelle de tautologie est exposée.

Un programme en PASCAL a été réalisé, et a permi de synthétiser de manière automatique le contrôleur d'un microprocesseur 16 bits, mettant ainsi en évidence l'aptitude du système ASYL à traiter des problèmes de taille réaliste.

Mots clés :

Compilateur de silicium, synthèse logique, synthèse séquentielle, fonctions Booléennes, PLA, contrôleur, automate, système de règles, test d'inclusion, preuve de tautologie.