Compilation du silicium : application à la compilation de partie contrôle

Patrick Varinot

THESE

présentée par
Patrice VARINOT

pour obtenir le grade de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**
(arrêté ministériel du 5 juillet 1984)

(spécialité: Microélectronique)

Compilation de Silicium: Application à la compilation de partie contrôle

Date de soutenance : le 2 Février 1987.

Composition du jury:

M. G.	MAZARE	Président
M. F.	ANCEAU	Examinateurs
M. B.	COURTOIS	
M. A.A.	JERRAYA	
M. C.	LANDRAULT	
M. J.P.	MOREAU	

Thèse préparée au sein du laboratoire: IMAG/TIM3

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année 1987

Vice - Présidents : René CARRE
Jean-Marie PIERRARD

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BESSON Jean	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
BUYLE-BODIN Maurice	ENSERG
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHERUY Arlette	ENSIEG
CHIAVERINA Jean	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Mar	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Pierre	ENSIEG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSEEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PAUTHENET René	ENSIEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SERMET PIERRE	ENSERG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année 1987

Vice - Présidents : René CARRE
Jean-Marie PIERRARD

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BESSON Jean	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
BUYLE-BODIN Maurice	ENSERG
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHERUY Arlette	ENSIEG
CHIAVERINA Jean	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Mar	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Pierre	ENSIEG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSEEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PAUTHENET René	ENSIEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SERMET PIERRE	ENSERG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

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 CATEGORIE

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 CATEGORIE

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

DIRECTEUR DE RECHERCHE

LESBATS Pierre	Métallurgie

MAITRE 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

Personalités habilitées à diriger des travaux de recherche

DRIVER Julian	Métallurgie
GUILHOT Bernard	Chimie
THOMAS Gérard	Chimie

Professeurs à l'UER de Sciences de Saint-Etienne

VERGNAUD Jean-Maurice	Chimie des Matériaux et Chimie Industrielle

Je tiens à remercier :

Mr. François Anceau, chef de la division architecture pour l'intelligence artificielle à Bull, pour m'avoir acceuilli dans son laboratoire et m'avoir fait découvrir les joies de la recherche, et d'avoir bien voulu me faire l'honneur de participer au jury de cette thèse.

Mr. Bernard Courtois, directeur du laboratoire IMAG/TIM3 qui m'a permis à la fois de continuer la recherche, et de m'avoir prodigué des encouragements tout au long de ce travail.

Mr. Ahmed Jerraya, chargé de recherche au CNRS, pour l'intérêt constant qu'il a porté à mon travail.

Mr. C Landrault, chargé de recherche au CNRS, pour avoir accepté d'être rapporteur de ce travail.

Mr. Guy Mazaré, professeur à l'ENSIMAG, de me faire l'honneur de présider le jury de cette thèse, et d'avoir accepté d'être rapporteur de ce travail.

Mr. Jean-Pierre Moreau, chef du département aide à la conception de circuits intégrés de Thomson Efcis pour avoir accepté d'être membre de ce jury.

Tous mes amis et collègues de l'équipe architecture des ordinateurs pour toutes les discussions et les critiques qui m'ont permis de mener à bien cette thèse.

Sandrine Bonnelli pour sa participation efficace à la frappe de ce document et à la saisie des dessins.

Pour le service de reprographie de l'IMAG pour avoir assuré le tirage de cette thèse.

A mes parents,
A mes amis.

RESUME :

Les problèmes posés par la conception de circuits VLSI de plus en plus complexes ont mis en relief la nécessité d'une automatisation de la tâche des concepteurs. Pour ce faire, de nombreux travaux de recherche ont porté sur l'étude et la réalisation d'un nouveau type d'outils : les Compilateurs de Silicium.

La première partie de cette thèse a pour objet de faire une synthèse de l'état de l'art en la matière, et de présenter le compilateur de silicium SYCO.

La seconde partie traite des architectures compilables de circuits intégrés. Cette étude ne concerne que les circuits de type microprocesseur, basés sur la machine de Von Neumann, et porte plus particulièrement sur l'architecture de parties contrôles compilables.

La troisième partie propose des schémas topologiques d'implantation pour chacune des architectures de parties contrôles compilables étudiées. Ces schémas sont proposés dans le cadre de la définition du compilateur de silicium SYCO.

Le résultat de cette étude se concrétise par la réalisation d'un ensemble de logiciels (GENCIRCUIT, GENPC...), intégrés autour d'une structure de données LDS. Des exemples de réalisation de circuits illustrent l'intérêt d'une telle étude.

MOTS CLES:

Compilation de Silicium, Partie Contrôle Compilable, PLA, ROM, Architecture de circuit, Topologie de circuit, Système de CAO, Structure de donnée.

ABSTRACT :

Problems posed by the ever increasing complexity of IC designer have shown the necessity of automatizing designer's work. A lot of research was spend on the study and realization of a new class of tools : silicon compilers.

The first part of this thesis is aimed to make a synthesis on the state of the art in this domain, and to present the SYCO silicon compiler.

The second part deals with the compilable architecture of integrated circuit. This study concerns the class of the microprocessor circuit, based on the Von Neuman machine. It emphasis on the compilation of the control part.

The third part presents different schemes of topological for control part implementation, particulary for the control parts introduced in the second part of this study. These schemes are developped in the SYCO environnement.

As a result, this study has led to the realization of different tools (GENCIRCUIT, GENPC...) integrated around of the data structure LDS. Examples of circuit realization show the interest of this study.

KEYWORDS :

Silicon Compilation, Control Part Compilation, PLA, ROM, Circuit Architecture, Circuit Topology, CAD Systems, Data Structure.

INTRODUCTION

Le développement de la technologie des semiconducteurs, plus particulièrement dans le cadre des circuits de type ASIC (Application Specific Integrated Circuit), a amené les concepteurs de tels circuits devant un dilemme: Comment concevoir un circuit ayant des fonctionnalités bien déterminées, dans un temps raisonnable (quelques semaines), pour un coût de conception (homme*année ingénieur) faible.

Le contexte de ce dilemme revient à définir l'intérêt de la conception de ces circuits à faible volume de production, concurremment à une approche 'carte' (utilisation de circuits du marché à fonctionalité déterminée et permettant par assemblage de réaliser l'application désirée) réalisant les mêmes fonctionnalités.

La réponse apportée par les concepteurs et fabricants pour augmenter l'efficacité de la conception fut de produire des outils de génération automatique de structures régulières (PLA, ROM...), puis de structures plus complexes: partie contrôle ou partie opérative. C'est cette évolution de la conception qui nous amène à introduire le terme de compilation de silicium: acte de produire un circuit à partir d'une description de haut niveau de la fonctionnalité du circuit, de manière automatique ou semi-automatique, c'est à dire avec une interaction minimum de la part d'une personne humaine durant la phase de compilation.

C'est dans le cadre de la compilation de silicium, et tout particulièrement dans le cadre de la compilation automatique de partie contrôle, que le travail ci après a été effectué.

Le chapitre I est une introduction au compilateur de silicium **SYCO** en le situant par rapport à son environnement. Le compilateur SYCO correspond à l'environnement dans le cadre duquel, le travail sur la compilation de partie contrôle a été fait.

Tout d'abord, une étude des différents styles de compilateurs du marché pour situer le compilateur de silicium SYCO est proposée. Les compilateurs de silcium nommés sont aussi bien universitaires qu'industriels.

En préliminaire de la description de SYCO, une présentation succinte de **SYCOMORE** et de ses objectifs est faîte.

Le compilateur **SYCO** est décrit de manière générale, englobant tous ses aspects, en partant de la description algorithmique jusqu'à la réalisation des masques.

Le chapitre II discute des différentes architectures de circuits de type microprocesseur compilables. Une présentation de quelques modèles de partie opérative est faîte pour montrer l'interaction entre complexité de la partie opérative et complexité de la partie contrôle.

Ensuite, differentes architectures de parties contrôles sont proposées, en étudiant particulièrement leurs aspects fonctionnel et structurel.

Enfin, une présentation bibliographique des différents compilateurs de parties contrôles est proposée comme introduction au chapitre suivant.

Le chapitre III se destine à la présentation de la compilation de partie contrôle dans le cadre de SYCO. Pour ce faire, il décrit d'abord le modèle global d'implantation retenu pour l'architecture cible choisie dans le cadre du compilateur de silicium SYCO, et les limitations introduites par ce modèle.

Ensuite, il propose le modèle temporel qui est associé au modèle topologique et fonctionnel, et les améliorations pouvant être apportées, tout en faisant remarquer les contraintes que cela suppose.

On présente ensuite les architectures de parties contrôles dans le cadre du compilateur SYCO, tout en insistant sur l'aspect topologique. Suit une présentation des parties contrôles de type monopla, et les résultats liés.

Le chapitre IV reprend les points importants, et les orientations définies au long de ce mémoire. En outre, une étude des évolutions à moyen et long terme du compilateur SYCO, autant sur le modèle de compilation que sur les différents types de parties contrôles réalisables, est proposée. Ces propositions sont pour certaines d'entre elles en cours d'évaluation.

CHAPITRE I

COMPILATION DE SILICIUM,

LE COMPILATEUR SYCO

1 INTRODUCTION AUX COMPILATEURS DE SILICIUM

La conception des circuits intégrés a évolué avec la complexité des circuits à réaliser.

Cette évolution a amené les concepteurs à chercher à automatiser de plus en plus leur tâche, en vue de pouvoir réaliser ces circuits complexes en un temps (homme/année ingénieur) raisonnable. De cette course à l'automatisation est née la notion de "Compilateur de Silicium".

1.1 Notions et définitions

Du cahier des charges au circuit intégré sur silicium, il y a plusieurs étapes définies manuellement. A chacune de ces étapes, de l'information est apportée: cette information est d'ordre logique, architectural et/ou technologique.

D'un point de vue général, un circuit intégré est tout d'abord décrit en un langage de haut niveau, et ce de manière fonctionnelle et/ou comportementale ou algorithmique. Une description fonctionnelle consiste à décrire le circuit sous forme d'un ensemble de fonctions à réaliser par ce circuit. Cette forme ne fait pas intervenir de notion architecturale. Une description comportementale permet de décrire le circuit par son comportement. Ce comportement est déterminé par rapport à un ensemble de vecteurs d'entrée qui peut être appliqué au circuit. Une description comportementale d'un circuit ne fait pas intervenir de notion architecturale. Une description algorithmique permet de représenter un circuit sous la forme d'un algorithme, algorithme pouvant être écrit dans un langage de programmation de type PASCAL par exemple. Cette forme de description ne fait intervenir aucune notion architecturale, et peut être assimilée à la forme de description de plus haut niveau, comparativement aux descriptions fonctionnelle et comportementale.

De cette description de haut niveau, un schéma logique est déduit, schéma que l'on traduit ensuite en terme de matériel (machine physique). Ainsi l'on assiste à un processus de compilation d'un langage source avec enrichissement du texte initial [GROS 83].

Le terme de compilation de silicium fut introduit la première fois par D. JOHANNSEN [JOHA 79], qui l'utilisa pour décrire le concept d'un assembleur de cellules paramétrisables. Cette notion, restrictive, fut étendue par [WALL 83] à "un programme ou un ensemble de programmes pouvant produire une description physique du circuit à partir d'une description de haut niveau".

[WERN 82] a décrit par un schéma explicite ce que peut être un compilateur de silicium idéal. Dans les autres cas, l'on a affaire à une classe d'outils capable d'effectuer une partie seulement du schéma de compilation (figure 1.1).

Nous en tirons la définition générale suivante:

Définition : *Un compilateur de silicium est un système ou progiciel (ensemble de programmes) qui permet de transformer une description de haut niveau d'un circuit (ou d'une partie de circuit) en un ensemble d'images géométriques (appelé layout). L'ensemble des images géométriques peut être directement utilisé pour générer les masques du circuit.*

Nous allons maintenant aborder le problème de la classification des différents types de compilateurs que l'on rencontre.

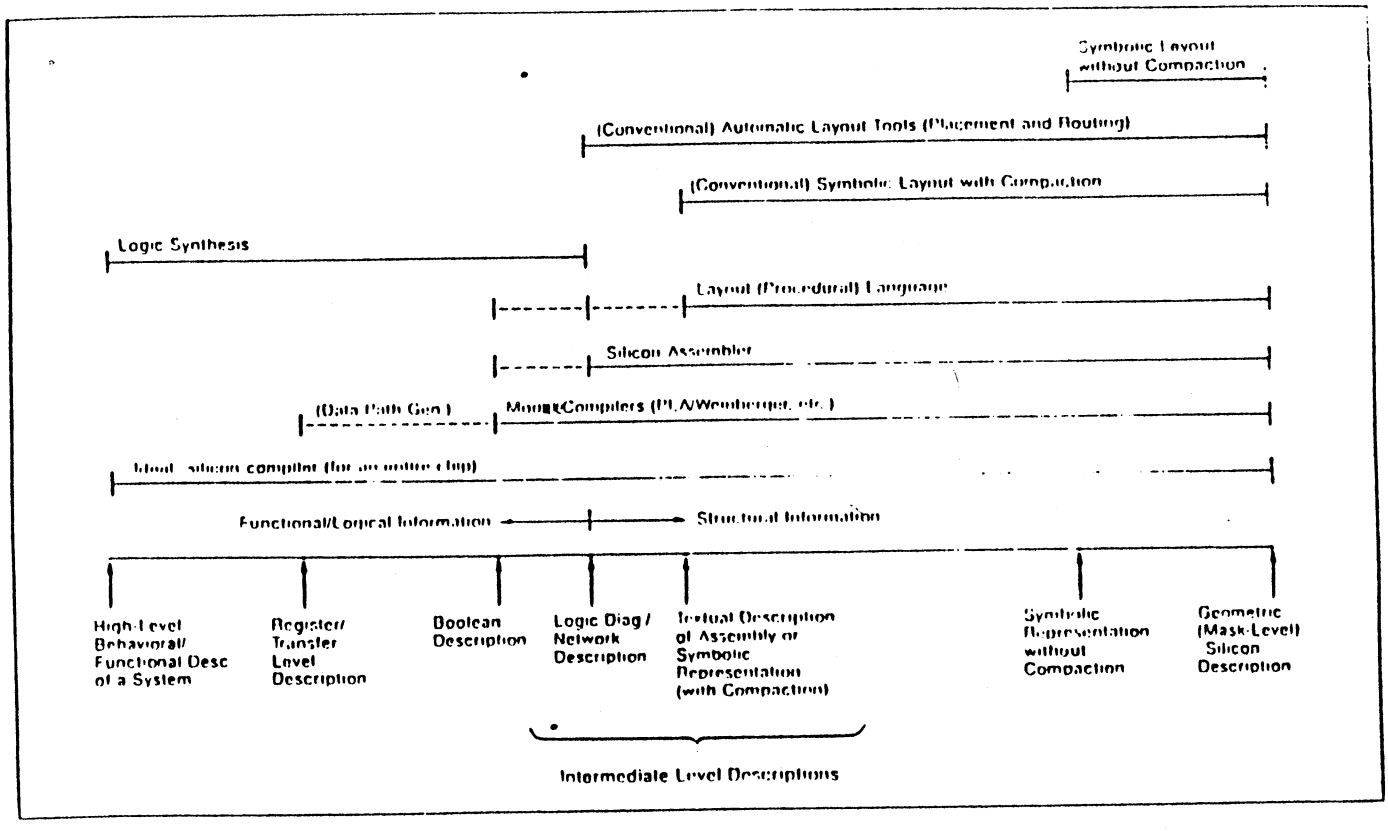


fig. 1.1 Un compilateur de silicium idéal embrasse la globalité du spectre de définition d'un circuit intégré [WERN 82]

1.2 Classification des compilateurs de silicium

On trouve actuellement dans la littérature ainsi que sur le marché, une grande variété de compilateurs de silicium. Ces compilateurs peuvent être classés selon trois critères indépendants les uns des autres:

* Le style de conception du circuit intégré,
* Le langage d'entrée du compilateur,
* L'aspect performances et convivialité.

1.2.1 Le style de conception du circuit:

Le style de conception est lié étroitement à la taille du circuit. En effet, certains styles limitent le nombre de portes qu'il est possible d'intégrer. On trouve les styles suivants:

- Les circuits prédiffusés ("Gate Array")
- Les circuits précaractérisés ("Standard Cell")
- Les circuits organisés à la demande ("Full Custom").

Ces trois styles de conception correspondent à une réalité physique du circuit, de même, l'on peut y adjoindre trois catégories de compilateurs. Les limites de chacun des styles sont présentées à la figure 1.2.

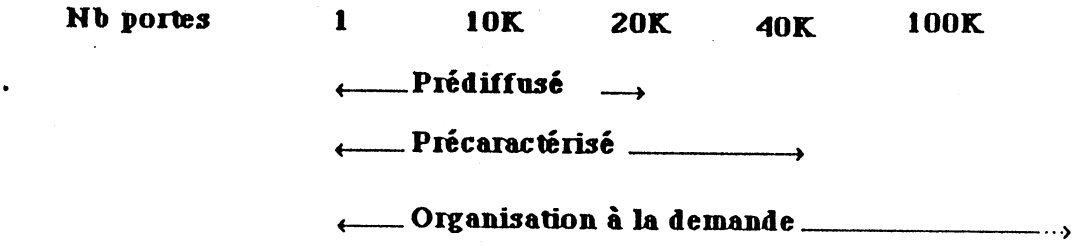


fig. 1.2 Complexité envisageable pour les différents styles de conception

1.2.2 Les langages d'entrée :

Les langages d'entrée de ces compilateurs peuvent, de même que les styles de conception, être classés en trois catégories:

- Langage de description du plan de masse: Le circuit est décrit par un ensemble de blocs interconnectés (où chaque bloc peut être décrit par un langage adapté au type du bloc). La plupart des compilateurs qui partent de ce type de description permettent de générer des circuits organisés à la

demande.
- Langage de description logique: Le circuit est décrit au niveau des portes (avec possibilité de hiérarchie). La plupart des compilateurs qui partent de ce type de description génèrent des circuits prédiffusés ou des circuits précaractérisés.

- Langage de description procédurale de layout (le terme anglais layout est employé pour représentation géométrique du dessin des masques): Dans ce cas, la conception d'un circuit revient à écrire un programme, qui, en s'exécutant, génère le layout du circuit. Ce programme peut faire appel à des cellules d'une bibliothèque et/ou à des programmes générateurs de structures particulières.

1.2.3 Aspect performance et convivialité:

Pour un utilisateur donné, pour lequel le type de circuit est fixé, et en général le langage d'entrée pour des raisons de compatibilité avec des outils déjà existants, les aspects de performances et de convivialité sont en général primordiaux. Les différents critères vus par l'utilisateur sont les suivants:

- Facilité de prise en main par le ou les concepteurs.
- Complexité du circuit réalisable à l'aide de ce compilateur. Cette complexité est liée à la technologie. Les limites actuelles sont bien sûr franchissables par évolution de la technologie (figure 1.2).
- Durée du cycle de conception: Le cycle de conception correspond au temps necessaire pour estimer la validité d'une solution donnée. Cette notion intègre aussi la facilité de modification d'un dessin ou de l'architecture d'un circuit.
- Surface perdue par utilisation d'un compilateur comparativement à une solution manuelle.
- Importance du coût de la conception (temps CPU et main d'oeuvre)
- Importance de la simulation: ce facteur est très variable en fonction du type de compilateur utilisé.
- Importance du test, notion de facilité de test par l'apport de la génération automatique, ainsi que le choix de structures facilement testables.
- Importance du délai d'obtention du premier prototype.

1.2.4 Représentation à l'aide de la carte en 'Y' de [GAJS 86a]:

Pour chaque type de compilateur dont l'étude suit, une représentation à l'aide de la carte en "Y" de [GAJS 86a] va être produite. Cette carte permet de représenter un compilateur suivant les trois domaines de son application, le domaine comportemental (et/ou fonctionnel), le domaine structurel et le domaine

géométrique. Par exemple, un compilateur idéal [WERN 82] sera représenté de la manière suivante (figure 1.3). Le compilateur part d'une description fonctionnelle ou algorithmique, construit la représentation structurelle, puis génère la description géométrique du circuit.

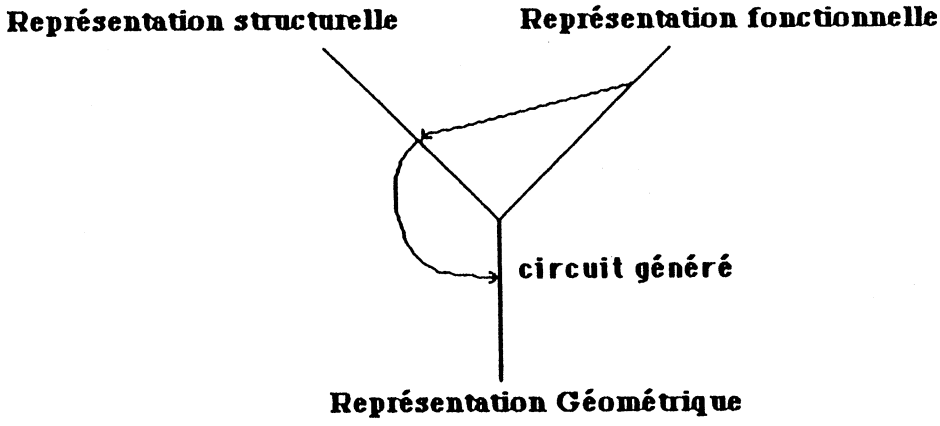


fig. 1.3 Représentation d'un compilateur "idéal" à l'aide de la carte en "Y" [GAJS 86a]

Par la suite nous allons présenter différents types de compilateurs, dont une étude brève sera faite.

2 ETUDE DE DIFFERENTS TYPES DE COMPILATEURS DE SILICIUM

2.1 Synthétiseurs de logique

Ce genre de compilateurs permet à partir d'une description de haut niveau (description comportementale, fonctionnelle ou algorithmique), d'extraire des équations booléennes, soit dans le cadre d'une structure d'accueil particulière, soit sous forme canonique implantable à l'aide de portes logiques de base (exemple du circuit prédiffusé).

Les synthétiseurs de logique tiennent compte du type de la technologie d'accueil (exemple en NMOS, l'utilisation de portes du type NOR est moins coûteuse en transistors que celle de portes OR ou AND, contrairement à la technologie bipolaire).

La programmation d'un tel type d'outil fait appel, dans le cadre des derniers outils sortis, à l'intelligence artificielle pour la synthèse logique. En effet, les règles de simplification logique sont construites en fonction de la technologie, ou des différents aspects des structures d'acceuils. A partir de ces règles, un programme d'extraction est lancé sur le texte source et produit un texte objet utilisé en entrée par les outils de génération de masques (voir P. Malardier [MALA 85a]).

A l'aide de la carte en "Y", le déroulement du processus d'un synthétiseur de logique se représente tel que la forme d'entrée est de type fonctionnelle. A partir de cette forme d'entrée, une description structurelle est fournie, en tenant compte des formes logiques de base implantables. Cette forme structurelle est alors traduite en une description géométrique du circuit (figure 1.4).

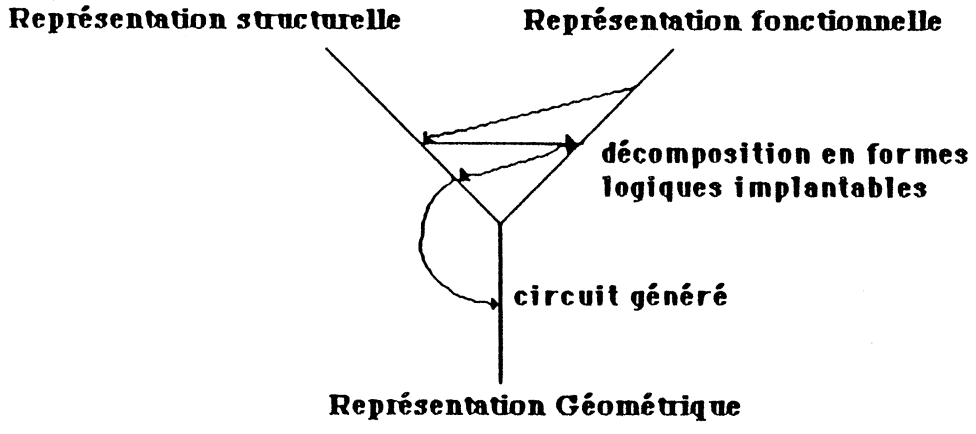


fig. 1.4 Représentation d'un synthétiseur de logique

2.2 Générateurs de modules

Dans la notion de générateurs de modules, l'on sous-entend l'idée que le circuit à réaliser est composé de blocs logico-fonctionnels, directement synthétisables sous une forme physique. Les générateurs de modules ont chacun une entrée spécifique, et génèrent en sortie le layout.

Plusieurs types de générateurs de modules ont été développés au long de la dernière décennie. Ils sont surtout destinés à produire du layout de structures régulières telles les RAM, ROM, PLA, WEINBERGER...[CHUQ 84] [MiSa 83] [SOUT 83] [WEIN 67].

La deuxième évolution de ces compilateurs s'est traduite par une génération de structures plus complexes, par exemple des structures de contrôle à un ou plusieurs PLAs, ou des structures de parties opératives. Cette deuxième génération s'appuie exclusivement sur la première génération de compilateurs de blocs [MEYE 84] (compilateurs spécialisés).

A l'aide de la représentation de [GAJS 86a], les générateurs de modules sont décrits de la manière suivante: partant d'une description structurelle du module à générer, le compilateur fournit la description fonctionnelle. A partir de cette forme fonctionnelle implémentable car correspondant à une structure pré-établie, on déduit la forme structurelle définitive, que l'on trraduit ensuite en une description géométrique.

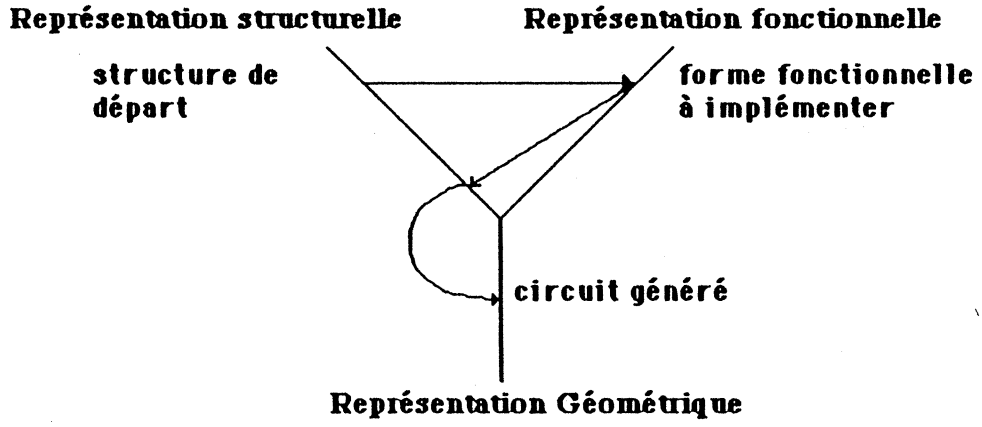
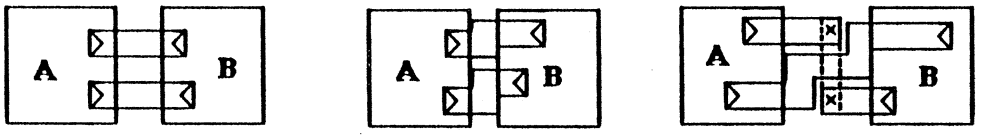


fig 1.5 Représentation d'un générateur de modules

2.3 Générateurs procéduraux de layout

Ce genre de compilateur ne tient pas compte en général de la logique des blocs, mais considère un bloc comme une entité possédant des frontières englobantes, et un certain nombre de connecteurs orientés (Nord, Sud, Est, Ouest).

Un assembleur se charge de relier ces différents blocs entre eux, suivant des lois strictes d'assemblage. Par exemple un bloc A peut être relié et placé à coté d'un bloc B par abouttement de ses connecteurs (figure 1.6.1) par dévoiement simple (figure 1.6.2) ou par routage bi-couche (figure 1.6.3).



(1) (2) (3)

fig. 1.6 Différents types d'assemblage

Une deuxième forme d'assemblage est connue sous le nom d'outil de placement-routage. Beaucoup plus souple que le précedent, il est moins économe en surface de silicium. Il consiste à placer un certain nombre de blocs, puis à router leurs connecteurs à l'aide de routeurs spécialisés utilisant des canaux de routage. Ils sont aussi utilisés dans le cadre des compilateurs de core microprocesseurs. Ces derniers correspondent à l'utilisation de circuits de base simples que l'on assemble pour construire des circuits plus complexes [NEMO 81] [KeKo 86] [ToKa 86] [BeDo 86].

Les méthodologies utilisant des assembleurs doublés de routeurs, tels les 'Gate array' ou 'Standard cell' construisent un circuit de manière hiérarchique à partir de cellules de base. La conception se déroule de la manière suivante, le concepteur référence les différentes cellules de base, en élabore la structure, en définit la fonction, et utilise ces résultats pour construire le niveau structurel le plus haut (le circuit) . A partir de ce niveau structurel, la conception se termine par la simulation, placement et routage des cellules constituant le circuit. Ces méthodes n'exploitent pas efficacement la hiérarchie puisque la simulation, le placement et le routage ne sont appliqués que sur le plus bas niveau de représentation et aussi le plus coûteux [GAJS 86b]. Ceci est illustré sur la représentation de la carte en 'Y' de la figure 1.7.

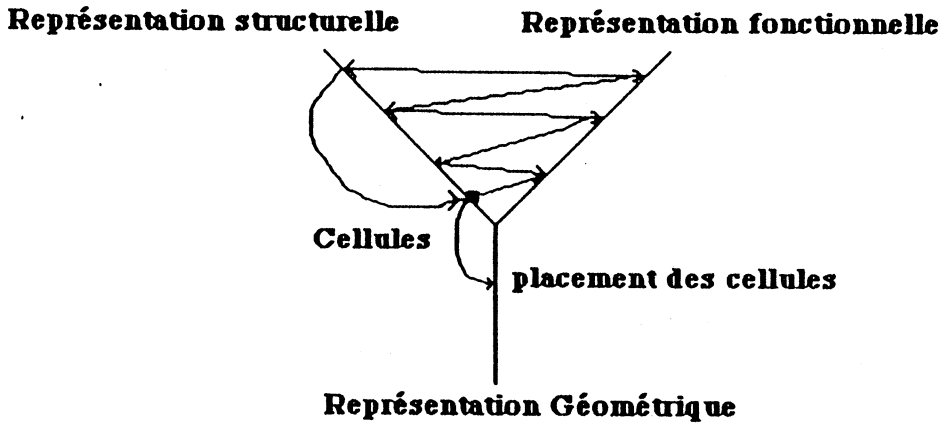


fig. 1.7 Représentation d'un générateur procédural de layout

Une troisième forme de générateurs procéduraux de layout est représentée par les outils type **STYX** [JeRo 85] (produit THOMSON/CSAO-INPG/TIM3) ou **ALI** [LiNo 82]. Ces outils permettent de générer le layout au niveau du rectangle. C'est à dire qu'un ensemble de procédures en s'exécutant produit le layout. Une couche supérieure introduit les fonctionnalités des assembleurs de blocs comme présenté ci-dessus. Cette troisième forme de générateurs est particulièrement intéressante, car elle conserve la hiérarchie de conception du circuit.

2.4 Compilateurs d'architectures

Les compilateurs d'architectures consistent à générer le layout à partir d'une description architecturale du circuit. Le circuit est décrit comme un ensemble de blocs structurels dont le contenu logique est connu, blocs sur lesquels on effectue un assemblage déterminé à l'avance par un plan de masse. La représentation en 'Y' suivante représente la démarche suivie par un compilateur d'architectures (figure 1.8).

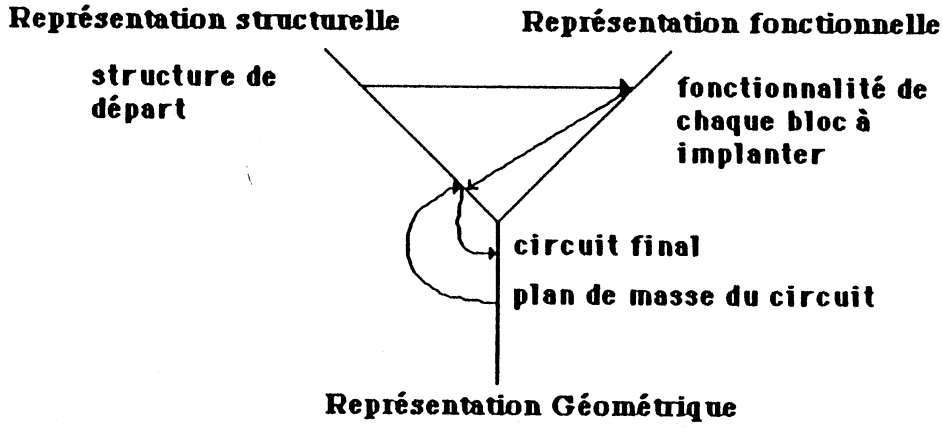


fig 1.8 Représentation d'un compilateur d'architectures

Deux compilateurs du commerce peuvent être classés dans ce genre: **CONCORDE** de **SST** et **GENESIL** de **SCI**. Ce type de compilateur peut être utilisé aisément par des ingénieurs peu familiarisés avec le détail de la conception de circuits intégrés.

Autour de l'environnemet **GTD** de **SDL** [ToKa 86] a été développé un compilateur d'architecture basé sur les core microprocesseurs SDL2000. Un "core" microprocesseur est une structure de coeur de microprocesseur dont l'architecture est strictement définie. La partie contrôle d'un "core" microprocesseur est souvent basée sur une solution à ROM. Le compilateur, à partir d'un algorithme, construit le microprogramme destiné à la ROM. Dans le cas du SDL2000, la partie opérative possède une structure de base pouvant être complexifiée par ajoût de fonctionnalité (en matériel cela se traduit par une UAL, Unité Arithmetique et Logique, plus complexe, et/ou des registres en plus). L'intérêt d'une telle approche réside dans:

- L'utilisation d'une machine standard de surface supérieure au plus à 30% d'une même réalisation produite manuellement [ToKa 86],
- L'utilisation de modules générateurs de base (PLA, Registres, UAL...) produisant un layout très régulier,
- Une architecture orientée ASIC, par la modularité de sa conception.

Une telle approche avait déjà été proposée par M. NEMOUR [NEMO 81] et M. MARQUES [MARQ 79] pour la réalisation de machines orientées traitement du signal. L'architecture de telles machines utilisait un ensemble de coeurs de ("core") microprocesseur interconnectés, dont la structure était redéfinissable et orientée vers l'utilisation de structures de base régulières (PLA, ROM).

2.5 Compilateurs spécialisés

On entend par compilateur spécialisé, un compilateur d'architecture lié à une architecture-cible particulière. Les compilateurs spécialisés permettent de compiler tout ou partie d'un circuit (compilateur de partie opérative, compilateur de contrôleur...). Le compilateur spécialisé permet de compiler une description algorithmique plus aisément que des compilateurs d'architectures universels. En effet, le choix d'une architecture-cible limite les ambiguïtés sur la traduction de la description algorithmique du circuit vers une description structurelle puis géométrique.

Le compilateur spécialisé, de par sa simplicité de fonctionnement, et la rapidité de la traduction d'une description algorithmique vers une description géométrique, permet à l'utilisateur de remodeler sa description algorithmique, jusqu'à l'obtention d'une description géométrique qui lui paraisse acceptable. La traduction correspond à un passage de la description initiale par différents outils, outils qui rajoutent de l'information à la description initiale. L'information est d'ordre structurelle,

logique et électrique, et en dernier lieu géométrique (figure 1.9). Une autre alternative est souvent offerte au concepteur: une variété importante d'architectures-cible, compilables à partir de la même description algorithmique. Ce choix permet au concepteur de tester les différents résultats obtenus par des compilations successives, et de choisir ainsi la "meilleure" structure pour un algorithme donné.

Une autre voie qui se dessine dans les compilateurs spécialisés est l'introduction de systèmes experts dans le choix de la structure réceptrice de la compilation d'une description de haut niveau [BrGa 86].

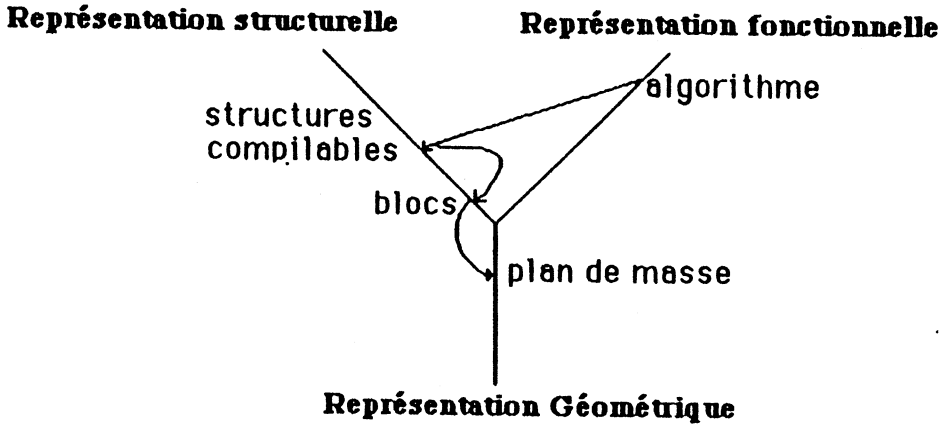


fig. 1.9 Représentation d'un compilateur spécialisé

Le compilateur **SYCO**, présenté par la suite, entre dans ce groupe de compilateurs.

3 LE COMPILATEUR DE SILICIUM "SYCO"

Avant de décrire le compilateur de silicium **SYCO**, nous allons présenter le cadre dans lequel sa conception s'est déroulée.

3.1 Présentation du projet SYCOMORE

Le projet **SYCOMORE** a pour objectif la réalisation d'une station de travail pour la conception de circuits intégrés VLSI complexes. Il a été lancé par un groupement d'industriels et de laboratoires publics: THOMSON, BULL SYSTEMES, INRIA ET INPG/TIM3. Il se propose d'étudier et de mettre au point :

- Des outils pour la compilation de descriptions procédurales avec gestion automatique de la correspondance entre les descriptions électrique et logique d'une part, et la description du layout. L'outil de layout STYX permet la description et l'assemblage du layout de manière symbolique. D'autre part le projet prévoit un éditeur de plan de masse et un éditeur d'interconnexion. Enfin le système contient une bibliothèque d'outils de routage [MASS 85].

- Des outils d'assistance pour le circuiteur. Ces outils permettent aux concepteurs de mettre au point des blocs particuliers ou des éléments de bibliothèque. Ces outils permettent la saisie et la mise au point de descriptions logique et électrique. Ils offrent des moyens de synthèse logique (passage d'une description logique à une description électrique). Ils permettent aussi l'analyse de temps et la simulation multi-niveaux (HADES [MALA 85a].

- Des outils opérant au niveau comportemental. Ces outils permettent la description, la mise au point et la simulation de circuits au niveau comportemental [LaNg 85].

- Un compilateur de silicium pour les circuits pouvant être décrits de manière algorithmique (circuits de type microprocesseur). Ce compilateur **SYCO** [JeVa 86] permettra de générer des masques d'un circuit en partant d'une description algorithmique.

Dans la suite, seule la partie concernant le compilateur **SYCO** sera développée.

3.2 Présentation du compilateur SYCO

Le compilateur de silicium **SYCO** est un système destiné aux circuits VLSI pouvant être spécifiés par un algorithme d'interprétation. Le système **SYCO** prend comme base de départ une description algorithmique de haut niveau et produit les masques d'un circuit de type microprocesseur qui réalise cet algorithme.

Un circuit de type microprocesseur est une machine physique réalisant un algorithme d'interprétation à l'aide d'un langage propre (figure 1.10).

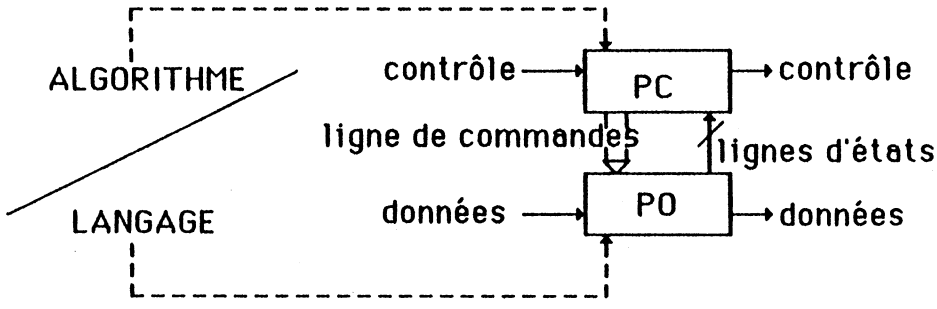


fig. 1.10 Définition de la partie contrôle et de la partie opérative

La partie contrôle active les opérations à effectuer par la partie opérative à l'aide d'un ensemble de lignes de commande. La partie contrôle exécute un ou plusieurs algorithmes de contrôle. Elle communique d'une part avec l'extérieur du circuit (lignes de contrôle externes et instructions), et d'autre part avec la partie opérative qui réalise les instructions de commande que la partie contrôle lui envoie. La partie opérative exécute le langage de base de la machine.

SYCO utilise une architecture-cible basée sur un modèle d'interprétation multiniveaux. L'interprétation d'un langage de commande donné est décomposée en un ensemble de niveaux d'interprétation. Chaque niveau d'interprétation transforme les commandes provenant du niveau supérieur en un ensemble de commandes pour l'interpréteur directement inférieur (hiérarchiquement). Chaque interpréteur est directement implanté en une structure physique constituant une tranche de layout du circuit. Ainsi un circuit contient un ensemble de tranches horizontales, l'une composant la partie opérative, et un nombre fini constituant la partie contrôle. Cet ensemble de traductions est réalisé automatiquement du niveau du langage algorithmique jusqu'au niveau du layout [JeVa 86].

Suivant le modèle de représentation de GAJSKI [GAJS 86b] (représentation en Y), le compilateur de silicium **SYCO** réalise le schéma suivant (figure 1.11):

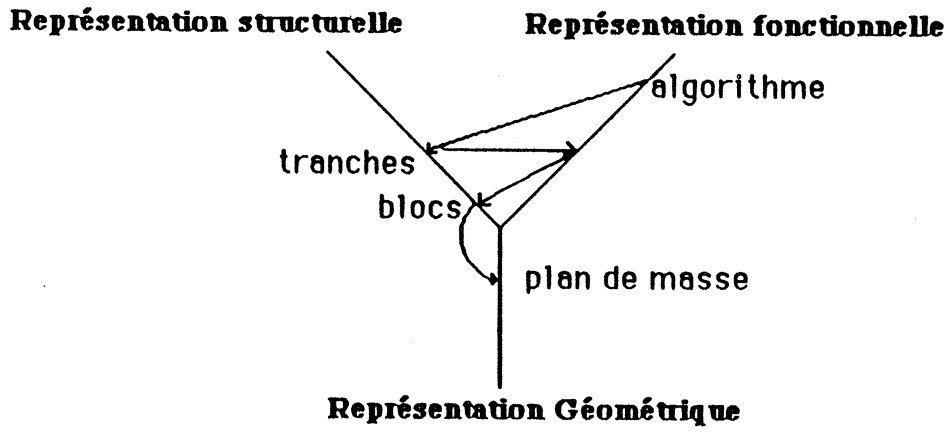


fig. 1.11 Représentation en Y du compilateur **SYCO**

A une description algorithmique donnée, correspond une structure de circuit en tranches. Fonctionnellement, chaque tranche est décrite. A partir de cette description structuro-fonctionnelle, on déduit un schéma en blocs, blocs que l'on implante suivant un plan de masse prédéfini.

3.2.1 Modèle fonctionnel du compilateur SYCO

Tout circuit de type microprocesseur peut être décrit à l'aide d'un algorithme d'interprétation de commandes. Cette méthode formalisée dès 1979 par F. Anceau [ANCE 79] [MARQ 79] [OBRE 82] est telle qu'un interpréteur d'un langage de haut niveau peut être décomposé en une série d'interpréteurs utilisant des langages intermédiaires. Cette technique, utilisée dans le monde de l'informatique des gros ordinateurs est appliquée au monde du VLSI [ANCE 86]. Un des exemples en est le 8085 de chez INTEL. Ce schéma récursif est le principe de base du système **SYCO** (figure 1.12).

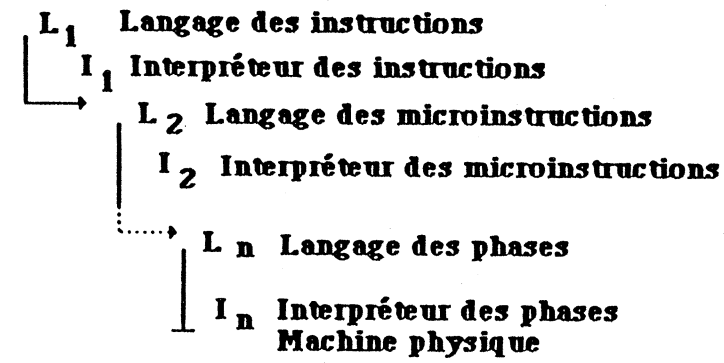


fig. 1.12 Schéma d'interprétation

Remarque:

Dans le compilateur de silicium SYCO, il n'y a en fait qu'un seul langage qui décrit le circuit à l'aide d'une hiérarchie de procédures. Il apparait qu'il y a une évolution par rapport au schéma de principe énnoncé ci dessus. Dans SYCO, chaque interpréteur est une machine physique qui sera implémentée. Le dernier interpréteur représente la partie opérative.

Le principe de la compilation consiste à produire un ensemble d'interpréteurs qui, assemblés, réalisent la même fonction que l'interpréteur spécifié par la description de départ. Ceci peut être formalisé de la manière suivante:

L'interpréteur d'un langage de niveau (L1) peut être décomposé en (n-1) interpréteurs travaillant sur (n) langages (L1,L2...,Ln). L'interpréteur de niveau (i) accepte le langage (Li) et génère un langage du niveau (Li+1). Les primitives du langage de niveau (n) constituent les opérations de base de la machine.

Chacun de ces interpréteurs est réalisé par une machine d'états finis. L'on distingue deux types de machine (au niveau de la finalité):

-Les machines de contrôle : Elles transforment une instruction de niveau i en une séquence d'instructions de niveau i+1.
- Les machines d'exécution (Partie opérative): Elles réalisent les opérations de base (transformation d'une instruction de niveau i+1, mais ici, le niveau i+1 correspond au niveau "données d'un programme utilisateur du processus).

L'équivalence fonctionnelle de ce schéma est telle qu'une machine de niveau i peut être décomposée en une machine de contrôle et une machine de niveau i+1 (figure 1.13).

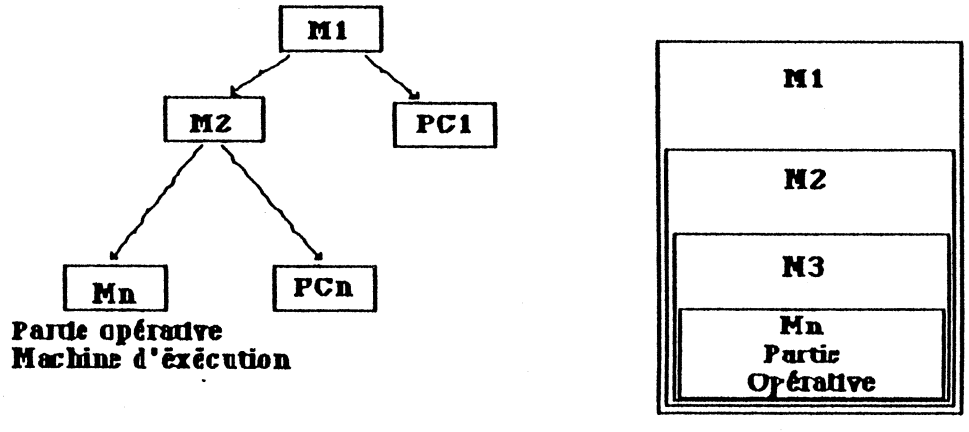


fig. 1.13 Equivalence fonctionnelle

Le modèle fonctionnel repose sur une structure d'appel , telle que les instructions de niveau i ne peuvent être décomposées qu'en une séquence d'instructions de niveau i+1 exclusivement. Ce modèle fonctionnel correspond à une description à l'aide d'un ensemble de procédures (ou fonctions) d'un circuit (figure 1.14).

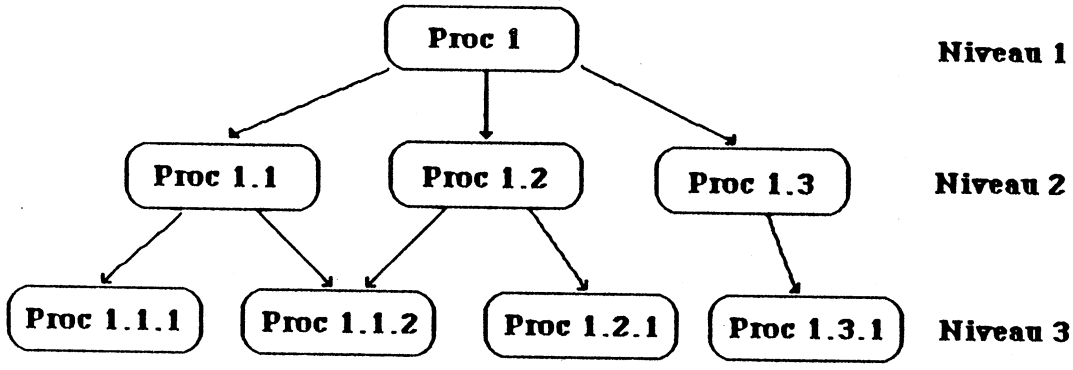


fig. 1.14 Description à l'aide de procédures (ou fonction) d'un circuit

3.2.2 Modèle architectural

Le modèle architectural est directement déduit du modèle fonctionnel. En effet, à chaque niveau fonctionnel (correspondant à un interpréteur), l'on associe un étage de contrôle. Cette solution a été choisie pour la facilité de réalisation qu'elle implique (figure 1.15).

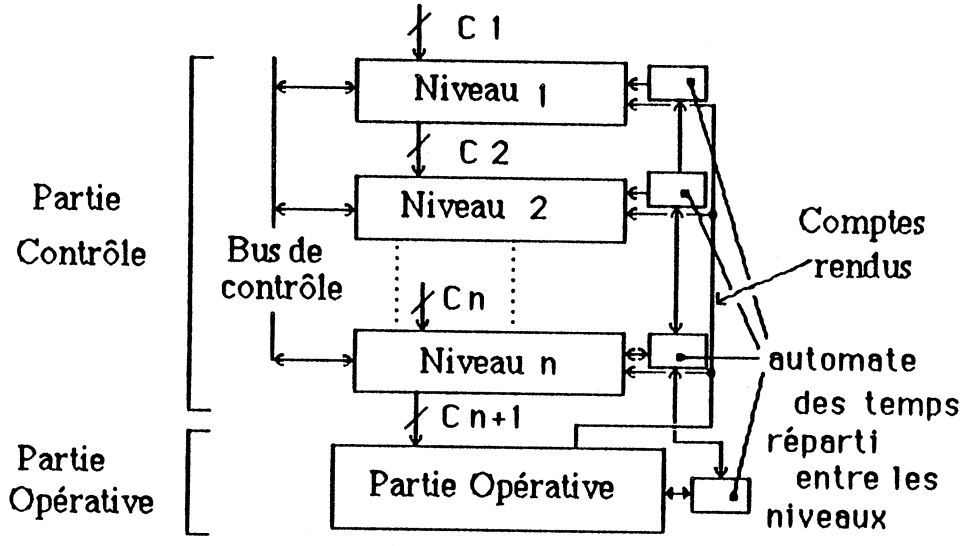


fig. 1.15 Représentation architecturale

Chaque niveau d'interprétation de la partie contrôle de la machine globale est constitué d'un bloc rectangulaire appelé tranche de contrôle. Le circuit est constitué d'un bloc rectangulaire résultant de l'empilement de plusieurs tranches de contrôle et de la tranche de partie opérative. SYCO fait appel à des modules générateurs de spécifications de masques spécialisés pour générer les différentes tranches. Les tranches sont ensuite assemblées automatiquement.

L'architecture cible de la partie opérative est basée sur le principe de "bit slice", c'est à dire que l'on construit une tranche de partie opérative de un bit, que l'on assemble ensuite pour former la partie opérative de n bits. Le modèle global comprend plusieurs sous-parties opératives permettant le travail en parallèle ou le transfert de données. Au niveau topologique, le "bit slice" est choisi bâti sur une structure à deux bus duaux.

La partie contrôle d'un circuit généré par SYCO est constitué d'une hiérarchie d'automates en cascade. Chaque étage de contrôle posséde une cellule de synchronisation. Cette cellule de synchronisation est une partie de l'automate général des temps (cf § II et III). L'automate des temps se charge de la synchronisation des différents étages de contrôle ainsi que de celle de la partie opérative. Il fournit les différentes horloges du circuit. Les sorties d'un automate de contrôle sont constituées :

- des commandes de l'étage immédiatement inférieur,
- d'un compte rendu d'exécution destiné à l'étage immédiatement supérieur, (qui transite par le bus de contrôle fig 1.15),
- des signaux de synchronisation,
- ainsi que des nouvelles valeurs des variables de contrôle et des signaux de contrôle d'écriture pour toutes les variables de contrôle modifiées à ce niveau.

Mentionnons un dernier point en ce qui concerne le modèle architectural de **SYCO**. **SYCO** permet l'emploi de procédures paramétrisées, ce qui évite de réécrire plusieurs procédures légèrement différentes. Pour cela, à chaque paramètre l'on associe une variable de contrôle. Ces variables sont réalisées à l'aide de bascules ou de registres selon que les paramètres sont booléens ou non. Ces variables de contrôle, ainsi que les signaux de contrôle externes, les comptes rendus de la partie opérative, les commandes de séquencement interne et externe les commandes de niveau supérieur, et le compte rendu d'exécution de l'étage inférieur constituent les entrées de l'automate de contrôle. L'automate des temps est défini par rapport au modèle temporel désiré (dans le cas de SYCO, le modèle temporel est de type procédural).

3.2.2.1 Modèle topologique:

Le modèle topologique général est tel que les différentes tranches composant le circuit, tranches de partie contrôle, ou tranche de partie opérative, sont topologiquement assemblées l'une au dessus de l'autre. Ce modèle est directement déduit du modèle architectural (hiérarchie d'interpréteurs). Autour du coeur du circuit ainsi créé, les plots sont assemblés (voir figure 1.15 du modèle architectural).

La tranche de partie opérative fixe la largeur du circuit. Par contre, en ce qui concerne les tranches de parties contrôles, leur taille est fonction de la complexité de l'algorithme à un niveau i, ainsi que du nombre d'états à générer. Leur taille varie généralement par une diminution en fonction de l'éloignement de la partie opérative. Cette approche se traduit par un effet pyramidal.

Pour limiter cet aspect, deux solutions sont utilisées par le compilateur **SYCO**:

- Transformation de la description d'origine [BEKK 86], afin de répartir sur l'ensemble des niveaux de contrôle la complexité de l'algorithme d'interprétation.
- Transformation topologique des niveaux de parties contrôles par jeu de déformation des blocs constituant une tranche de partie contrôle. Par exemple le niveau i est déformé de telle façon que la largeur soit égale à la largeur de la partie contrôle i+1, d'où globalement, tout s'aligne sur la taille de la partie opérative. Par exemple une structure de partie contrôle monoPLA peut être déformée topologiquement pour adapter le PLA à son environnement, et ce à l'aide d'outils type PAOLA [CHUQ 84].

3.2.2.2 Modèle temporel

L'horloge de base du circuit est décomposée en quatre phases. L'ensemble de ces phases correspond à l'exécution d'un transfert dans la partie opérative. Le temps d'exécution d'une instruction au niveau d'une tranche de partie contrôle i est équivalent à un cycle (quatre phases).

Le circuit est considéré comme une zône isochrone, c'est à dire que l'horloge ne subit pas de déphasage d'une extrémité à l'autre de la zone. On considére que le modèle temporel est de type synchrone (la même horloge pour tout le circuit). Cette hypothèse est restrictive sur la taille maximale des circuits réalisables à l'aide de SYCO (conservation de la zone isochrone).

Modèle temporel de type procédural

A chaque cycle, une seule tranche de partie contrôle est active. Lorsqu'une tranche de partie contrôle exécute une procédure, elle doit attendre que l'exécution

de cette procédure soit terminée, avant d'en exécuter une nouvelle. Ce modèle temporel est tel que la partie opérative se trouve sous employée, ceci s'amplifiant de manière linéaire en fonction du nombre d'étages de partie contrôle.

En effet si un étage i déclenche une opération, son exécution par la partie opérative se fera n-i cycles plus tard, et le début d'une nouvelle instruction n-i+1 cycle plus tard.

Ce modèle temporel peut être amélioré si l'on introduit la possibilité que plusieurs étages soient actifs au même moment. Deux modèles temporels seront présentés plus en détail au chapitre 3.

3.2.3 Langage d'entrée, présentation de LDS

Le langage d'entrée du compilateur doit permettre de décrire l'algorithme de fonctionnement d'un circuit. On doit disposer en entrée d'un langage algorithmique permettant de spécifier des transferts, des actions conditionnelles et de séquencement. De plus le langage doit disposer de fonctions et de procédures permettant de hiérarchiser l'algorithme comme spécifié par le modèle fonctionnel.

Le compilateur **SYCO** utilise dans sa version actuelle un sous-ensemble du langage de description de circuit **LDS** [LaNg 85]. Historiquement, le premier langage de description utilisé dans le cadre de **SYCO** fut **IRENE** [MARI 86]. **IRENE** est un langage de niveau RTL (Register Transfert Level). C'est un langage modulaire et structuré, autour duquel a été développé un simulateur fonctionnel [BOUR 84], et un extracteur de partie opérative et de partie contrôle automatique [MART 85] [MHAY 85].

Ces deux langages de type procédural permettent de décrire le comportement et la structure de systèmes digitaux. La description qu'elle soit comportementale ou structurelle, est de niveau transfert de registres (RTL), c'est à dire qu'elle fait intervenir des types d'objets tels les registres ou mémoires.

Le langage **LDS** est un langage multi-sémantique à syntaxe unique qui permet la manipulation de la description d'un modèle par plusieurs processeurs porteurs de leur propre sémantique [BOUR 86].

LDS contient en son sein deux processeurs de base: Un compilateur et un simulateur.

- Le compilateur génère à partir d'une description texte source une structure intermédiaire sur laquelle travaillent le simulateur et le compilateur **SYCO**.
- Le simulateur est de type fonctionnel et permet de simuler à partir de la structure compilée le comportement du circuit, et de vérifier ainsi la cohérence de la description faite par le concepteur.

Le langage **LDS** permet de décrire le circuit de manière structurelle. La description structurelle consiste en la décomposition du circuit en modules interconnectés entre eux par des signaux. La description est hiérarchisée : un module peut en appeler un autre.

Afin de distinguer la description structurelle de la description comportementale, **LDS** définit deux types de modules:

- Les modules structuraux (SMODULE)
- Les modules comportementaux (CMODULE)

Un circuit peut ainsi être vu comme un ensemble de blocs (ou modules) interconnectés entre eux. Ces blocs possèdent une notion structurelle et comportementale. A chaque module structurel de base est associé un module comportemental qui permet de définir son fonctionnement. L'évolution de la sortie en fonction des entrées doit être spécifiée.

Le compilateur **SYCO** utilise un sous ensemble de **LDS** , lequel impose des restrictions sur les modules structuraux: Un seul module structurel est déclaré, celui correspondant au circuit. A ce module est associé un module comportemental pouvant appeler d'autres modules comportementaux.

Cette hiérarchie dans la description correspond de manière directe au modèle fonctionnel de **SYCO**. Un niveau i de la description fonctionnelle LDS est constitué d'un ensemble {Mi} de modules comportementaux (CMODULE). Cet ensemble {Mi} constitue l'ensemble des instructions d'un langage de niveau i-1 $\{I_{Li-1}\}$. L'ensemble des instructions des modules de niveau i $\{I_{Li}\}$ correspond à un langage de niveau i. Ainsi l'ensemble $\{I_{Li-1}\}$, ou langage de niveau i-1, est interprété et produit un ensemble $\{I_{Li}\}$ constituant le langage de niveau i.

Constitution d'un SMODULE

Un SMODULE est caractérisé par :

- La liste de ses bornes ou variables d'entrées/sorties
- La déclaration des variables utilisées

Les variables sont classées en deux types :

- les variables de données : ce sont les registres de la partie opérative. Ces registres sont de type point mémoire avec mémorisation sur un état et non sur un front.
- les variables de contrôle : ce sont des variables qui sont réalisées par des bascules ou des registres selon que ces variables sont booléennes ou non. Ces bascules ou registres sont situés dans la partie contrôle et servent entre autres à mémoriser les paramètres des procédures.

Remarque

Cette distinction entre variables de contrôle et variables de données a été introduite pour les besoins du compilateur de silicium. En effet, il est peu intéressant d'intégrer les variables de contrôle booléennes ou qui portent sur un faible nombre de bits dans une partie opérative dont le nombre de bits est bien supérieur. Cet argument ne concerne que les variables de la partie déclaration. Il est bien sûr nécessaire d'établir une différence fondamentale entre les entrées/sorties de type données et les entrées/sorties de type contrôle.

Constitution d'un CMODULE:

La description d'un circuit est constituée d'une part d'un SMODULE, d'autre part d'un CMODULE qui lui est associé par l'instruction LINK. Un CMODULE est constitué d'une suite de séquences. Une séquence est une suite d'instructions étiquetées par le nom de la séquence. Les instructions sont exécutées séquentiellement, sauf si une action de branchement est notifiée. L'appel d'un CMODULE (EXECUTE) provoque le déroulement des séquences de ce module. A la fin de son exécution le processus appelant qui était jusqu'alors bloqué, reprend son déroulement.

Une instruction (PINST en LDS) d'une séquence est constituée d'un ensemble d'actions se déroulant de façon concourante.

Les actions peuvent être conditionnelles (IF, CASE) ou inconditionnelles. Parmi les actions inconditionnelles, on distinguera les affectations : actions opératives (transferts, opérations) et actions de contrôle (SET, RESET) qui permettent d'affecter des valeurs aux variables de contrôle (d'entrée-sortie ou internes). Les actions de séquencement : le branchement à l'intérieur d'un CMODULE (NEXT) et le retour au niveau supérieur ou échappement d'un CMODULE appelé pour revenir au niveau de l'appel (EXIT) font aussi partie de la classe des actions inconditionnelles. Seules les actions opératives d'une part et les affectations des variables de contrôle d'autre part peuvent se dérouler en parallèle, une seule tranche (de contrôle ou opérative) fonctionnant à un cycle donné (modèle temporel).Les actions qui se déroulent en parallèle sont regroupées et sont délimitées par des parenthèses.

Exemple de description d'un Cmodule

```
CMODULE          MICROP;

<rest>      IF (restart = 0) NEXT rest;  END;

<start>     pc :=0;
<newinstr>          r :=0;
```

```
                    EXECUTE fetch;

<admode> CASE (ir 4:2)
                    WHEN ('00')  NEXT decode;
                    WHEN ('01')  EXECUTE fetch; ad :=ir;
                    WHEN ('10')  EXECUTE fetch; ad :=pc+ir;
                    WHEN ('11')  EXECUTE fetch; ad :=pc-ir;
                    END;
<decode>  CASE (ir 0:4)
                    WHEN ('00 00')  EXECUTE ada;
                    WHEN ('00 01')  EXECUTE lda;
          ...
                    ...
                    END;
                    IF (restart = 0) NEXT start; ELSE NEXT newinstr; END;

END;
```

Remarque:

Cette description LDS correspond à l'exemple de microprocesseur traité dans [ANCE 86] (cf ANNEXE C).

3.2.4 Modèle de compilation

3.2.4.1 Algorithme de traduction

La traduction consiste à passer d'une description comportementale à une description structurelle, puis d'une description structurelle à une description géométrique. Le compilateur de silicium est basé sur une architecture -cible, ce qui rend plus aisée la traduction.

Le langage d'entrée du compilateur à été conçu en intégrant les problèmes de la traduction. Cela simplifie d'autant l'algorithme de traduction. La simplicité de l'algorithme est cependant essentiellement basée d'une part sur la correspondance bijective entre la hiérarchie des appels de procédures et la hiérarchie des tranches de contrôle et d'autre part sur la correspondance bijective entre le nombre maximum d'opérations en parallèle et le nombre de sous parties opératives. Cette correspondance bijective permet bien sûr au concepteur de contrôler le fonctionnement du compilateur puisque le mécanisme de traduction bijectif est clairement défini. Toutefois, une phase d'optimisation peut être déclenchée afin d'aboutir à un meilleur compromis surface vitesse. Les différentes étapes de la traduction sont les suivantes :

3.2.4.2. Extraction des descriptions comportementales des tranches de contrôle et de la partie opérative [MART 85], [BOUR 86]

Cette étape de la compilation est actuellement la première étape. Elle sera précédée dans un proche avenir par une étape d'optimisation de la description originale. Le but de l'extraction est de séparer les instructions des différents niveaux (niveau opératif et niveaux de contrôle). La procédure la plus simple consiste à effectuer un traitement récursif. On extrait d'abord la description de la partie opérative, puis la description de la tranche de contrôle de plus bas niveau, puis celle de niveau immédiatement supérieur et on continue ainsi jusqu'à ce que l'on obtienne la description de toutes les tranches de contrôle.

Avant de procéder à cette extraction, les instructions comportant des actions conditionnelles sont mises sous forme canonique.

(COND

(cond 1 action 11 action 12... action $1p_1$)
(cond 2 action 21 ..... action $2p_2$)
(cond n action n1..... action np_n))

Toute instruction conditionnelle, c'est à dire comportant des actions conditionnelles (if ou case) en parallèle, même lorsqu' elles comportent des actions conditionnelles imbriquées, est transformée en une suite de couples (condition instructions_inconditionnelles) où les différentes conditions sont exclusives et où les instructions sont constituées d'une suite d'actions inconditionnelles à exécuter en parallèle. On obtient ainsi une mise à plat de l'algorithme permettant d'évaluer le parallélisme maximum pour un niveau.

3.2.4.2.1.Génération de la description comportementale de la partie opérative

Une instruction opérative est constituée d'un ensemble d'actions opératives. Une action opérative est caractérisée d'une part par le fait qu'elle se trouve dans la partie exécution d'une instruction conditionnelle ou dans une instruction inconditionnelle, d'autre part par le fait qu'elle fait intervenir au moins un registre de la partie opérative .

On associe un numéro à chaque instruction opérative. A une instruction conditionnelle à n alternatives correspond n instructions opératives . D'une part on génère un tableau des instructions opératives avec leurs numéros et d'autre part on remplace dans la description originale les instructions opératives par leurs numéros dans le but de générer la description des tranches de contrôle .

3.2.4.2.2.Génération de la description comportementale des différentes tranches de contrôle

Les instructions de contrôle de même niveau doivent être regroupées. La description comportementale peut être représentée par un arbre d'appels de CMODULEs. On suppose que les instructions opératives ont été remplacées par des C MODULEs opératifs.

Exemple:

```
CMODULE ada;
<e1>                          (r:=2;EXECUTE fetch;)
                              a:=a+b;
END;
```

est remplacé par:

```
CMODULE ada;
<e1> (r:=2; EXECUTE fetch;)
                              EXECUTE instr1;
END;
```

avec

```
<instr1> a:=a+b;
```

Une instruction de contrôle est un ensemble d'actions de contrôle pouvant se dérouler en parallèle. De ce fait elle ne peut être constituée que d'un appel à un CMODULE, d'un ordre de branchement et d'affectations de variables de contrôle. En effet, deux CMODULEs ne peuvent être exécutés simultanément. Le niveau d'une instruction de contrôle est donc donné par le niveau du CMODULE appelé s'il s'agit d'une instruction inconditionnelle et sinon par le niveau du CMODULE appelé de plus haut niveau parmi les CMODULEs appelés dans les différentes alternatives d'une instruction conditionnelle.

Exemple:

```
CMODULE ada;      ! cmodule de niveau i
      EXECUTE  a      ! instruction de niveau i+1
      if e1 EXECUTE b; else EXECUTE c; end; ! instruction de niveau i+1
END;
```

avec

```
CMODULE a; ! cmodule de niveau i+1
END;

CMODULE b; ! cmodule de niveau i+1
END;

CMODULE c; ! cmodule de niveau i+3
```

END;

Si une instruction de contrôle ne fait pas appel à un CMODULE, deux cas peuvent se présenter. Soit l'instruction comporte un ordre de branchement et le niveau de l'instruction est égal au niveau du CMODULE qui l'englobe, soit l'instruction ne comprend que des affectations de variables de contrôle, elle peut alors figurer dans n'importe quel niveau de contrôle puisque l'affectation d'une variable de contrôle peut se faire à n'importe quel niveau; on donne alors à cette instruction le niveau maximal qu'elle peut avoir, c'est à dire le niveau du CMODULE qui l'englobe afin de ne pas créer d'instructions intermédiaires inutiles.

Exemple:

```
CMODULE ada;  ! cmodule de niveau i
<test>IF (RESTART=0) NEXT start;end; ! instruction
      de niveau i
<ctrl>(SET var1); ! instruction de niveau i
<start>pc:=0;
END;
```

L'algorithme d'extraction est direct. On remplace dans la description chaque instruction de contrôle de niveau directement inférieur au dernier niveau extrait (c'est à dire une instruction appelant un CMODULE de niveau du dernier niveau extrait) par l'appel d'un CMODULE dont le nom est le point d'entrée dans la table des instructions de contrôle du niveau en cours d'extraction. Les différentes instructions d'un CMODULE ne sont pas forcément de même niveau. On crée alors des CMODULES intermédiaires afin que toutes les instructions aient même niveau (figure 1.16).

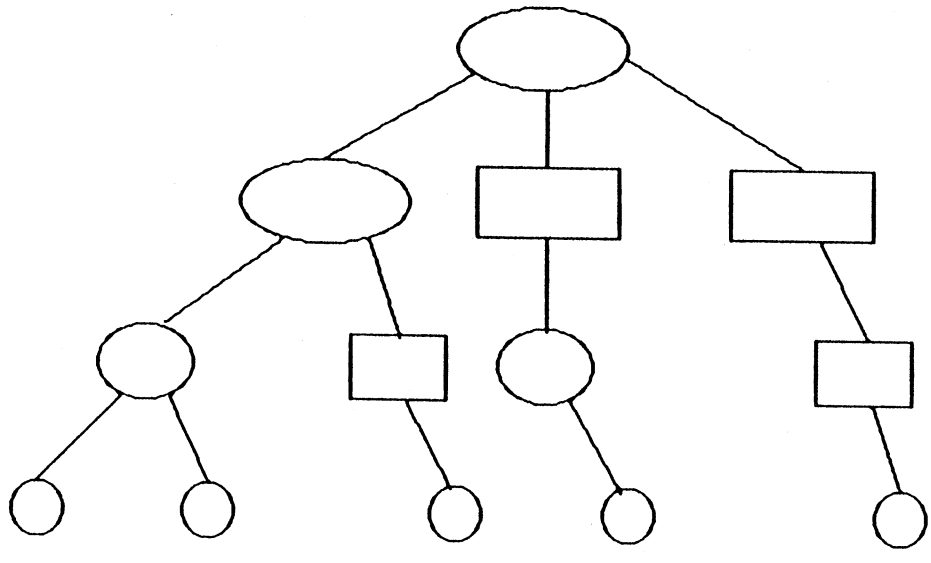


fig. 1.16 Création de CMODULES intermédiaires

Exemple:

```
CMODULE ada;
<e1> (r:=2; EXECUTE fetch;)
                              EXECUTE instr1.niv1;
END;

CMODULE instr1.niv1;
<instr1.niv1>  EXECUTE instr1;
END;
```

On introduit le CMODULE inst.niv1 afin d'amener au même niveau les deux instructions de contrôle.

3.2.4.3. Synthèse des tranches de contrôle [MHAY 86]

L'outil de synthèse des tranches de contrôles permet dans la version actuelle de générer les spécifications de tranche de contrôle de type monoPLA. Un PLA est constitué d'une matrice ET et d'une matrice OU. Chaque tranche de contrôle est assimilable à un automate dont on recensera les vecteurs d'entrée et de sortie. Etant donné qu'un étage de contrôle est construit à l'aide d'un PLA unique, il sera synthétisé de la manière suivante.

Les colonnes de la matrice ET du PLA sont constituées des entrées de l'automate. Ce sont :

- les comptes rendus de la partie opérative,
- les champs du registre instruction,
- les requêtes de contrôle externes et internes,
- l'état de l'automate lui-même constitué de 2 champs :
    * le séquencement "externe" : l'appel d'un CMODULE par le niveau supérieur,
    * le séquencement "interne" nécessaire au déroulement du CMODULE appelé.

Les lignes de la matrice ET correspondent aux vecteurs d'entrée de la machine d'états finis et les monômes correspondent aux différentes transitions de l'automate de contrôle. Les colonnes de la matrice OU sont constituées des sorties de l'automate. Ce sont :

- les ordres d'affectation des variables de contrôle internes et externes,
- les appels aux CMODULEs de niveau directement inférieur ou les commandes de la partie opérative s'il s'agit du dernier étage de contrôle,

- le nouvel état du CMODULE en cours d'exécution,
- et des signaux de synchronisation qui déclenchent le passage à un nouvel état de l'automate de contrôle supérieur si l'exécution du CMODULE est terminée ou qui passent le contrôle à l'étage inférieur si un nouveau CMODULE doit être exécuté. On suppose que la partie contrôle ne fonctionne pas en mode pipeline.

La synthèse est effectuée à partir de la strcuture de donnée LDS obtenue après compilation du texte source LDS. La structure est préalablement traitée pour extraire les différents niveaux d'interprétation [MHAY 86].

3.2.4.4.Evaluation, optimisation et compromis surface performance : [BEKK 86]

La tâche de l'évaluateur consiste à déterminer si la description du concepteur conduira à un bon compromis surface-vitesse. L'évaluation doit être effectuée avant que les masques ne soient générés, la génération des masques étant une des étapes les plus coûteuses en temps CPU. L'évaluation peut porter sur deux types de description:

- la description comportementale des tranches de contrôle et de la partie opérative après la phase d'extraction; il s'agit alors d'une évaluation grossière qui permet de déterminer le nombre d'étages de contrôle et une estimation du nombre de sous parties opératives,
- la description structurelle des PLAs de contrôle et de la partie opérative après la phase de synthèse. Il s'agit d'une évaluation plus fine qui permet de déterminer le nombre et la taille des différents PLAs de contrôle ainsi que le nombre de sous parties opératives et la taille de la partie opérative.

Suivant le résultat de l'évaluation, deux types de modifications de la description comportementale peuvent être effectués :

- des optimisations,
- des transformations visant à la recherche d'un meilleur compromis surface performance.

On entend ici par optimisation un ensemble d'actions dont le but est d'améliorer à la fois la surface occupée et les performances du circuit, ou du moins qui améliorent l'un sans dégrader l'autre. Cette phase de la compilation n'est déclenchée que sous le contrôle de l'évaluateur car l'optimisation est en général une tâche très coûteuse en temps CPU. La recherche d'un meilleur compromis surface-performance consiste par contre à échanger une perte de performance contre un gain de surface ou vice versa.

Optimisation

Nous avons déjà mentionné 2 types d'optimisation :

- la réduction du nombre des états d'un automate
- le choix d'un codage des états qui minimise le nombre des monômes d'un PLA.

Nous mentionnerons ici 2 autres types d'optimisation :

- l'optimisation de code qui est une technique bien connue des concepteurs de compilateurs de langages évolués [AhUl 77]
- le compactage de microcode dont le but est de combiner les microopérations en le plus petit nombre de microinstructions, afin de réduire la taille et le temps d'exécution de l'algorithme d'interprétation des instructions

Optimisation de code [AhUl 77], [WaTh 83]

Les transformations que l'on peut appliquer à la description originelle sont les suivantes :

- propagation des constantes
- élimination des expressions redondantes
- déplacement des actions invariantes à l'extérieur des boucles
- expansion de procédures appelées une seule fois.

Compactage de microcode [DAVI 81], [AGER 76], [FISH 81], [ISOD 83], [TOKO 81].

Les techniques de compactage de microcode peuvent être classées dans la rubrique optimisation ou dans la rubrique compromis surface-vitesse selon qu'elles sont appliquées après la construction de la partie opérative ou dès le début de la compilation.

Dans le premier cas, la mise en parallèle des microopérations dépend à la fois des relations de dépendance des données et des possibilités de parallélisme de la partie opérative alors que dans le second cas seules les relations de dépendance entre les données sont à prendre en compte. Le compactage de microcode peut être effectué à la fois avant et après la synthèse de la partie opérative. Dans le premier cas, il s'agit de paralléliser au maximum la description, dans le second il s'agit d'exploiter au maximum les ressources de la partie opérative.

La description comportementale est divisée en segments ne comportant qu'un seul point d'entrée et un seul point de sortie qui sont situés aux extrêmités des segments.

On distingue deux types de compactage :

- le compactage local qui optimise la description à l'intérieur des blocs de base définis précédemment.
- le compactage global qui optimise la description dans son ensemble. Les segments successeurs et prédécesseurs de chaque segment doivent être déterminés afin de pouvoir effectuer une optimisation globale. Un des cas de figures qui peuvent se présenter est le suivant [TOKO 81].

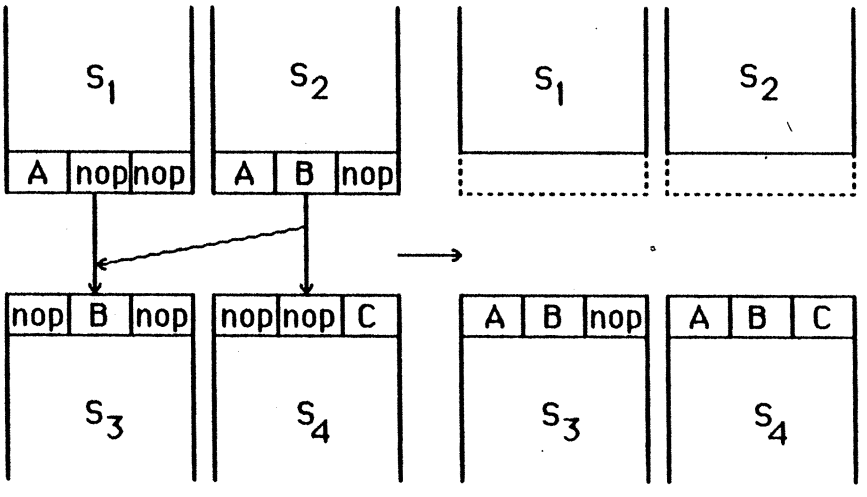


fig. 1.17 Compactage global

Compactage local

Si les microinstructions d'un même bloc de base dans la description de départ contiennent déjà des microopérations en parallèle , il est nécessaire de décomposer ces microinstructions en microopérations afin d'obtenir le minimum absolu du nombre des microinstructions.
Davidson et al comparent quatre méthodes de compactage local dans [DAVI 81].

- la méthode du premier venu, premier servi (FIFO): les microopérations sont traitées dans l'ordre de départ. Chaque microopération est placée dans la première des microinstructions (en cours de formation) possible.
- la méthode du chemin critique : c'est la méthode inverse de la précédente. Il s'agit de déterminer le nombre maximum de microinstructions que l'on peut exécuter avant une microopération sans provoquer d'augmentation du nombre de microinstructions.
- la méthode "Branch and Bound" : on construit un arbre dont les nœuds sont les microinstructions. On part des microopérations qui n'ont pas de prédécesseur dans le graphe de précédence. Des branches sont créées chaque fois qu'il y a plus d'une microinstruction qui peut être constituée à ce niveau. Toutes les branches de l'arbre qui peuvent conduire à une

solution optimale sont parcourues. Seul cet algorithme dont le temps d'exécution est d'ailleurs une fonction exponentielle du nombre de microopérations garantit l'obtention de la solution optimale.

- la méthode "List scheduling" : seule la meilleure branche de l'arbre de la méthode Branch and Bound est formée à chaque niveau. La meilleure peut être par exemple celle dont les microopérations qui constituent la microinstruction ont le plus de descendants (directs ou indirects) dans le graphe de dépendance.

Compactage global

En particulier si l'on compacte non pas la description d'entrée de **SYCO** mais les descriptions des tranches de contrôle après extraction, une forme de compactage globale consiste à éliminer les instructions qui ne comportent qu'un ordre de branchement NEXT. Les instructions de contrôle peuvent en effet être constituées d'un appel de CMODULE : EXECUTE, d'affectations de variables de contrôle : SET et RESET et d'un ordre de branchement : NEXT ou QUIT.

Une instruction ne comportant qu'un NEXT peut être remplacée par la première instruction de la séquence commençant par l'étiquette de branchement du NEXT. Il est d'autant plus important qu'il n'y ait pas d'instructions NOP (no opération) que le nombre d'étages de contrôle est plus élevé. Il est donc souhaitable dans la mesure du possible que chaque instruction de contrôle comporte une partie EXECUTE.

Compromis surface-vitesse [BEKK 86]

Parmi les transformations que l'on peut appliquer à un algorithme, un certain nombre peuvent conduire à un compromis : augmentation des performances mais aussi augmentation de la surface occupée par le circuit ou vice-versa.

- l'expansion de procédures ou la formation de procédures.
- transformation d'une instruction comportant des IF en parallèle en une suite de IF, factorisation d'un CASE .

Expansion/formation de procédures

L'expansion de procédures consiste à remplacer les appels de procédures par des copies de procédures. Cette transformation conduit à une augmentation de la surface nécessaire à la mémorisation des commandes de la partie opérative mais aussi à une augmentation des performances car il n'y a pas de perte de temps pour l'appel des procédures (empilement et dépilement de l'adresse de retour). En ce qui concerne **SYCO**, le temps gagné se traduit en fait par une diminution du nombre des étages de contrôle, ce qui est équivalent à la diminution du temps de traversée de la structure de contrôle quel que soit d'ailleurs le mode de fonctionnement (procédural ou

pipeline).

La formation de procédures consiste à remplacer des portions de code identiques par des appels à des procédures. Les effets de cette transformation sont inverses de ceux d'une expansion.

Bekkara [BEKK 86] propose un compromis surface-performance plus fin pour la décomposition de la partie contrôle. Pour diminuer d'un le nombre d'étages, seuls les appels des Cmodules de l'étage que l'on veut faire disparaître seront remplacés par des copies des Cmodules.

Diminution du parallélisme par décomposition des instructions opératives

En ce qui concerne la partie opérative, les étapes de compactage ont pour effet de paralléliser au maximum la description comportementale, donc d'exiger le maximum de ressources opératives. Si l'on ne veut pas dépasser une limite donnée pour la partie opérative, il peut être nécessaire de décomposer certaines instructions opératives.

Il existe aussi un autre type de compromis dont la recherche est intéressante. C'est le compromis largeur/hauteur. La déformabilité des blocs est une des conditions nécessaires à la minimisation de la surface des zones de connexion entre les blocs.

4 CONCLUSION

Nous avons essayé de présenter le compilateur **SYCO** comparativement à d'autres compilateurs. Il peut être classé dans les compilateurs spécialisés à architecture-cible. Son modèle fonctionnel et architectural lui permet une grande flexibilité.

Nous allons résumer les particularités, avantages et limitations du compilateur de silicium **SYCO**:

Particularités :

Le compilateur de silicium SYCO intègre autour d'une structure de donnée, un nombre important de processeurs spécialisés. Toutes les étapes principales de la compilation sont strictement définies:

* Modèle fonctionnel de départ avec utilisation d'un langage d'entrée LDS (utilisation d'IRENE [MARI 86] dans une première version),
* Modèle architectural général avec une correspondance immédiate avec le modèle fonctionnel. Autour du modèle architectural, il existe un modèle temporel, actuellement de type procédural, dont la particularité est qu'il est défini comme un automate des temps réparti de manière topologique au niveau de chacune des parties contrôles composant le circuit.
* Modèle topologique global qui permet d'intégrer diverses topologies de parties opératives et de parties contrôles.

Avantages :

Les avantages du modèle de compilation SYCO proviennent des spécifications strictement définies des différents modèles (architectural, fonctionnel et topologique) autour desquels le compilateur est construit:

* Le modèle fonctionnel permet une description simple d'un circuit (ou d'une partie de circuit). Le concepteur (ou programmeur) peut ne pas tenir compte de l'implémentation finale de ses algorithmes.
* Le modèle architectural et le modèle topologique permettent une évaluation rapide de la surface occupée par telle ou telle description, et permet ainsi une comparaison des différents résultats de compilation.
* Le modèle de compilation de SYCO permet de compiler des "gros circuits" grâce à la hiérarchie définie dans les modèles fonctionnel et architectural.

* Le compilateur de SYCO est orienté compilation de circuit de type microprocesseur.
* Le compromis surface/temps s'exprime selon deux ordres:

  * Plus le nombre de niveaux est important, plus le temps d'exécution d'une instruction de niveau le plus élevé augmente (chute des performances).
  * Plus le nombre de niveau est faible, plus la surface respective des différents niveaux de contrôle augmente, ayant pour effet d'augmenter le temps de cycle d'horloge à structure de partie contrôle fixée (chute des performances due à l'augmentation de surface d'un niveau de contrôle)

Limitations :

* Le compilateur de silicium SYCO ne permet de compiler qu'un type de circuit : les circuits de type microprocesseur.
* Les parties contrôles générées, en ce qui concerne la première version, sont strictement du type monoPLA. Cette limitation sera levée dans les versions futures de SYCO.

Le compromis surface /temps est tel que si l'on veut un circuit complexe rapide et de surface faible, il est nécessaire de disposer d'un nombre d'étages réduit. Par exemple, dans les processeurs existants sur le marché, la limitation en nombre de niveau de contrôle est de deux à trois niveaux (le Z80 de chez ZILOG ou le 8085). Mais ce compromis peut s'exprimer uniquement en terme de surface, et il est possible de disposer d'un petit circuit avec un nombre important d'étages.

CHAPITRE II

ARCHITECTURE ET COMPILATION

DE

PARTIES CONTROLES

1 ARCHITECTURE DE CIRCUIT DE TYPE MICROPROCESSEUR

Un microprocesseur, comme toute machine informatique, est défini par son langage, c'est à dire par le jeu d'instructions qu'il exécute. Il constitue l'une parmi de nombreuses réalisations matérielles possibles de l'algorithme d'interprétation de ces instructions. Cette réalisation matérielle est déterminée par le concepteur lors du processus de conception de l'architecture interne.

Au cours de la définition de l'architecture interne, le concepteur est amené à effectuer un certain nombre de choix pour respecter les contraintes qui lui ont été imposées. Dans le cas d'architectures compilables, les contraintes imposées sont souvent draconiennes, et le choix des réalisations matérielles limité.

Différentes règles de conception proviennent d'une étude de circuits du marché ("reverse engineering"). Ces règles de conception spécifiées par [ANCE 86] sont les suivantes:

- La recherche d'une optimisation globale de la surface [AnRe 82], en opposition à une optimisation locale des blocs constitutifs d'un circuit.

- La construction d'un bloc est défini à partir du plan de masse. Durant la mise au point du plan de masse, un bloc existe comme une entité possédant soit une notion de dureté (bloc fontionnel topologiquement indéformable), soit une notion de malléabilité (bloc fonctionnel topologiquement déformable) [SUPR 85]. Fixer un pan de masse, c'est durcir les blocs constitutifs du circuit.

- Optimisation de la surface des interconnexions:
  * Utilisation de connexions directes entre cellules, impliquant le dessin des cellules en fonction des contraintes de leur environnement,
  * Utilisation de la transparence des cellules (passage d'un fil physique à travers une cellule sans modification de sa fonctionnalité).

- Une stratégie d'utilisation des niveaux de connexion pour le croisement de deux flots d'informations (flot donnée et flot contrôle par exemple) (figure 2.1).

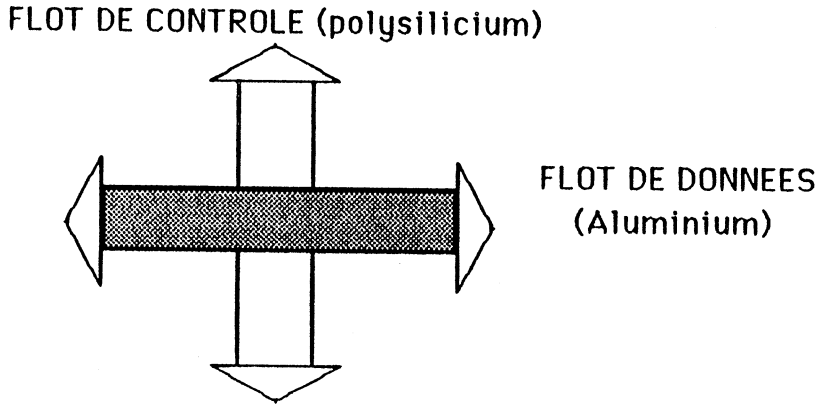


fig. 2.1 Croisement de deux flots d'information avec utilisation de deux niveaux de masque

Au cours de ce chapitre, divers aspects concernant les contraintes sur les choix pour telle ou telle architecture de partie contrôle vont être exposés. Ces différentes architectures proviennent de l'analyse de circuit, analyse qui avait été faite dans le cadre de la définition de la méthodologie de conception de circuit CAPRI [ANCE 83].

1.1 Architectures générales

1.1.1 Notions de machines d'états finis

Tout réseau séquentiel peut être décrit mathématiquement comme un automate d'états finis [BARA 79]. Il est représenté par une collection de six êtres mathématiques notés

$S = (A, Z, W, \delta, \lambda, a1)$

où

* $A = \{a_1, ..., a_m, ..., a_n\}$ est l'ensemble des états internes ;

* $Z = \{z_1, ..., z_f, ..., z_F\}$, l'ensemble des signaux d'entrées ;

* $W = \{w_1, ..., w_g, ..., w_G\}$, l'ensemble des signaux de sortie ;

* $\delta : A \times Z \rightarrow A$, la fonction des transistions. Cette fonction réalise l'application $D_\delta \subseteq A \times Z$ dans A. Autrement dit, la fonction δ fait correspondre à un couple état interne - signal d'entrée (a_m, z_f) un état $a_\delta = \delta(a_m, z_f)$ de l'automate, $a_\delta \in A$;

* $\lambda : A \times Z \rightarrow W$, la fonction des sorties. Cette fonction réalise l'application $D_\delta \subset A \times Z$ sur W. A un couple état interne - signal d'entrée, la fonction λ fait correspondre des signaux de sortie de l'automate $w_g = \lambda(a_m, z_f)$;

* $a_1 \in A$, l'état initial de l'automate.

Un automate d'états finis comprend un ensemble d'entrées et un ensemble de sorties. Il fonctionne dans un temps discret qui prend des valeurs entières non négatives A = 0, 1, 2, ...
A chaque instant t du temps discret, l'automate est dans un état a(t) appartenant à son ensemble des états (figure 2.2).

A l'instant t = 0, il se trouve à l'état initial $a(0) = a_1$

à t, $w(t) = \lambda(a(t), z(t))$,
$a(t+1) = \delta(a(t), z(t))$,
$a(t) \in A$, $w(t) \in W$

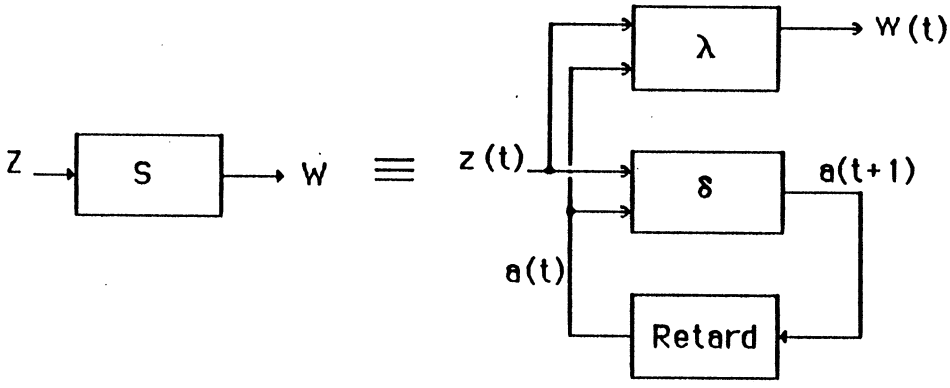


fig. 2.2 Représentation symbolique d'un automate d'états finis (automate de Mealy)

Un réseau combinatoire est un automate dont la sortie est indépendante du passé, n'étant définie à tout instant que par le signal d'entrée à ce même instant. Au niveau abstrait, on peut le considérer comme un automate à un seul état.

Il peut être défini par le triplet $S \triangleq (Z,W,\lambda)$ ou $\lambda = Z \rightarrow W$ fonction des sorties (figure 2.3).

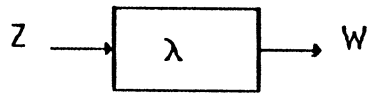


fig. 2.3 Représentation d'un réseau combinatoire

1.1.1.1. Automates de Moore et de Mealy.

Les deux classes d'automates le plus fréquemment rencontrés dans la pratique sont les automates (ou machines) de Mealy et de Moore.

Le fonctionnement d'un automate de Mealy se définit par les équations:

$a(t+1) = \delta(a(t), Z(t))$;
$w(t) = \lambda(a(t), Z(t))$;
$t = 0, 1, 2, ...,$

La figure 2.1 représente un automate de Mealy.

Le fonctionnement d'un automate de MOORE, par les équations :

$a(t+1) = \delta(a(t), Z(t))$;

$w(t) = \lambda(a(t))$;

$t = 0, 1, 2, ...,$

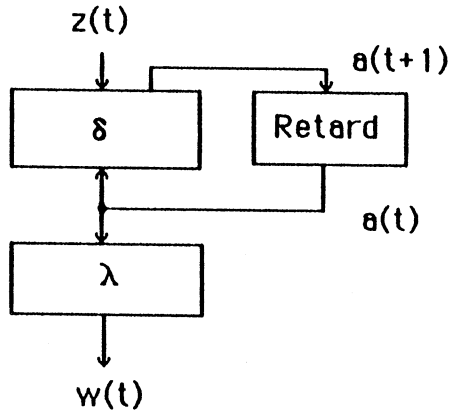


fig. 2.4 Représentation d'un automate de Moore

Les automates de Moore et Mealy ont une réponse équivalente à une séquence d'entrée, et ne diffèrent que par leurs fonctions λ et δ. Le passage de l'un à l'autre est ainsi totalement réalisable.

On dit d'un automate qu'il est fini si les ensembles A, Z, W le sont.

Le fonctionnement d'un automate peut être décrit de plusieurs façons. Le plus souvent, on a recours à un graphe (PETRI, Graphe d'état) (figure 2.5), ou à une table (KARNAUGH ...). Les représentations par graphe de PETRI ou par organigramme sont les plus courantes, par exemple dans un graphe de PETRI, dans le cas d'un automate de Moore, les sorties sont liées à l'état, et dans le cas d'un automate de Mealy, les sorties sont liées à la transition. La table de Karnaugh est peu utilisée, car sa représentation devient très complexe pour un grand nombre d'entrées et d'états.

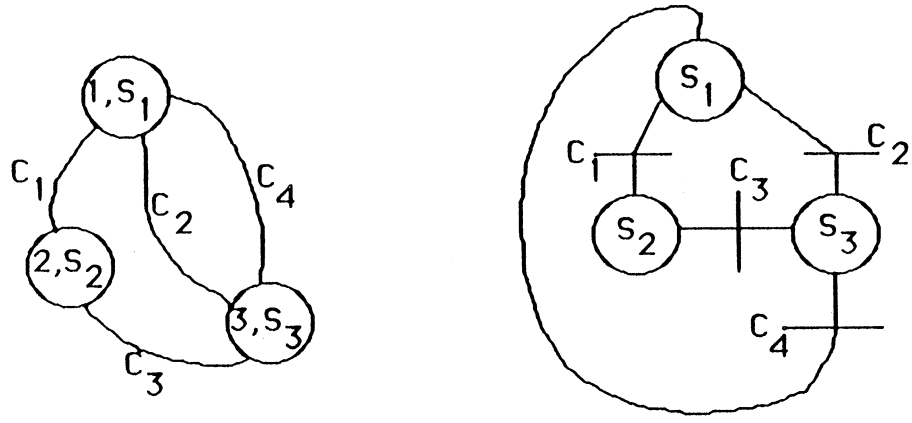


fig. 2.5 Représentation d'un automate sous forme de PETRI (à droite) et de Graphe d'état (à gauche)

1.1.1.2 Composition et décomposition des automates

Les automates peuvent être rassemblés en réseaux, les trois principaux types de connexions étant : en parallèle, en série, et avec boucle de retour.

1.1.1.2.1 Connexion en parallèle (figure 2.6)

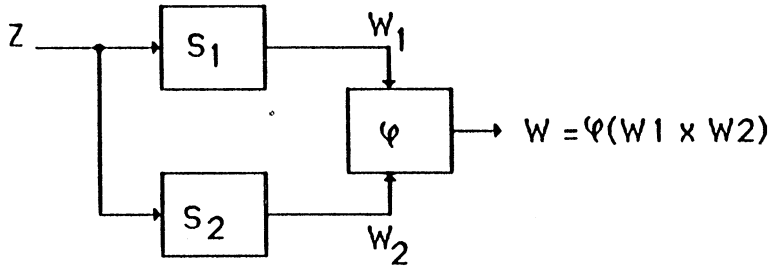


fig. 2.6 Connexion d'automates en parallèle

Ici, $S_1 = (A_1, Z, W_1, \delta_1, \lambda_1)$ et $S_2 = (A_2, Z, W_2, \delta_2, \lambda_2)$. L'entrée est commune aux deux automates, les sorties de S_1 et S_2 sont reliées à un traducteur de fonction φ (Automate sans mémoire) qui réalise l'application

$\varphi : W_1 \times W_2 \rightarrow W$

Ce traducteur fournit la réponse (sortie) du système global S en fonction des sorties de S_1 et de S_2.

L'automate résultant de la connexion en parallèle de S_1 et S_2 est l'automate $S = (A, Z, W, \delta, \lambda)$ dans lequel :

*L'ensemble des états $A = A_1 \times A_2$ est

$A = \{a_m = (a_{m1}, a_{m2}) / a_{m1} \in A_1, a_{m2} \in A_2\}$

*L'ensemble des signaux d'entrées est l'ensemble Z des automates S_1 et S_2

*L'ensemble des signaux de sorties est $W = \varphi(W_1 \times W_2)$, φ étant l'application à réaliser.

*La fonction des transitions $\delta : A \times Z \rightarrow A$ se définit comme :

$\delta(a_m, z_f) = (\delta_1(a_{m1}, z_f), \delta_2(a_{m2}, z_f))$ $z_f \in Z$

ou $\delta(A \times Z) = (\delta_1(A_1 \times Z), \delta_2(A_2 \times Z))$

* La fonction des sorties $\lambda : A \times Z \rightarrow W$ est

$\lambda(a_m, z_f) = \varphi(\lambda_1(a_{m1}, z_f), \lambda_2(a_{m2}, z_f))$,

ou $\lambda(A \times Z) = \varphi(\lambda_1(A_1 \times Z), \lambda_2(A_2 \times Z))$

1.1.1.2.2 Connexion en série (figure 2.7)

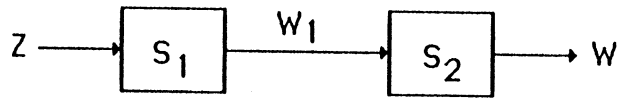


fig. 2.7 Connexion d'automates en série

Ici, $S_1 = (A_1, Z, W_1, \delta_1, \lambda_1)$ et $S_2 = (A_2, W_1, W, \delta_2, \lambda_2)$

L'automate résultant de la connexion en série de S_1 et S_2 est

$S = (A, Z, W, \delta, \lambda)$, tel que :

$A = A_1 \times A_2$, ou $A = \{a_m = (a_{m1}, a_{m2}) / a_{m1} \in A_1, a_{m2} \in A_2\}$

$Z = Z$

$W = W$

δ : A x Z→A se définit comme :

$$\delta(A \times Z) = (\delta_1(A_1 \times Z), \delta_2(A_2 \times \lambda_1 (A_1 \times Z))$$

λ : A x Z →W se définit comme :

$$\lambda(A \times Z) = \lambda_2(A_2 \times \lambda_1(A_1 \times Z))$$

1.1.1.2.3 Connexion à boucle de retour (figure 2.8)

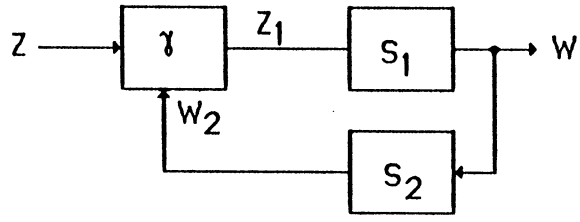


fig. 2.8 Connexion à boucle de retour

Ici $\delta_1 = (A_1, Z_1, W, \delta_1, \lambda_1)$ et $\delta_2 = (A_2, W_1, W_2, \delta_2, \lambda_2)$.

Le traducteur de fonction (automate sans mémoire) réalise l'application

$\gamma : Z \times W_2 \rightarrow Z_1$

Dans le cas d'une connexion à boucle de retour, l'un des automates au moins, S_1 ou S_2, est toujours un automate de Moore.

En effet, dans le cas où les deux automates seraient de type Mealy, on aurait alors

$$W_2 = \lambda_2(A_2 \times W),\ W = \lambda_1(A_1 \times Z_1)$$

le signal $Z_1 = \gamma(Z \times W_2) = \gamma(Z \times \lambda_2(A_2 \times W) = \gamma(Z \times \lambda_2(A_2 \times \lambda_1(A_1 \times Z_1)))$,

donc $$\underline{Z_1} = \gamma(\ldots \underline{Z_1})$$

Comme Z_1 à un instant donné dépend de Z_1 au même instant, la stabilité du réseau ainsi constitué est compromise,

si on considère qu'un des automates est de type Moore, on a alors:

** cas ou S_1 est un automate de Moore ; alors

$w = \lambda_1(A_1)$ et

$Z_1 = \gamma(Z \times W_2) = \gamma(Z \times \lambda_2(A_2 \times W))$

$= \gamma\ (Z \times \lambda_2\ (\ A_2 \times \lambda_1\ (\ A_1\ )))$

le réseau est alors stable.

Si S_2 est un automate de Moore ($W_2 = \lambda_2(A_2)$), l'automate résultant de la connexion à boucle de retour est

$\delta = (A, Z, W, \delta, \lambda)$ tel que

$A = A_1 \times A_2$, ou $A = \{a_m = (a_{m1}, a_{m2}) / a_{m1} \in A_1, a_{m2} \in A_2\}$

$Z = Z$

$W = W$

$\delta : A \times Z \rightarrow A$ se définit comme

$$\delta(A \times Z) = (\delta_1(A_1 \times \gamma(Z \times \lambda_2(A_2))),$$
$$\delta_2(A_2 \times \lambda_1(A_1 \times \gamma(Z \times \lambda_2(A_2)))$$

$\lambda : A \times Z \rightarrow W$ se définit comme

$$\lambda(A \times Z) = \lambda_1(A_1 \times \gamma(Z \times \lambda_2(A_2)))$$

1.1.2 Stratégies de conception et architecture de circuit

Si d'un point de vue conceptuel et théorique, l'on sait définir un circuit de type microprocesseur, la réalisation matérielle est quant à elle dépendante de la méthodologie de conception choisie [MeCo 81].

Machine de Von Neumann: Une machine de Von Neumann est un automate programmable, dont le programme et les données sont enregistrées en mémoire. Un circuit de type microprocesseur représente une partie de la machine de Von Neumann: la partie contrôle représentant le séquenceur de commandes de la machine de Von Neumann, et la partie opérative correspondant au chemin de données de la machine de Von Neumann (figure 2.9). Le programme et les données sont dans une mémoire extérieure au circuit de type microprocesseur.

Un circuit de type microprocesseur est ainsi assimilable à un automate d'états finis (automate programmable). La décomposition de cet automate fait apparaître deux automates à boucle de retour (§II 1.1.1.2.3). Ces deux automates sont respectivement nommés partie opérative et partie contrôle (figure 2.9):

- La partie opérative est assimilable à un réseau combinatoire pur (§II 1.1.1): ou l'automate de partie opérative est représenté par

$S=(Z,W,\lambda)$,

avec

Z={Données d'entrée,Commandes provenant de la Partie Contrôle},

W={Données de sortie, Comptes rendus de la partie opérative},

$\lambda = Z \mapsto W$.

- La partie contrôle permet le séquencement de l'automate de la partie opérative et correspond à un automate d'états finis $S=(A,Z,W,\lambda,\delta)$ tel que l'automate S regroupe l'automate S_1 et le traducteur de fonction γ de la figure 2.8:

Z={CTRL, Comptes rendus},

W={CTRL sortants,Commandes vers la partie opérative},

λet δ sont les fonctions d'un automates sous forme de Moore ou de Mealy.

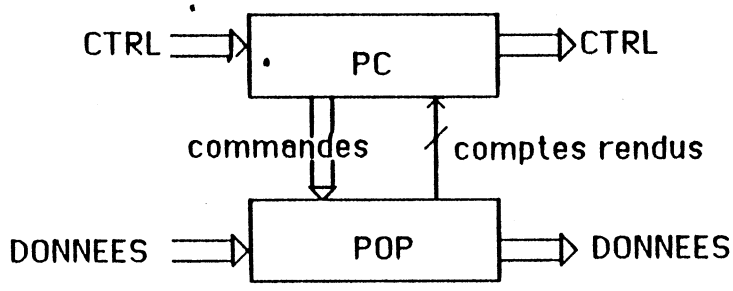


fig. 2.9 Circuit de type microprocesseur

L'orientation pour définir une stratégie de conception de circuit de type microprocesseur s'est au début basée sur les connaissances accumulées dans la conception des cartes processeur des gros ordinateurs. Cette conception utilisait comme matériaux de base des circuits MSI ou LSI (Medium and Large Scale Integration). Cette orientation s'est traduite par des surfaces importantes perdues par routage entre les différents éléments du circuit implémenté. Cette surface ajoutée est, dans le cas d'un circuit intégré, préjudiciable au rendement de production: elle implique un coût plus élevée, et une baisse du nombre de circuit "bon" à la sortie d'une chaine de production. Cette divergence a entraîné des choix architecturaux spécifiques et plusieurs stratégies de conception se sont développées [ANCE 86].

D'autre part, le fait de l'intégration n'est pas le seul à influer sur les choix architecturaux, la technologie employée provoque, de même, des orientations différentes [ANCE 82] [ANCE 79].

Toute résolution du problème architectural suppose la connaissance préalable des "ressources" utilisables pour résoudre le problème donné. Cette résolution peut être abordée de deux manières [ANCE 74].

Méthode ascendante

Cette méthode consiste à définir successivement des ressources par regroupement de ressources élémentaires déjà définies, en partant de ressources primitives données, jusqu'à ce que cet assemblage résolve le problème donné. L'implémentation d'un circuit intégré se déroule toujours de manière ascendante:

- assemblage de cellules de base,
- puis remontée de l'arbre de construction du circuit jusqu'au sommet.

Ce principe est utilisé dans la conception des gros systèmes: assemblage de blocs de registres, de processeurs,..., et ce en éléments discrets. L'inconvénient de cette méthode est qu'on obtient généralement un sur-ensemble de ce qui était visé.

Méthode descendante

Cette méthode consiste à décomposer progressivement le problème posé, de manière à ce que l'assemblage des ressources ultimes réalise l'ensemble des ressources primitives données. Les compilateurs de silicium, comme l'analyse de circuit se déroule de manière descendante. Cette méthode correspond à une décomposition algorithmique d'un problème.

1.1.3 Architectures cibles compilables

Définir une architecture cible pour la compilation, c'est définir une méthodologie descendante de la conception de tout ou partie d'un circuit. Les différents critères sur lesquels l'une ou l'autre des architectures cibles compilables sont choisies sont les suivants : [MARQ 79]

1) Facilité d'implantation :

Dans les circuits complexes du type microprocesseurs, des solutions matérielles pouvant être facilement implantées de manière automatique doivent être préférées, car le temps d'implantation représente une grande part du temps de conception. Le circuit doit être aussi le plus modulaire possible, pour qu'en cas de modifications, une partie du circuit seulement soit modifiée (perturbation minimum sur l'ensemble du circuit).

2) Critère de surface :

La surface finale du circuit est prédominante dans l'évaluation des coûts de fabrication (figure 2.10). La surface n'est pas liée, dans le cas des circuits intégrés, seulement au nombre de transistors à implanter, mais aussi au type d'architecture choisie pour l'implantation des différents modules.

Ce critère est en opposition sur les choix architecturaux, avec le critère précédent. En effet, les structures très régulières sont en général fort consommatrices de surface. La solution optimum est donc un compromis entre ces deux critères.

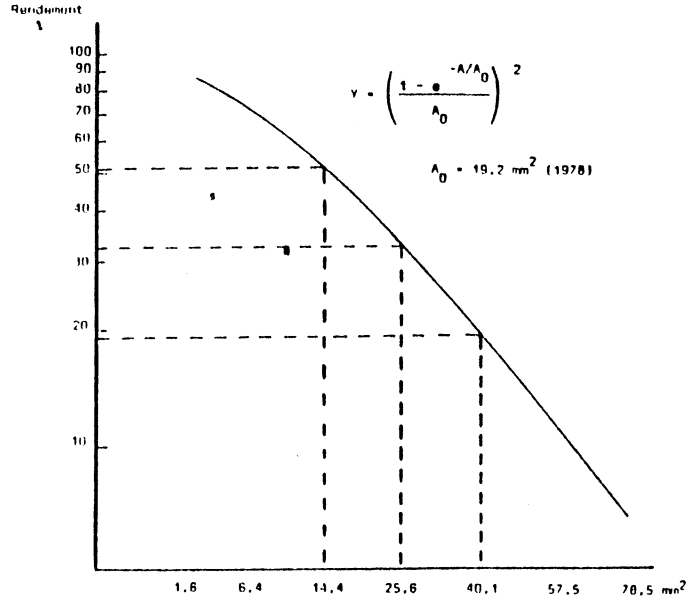


fig. 2.10 Rendement de production

3) Critère de performance :

Dans certains cas, selon la finalité des circuits de type microprocesseur, le critère de performance influence le choix dans l'architecture cible. Par exemple un circuit destiné à un contrôle d'automatisme temps réel sera conçu en tenant compte prioritairement de ce critère.

1.1.3.1 Style de conception basé sur l'assemblage ou connectique de cellules prédéfinies.

Les styles de conception basés sur l'assemblage avec routage de cellules de base sont classiquement le circuit prédiffusé ou "Gate Array" et le circuit précaractérisé ou "Standard Cell".

Dans le cas du circuit prédiffusé, on dispose d'un certain nombre de cellules de base (ou portes logiques) préimplantées. L'automate à implanter est donc décomposé sous forme logique implantable à l'aide des portes disponibles, puis les portes sont routées entre elles pour réaliser les fonctions demandées. Ce type de conception est alors proche de celui utilisé pour concevoir les systèmes à partir de portes de base type TTL. La séparation entre partie opérative et partie contrôle n'est pas nécessairement apparente.

Le précaractérisé est une approche optimisée dans ce style de conception [JOHA 79]. Il repose sur l'utilisation d'un certain nombre de cellules de base (ex : UAL, Unité Arithmetique et Logique, registres, ROM...), à partir desquelles l'on réalise le circuit à implanter. Ensuite, ces cellules subissent un placement, si possible optimum, sur la surface allouée pour construire le circuit, puis un routage global est effectué (le routage correspond à relier les différentes entrées/sorties des cellules entre elles, selon un plan de connection pré-établi).

L'avantage premier de ce style de conception en est son automatisation aisée grâce à l'utilisation de nombreux outils tels : outils de placement automatique, outils de routage... De plus les cellules de base disponibles peuvent être très complexes. Les critères de facilité d'implantation et de performance sont optimisés.

Par contre, ce style de conception nécessite une bibliothèque importante de cellules complexes, bibliothèque difficile à maintenir, et très liée à la technologie.

1.1.3.2 Architecture libre ("full custom")

L'architecture libre sous entend que l'on n'utilise pas une bibliothèque de cellules, mais que l'on implante les automates soit à l'aide de logique anarchique, soit à l'aide de structures régulières, en faisant prédominer soit l'aspect surface, soit l'aspect performance, soit l'aspect coût (en temps) de conception.

L'architecture libre demande une approche de conception descendante. Elle nécessite de déterminer de manière précise l'automate de contrôle et l'automate de partie opérative. On définit les choix structurels de la partie contrôle et de la partie opérative à partir des critères prioritaires définis ci-dessus (critères de performance, de surface et de facilité d'implantation).

Par convention dans une architecture de type microprocesseur, on implante la partie contrôle au-dessus de la partie opérative [MeCo 81].

D'autre formes d'implantation de la partie contrôle relativement à la partie opérative peuvent être imaginées [MARC 86b]: trois solutions sont alors envisageables:

1) - La partie contrôle est répartie sur le pourtour de la partie opérative,
   - La partie opérative occupe le centre du circuit => le problème provient des nombreuses connexions entre la partie opérative et ses plots (adresse et données). En outre cela suppose une partie opérative à faible complexité (peu de registres...) (figure 2.11).

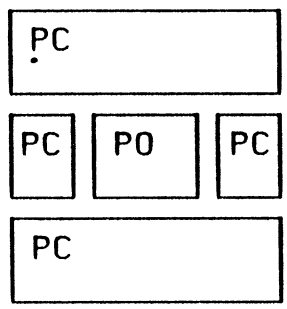


fig. 2.11 Partie contrôle répartie sur le pourtour de la partie opérative

2) - La partie opérative occupe l'extérieur du circuit => il existe une bonne connectique entre les plots et la partie opérative, mais il y a une perte de surface due aux coudes formés par la partie opérative, ainsi qu'un problème lié à la complexité de la partie opérative (la partie opérative doit être assez conséquente pour occuper le tour du circuit). Pour résoudre le dilemme posé par le coude, on peut utiliser des opérateurs triangulaires (ayant ses connecteurs à angle droit) tel un décaleur barillet [HOCH 87].

- La partie contrôle occupe le centre du circuit => la connectique entre la partie contrôle et ses plots n'est pas compromise, car cette connectique est faible. Par contre, il est nécessaire d'avoir une partie contrôle de complexité faible (par exemple une partie contrôle d'un processeur en architecture RISC, Reduced Instruction Set Computer. Un processeur RISC est un processeur posédant une partie opérative de grande surface (un grand nombre de registres, des opérateurs complexes), et une partie contrôle à fonctionnalité faible, donc faible en surface (figure 2.12).

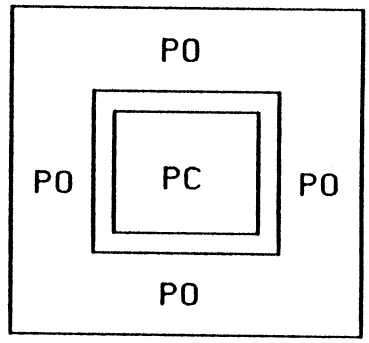


fig. 2.12 Partie opérative occupant le pourtour du circuit

3) Une solution intermédiaire aux deux solutions précedentes:
- La partie opérative n'occupe qu'une partie du pourtour du circuit => même problème que pour la proposition (2), la partie opérative doit posséder des opérateurs triangulaires pour occuper la surface perdue dans les coudes [OSSE 86],
- La partie contrôle occupe le centre du circuit (figure 2.13).

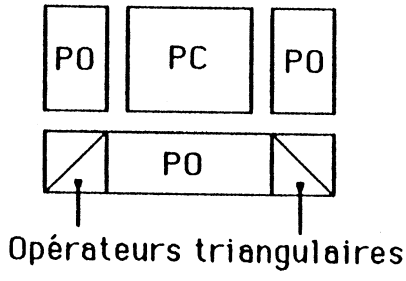


fig. 2.13 Proposition intermédiaire

Considérons une forme classique d'implantation de circuit telle définie dans [MeCo 81], avec la partie contrôle implantée au dessus de la partie opérative. La forme globale du circuit résultant à donné lieu à discussion dans [SCHO 85] et [MARI 86]. Lors de ces discussions, ce sont les relations entre performances et

complexités (surfaces) de la partie contrôle et partie opérative qui en ont été le centre.

* En effet la hauteur de partie opérative est fixée H(POP) par la taille en nombre de bits du mot à chaque opération (8, 16, 32...). Cette taiile ne correspond à une définition vraie que dans le cas ou la machine ne traite pas des instructions en demi-format (exemple un processeur posséde une partie opérative de quatres bits, mais est extérieurement vu comme un processeur huit bits, le Z80 par exemple). La performance (ou complexité en terme de nombre d'opérateurs différents) à atteindre est liée à la longueur de la partie opérative L(POP) correspondant aux ressources de celle-ci (figure 2.14).

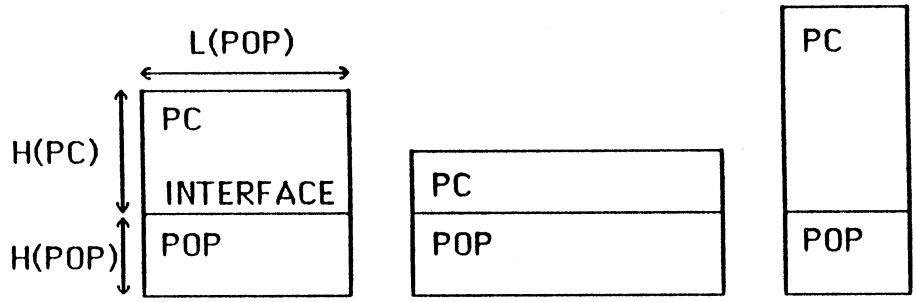


fig. 2.14 Variations de la forme globale d'un circuit en fonction du compromis entre la complexité du contrôle et la complexité des ressources opératives

* En outre la complexité de la partie opérative L(POP), en tenant compte du facteur L(POP)=L(PC), interragit avec la hauteur de la partie contrôle. Lorsqu'on augmente la longueur de la partie opérative, donc le nombre de ses commandes, le nombre total de cycle d'un algorithme (invariant) diminue, amenant la hauteur de la partie contrôle à diminuer.

Comment définir une bonne architecture pour un circuit de type microprocesseur?

Une bonne architecture prend en compte les différents critères énnoncés par avant (voir §II 1.1.3, critère de performance, de surface et de facilité d'implantation).

Les contraintes de performances sont liées à la traduction de l'algorithme d'interprétation de l'automate vers la solution matérielle. L'influence sur la surface est telle que cité dans [SCHO 85] [MARI 86].

Un deuxième facteur joue sur l'aspect surface : les choix topologiques pour l'implantation d'une ressource matérielle. Les différents choix topologiques

d'implantation d'un automate de contrôle fourniront des résultats plus ou moins optimaux en terme de surface.

Comment comparer l'influence de ces facteurs sur la topologie globale ?

Une proposition de [SCHO 85] est d'assimiler la surface d'un circuit à une trame virtuelle.

Cette trame virtuelle est construite à partir des fils de commande de la partie opérative. Elle est donc établie une fois la partie opérative construite, et est indépendante de la technologie. Cette trame virtuelle définit une structure d'accueil pour la partie contrôle. Le nombre de ces fils virtuels s'obtient comme fonction du nombre N de fils de commande, du pas moyen D de ces fils, et du pas PM (pas d'un niveau métal) des fils virtuels.

Nombre de fils virtuels: N x D/PM = (D/PM) x N = K x N

- D/PM est une constante dépendante de la technologie.
- s'il y a N commandes à générer, on dispose de (K x N) fils virtuels pour implanter la partie contrôle (on dit fils virtuels car ils ne sont pas forcément tous utilisés).

1.2 Interaction entre Partie contrôle et Partie opérative

Bien que ce ne soit pas le sujet essentiel de cette étude, il ne faut pas négliger son aspect fondamental. En effet, c'est à partir de la définition de la partie opérative et de sa réalisation physique, que l'on peut concevoir et construire la partie contrôle.

Si une partie opérative peut être réalisée à l'aide de logique anarchique, les concepteurs se sont tournés en premier lieu vers une implantation des ressources matérielles par blocs de fonctionnalités, connectés entre eux par l'intermédiaire de nappes de routages [MARC 86]. Ensuite, les concepteurs se sont tournés vers une solution architecturale autour d'un ou n bus, plongés dans la logique (c'est-à-dire que les bus ne "courent" pas sur du silicium avec aucun élément actif en dessous, mais sont directement intégrés dans la conception des cellules constitutives de la partie opérative).

La continuité de cette approche a vu l'élaboration de la solution de conception en tranches de un bit, tranches assemblées l'une après l'autre jusqu'à concurrence du nombre de bits de la partie opérative (étude effectuée sur les partie opératives bit-slice: cf [SUZI 81]).

L'approche de type "bit-slice" est universelle pour la constitution de partie opérative à un ou deux bus (ou plus si nécessaire). La méthodologie qu'elle suppose est de type descendant. Elle nécessite la définition des ressources, mémoires et opératives, puis la construction d'une tranche de un bit à répéter autant de fois que nécessaire.

Si les premières utilisations de cette technique (voir [SUZI 81]) utilisent une solution à un seul bus, les circuits actuels utilisent plus souvent une solution à deux bus, comme le circuit 68000 de Motorola ou la partie opérative du processeur PASCAL intégré POPI [SCHO 85].

A partir d'une architecture bit-slice, à un ou deux bus, plusieurs solutions sont possibles pour l'implantation d'une série d'opérations. Nous allons en présenter quelques unes en appuyant sur l'influence de leur architecture sur la partie contrôle.

Soit l'opération est de type unaire ou binaire, ou est constituée d'un ensemble d'opérations unaires ou binaires en parallèle.

* Dans le cas d'une opération unaire unique, une partie opérative à un seul bus est potentiellement :
  - performante (deux cycles nécessaires, un cycle pour la sélection des opérandes au niveau de l'UAL et le calcul, un cycle pour la sauvegarde du résultat).

- de coût surface et complexité le plus faible

* Dans le cas ou au moins une opération binaire est à effectuer par la partie opérative, alors nous pouvons recenser deux cas possibles:
  - soit l'opération se réalise en deux cycles (un cycle pour la sélection des opérandes et calcul, un cycle pour la sauvegarde des résultats), et seule une partie opérative à 2 bus peut réaliser une telle opération,
  - soit trois cycles peuvent être utilisés (un cycle pour amener la première opérande au niveau de l'UAL et la mémoriser, un cycle pour amener la seconde opérande et effectuer le calcul, un cycle pour sauvegarder le résultat), et alors la solution à un bus est préférable et suffisante.

Nous en déduisons alors :
- si le facteur performance prédomine -> choix d'une partie opérative à 2 bus.
- si le facteur surface prédomine -> choix d'une partie opérative à 1 bus.

Il apparaît en outre que la complexité de la partie opérative à 2 bus simplifie la gestion de la partie contrôle. En effet, dans le cas d'une partie opérative à 1 bus il est nécessaire de traiter deux états au lieu d'un seul pour la même opération binaire avec une partie opérative à deux bus (deux cycles de sélection d'opérandes pour une partie opérative à un bus). Le gain en surface n'est donc pas forcément réel.

* Dans le cas ou une opération est la combinaison de plusieurs opérations binaires ou unaires en parallèle alors : trois possibilités sont offertes, en partant, pour simplifier, d'une architecture de base à 2 bus:
  - considérer la partie opérative comme un assemblage de sous-parties opératives permettant de traiter les opérations unaires ou binaires en parallèle (figure 2.15) [JaJe 85].

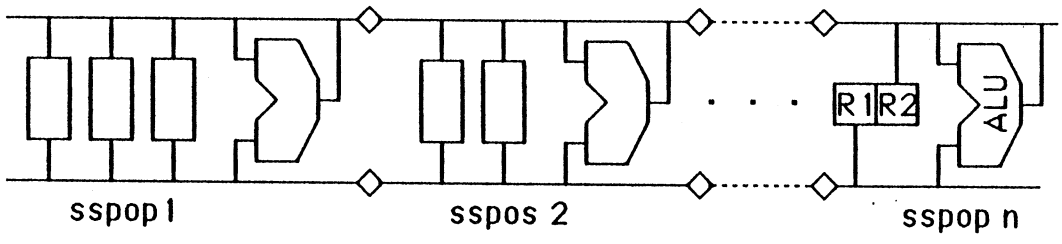


fig. 2.15 Partie opérative composée de n sous parties opératives en parallèle

Cette solution peut s'avérer coûteuse en matériel donc en surface [BEKK 86] dans le cas d'une faible occurence de ce genre d'opération. Par contre,

le facteur performance est optimum. Cette solution, déjà employée pour la construction du microprocesseur 68000 de Motorola, est à la base du développement de l'outil "APOLLON" [JaBe 86], de conception automatique de partie opérative à partir d'un langage de haut niveau type RTL.

- Une deuxième voie consiste à construire une partie opérative à plusieurs niveaux (partie opérative pipe-line), dans laquelle les opérations parallèles seraient introduites séquentiellement [PaPa 86a] (figure 2.16). Le pipe-lining se trouve au niveau de l'UAL et au niveau des mémorisations d'entrées et de sorties de l'UAL.

a) Les performances d'une telle partie opérative sont, hors le temps de latence de la première opération à effectuer, d'une opération réalisée par cycle. Le temps de latence correspond au transit de la première opération à travers tous les niveaux du pipe-line avant la sortie du résultat. Il est proportionnel au nombre de niveaux.

b) La surface est liée au nombre de niveaux de pipe-line choisi. Comme précédemment, le gain en surface de la partie opérative est relativisé par la perte de surface de la partie contrôle due à l'augmentation de la complexité de son algorithme.

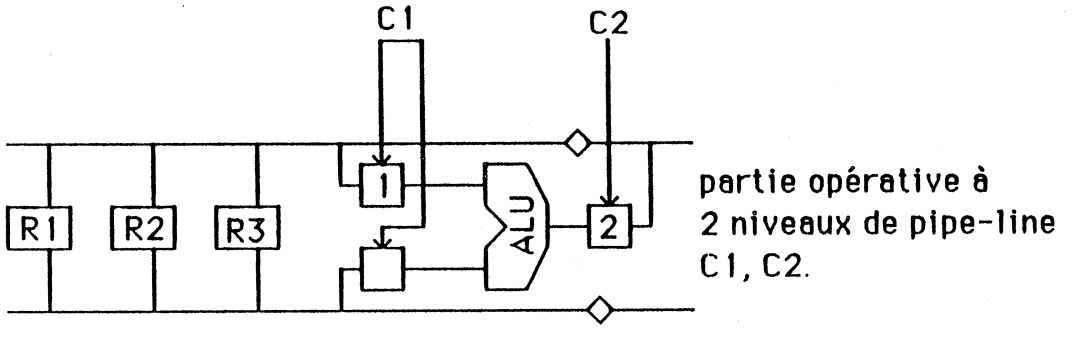


fig. 2.16 Partie opérative pipe-line

- Une troisième solution consiste à n'utiliser qu'une partie opérative, et à séquentialiser les opérations parallèles en opérations séquentielles. La complexité de la partie contrôle est donc considérablement augmentée (comme c'est le cas pour une partie opérative pipe-line) sans contrepartie au niveau des performances. La surface occupée par la partie opérative est par contre minimum.

A partir des remarques précédentes, on peut construire un tableau récapitulatif sur les différents types de parties opératives, que l'on comparera suivant les critères

(figure 2.17):

- surface de la partie opérative
- performances
- accroissement de la surface de la partie contrôle

	POP à ss.pop parallèles	POP pipelinée	POP unique
Surface	importante	moyenne	faible
Performance	2 cycles	apparent : 1 cycle réel: p cycles	n cycles
accroissement de la surface PC	Surface PC faible	PC à complexité importante	PC à complexité importante

fig. 2.17 Tableau comparatif entre différents types de parties opératives

Dans le cas d'une partie opérative pipe-line, la différence entre le nombre de cycles apparents et réels provient du fait que l'utilisateur à l'impression qu'un résultat d'opération sort à chaque cycle, bien que chaque opération se déroule en un nombre de cycle = p cycles, p correspondant au nombre de niveaux p du pipe-line de la partie opérative.

Séquencement de la partie opérative : Un seul exemple de séquencement (ou timing) sera donné. La partie opérative sera séquencée par une horloge à 4 phases non recouvrantes (figure 2.18) [GALL 84].

Les quatre phases sont générées à partir de deux horloges Ø1 et Ø2 sans recouvrement et de même période T.

Si dans le séquencement décrit par [GALL 84] [MARC 86] les quatre phases de l'horloge servent respectivement à :

H1 = Précharge du bus
H2 = Lecture sur les bus
H3 = Ecriture du bus dans récepteur
H4 = Calcul sur UAL

les quatre phases utilisées par "APOLLON" servent respectivement à :

H1 = Précharge du bus
H2 = Lecture sur les bus
H3 = Amplification
H4 = Ecriture du bus dans récepteur et en parallèle calcul sur UAL

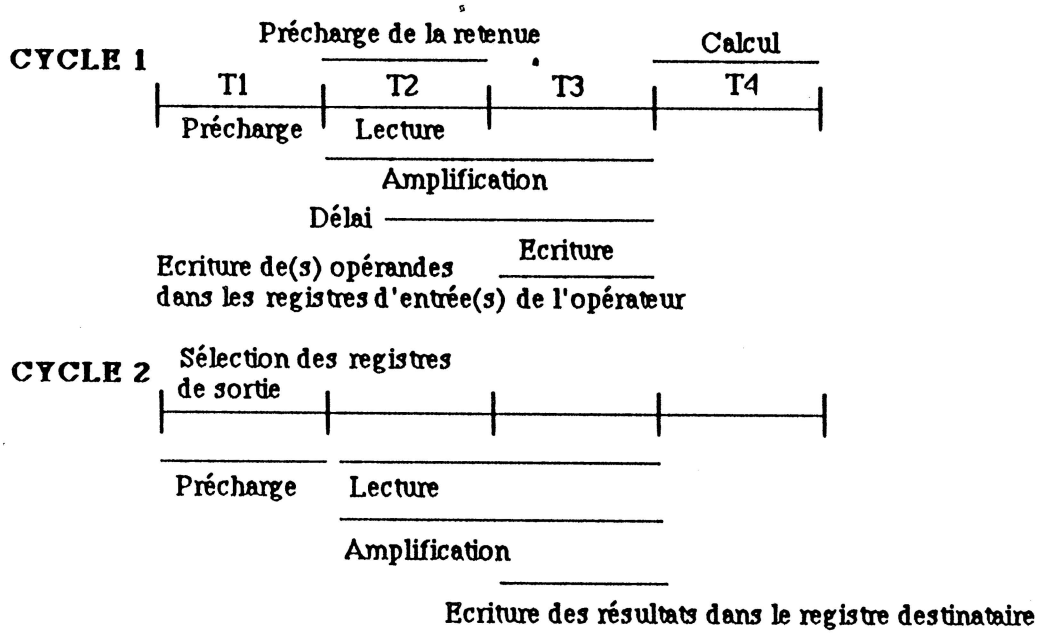


fig. 2.18 Horloge de base du circuit

Les parties opératives ne concernent que la partie traitement de l'information du microprocesseur. Il reste à définir la partie contrôle, automate séquençant la partie opérative.

2 ARCHITECTURES DE PARTIES CONTROLES COMPILABLES

2.1 Présentation de modèles généraux de compilation

Une partie contrôle consiste en un automate, ou ensemble d'automates permettant de réaliser l'organigramme d'un algorithme d'interprétation, en fournissant le séquencement de la partie opérative.

De manière schématique, la partie contrôle peut être décrite comme suit (figure 2.19):

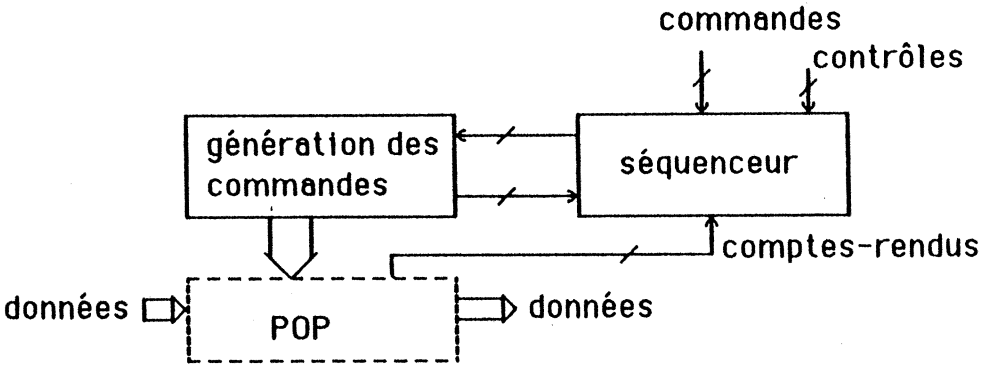


fig. 2.19 Représentation schématique de la partie contrôle d'un circuit de type microprocesseur

L'automate de partie contrôle peut être décrit à l'aide d'un graphe de PETRI, ou déduit à partir d'un algorithme d'interprétation écrit dans un langage de haut niveau (ex. PASCAL...).

Il peut être procédé à toute transformation série<->parallèle, composition/décomposition de réseaux selon les lois de transformations définies pour les réseaux d'automates d'états finis (cf §II 1.1.1).

Pour respecter le modèle temporel de SYCO, vrai pour une zone isochrone, nous considèrerons que tous les automates sont synchrones.

Le réseau d'automate tel qu'il est utilisé pour le compilateur de silicum **SYCO** [JeVa86] (figure 2.20) est un sextuplet $N = (Z,\{Si\},W,\{fi\},\{\Psi i\},g)$ ou

* Z est l'ensemble des signaux d'entrée
* $\{Si=(Ai,Zi,Wi,\delta,\lambda,ai1)\}$ $1\leq i\leq n$ est un ensemble d'automates rebouclés sur $\{fi\}$, $1\leq i\leq i$; automate composant du réseau; chaque automate Si est tel que $Zi=\{Z'i \times Z''i$ avec $Z'i\neq\emptyset$ ou $Z''i$ pour $Z'i=\emptyset\}$
* W est l'alphabet de sortie du réseau
* $\{f1 : W \rightarrow Z'1, fi : Wi \rightarrow Z'i\}$ est l'ensemble des fonctions de connexion des

automates composant le réseau
* {Ψi : Z -> Z"i}, 1≤i≤n est l'ensemble des fonctions d'entrées
* g : An -> W est la fonction de sortie du réseau

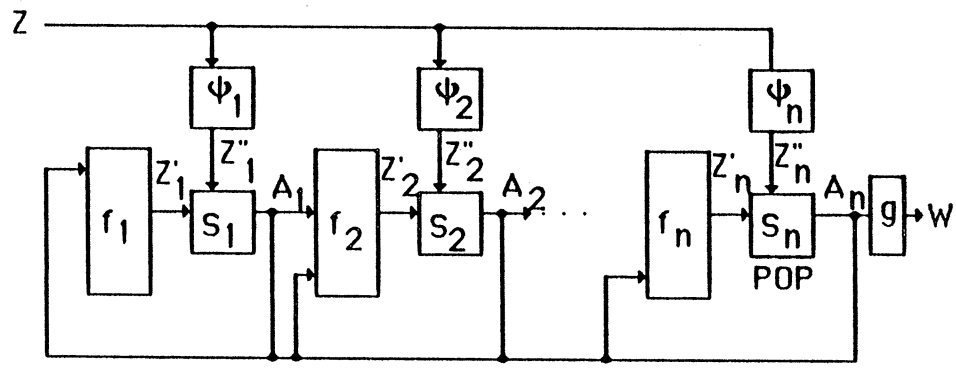


fig. 2.20 Description sous forme de réseau d'automates du modèle **SYCO**

Il apparaît, bien que théoriquement l'on soit capable de décrire une partie contrôle sous forme de réseau d'automates, qu'il n'est pas aisé d'en proposer une implantation "propre". Une implantation propre correspond à une surface de zône d'interconnexion faible, et un nombre de blocs composant la structure faible.

Pour implanter une partie contrôle de type **SYCO**, deux voies extrèmes sont possibles. On prendra pour la représentation les conventions suivantes:

- les n-1 automates de contrôle composant le réseau sont nommés
  PCi; i=1,.., n-1
- l'automate n de partie opérative est nommée POP

Les deux cas possibles d'implantation des automates de contrôles sont les suivants:

- soit les n-1 automates sont implantés les uns au dessus des autres, et l'ensemble au dessus de la partie opérative (figure 2.21 a),
- soit les n-1 automates sont implantés les uns à côté des autres, et l'ensemble au dessus de la partie opérative (figure 2.21b).

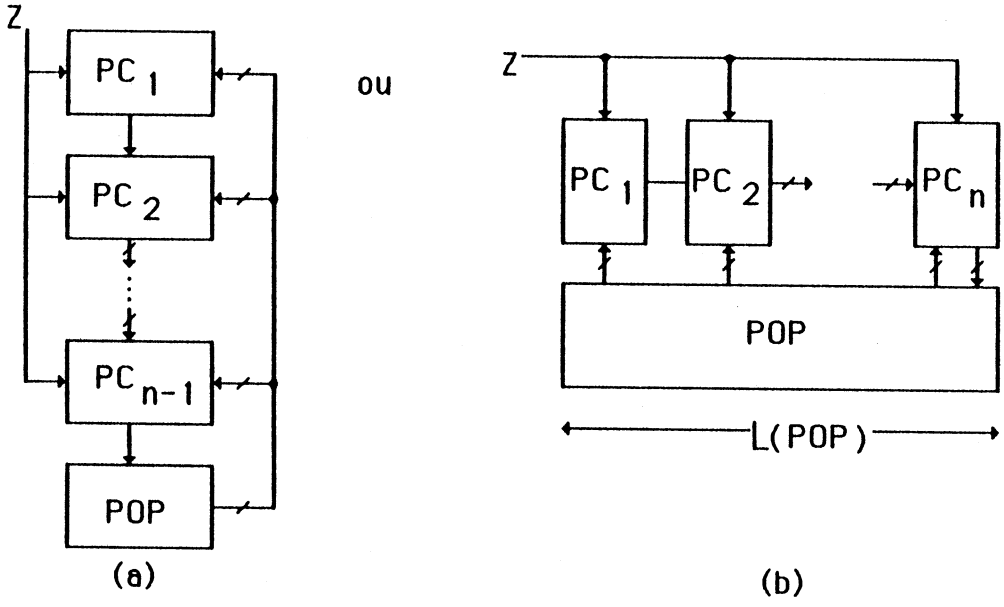


fig. 2.21 Modèles topologiques globaux possibles pour le modèle type **SYCO**

La longueur L (POP) détermine la largeur du circuit, et par conséquent la base pour l'implantation de la partie contrôle.

Le choix d'implantation entre ces deux solutions est fonction de la complexité et du nombre d'états des automates du réseau. En effet, plus la complexité et le nombre d'états sont importants, plus la surface de la réalisation matérielle de l'automate augmente.

Dans le cas d'une structure implantée horizontalement (figure 2.21a), les parties contrôles successives doivent être déformées verticalement. L'inconvénient majeur est produit par la partie contrôle la plus proche de la partie opérative du point de vue fonctionnel. En effet sa forme implique une adaptation topologique de ses sorties vers la partie opérative. Dans le cas de la non adaptibilité il se produit une zône de routage non désirable.

Dans le cas d'une stucture implantée verticalement (figure .21b), les zones de routages sont minimisées. Pour construire les parties contrôles, l'on établit tout d'abord la trame virtuelle définie par les informations inter-parties [SCHO 85], trame sur laquelle sont implémentées les parties contrôles. Cette solution est ainsi topologiquement préférable à la première.

Nous allons dans la suite de ce paragraphe proposer différents types de structures-cibles possibles pour implanter les (n-1) automates de partie contrôle du réseau. Pour cela, on considèrera uniquement un automate, pour lequel on définira des structures-cibles. Ces structures cibles sont destinées à être intégrées dans une structure plus complexe type **SYCO**.

Vers quel type de structure , les concepteurs des microprocesseurs actuels se dirigent-ils ?

Historiquement, les premiers automates de partie contrôle furent implantés à l'aide de portes logiques: structures non régulières, que l'on a appelé par la suite logique anarchique. Cela découlait du fait que les concepteurs des premiers circuits concevaient comme l'on conçoit des cartes à l'aide de circuits SSI et MSI. Ils utilisaient des bibliothèques de cellules [M6800, Z80...]. Une autre raison de l'utilisation de la logique anarchique, c'est que l'on câblait directement l'algorithme décrit sous forme de Moore (W=f(A)). Les générations suivantes se sont dirigées vers des solutions utilisant un nombre important de structures régulières tels les PLA's, ROM, RAM...

Nous allons essayer de présenter les différentes solutions en tenant compte des facteurs :

- performance,
- régularité,
- temps de conception.

2.2 Partie contrôle implantée à l'aide d'une structure à logique anarchique.

Une logique anarchique est une logique ou le facteur de répétitivité est faible sinon nul. Ce facteur de répétitivité se traduit au niveau topologique par une faible occurence d'une cellule dessinée (en fait, plus la répétitivité est élevée, plus la facilité d'implantation d'une structure est élevée). En outre le routage entre cellules est important (les zônes de routage peuvent éventuellement être placées au-dessus des cellules).

Une réalisation en logique anarchique peut indifféramment être basée sur une description de Moore ou de Mealy (§II 1.1.1.1). De par le fait qu'un automate de Moore posséde plus d'états que l'automate de Mealy équivalent, l'implémentation matérielle d'un automate de Moore est plus coûteuse puisqu'il faut faire correspondre à chaque état une bascule d'état (bascule D par exemple). La génération des commandes est néanmoins simplifiée car les commandes ne dépendent que de l'état dans lequel l'automate se trouve à un instant 't' (figure 2.22).

En général, les automates complexes sont implémentés sous forme d'automate de Mealy. Un état de l'automate de Mealy correspond à plusieurs états d'un automate de Moore équivalent, et les commandes sont validées en fonction des conditions calculées extérieurement.

Performance d'un automate de partie contrôle réalisé en logique anarchique

Le temps de calcul d'une commande est lié d'une part au temps de transit d'un état i vers un état j, d'autre part au temps nécessaire à la validation de la commande par les conditions. De manière générale, le temps de transfert à travers les portes de validation des commandes est inférieur à celui d'une structure logique régulière du type PLA, ce qui se traduit par un fréquence d'horloge plus élevée [OBRE 82] (un PLA à 50 lignes de monômes, 25 colonnes d'entrées/sorties représente une capacité à charger plus importante que dans le cas ou l'on a une ou deux portes logiques à traverser: la solution utillisant des portes logiques est donc plus rapide).

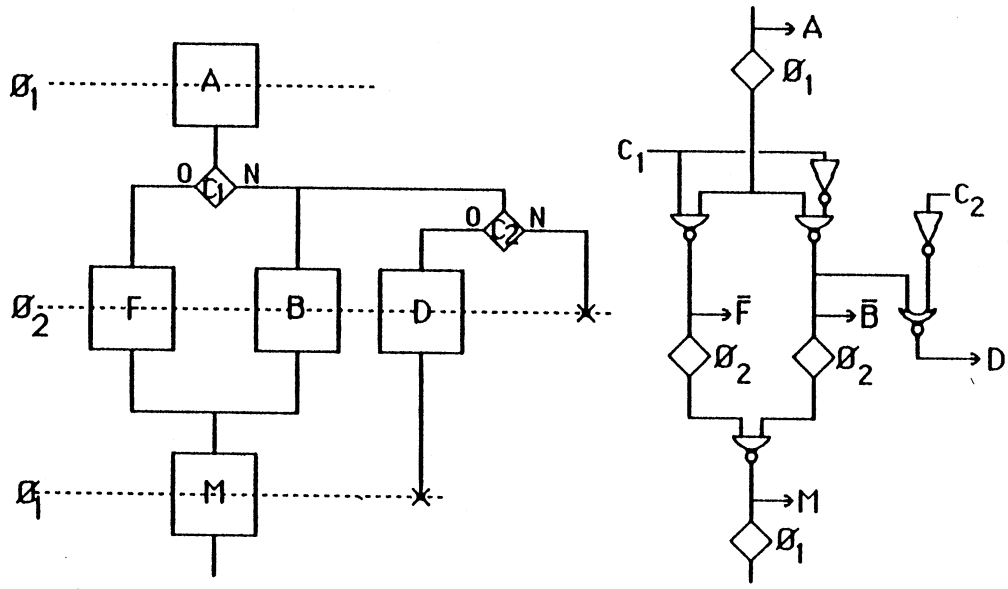


fig. 2.22 Exemple d'une partie d'algorithme et de sa réalisation matérielle en logique anarchique

Surface d'un automate de partie contrôle réalisé en logique anarchique

Il n'est pas aisé de trouver une formule correcte permettant d'estimer la surface d'un automate réalisé à l'aide d'une telle structure: [OBRE 82] propose la formule suivante:

$S_{AUT} \approx g_A \times NE$

où S_{AUT} est la surface de l'automate,

g_A est une constante caractérisant la surface virtuelle d'un état, g_A a été évalué à 25149,7μm^2 en technologie NMOS GSN3, dans le cas du 6800.

NE = nombre d'états de l'automate.

Cette formule ne traduit pas réellement la surface nécessaire pour l'implémentation d'un automate de Moore ou de Mealy. Elle a été élaborée à partir d'un exemple du 6800, dont on connait la surface, et du nombre d'états de son automate implémenté sous forme de Moore.

Une autre estimation empirique pour un circuit réalisé en logique anarchique a été proposée par [MARQ 80]. Cette formule lie la surface totale au nombre de portes logiques à implanter par un coefficient G_p (surface moyenne occupée par une porte dans une technologie donnée):

$S_{LA} = G_p * N_p$
avec N_p : nombre de portes logiques,
et $G_p = 4160\ \mu m^2$ en technologie GSN3 pour le 6800.

Dans les deux cas, il existe un facteur (g_a ou G_p) lié à la technologie. Ces facteurs doivent être redéfinis pour chaque technologie, c'est pourquoi il est assez difficile de comparer deux microprocesseurs non réalisés dans la même technologie.

Facilité d'implantation

Le temps d'implantation d'une structure en logique anarchique est fonction du nombre de portes à implanter (Dans le cas du 6800, une porte comporte en moyenne de 4 à 5 transistors). Classiquement, le nombre de 7 transistors/jour/homme est donné [MARQ 79] [OBRE 82], ce qui se traduit par un temps considérable si l'on désire implanter un automate complexe du type partie contrôle du 68000 de Motorola. En outre l'irrégularité de la logique anarchique impose une refonte totale de l'implémentation d'un automate en cas de changement d'une ou plusieurs parties de son algorithme (ou par la détection d'une éventuelle erreur).

L'inconvénient majeur de cette structure provient de la difficulté rencontrée dans l'automatisation du processus de dessin. Différentes approches ont été recensées et proposées par [MARI 82] et [BIAN 81]. Toute recherche sur une méthode d'implantation automatique de logique anarchique se traduit fatalement par une perte de surface, car l'on recherche à ré-introduire de la régularité dans le dessin. Classiquement, ces méthodes d'implantation consistent à utiliser une structure en bandes, bandes constituées par les alimentations ou la connectique entre cellules. Sous cette structure en bandes, la logique est implantée [PAIL 84].

2.3 Partie contrôle réalisée à l'aide d'une structure à PLA unique

Un PLA (Programmable Logic Array) fait partie du groupe de Matrices Logiques Programmables (MLP) constitué par les ROM, PLA, SLA (Structure Logic Array)... Il permet de réaliser la synthèse de n'importe quelle fonction logique sous la forme d'une somme de produit des termes d'entrées. L'implantation d'une telle forme logique se réalise à l'aide de deux matrices de transistors, respectivement matrice ET et matrice OU [CHUQ 84] (figure 2.23).

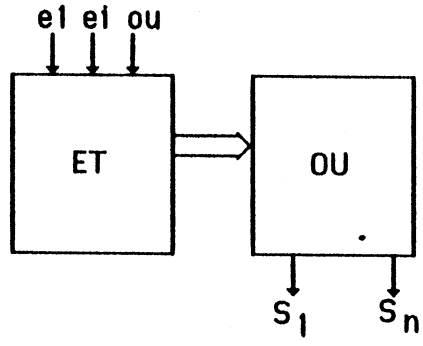


fig. 2.23 Structure logique de PLA

Une telle forme logique ne peut pas être implantée telle quelle, sans tenir compte de la technologie. Par exemple en technologie NMOS, la forme canonique implantée est de type NOR-NOR avec inversion des sorties (la logique NMOS est une logique négative: on implante facilement les portes NOR et NAND). De plus, fabriquer un NAND avec un nombre d'entrées supérieur à 5 est technologiquement très coûteux en surface, et au niveau électrique (que la technologie soit NMOS ou CMOS) quasi impossible. Cela est du au phénomène tension-source en chaque MOS en série (perte de seuil) [MeCo 81] [GlDo 85].

L'intérêt d'une telle structure logique est que pour l'implantation d'une fonction logique, son utilisation est plus avantageuse que les ROM ou les circuits combinatoires à structure cablée [ATKI 81], [LaRi 81], [LaSh 82] (un PLA n'est autre qu'une ROM à décodeur incomplet). Dans la conception d'un circuit il améliore sa sureté de conception et en diminue les coûts [FlMa 75], [JONE 75].

Un autre intérêt de ce type de structure, en est les nombreuses techniques et caractéristiques dans le cadre de l'optimisation logique des fonctions à implanter et topologique pour la réduction de la surface.

Ils peuvent être considérés comme des blocs topologiquement adaptables à leur environnement [CHUQ 84].

Comment passer d'une description d'un automate, à la structure d'accueil monoPLA?

Une équation de fonction s'obtient à partir de la description des algorithmes. Cette description s'obtient elle-même par l'analyse des états dans lesquels la sortie concernée doit être activée. On tiendra compte, le cas échéant, des conditions d'activation associées. De cette façon on obtient l'ensemble des équations qui décrit le comportement de l'automate de séquencement. A chaque pas de l'algorithme, toutes les fonctions sont donc évaluées.

$w(t) = \lambda\,(a(t),z(t))$; $\lambda = \Sigma\Pi(z(t),a(t))$
$a(t) = \delta\,(a(t-1),z(t))$; $\delta = \Sigma\Pi(z(t),a(t-1))$
$S(\Lambda,Z,W, \delta, \lambda)$.

Surface d'une structure MonoPLA

Une telle structure monoPLA possède un taux de remplissage de la matrice OU relativement faible (matrice creuse), d'où une possibilité d'optimisation topologique importante de cette matrice. Pour plus de détail sur les méthodes d'optimisation des PLA's, on se reportera à [CHUQ 85] [OBRE 82] [VARI 85a]. Une telle structure monoPLA a d'ailleurs été employée par NS (Nationnal Semiconductor) pour les microprocesseurs SC/MP et INS 807X.

Pour un automate de Mealy (implémentation choisie pour le compilateur **SYCO**), nous obtenons le schéma équivalent suivant (figure 2.24):

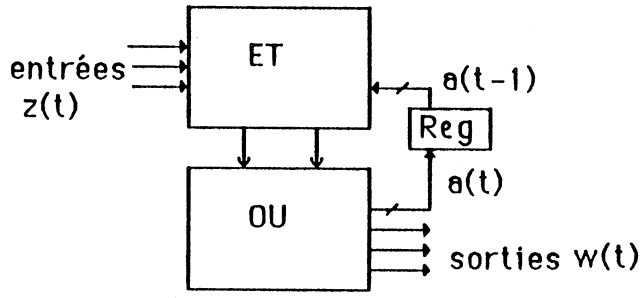


fig. 2.24 Structure d'une partie contrôle monoPLA sous forme de Mealy

L'inconvénient de ce type d'automate (Mealy) est du au risque d'aléas en cas de changement du vecteur d'entrée Z durant l'établissement des sorties (Dans le cadre de **SYCO**, ou l'automate implanté est de type Mealy, toutes les fonctions de sortie sont resynchronisées par une horloge, et les entrées sont mémorisées durant le temps de calcul des sorties).

Sous forme de Moore, l'automate est resynchronisé. Nous obtenons alors le schéma structurel suivant pour un automate de Moore (figure 2.25):

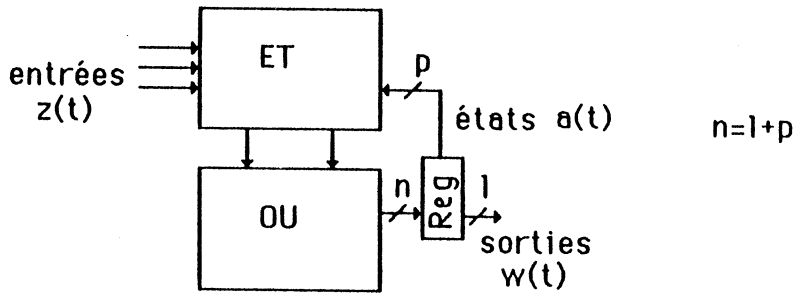


fig. 2.25 Structure d'une partie contrôle monoPLA sous forme de MOORE

Facilité d'implantation

L'implantation d'une telle structure, de par sa régularité intrinsèque, est compilable automatiquement. Des générateurs de PLA's existent [CHUQ 85] [MiSa 83], et permettent en outre une optimisation topologique et logique.

Performance d'une structure MonoPLA

La performance d'une structure de partie contrôle MonoPLA est liée à la taille du PLA, et donc à la complexité de l'automate à implanter. La complexité est fonction du nombre d'états différents de l'automate de contrôle. La raison de l'incidence de la taille du PLA sur la performance de la partie contrôle est due au fait que plus le nombre de monômes et d'entrées/sortie est important, plus la longueur des lignes à charger (ligne d'entrée en polysilicium en technologie NMOS) est importante, donc plus la capacité est élevée.

Le séquencement ('timing') d'une telle partie contrôle se résume par le besoin d'une horloge validant les données du changement d'état. Dans le modèle de compilation **SYCO**, pour éviter les risques d'aléas propres à une implémentation de type Mealy, les entrées et les sorties sont mémorisées à chaque pas de l'algorithme, aussi, deux phases d'horloges sont nécessaires, une pour valider l'entrée, une pour valider les sorties. La mémorisation est réalisée à l'aide de registres (bascule D) en entrée et en sortie.

2.4 Partie contrôle réalisée à l'aide d'une structure à PLA's multiples

La structure à PLA unique possède un inconvénient majeur: si le nombre d'états de l'automate est important et que le séquencement est complexe, le PLA occupe une surface très importante [OBRE 82] [DERA 85]. Cela provient aussi du fait que la matrice OU du PLA est très creuse.

La première solution présentée (§II 2.4.1) illustre comment on peut modifier l'algorithme d'interprétation des instructions, si on prend d'abord en considération les propriétés globales de l'instruction ou des groupes d'instructions, puis les propriétés locales de certains blocs fonctionnels.
La deuxième solution (§II 2.4.2) est bâtie sur une extrapolation du principe d'utilisation des propriétés locales, c'est-à-dire des paramètres [OBRE 82]. Dans les deux cas l'automate est décrit sous forme de Mealy.

2.4.1 Structure multi-PLA avec extraction des propriétés :

Cette solution correspond à transformer l'automate de Moore, utilisé dans la solution ROM ou MonoPLA, en automate de Mealy équivalent. Cette transformation diminue le nombre d'états d'algorithme (cas de groupe d'instructions de microprocesseur), mais les sorties générées dans chaque état ne dépendent plus uniquement de cet état. Elles dépendent aussi d'autres valeurs qu'il faut calculer séparément. Ce sont ces valeurs qui sont appelées propriétés [DERA 85].
Le PLA de séquencement est simplifié par ce codage puisqu'il ne doit reconnaître que des familles d'instructions, et non des instructions séparées. En outre, le fait de la diminution du nombre d'états différents (automate de Mealy) implique une diminution de la taille de la matrice ET qui reconnait l'état courant, et celle de la matrice OU qui génère l'état suivant.

Schéma fonctionnel de la solution à PLA's multiples avec extraction des propriétés (figure 2.26).

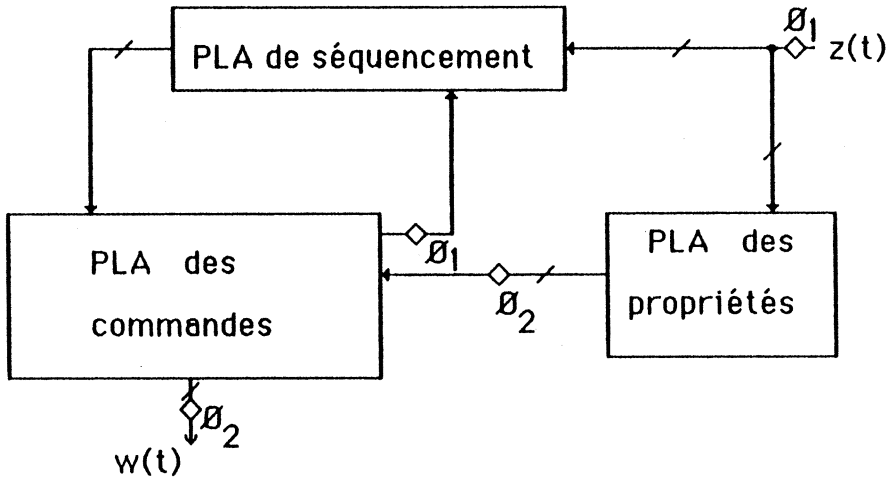


fig. 2.26 Structure d'une partie contrôle à multi PLA avec extraction des propriétés

Le PLA des commandes représente le PLA le plus important, bien que comparativement peu important par rapport au PLA d'une structure monoPLA. Cela est dû à la mise sous forme de Mealy de l'algorithme.

Le séquencement (ou activation temporelle) utilise une horloge biphasée, où la première phase sert à valider les résultats du PLA de propriété et de celui de séquencement, la seconde phase à valider les résultats du PLA de commande.

2.4.2 Structure multi-PLA avec extraction des paramètres :

Cette solution pour implémenter un automate est surtout valable pour générer les commandes d'une partie opérative (figure 2.27). Les paramètres concernent un ensemble des commandes agissant sur une unité fonctionnelle de la partie opérative qui n'exécute qu'une fonction spécifique au cours de l'interprétation d'une instruction donnée, le reste du temps, une fonction par défaut est exécutée.

De ce fait, cet ensemble de commandes peut être généré statiquement à partir du code opération dès la lecture de l'instruction et les valeurs de commandes sont alors appelées paramètres.

Dans la description de l'état de l'algorithme, toutes les commandes formant cet ensemble sont enlevées et remplacées par une seule commande séquencée de sélection. Cette commande de sélection permet de faire le choix entre les paramètres et les valeurs par défaut générées à l'entrée du multiplexeur.

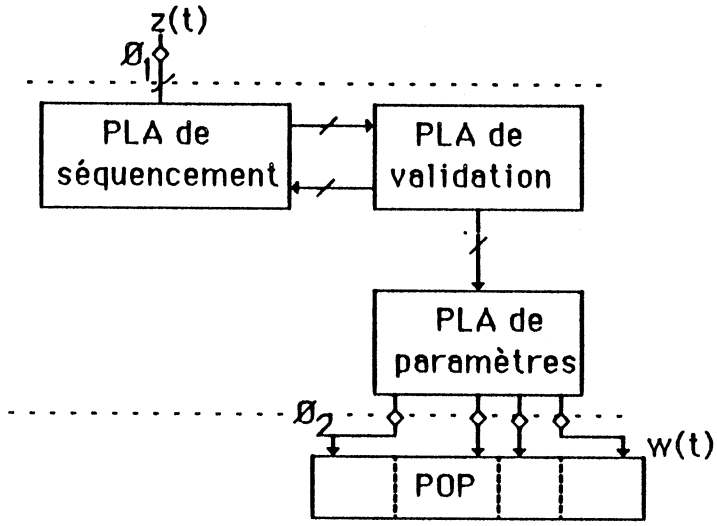


fig. 2.27 Schéma fonctionnel de la solution multiple avec extraction des paramètres

2.4.3 Structure générale à PLA's multiples:

En fait, si l'exécution des propriétés reflète les caractéristiques globales de l'instruction, l'extraction des paramêtres correspond aux caractéristiques particulières des unités fonctionnelles discernées au sein de la partie opérative.

Nous pouvons alors déduire des deux cas précédents un schéma global qui comporte :

- Un PLA de séquencement,
- Un PLA de génération des commandes,
- Un PLA de propriétés,
- Un ou plusieurs PLA's de paramêtres.

La taille des PLA's dépend essentiellement des caractéristiques de l'organigramme de l'algorithme considéré (figure 2.28)

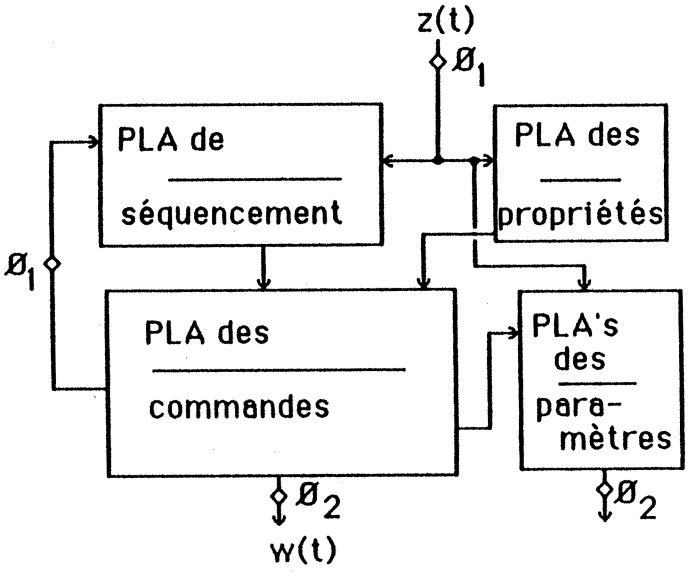


fig. 2.28 Structure d'une partie contrôle multi-PLA's générale

Respect du facteur performance:

L'utilisation d'une solution multi-PLA's améliore le facteur performance par la diminution de la taille des PLA's. En effet, plus un PLA est gros, plus les lignes d'entrées et de monômes possèdent une capacité et une résistance importantes, d'où un temps de calcul global du PLA augmenté (la charge d'une capacité est une fonction exponentielle de la valeur de la capacité, cela implique que la charge de deux petites capacités, deux petits PLA's, est plus rapide que la charge d'une seule grosse capacité, solution monoPLA).

Respect du facteur de surface:

S'il y a une perte de régularité de l'ensemble par rapport à une solution à PLA unique, le gain en surface sur chaque PLA équilibre cette perte de régularité. Cette solution parait plus intéressante dans le cas d'une adaptation topologique globale des différents PLA's entre eux.

Facilité d'implantation

La compilation d'une telle structure est logiquement et géométriquement réalisable avec les outils disponibles actuellement. Il existe de nombreux outils de génération automatique de PLA. Dans le cadre de **SYCO**, le fait de disposer d'une topologie cible simplifie d'autant le traitement topologique à effectuer sur les divers PLA's constituant la structure.

D'autre part la structure multi-PLA est très avantageuse dans le cas de changement d'une partie de l'algorithme, ainsi que dans la mise au point du circuit, car la modification de l'implantation n'intervient généralement que sur une partie de la structure (un PLA ou une fonction de PLA), ceci n'entrainant donc pas une refonte globale du circuit.

2.4.4 Structure à générateur de temps:

La traduction d'un algorithme dans une structure à générateur de temps représente une autre façon de voir le problème de l'interprétation des commandes par rapport aux solutions présentées ci avant. Le principe consiste à considérer qu'une commande de sortie d'un automate est caractérisée par le ou les instants de son activation au cours de l'exécution d'une valeur de commande d'entrée [SCHO 85] [OBRE 82] [ANCE 86].

Le déroulement d'un vecteur de commande d'entrée peut être décrit par deux ensembles:

- L'ensemble de toutes les commandes à activer pour ce vecteur d'entrée,
- L'ensemble des instants de leur activation,

Aussi pour réaliser un algorithme d'interprétation, il faut disposer:

- D'un dispositif qui élaborera les commandes à activer (PLA des propriétés)
- D'un dispositif qui délivrera les instants définissant le temps (générateur de temps),
- D'un dispositif qui validera les commandes au bon moment (PLA des commandes)

PLA des propriétés : le PLA des propriétés définit à partir du vecteur de commande d'entrée:

- Les propriétés particulières du vecteur d'entrée. Ces propriétés décrivent les commandes de sortie à générer (attributs)
- Les propriétés globales qui décrivent les caractéristiques de la famille à laquelle appartient le vecteur d'entrée. Ces caractéristiques sont en particulier , le nombre et le type des intervalles de temps nécessaire à l'exécution du vecteur d'entrée, composant le schéma temporel de cette famille.

Générateur de temps : Un générateur de temps est un automate assurant la progression de l'interprétation du vecteur d'entrée, ainsi que l'identification de chaque instant d'exécution grâce aux signaux de séquencement qu'il délivre. Il tient compte des propriétés du vecteur d'entrée, de l'état de la machine et des contrôles extérieurs. Il réalise un algorithme de contrôle sous forme de Mealy très paramétrisé. Il peut être réalisé à l'aide d'un PLA (ou d'une manière topologique plus efficace en terme de surface à l'aide d'un compteur et d'un PLA).

PLA des commandes : Le PLA des commandes fournit les vecteurs de sortie en fonction des informations de temps et des informations de propriétés particulières.

On obtient alors le schéma structurel suivant (figure 2.29):

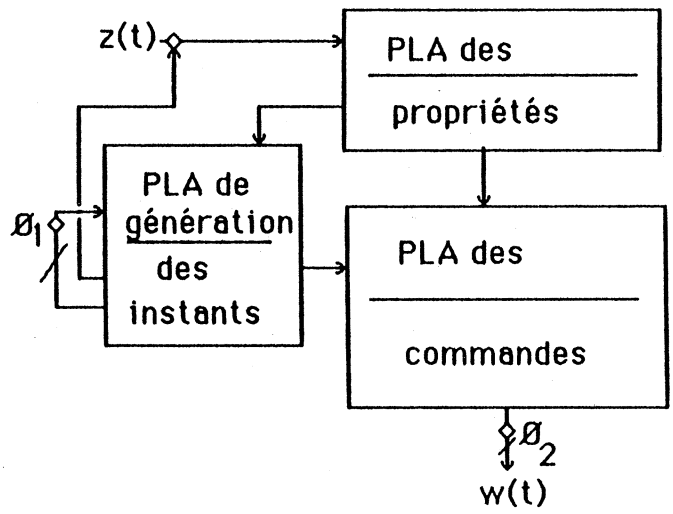


fig. 2.29 Structure d'une partie contrôle à générateur de temps

Performance

L'intérêt d'une telle structure réside dans le cas d'un séquencement systématique, c'est-à-dire ne nécessitant pas un grand nombre de groupes d'instants différents. Par exemple le 8048 de chez INTEL possède des instructions s'exécutant soit en dix , soit en vingt instants, sans séquencement conditionnel.

Surface/ Facilité d'implantation

L'inconvénient d'une telle structure est son irrégularité face à une solution MonoPLA ou même multi-PLA sans générateur de temps (Dans [OBRE 82], on considére le 'TIMING GENERATOR' ou générateur de temps comme réalisé en logique anarchique, et il est adjoint un facteur de régularité de 33% à une structure de partie contrôle à générateur de temps). Cette solution apporte malgré tout un gain en surface dans des cas précis comme celui du 8048, et permet dans ces cas des gains en performance, gains déterminés par des tailles de PLA plus faibles (à l'extrême, si le générateur d'instant ne doit générer qu'un style de séquencement, seul un compteur est nécessaire).

2.5 Partie contrôle réalisée à partir d'une structure à base de ROM (Read only memory)

Une ROM permet d'implanter une fonction logique sous forme canonique. Elle est pour cela consituée d'un décodeur d'entrée, le décodeur d'entrée correspondant aux décodeur ligne et décodeur colonne (Matrice ET PLA), d'un plan de matrice (Matrice OU PLA) ou zône de mémorisation. Le plan de matrice étant organisé en n.uples mots, n=1,...p, il faut donc lui associer un démultiplexeur en sortie, faisant le choix d'un mot parmi n. Pour raison d'optimisation de surface, on ajoute généralement un décodeur des commandes du démultiplexeur : le nombre de commandes en entrée de ce décodeur est alors $Log_2(Log_2(n))$ fils).

L'utilisation d'une ROM pour implémenter un algorithme d'interprétation est lié au concept de la microprogrammation: Cela veut dire qu'une microinstruction d'une ROM contient l'information liée aux commandes de sortie, ainsi que l'information permettant de déterminer l'adresse de la microinstruction suivante.

Une ROM implémente classiquement un algorithme sous forme de Moore. Dans le cas du 68000, la ROM implémente un algorithme sous forme de Mealy.

Du point de vue logique, la solution ROM est proche de la solution multiple PLA.

Un automate utilisant une ROM peut être implémenté de deux manières:

1) Soit son algorithme d'interprétation est mis à plat, sous forme de Moore, sans recherche de propriété propre aux instructions ou à un groupe d'instructions. Un état de l'algorithme est assimilable alors à une adresse de microinstruction ou toute l'information concernant cet état est écrite. Dans ce cas, comme dans le cas de la solution à PLA unique, nous nous retrouvons avec une surface de ROM importante.

Une microinstruction est constituée de trois champs:

- Un champ de séquencement,
- Un champ définissant l'adresse de la microinstruction suivante,
- Un champ test décidant si la microinstruction suivante doit être déduite du champ adresse, ou de conditions extérieures.

Ce cas de programmation s'appelle la "microprogrammation horizontale" [GlDo 85].

2) Dans le cas de la microprogrammation verticale, les différents champs de la microinstruction subissent un codage nécessitant des décodeurs en sortie de la ROM. Une première amélioration consiste à coder les microinstructions avec des champs à formats variables pour diminuer les tailles des microindtructions.Un champ est alors ajouté à la microinstruction pour retrouver le formatage effectué.

Une deuxième amélioration consiste à supprimer le champ séquencement de la

microinstruction. On considère alors que les microinstructions sont exécutées en séquence, et qu'en cas de rupture de séquence, on utilise une microinstruction de contrôle qui contient la microadresse de branchement ou l'information nécessaire au calcul de la microadresse suivante. Une partie du traitement du séquencement est ainsi reportée dans un PLA de séquencement. Nous obtenons alors le schéma fonctionnel suivant (microprogrammation verticale). Dans ce cas l'automate est plus performant sous forme de Mealy, avec recherches des propriétés et paramétrisation(voir §II 2.4.2) (figure 2.30) [OBRE 82] [GIDo 85] et documentation Motorola sur le 68000.

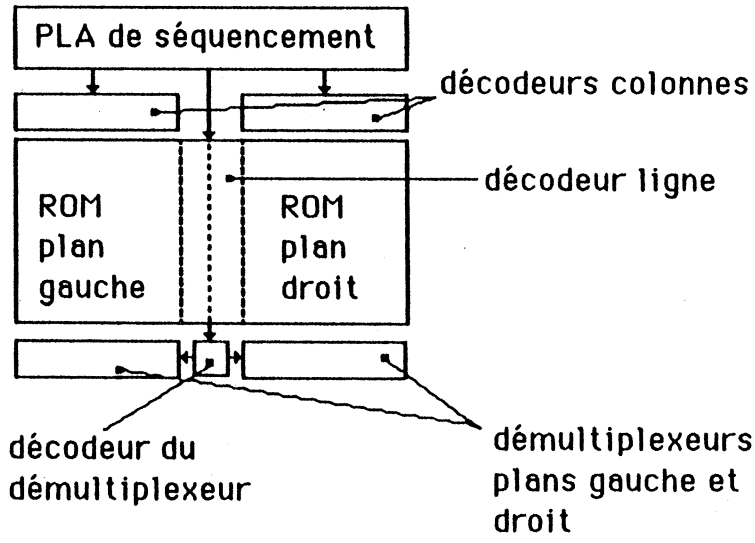


fig. 2.30 Structure d'une partie contrôle réalisée à l'aide d'une ROM à Microprogrammation verticale.

Le PLA de séquencement peut être simplifié par ajout d'un compteur. Il est à noter que comme pour la solution à multiple PLA's, l'ajout d'un compteur diminue la régularité de la topologie.

Inconvénients d'une solution à ROM :

Une solution à ROM amène à une surface généralement plus importante que toutes solutions à PLAS multiples, ou à logique anarchique. La taille de la ROM de microprogramme provoque aussi généralement une diminution de la fréquence d'horloge, due au transit de l'information à travers la ROM.

Avantages d'une solution à ROM :

La régularité dans la conception est la plus élevée de toutes les solutions proposées précédement. En outre, on laisse généralement dans une ROM de microprogramme de la place libre pour du code supplémentaire. La modification complète du jeu d'instruction est aisé, et l'on peut ainsi reprogrammer la ROM, et donc le circuit pour une application spécifiques [DeMa 86].

2.6 Evaluation comparative

Des travaux de [OBRE 82], on peut tirer l'évaluation comparative des différents types de parties contrôles.

Cette étude a été menée dans le cadre d'un circuit du commerce le 6800 (ainsi que le 6809) de Motorola. Ce circuit a été redessiné avec chacun des types de partie contrôle définis plus haut. La figure 2.31 présente l'influence des formes d'implantation sur la surface d'une partie contrôle. Par contre, la différence enregistrée pour le temps de cycle entre telle ou telle architecture de partie contrôle apparait négligeable, et ne doit pas influencer le choix, autrement que pour des cas critiques tels le traitement de signal, ou les co-processeurs spécialisés (un seul cas critique, la solution à PLA unique pour implémenter une partie contrôle). D'après la figure 2.32, on s'aperçoit clairement que le gain en surface se traduit par une diminution de la régularité. L'ensemble de ces deux figures (figures 2.31 et 2.32) démontre l'influence des formes régulières sur la surface finale de la partie contrôle (plus la partie contrôle est régulière, plus la partie contrôle occupe une surface importante), mais aussi l'influence du nombre de niveaux d'interprétation qui se traduit par une diminution de la surface de contrôle. Ce dernier point démontre l'intérêt de l'approche du compilateur SYCO, et de son modèle fonctionnel et architectural à plusieurs niveaux d'interprétation.

A partir des remarques précédentes, nous définirons que dans le cadre d'architectures compilables, il est nécessaire d'utiliser des formes régulières pour implémenter une partie contrôle. D'autre part, la perte en surface par rapport à l'emploi d'une logique anarchique, n'est pas comparable au gain de temps enregistrable sur le temps de conception dans le cas ou on choisit une partie contrôle employant des structures régulières.

Aussi, la structure à logique anarchique apparait inintéressante pour définir une architecture compilable de partie contrôle. En effet, son emploi systématique dans un compilateur de partie contrôle nécessiterait une structuration de cette logique, avec ainsi une perte de surface par rapport à la solution manuelle (cf § II 2.2).

Les architectures de partie contrôle utilisant des structures régulières sont au contraire toutes compilables. Il est nécessaire néanmoins de définir pour chacunes d'elles (multi-PLA, générateur de temps...) des modèles topologiques simplifiant la compilation automatique. Ces modèles topologiques permettent, outre le fait d'en simplifier la compilation, la possibilité d'en estimer la surface aisément, et ainsi de pouvoir comparer, pour un algorithme donné, la meilleure implémentation.

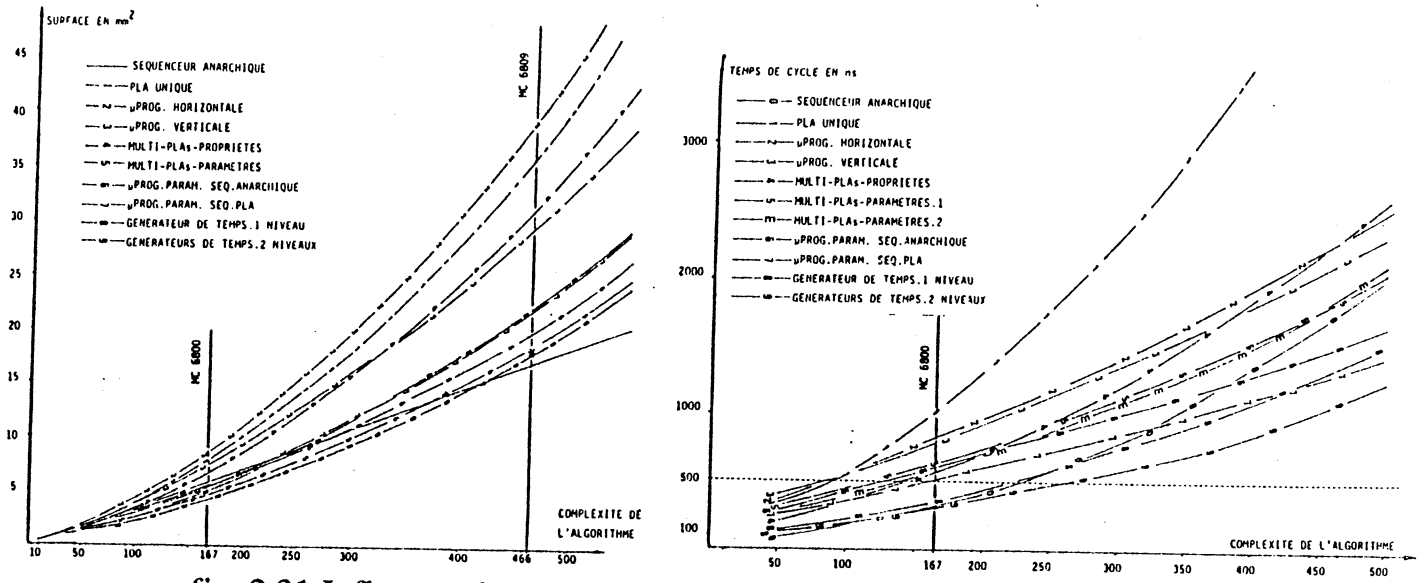


fig. 2.31 Influence des types d'implantation de partie contrôle sur les temps de cycles et la surface occupée [OBRE 82]

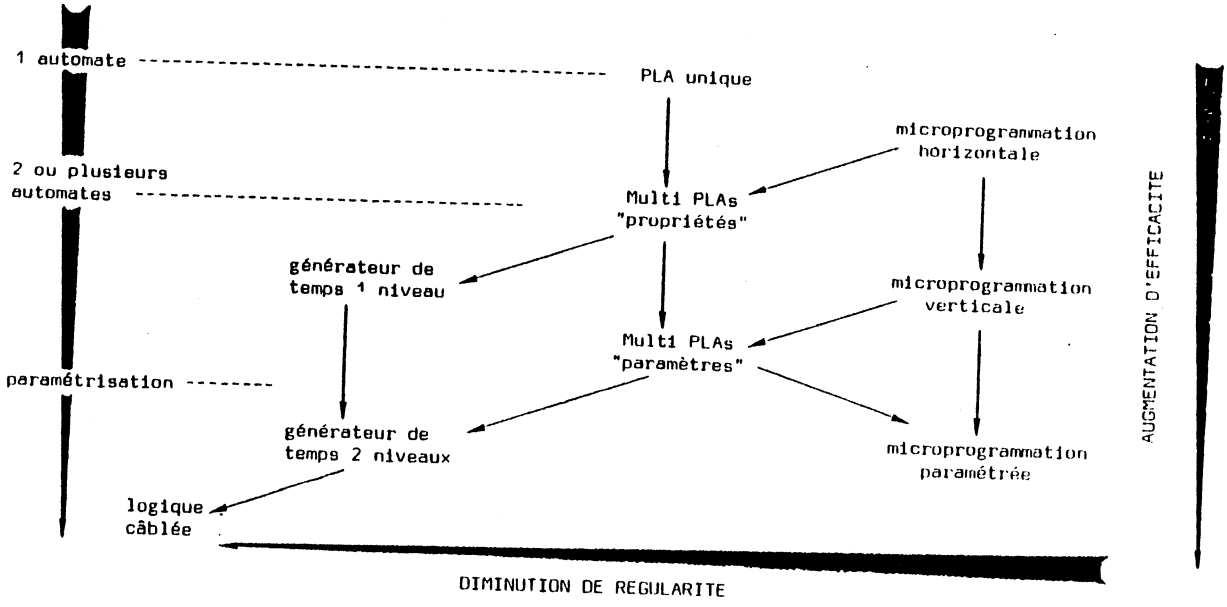


fig. 2.32 Evolution des styles de conception

3 ETUDE DE DIFFERENTS COMPILATEURS DE PARTIES CONTROLES EXISTANTS

En majorité, les compilateurs de parties contrôles existants s'appuient sur des structures régulières comme les ROM, RAM, PLA, structure WEINBERGER...

Le compilateur de partie contrôle part généralement d'une description d'entrée proche de la structure à générer, par exemple où le séquencement et les commandes sont explicitement déclarés, et à partir de cette description fonctionnelle, il construit la structure cible. Le choix de l'implémentation entre différentes structures cibles est fait entre les deux voies extrèmes qui sont la logique anarchique et la microprogrammation [MALA 85b]. Le choix d'une structure microprogrammée fut la première fois proposée par WILKES. Les avantages d'une telle implémentation sont:

- la régularité opposée à l'irrégularité de la logique anarchique,
- l'écriture des microprogrammes est directement traduisible en terme de matériels puisqu'implantable dans une ROM,
- la structure d'accueil permet de modifier le microprogramme sans changer l'implémentation (à complexité de microprogramme égale),
- une structure à microprogramme est plus facilement testable, qu'une structure à logique anarchique.

Les solutions basées sur une structure à PLA représentent une alternative aux deux extrèmes présentés ci-dessus. Le grand avantage des PLA's réside dans leurs modularité ainsi que dans les multiples optimisations logiques et topologiques possibles dont la littérature est abondante. On trouve la plupart des références sur ce domaine dans [CHUQ 84].

La majorité des compilateurs de silicium, pour la partie contrôle, sont développés autour d'un solution à un ou plusieurs PLA's. Certains de ces compilateurs (CATHEDRAL II [DeMa 86], [FLAM 84],...) possèdent des optimiseurs logiques et topologiques.

L'étude suivante présente un type de compilateur par type de structure de partie contrôle. Pour chaque compilateur, cette étude fait ressortir les points importants caractéristiques. Ces point sont d'ordre topologique, logique et structurel. Par exemple les compilateurs tels CATHEDRAL II et celui travaillant à partir de la structure CASSIOPE possèdent chacun une topologie cible, et permettent d'obtenir de meilleurs résultats de compilation du point de vue génération du layout.

Le compilateur MACSIM développé au Laboratoire d'Architecture des Ordinateurs INPG/TIM3 [SCHO 85], part d'un texte source IRENE [MARI 86] (Langage de niveau RTL) et construit une structure à générateur de temps à l'aide de compteurs, et de deux PLA's: un PLA de séquencement et un PLA de génération des commandes. Ce compilateur est destiné à utiliser l'outil PAOLA pour générer

les PLAs. Ce compilateur ne possède pas d'optimiseur logique pour l'instant. Un exemple de réalisation avec ce compilateur est le circuit 80C48 de chez INTEL (refonte en CMOS du 8048 conçu en NMOS) réalisé par F. BERTRAND [BERT 86] et M. SABHATOU [SABH 84]. L'inconvénient majeur de MACSIM provient du fait qu'il n'utilise pas de modèles topologiques cibles. En effet, si les PLA's sont réalisés à l'aide de l'outil PAOLA, le manque de topologie cible entraîne une perte importante de surface au niveau des interconnexions entre les éléments constitutifs de la partie contrôle générée automatiquement.

Un autre type de compilateur de partie contrôle, développé par [PaCo 85], à partir d'un langage comportemental construit une structure de donnée représentant les états de la machine. Une analyse de cette structure produit un code optimisable puis assimilable par un générateur de PLA. Dans l'optimisation du code, il est possible de procéder à un partitionnement aboutissant à une structure à multi-PLA. Autour de la structure, et avant la génération du layout des PLAs, il peut être procédé à une simulation de la structure obtenue. Les PLA's sont générés de manière classique, ce qui provoque une perte importante de surface de routage entre les différents éléments constitutifs de la partie contrôle.

Un autre compilateur générant une structure à un PLA a été réalisé au CNET [FLAM 84], à partir d'une base de donnée CASSIOPEE [BeHe 82]. Il utilise pour générer le PLA un générateur propre à CASSIOPEE sans optimiseur topologique. L'ensemble du processus est automatique. Le résultat topologique est optimisé globalement. En effet le placement des registres d'états est fonction de l'emplacement des sorties du PLA. Le schéma cible de ce compilateur est de réaliser des automates de contrôle pour des processeurs de traitement du signal. L'évolution des implantations de partie contrôle se fera vers des solutions à base de ROM (figure 2.33).

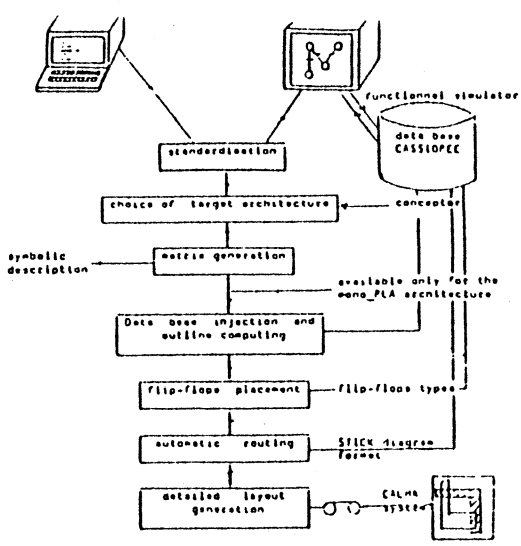


fig. 2.33 Compilation à partir du système CASSIOPEE

Dans le cadre des compilateurs de partie contrôle à un PLA, on peut citer un générateur produit par De MICHELLI [DEMi 84], qui génère une structure optimisée logiquement. Il utilise un générateur de PLA permettant de surcroît une optimisation topologique. Par contre si le bloc est optimisé localement, la non prise en compte de l'environnement implique des pertes de surface par routage avec les autres parties du circuit.

Parmis les compilateurs de silicium dont la structure cible est architecturée autour d'une ROM, on peut citer le compilateur CATHEDRAL II [DeMa 86], destiné à la compilation de processeurs dédiés au traitement du signal. Ce compilateur utilise une structure de contrôle à ROM additionnée d'une mémoire contenant les adresses de branchement conditionnels et d'un multiplexeur (figure 2.34). En outre, il y a définition d'une topologie cible, redéfinissable par programme. Celle-ci permet d'adpater les blocs constituant le contrôle au bloc constituant la partie opérative.

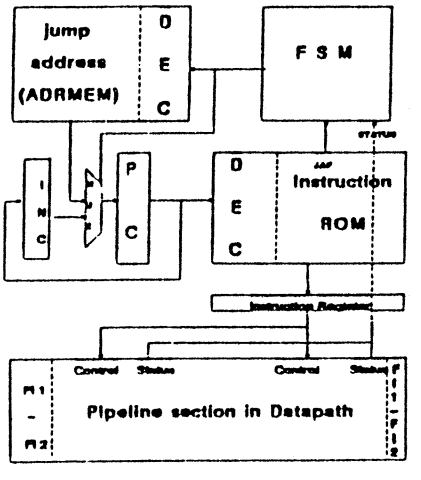


fig. 2.34 Architecture de partie contrôle générée par le compilateur CATHEDRAL II

L'intérêt d'un tel compilateur provient de sa structure cible (core microprocesseur), additionnée d'une topologie déterminée mais évolutive suivant les conditions posées par le circuit à implanter.

Le compilateur SYCO tient compte des remarques précédentes dans son processus de compilation. Il possède une topologie globale cible. Il est organisé autour d'une structure de donnée unique LDS. Il possède un extracteur de partie contrôle, ainsi qu'un générateur de layout de partie contrôle à structure monoPLA.

Son évolution actuelle est destinée à intégrer un optimiseur logique et topologique par modification du nombre de niveaux d'interprétation de la partie contrôle.

4 CONCLUSION

L'architecture de circuit fait appel à de nombreux domaines : automatisme, circuiterie, technologie des composants.... Bien que dans l'étude précédente, l'on se limite à des circuits de type microprocesseur, les architectures-cibles définissables sont très nombreuses. C'est pourquoi, un choix préalable à la réalisation d'un compilateur optimisé a du être fait. L'architecture cible choisie est une architecture à plusieurs niveaux d'interprétation. Ces niveaux correspondent en fait a une décomposition algorithmique d'un problème (interprétation d'un jeu d'instructions). La structure correspond donc à une structure analogue au fameux "Divide and Conquer" [KNUT 79].

Les différents niveaux d'interprétation sont réalisés à l'aide de n automates en série (cf §II 2.1). Chacun des automates constituant ce réseau peut être implantée à l'aide d'une structure particulière. C'est autour des différents choix de structures compilables que la discussion a été menée. Il ressort de cette discussion les points suivants:

Choix d'une structure cible

Le choix pour telle ou telle structure est lié à la forme et à la complexité de l'algorithme à implanter. En outre, suivant la surface disponible pour implanter un algorithme, le choix d'une structure évoluera. Il apparait que plus une structure est régulière, plus la surface nécessaire à son implantation sera importante. D'autre part, si l'on choisit une structure de partie contrôle à plusieurs niveaux d'interprétation, la surface nécessaire pour implanter cette partie contrôle sera plus faible que si l'on choisit une partie contrôle à un seul niveau ([OBRE 82] [ANCE 86]...).

Performance d'une structure

La performance, en terme de vitesse d'exécution pour interpréter une instruction, est souvent prédominante dans le choix d'une structure préférentiellement à une autre. Ce facteur intervient dans le cas de processeurs destinés au traitement du signal [DeMa 86], ou dans le cas de coprocesseurs arithmétiques [ZYSM 87]. Il apparait dans ce cas, que plus une structure est régulière, plus sa performance décroit.

Compilabilité d'une structure

Une structure est d'autant plus facilement compilable que sa régularité est importante. En effet, plus le nombre de blocs constituant une structure est important, plus il est difficile d'en générer la description et/ou l'implantation automatique. Par exemple, les informations nécessaires à

générer une structure MonoPLA sont plus aisées à déduire de l'algorithme d'interprétation, et à fournir par le compilateur pour l'implantation de la partie contrôle, que les informations nécessaires pour décrire et générer une partie contrôle à PLA multiples.

En déduction des points précédents, il apparait que la génération automatique d'une partie contrôle en logique anarchique est difficile à mettre en oeuvre dans le cadre de l'étude menée sur le compilateur de silicium SYCO. La régularité d'une telle solution est quasi nulle, par contre, sa forme logique est facile à extraire de l'algorithme d'interprétation (Cette solution reste viable dans le cas de génération d'une partie contrôle avec un style de conception de type circuit précaractérisé).

Les autres structures de parties contrôles présentées au long de ce chapitre sont par contre implémentables automatiquement, et il existe une forme topologique pour chacunes d'elles dans le cadre du compilateur SYCO (cf chapitre III). On peut dire que la complexité de leur extraction logico-fonctionnelle suit l'ordre suivant:

1) La solution MonoPLA ou à ROM microprogrammée horizontalement,
2) La solution à PLA's multiples et la solution à ROM microprogrammée verticalement,
3) Les solutions utilisant un générateur de temps.

Toutes ces solutions font par ailleurs l'étude de recherches pour leur extraction et leur génération automatiques [SCHO 85] [FLAM 84] [PaCo 85]....

CHAPITRE III

COMPILATION DE PARTIES CONTROLES

1 MODELE GENERAL

1.1 Modèle à plusieurs niveaux d'interprétation, modèle fonctionnel

Ainsi que cité dans le chapitre I, dans le cadre du compilateur de silicium **SYCO**, le modèle de compilation de la partie contrôle choisi est un modèle à plusieurs niveaux d'interprétation. Le choix de ce modèle permet de produire une architecture de partie contrôle à plusieurs niveaux d'automates en série (cf §II 2). La simplicité de ce modèle permet de traduire rapidement une description algorithmique de haut niveau, en une description structurelle puis topologique du circuit.

Le choix d'un tel modèle fonctionnel impose de répertorier tous les types d'informations véhiculées entre les différents niveaux de partie contrôle, ainsi qu'entre les plots, la partie opérative et la partie contrôle globale. Ces informations sont de type :

- Contrôle: informations d'échanges entre les différents étages de partie contrôle, ou avec l'extérieur (plots...).

- Comptes-rendus: informations validées en résultat d'une opération effectuée par la partie opérative. Ces informations sont utilisées par tous les niveaux de partie contrôle.

- Synchronisation: le modèle temporel étant de type synchrone, toutes les parties contrôles sont rythmées par la même horloge, et se synchronisent entre elles à l'aide d'informations de contrôle.

1.2 Architecture-cible

Le choix d'une architecture-cible permet de simplifier la traduction d'une description algorithmique jusqu'à la réalisation des masques. Ce choix permet aussi au concepteur de parfaire sa description et de comparer l'évolution de sa réalisation à chaque retour arrière.

Le modèle fonctionnel fournit en lui-même une solution architecturale pour l'implémentation des niveaux de partie contrôle. Cette solution consiste à empiler, au moment de la réalisation, les divers étages de contrôle en prenant la partie opérative comme base.

Une telle solution implique des choix architecturaux permettant de ne pas souffrir de l'effet "pyramide" introduit par le modèle fonctionnel procédural. En fait, la

description fonctionnelle est telle que le niveau le plus haut se retrouve être le moins important en code. Traduit en matériel, un tel schéma produit un étage de partie contrôle de faible surface. Pour pallier à cet état de fait, des notions topologiques telles que déformabilité et transparence de bloc sont utilisées [ANCE 83].

- La déformabilité d'un bloc consiste à l'adapter à son environnement. Dans notre cas, les blocs de parties contrôles ont leurs bases (largeurs) adaptées à celle de la partie opérative, de telle manière que leurs surfaces soient si possible constantes. Cela implique une diminution de la hauteur de chaque étage de partie contrôle, et ainsi un circuit de forme plus carrée (une forme plus carrée du circuit implique de garantir plus facilement la notion de zône isochrone [MARQ 79]). Une forme plus carrée implique aussi une optimisation de la surface du circuit une fois le nombre de plots connus. En effet, de tous les parallélogrammes de même périmètre, le carré est celui qui possède la surface la plus importante (=> une surface plus importante pour la circuiterie interne).

- L'ordonnancement des connecteurs consiste à adapter la connectique de chaque bloc à la connectique des blocs contigüs. Cela s'exprime par la connexion des blocs si possible par aboutement des connecteurs entre eux.

- La transparence consiste à permettre le passage d'une ou plusieurs lignes d'informations à travers un bloc, sans modification de sa fonctionnalité.

Ces solutions topologiques se traduisent par une architecture de circuit où les bus d'informations constituent la trame du circuit sur laquelle les parties contrôles sont construites. Cette forme d'implantation se rapproche de celle proposée par J.P. Schoellkopf [SCHO 85], qui proposait de construire les parties contrôle sur la trame constituée par les comptes rendus de la partie opérative (voir chapitre II 1.1.3.2).

Dans le cadre de la première version de **SYCO**, la transparence de bloc n'est pas utilisée, aussi les lignes d'informations sont constituées en bus, répertoriés par leur fonctionnalités. Les bus ainsi formés sont:

- Le bus de contrôle, transitant les informations de contrôles destinées aux plots de contrôles, ou aux variables internes utilisées par les étages de partie contrôle (paramêtres de procédures en LDS).
- Le bus de compte-rendus, transitant les informations résultats d'opérations effectuées par la partie opérative. Ces informations sont utilisées par les étages de partie contrôle.
- Les informations de synchronisation, générées par les cellules de synchronisation réparties à chaque niveau de partie contrôle (cellules formant l'automate général des temps du circuit), et contenant aussi les horloges générales du circuit ainsi que le RESET matériel.

Le modèle architectural est donc représenté par le schéma suivant (figure 3.1):

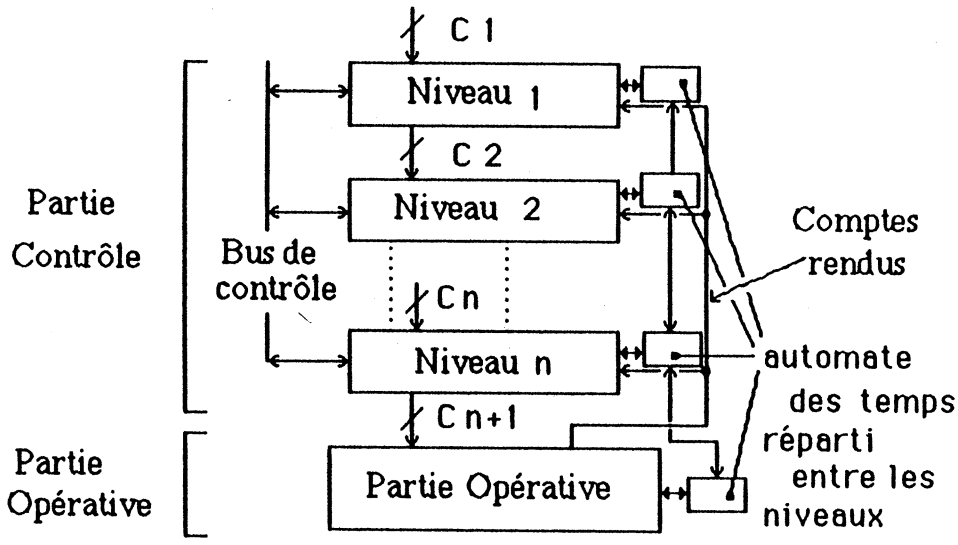


fig. 3.1 Modèle architectural général

1.3 Etude d'un modèle topologique

Le modèle topologique cible implante les différentes tranches de contrôle de la manière suivante (cf §II 2.1):

- La partie opérative forme la base du circuit,
- Les différentes tranches de contrôle sont superposées les unes au dessus des autres, en partant de la tranche "n" de partie contrôle, tranche directement supérieure à la partie opérative, jusqu'à la tranche "1" de partie contrôle.
- Les variables de contrôle internes, variables utilisées pour mémoriser les paramêtres de procédures (paramêtres de branchement, ou de retour de branchement), ou pour échanger des informations de contrôle entre les différentes tranches de contrôle, sont implantées (telles un banc de registre à un bit) au dessus du bloc formé par l'assemblage des tranches de partie contrôle et de la partie opérative.

Dans une première étude, la notion de transparence à travers les blocs (partie contrôle ou partie opérative) n'a pas été retenue. Cette notion n'a pas été utilisée pour raison de limitation des outils utilisés pour l'assemblage des blocs du circuit.

En tenant compte de cette remarque:

- Les fils de contrôle (véhiculant l'information des variables de contrôle internes et externes) sont regroupés en un bus de contrôle implanté à gauche (ouest) des tranches de contrôle composant le circuit.
- Les fils de compte rendu provenant de la partie opérative, et véhiculant les informations relatives à une exécution d'une ou plusieurs opérations de la

partie opérative, sont regroupés en un bus de compte rendu, et implantés à droite (est) des tranches de contrôle.

- Les fils de synchronisation provenant des différentes cellules de synchronisation, cellules réparties au niveau de chaque tranche de contrôle, sont regroupés en un bus de synchronisation implanté à droite (est) des tranches de contrôle entre la partie contrôle générale et le bus de compte rendu.

Autour du coeur du circuit ainsi formé, les plots de contrôles, les plots d'horloges, les plots de données et d'adresses sont assemblés. Les plots occupent ainsi le pourtour du circuit. La liaison entre les tranches de contrôle et les variables de contrôles internes et externes se déroule à travers le bus de contrôle. Aussi le bus de contrôle occupe une partie de la périphérie du coeur du circuit (cf figure 3.2).

Nous obtenons le schéma topologique suivant (figure 3.2):

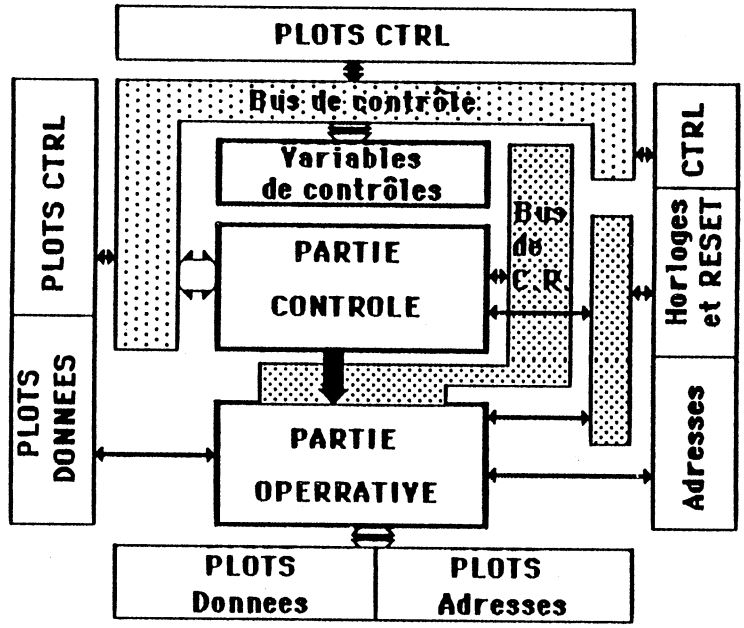


fig. 3.2 Modèle topologique général

Les tranches de partie contrôle constituant la partie contrôle globale respectent un schéma d'implantation (figure 3.3) leur permettant de recueillir et de fournir les informations nécessaire à l'exécution de l'automate de contrôle.

Les informations d'entrée sont:

- Les informations de contrôle, les informations de compte rendu de la partie opérative, les informations de commande du niveau supérieur, et les

informations de synchronisation.
- Les information de contrôle proviennent du bus de contrôle à gauche. Elles sont alors, suivant le type de partie contrôle (cf §III 3), introduites par le haut de la tranche de contrôle, ou par le côté gauche. L'introduction par le côté gauche diminue la zône de routage (pas de coude pour apporter les informations de contrôle).
- Les informations de compte rendu (C.R.) proviennent du bus de compte rendu à droite de la tranche de contrôle. Ces informations sont introduites dans la tranche de contrôle par le haut ou la droite de la tranche.
- Les informations de synchronisation provenant du bus de synchronisation, sont introduites dans la cellule de synchronisation, cellule disposée en haut à droite de la tranche de contrôle. Ces informations sont introduites soit à droite soit en haut de la cellule de synchronisation.
- Les commandes provenant du niveau de contrôle supérieur (du point de vue topologique) sont introduites par le nord de la tranche de contrôle.

Dans le cas des informations de sortie:

- Les informations de sortie regroupent les informations de contrôle, les informations de synchronisation, et les informations de commandes pour l'automate de contôle de niveau inférieur.
- Les informations de contrôle sont dirigées vers le bus de contrôle à gauche, et sortent de la tranche de contrôle par le bas ou la gauche de cette tranche.
- Les informations de synchronisation sont dirigées vers le bus de synchronistaion à droite de la tranche de contrôle, et sortent à droite ou en haut de la cellule de synchronisation.
- Les informations de commandes sont dirigées vers la tranche de contrôle directement inférieure, et sortent par le bas (ou sud) de la tranche de contrôle.

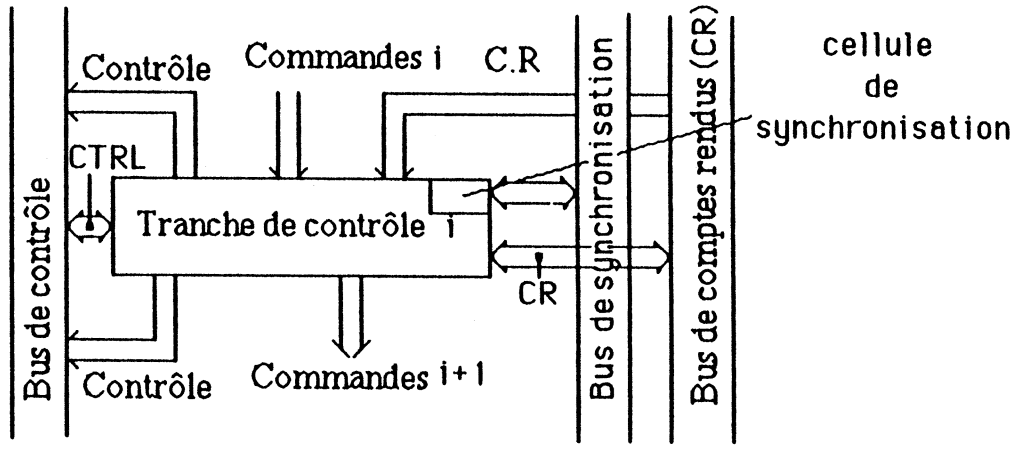


fig. 3.3 Modèle topologique d'une tranche de partie contrôle

Chaque tranche de contrôle peut utiliser une structure cible différente (cf § II), si

et seulement si son implantation respecte le schéma topologique ci-dessus. Le choix de la structure cible et de son implantation pour une tranche de partie contrôle reste à l'appréciation du compilateur de silicium, qui choisit parmi les différents types qui vont être décrits ci-après. Les critères de choix du compilateur entre les différentes structures cibles sont établis en fonction :

- de la forme de l'organigramme de l'algorithme à implanter,
- de la performance désirée pour le circuit,
- de la surface finale de la tranche de partie contrôle à réaliser.

Les critères précédents sont évalués à partir de la forme compilée de la description fonctionnelle du circuit [BEKK 86].

Dans le cadre actuel de SYCO, seules les tranches de partie contrôle de type MonoPLA à l'aide de PLA classique sont réalisées et implémentées (cf §III 5.2).

2 ETUDE DE MODELES TEMPORELS GENERAUX

2.1 Modèle global

Le modèle temporel proposé est destiné à synchroniser les différentes parties contrôles, ainsi que la partie opérative. Le circuit est considéré comme une zône isochrone, c'est-à-dire que sur la surface active considérée, un signal quelconque ne subit pas entre son origine et son utilisation de déphasage, susceptible de produire un état aléatoire dans le cadre d'un automate synchrone [MARQ 79].

2.1.1 Horloge générale du circuit:

L'horloge générale du circuit correspond à l'horloge de base de la partie opérative. La partie opérative est synchronisée sur une horloge à quatre phases non recouvrantes. Ces quatres phases sont produites à partir d'une horloge à deux phases non recouvrantes [GALL 84] [JaBe 86]. Un cycle de base correspond à l'exécution d'un transfert au niveau de la partie opérative (figure 3.4). Une opération est effectuée par la partie opérative en au moins deux cycles.

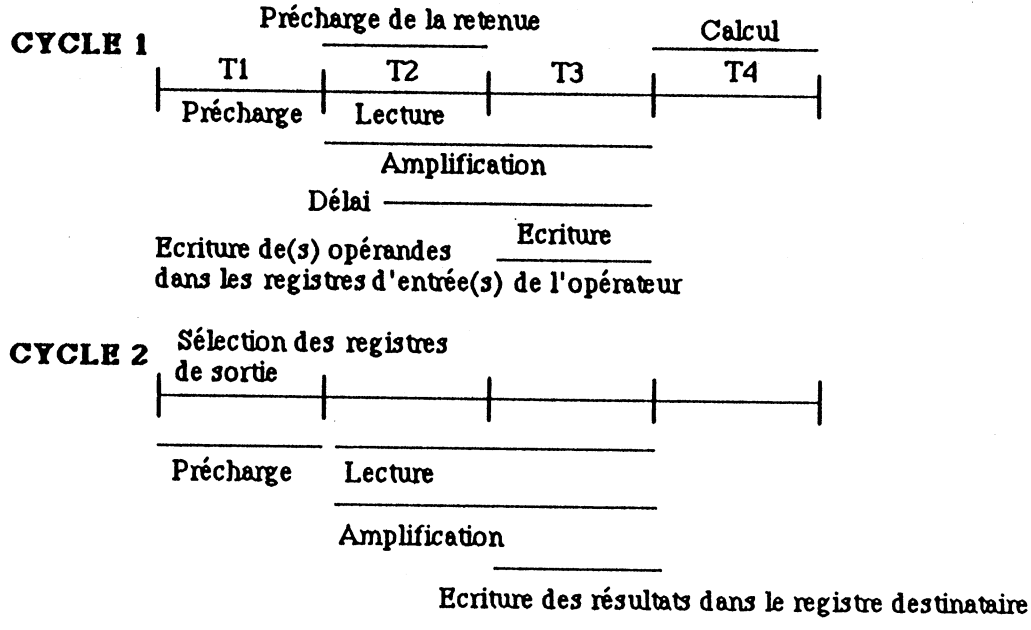


fig. 3.4 Horloge de base du modèle temporel

Les modèles temporels présentés sont de type statique. Toutes les informations transitant entre chaque tranche de partie contrôle, ou vers la partie opérative sont, à un instant "t" donné (correspondant à une phase de l'horloge générale du circuit), mémorisées, ceci évitant toutes créations d'aléas dans les automates de partie contrôle.

2.1.2 Modèle temporel procédural:

Définition :

On appelle modèle temporel procédural, un modèle temporel permettant de synchroniser les différentes tranches de contrôle du circuit entre elles, et dont, à un instant donné (durant un cycle d'horloge, un cycle correspondant à quatres phases non recouvrantes), seule une tranche est active. L'exécution de l'algorithme d'interprétation à l'aide de ce modèle temporel est donc purement séquentielle.

Cette solution a été choisie pour permettre de valider le compilateur de silicium **SYCO**. Ce modèle, généralisé à n tranches de contrôle, est proche du modèle temporel utilisé pour synchroniser la partie contrôle à deux niveaux d'interprétation du microprocesseur Z80 de ZILOG.

Informations affectant la synchronisation du modèle temporel:

- Les informations affectant la synchronisation du modèle temporel sont composées:

  - Des informations de commande provenant d'un niveau i de partie contrôle, et destinées à un niveau i+1 de partie contrôle. Ces informations doivent être validées uniquement si il y a eu modification. Il ne doit pas y avoir à chaque cycle de l'horloge de base une remémorisation des informations sortant de la partie contrôle de niveau i s'il n'y pas eu de progression dans son graphe d'état.

  - Des informations provenant d'un résultat d'une opération effectuée par la partie opérative, opération demandée par un niveau i de partie contrôle, et provoquant une modification dans l'exécution séquentielle de l'algorithme à ce niveau i de partie contrôle. Ces informations correspondent aux informations transitant par le bus de comptes rendus. Ces informations sont considérées comme valides soit à partir de deux cycles de l'horloge de base du circuit (premier modèle temporel procédural), soit à partir du premier cycle d'une opération (second modèle temporel procédural).

La synchronisation d'un tel modèle, nécessite de disposer de deux signaux de synchronisation:

* Synchronisation descendante :Le premier signal permet valider les données calculées par un étage de niveau i, pour l'étage de niveau i+1.

* Synchronisation montante: Le deuxième signal prévient les étages supérieurs à l'étage considéré que l'exécution de la séquence en cours est terminée. Il est alors possible aux étages supérieurs de calculer une nouvelle donnée pour cet étage. Ce signal est équivalent à un signal (ou instruction) signalant une fin de procédure dans un langage évolué (FORTRAN).

2.1.2.1 Description du premier modèle temporel procédural

Dans le premier modèle temporel procédural, une opération est effectuée par la partie opérative en **deux** cycles. Les comptes rendus de la partie opérative sont validés à la fin du second cycle de l'opération. Aussi, pour l'exploitation des comptes rendus par les tranches de partie contrôle supérieures, un cycle d'horloge est perdu (un cycle correspond à quatre phases). Durant ce cycle perdu, l'information concernant les comptes rendus est mémorisée dans un registre interne de la partie opérative [JAMI 86].

Influence du modèle temporel de la partie opérative sur la partie contrôle:

Si l'on considère la partie contrôle "n" directement supérieure à la partie opérative (dénommée partie contrôle de génération des étapes dans la syntaxe APOLLON [JAMI 86]), la séquence minimale est de trois cycles. En effet, si les deux premiers cycles correspondent aux commandes pour effectuer l'opération proprement dite, le dernier cycle est un cycle d'attente. Ce cycle d'attente, ou aucune opération n'est effectuée, correspond à l'attente de validation des comptes rendus de la partie opérative. Sur ce cycle d'attente, le signal de synchronisation montante est validé (START i validé sur H4..H2). Le changement d'état de l'automate de niveau n est fonction du signal $INVALID_{n-1}$ en cas de début de séquence, et dans le cas ou l'automate se trouve en milieu de séquence, alors le séquencement est automatique durant trois cycles (durée d'une séquence dans le cas de l'automate de niveau n).

Dans le cas des tranches de partie contrôle de 2 à "n-1", une partie contrôle "i" n'effectue un calcul (génération des commandes pour le niveau inférieur) que si le signal de synchronisation montante du niveau "i+1" est valide, ou que le signal de synchronisation descendante (INVALID i-1) est valide. Le choix entre les deux signaux de synchronisation est fonction de l'état dans lequel l'automate se trouve:

- Si l'automate est en attente de commande du niveau supérieur, c'est à dire qu'il n'a pas débuté une séquence, alors le signal de synchronisation INVALID i-1 est sélectionné (exemple états 1,2 et 3 de la partie contrôle 2 de la figure 3.5).
- Si l'automate est en attente, mais a entamé une séquence, alors le signal START i+1 est choisi (exemple 2' de la partie contrôle 2 de la figure 3.5).

Dans le cas de la partie contrôle de niveau 1, étant donné qu'il n'existe pas de partie contrôle supérieure, un changement d'état de son automate (transition) est fonction du signal de synchronisation montante $START_2$, ou du RESET en cas d'initialisation du circuit (exemple état 1,2 et 3 de la partie contrôle 1 de la figure 3.5).

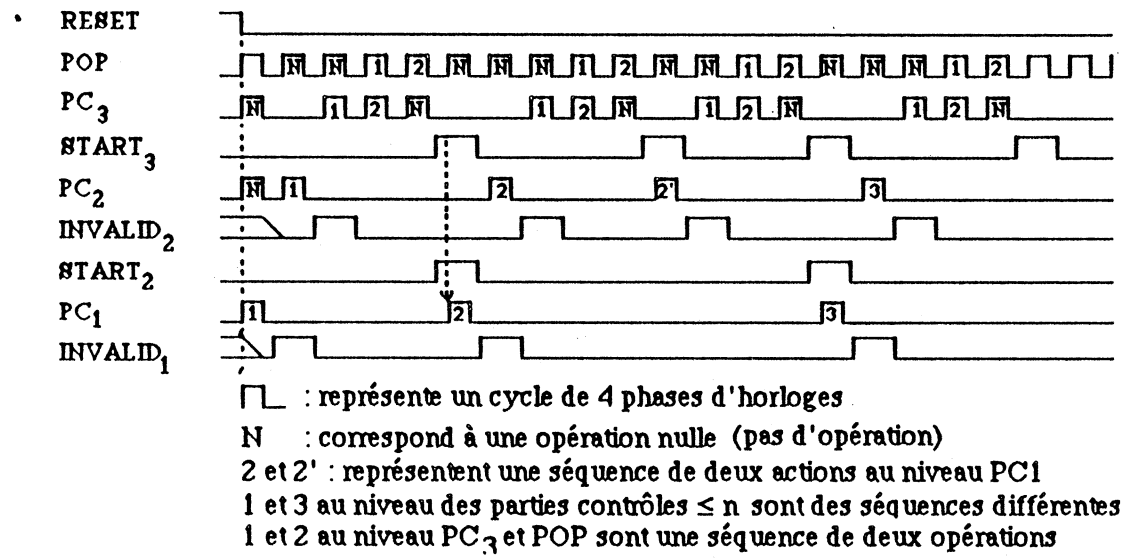


fig. 3.5 Modèle temporel procédural

Il apparait qu'il existe trois types de cellules de synchronisation:

* Une cellule de synchronisation destinée à l'étage de partie contrôle 1. Cette cellule génère les deux horloges de commandes des mémoristation d'entrée et de sortie de l'étage de contrôle1 (respectivement $PCH_{1\_entrée}$ et PCH_{1_sortie}), ainsi que le signal de synchronisation descendante $INVALID_1$. D'autre part cette cellule ne valide les horloges de mémorisation que sur RESET ou $START_2$ comme précisé plus haut.

* Une cellule de synchronisation destinée à l'étage PC_n (partie contrôle directement supérieure à la partie opérative). Cette cellule génère les deux horloges de mémorisation ($PCH_{n\_entrée}$ et PCH_{n_sortie}), ainsi que le signal $START_n$, signal fonction de la validation de la fin d'une séquence de trois cycles consécutifs de cet étage.

* Une cellule destinée aux étages de contrôle i, i variant de 2 à n-1. Cette cellule génère les deux horloges de mémorisation, ainsi que les signaux $START_i$ et $INVALID_i$. Le signal $START_i$ est fonction de l'indication de fin de séquence de cet étage i et du $START_{i+1}$ du niveau inférieur. Le signal $INVALID_i$ est fonction de la validation des sorties de l'étage i, du $START_{i+1}$ et du $INVALID_{i-1}$.

Pour définir ces trois cellules de synchronisation, on en décrit le graphe d'état permettant de générer les différents signaux de synchronisation. Pour simplifier la description, il a été introduit un automate "to_i" qui évolue sur deux états "to_i1" et

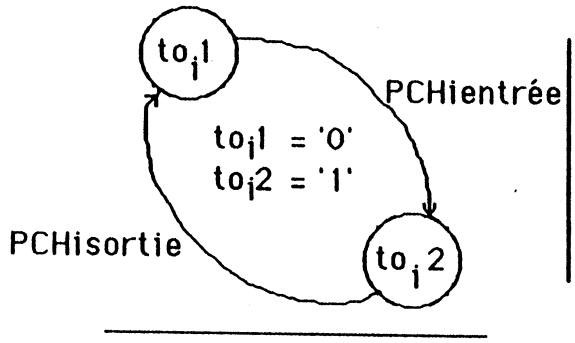
"to_i2" en fonction des horloges de mémorisation $PCH_{i\_entrée}$ et PCH_{i_sortie}. Cet automate symbolise le fonctionnement de l'étage de niveau i, l'état "to_i1" correspondant à l'état inactif de l'étage de niveau i, et l'état "to_i2" correspondant à l'état actif de l'étage de niveau i (figure 3.6).

- Cas ou l'on a une partie contrôle de niveau 1, alors:

$$PCH_{1\_entrée} = (\text{NON(RESET) OU } START_2) \text{ ET } H1$$

- Cas ou l'on a une partie contrôle de niveau n, alors:

$$PCH_{n\_entrée} = (INVALID_{n-1} \text{ OU NON}(fin\_de\_séquence_n)) \text{ ET } H1$$



Pour i=1

$PCH_{i\_entrée} := H1.(\overline{RESET} + (START_{i+1}))$

$PCH_{i\_sortie} := H4 * toi_2$

Pour 1<i<n

$PCH_{i\_entrée} := H1.(INVALID_{i-1} + (START_{i+1} * \overline{findeséquence_i}))$

$PCH_{i\_sortie} := H4 * toi_2$

Pour i=n

$PCH_{i\_entrée} := H1.(INVALID_{i-1} + \overline{findeséquence_i})$

$PCH_{i\_sortie} := H4 * toi_2$

fig.3.6 automate "to_i"

Le signal $INVALID_i$ est fonction de l'état de l'automate "to_i", et des phases H3 et H2 de l'horloge de base. Ce signal est mis à "1" logique pour signaler à l'étage inférieur que les données sont valides (en sortie de l'automate de partie contrôle), donc quand l'automate to_i se trouve dans l'état "to_i2". Il est remis à zéro si l'automate se retrouve dans l'état "to_i1" et que le signal se trouvait précédement à "1". Dans le cas de l'étage de partie contrôle de niveau n, le signal $INVALID_n$ n'existe pas (figure 3.7).

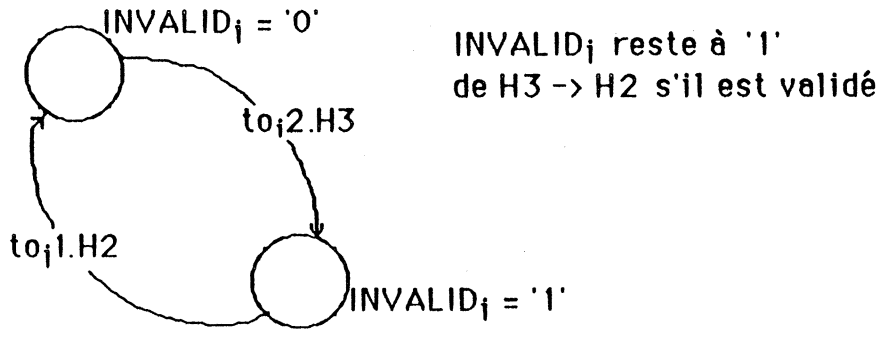


fig.3.7 Automate de génération du signal INVALIDi

Le signal $START_i$ est fonction de l'indication de fin de séquence de l'étage de niveau i, et du signal $START_{i+1}$ de l'étage inférieur. Dans le cas de l'étage n, il est validé sur l'indication de fin de séquence et H4, et dévalidé sur la phase d'horloge H2 suivante. Dans le cas de l'étage de niveau 1, le signal $START_1$ n'existe pas (figure 3.8).

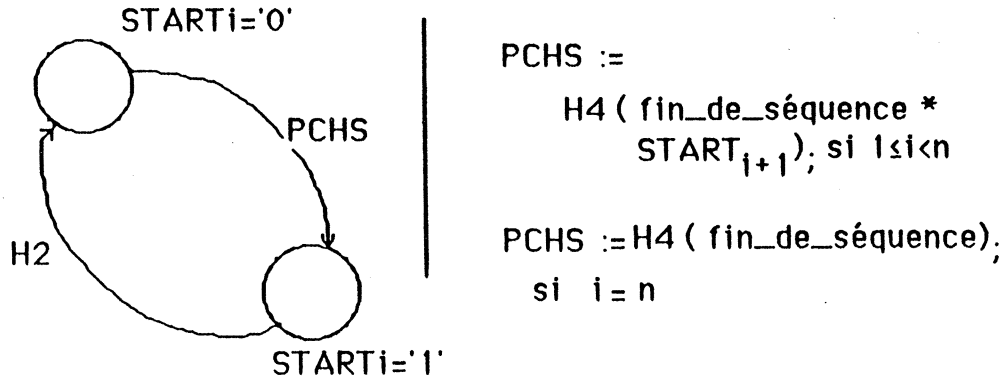


fig. 3.8 Automate de génération du signal $START_i$

On obtient alors les trois cellules de synchronisation suivantes (figure 3.9 a,b,c). La figure 3.9a correspond à la cellule de synchronisation de l'étage de niveau 1, la figure 3.9b correspond à la cellule de synchronisation de l'étage de niveau i, $2 \leq i \leq n-1$, la figure 3.9c correspond à la cellule de synchronisation de l'étage de niveau n.

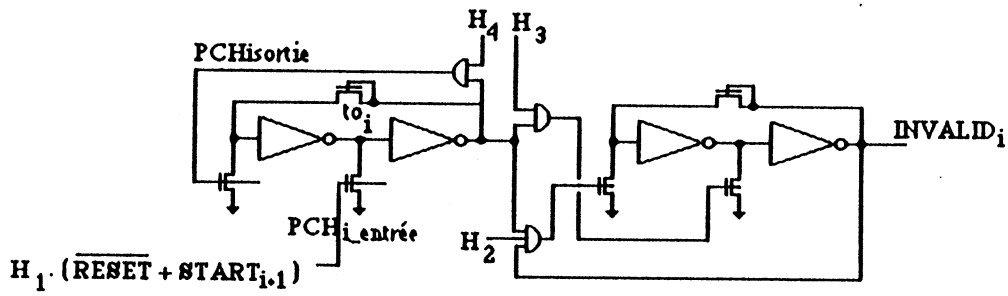


fig. 3.9a Cellule de synchronisation de l'étage de contrôle de niveau 1

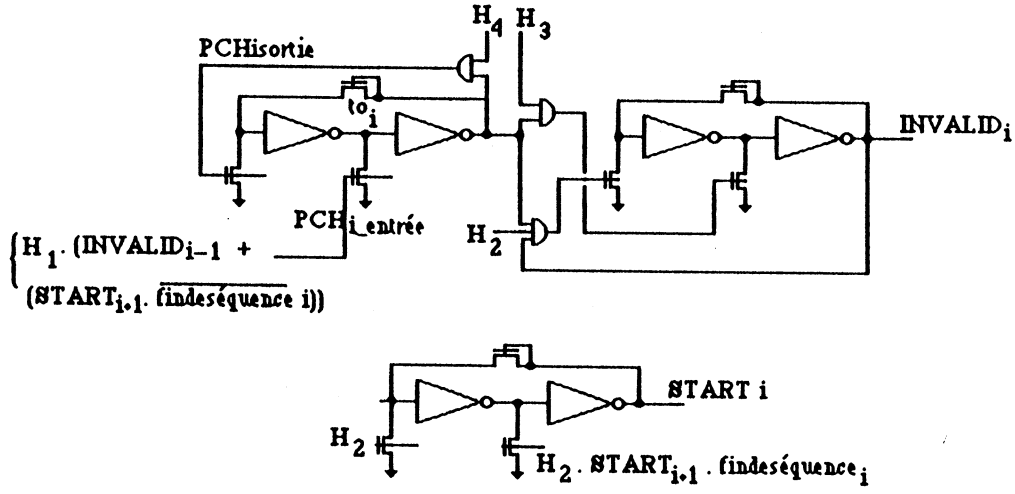


fig. 3.9b Cellule de synchronisation de l'étage de contrôle de niveau i

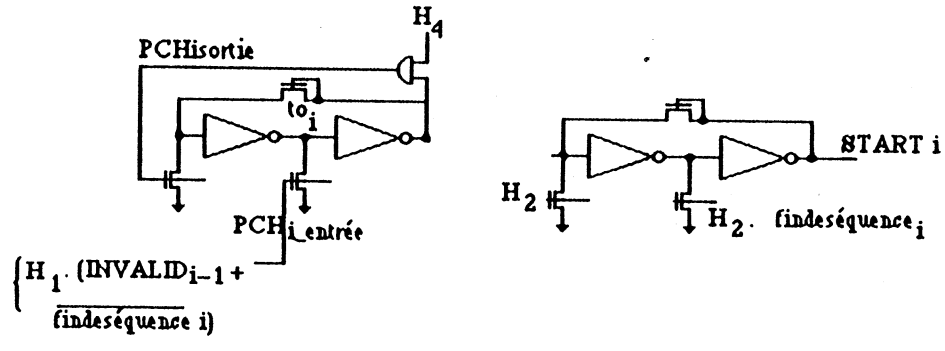


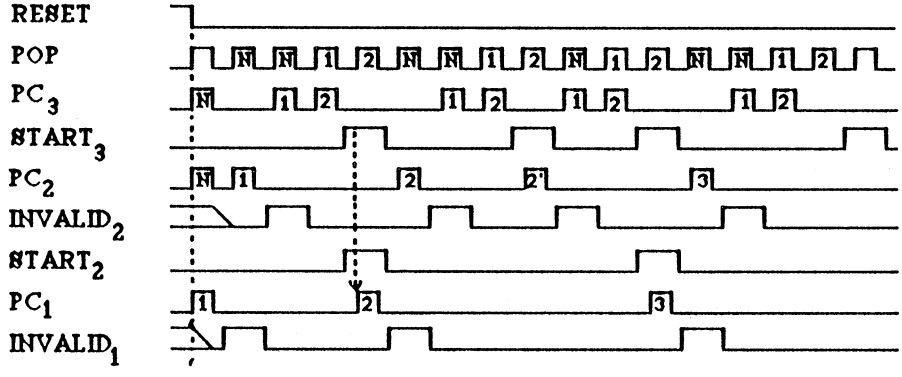
fig. 3.9c Cellule de synchronisation de l'étage de contrôle de niveau n

2.1.2.2 Description du second modèle temporel procédural:

Dans ce second modèle temporel procédural, une opération est effectuée par la partie opérative en un ou plusieurs cycles (un cycle est composé de quatre phases). Les comptes rendus de la partie opérative sont valides sur la première phase suivant la phase de calcul. Cela correspond dans le modèle temporel de la partie opérative, à la phase H1 suivant la phase H4 de calcul (figure 3.4). Il y a donc modification du modèle temporel de la partie opérative (cf §II 2.1.2.1).

L'avantage d'une telle modification provient du gain apporté en nombre de cycle par rapport à la solution précédente: il y a gain de un cycle à chaque opération. D'autre part, il n'y a pas modification du modèle temporel procédural général tel défini au chapitre III 2.1.2.1. Il y a donc conservation des cellules de synchronisation définies plus avant (figure 3.9 a,b,c).

Si l'on compare la figure 3.10 à la figure 3.5, on aperçoit le gain de un cycle effectué à chaque opération. La partie opérative exécute une opération à deux cycles, opération demandée par l'étage de contrôle de niveau 1(état 1), trois cycles plus tard. Il y a ensuite deux cycles d'attente (N) pour une nouvelle opération demandée par l'étage de niveau 1, contrairement au modèle de la figure 3.5, où il y avait trois cycles d'attente.



⊓⊔ : représente un cycle de 4 phases d'horloges
N : correspond à une opération nulle (pas d'opération)
2 et 2' : représentent une séquence de deux actions au niveau PC1
1 et 3 au niveau des parties contrôles ≤ n sont des séquences différentes
1 et 2 au niveau PC_3 et POP sont une séquence de deux opérations

fig. 3.10 Modèle temporel procédural (b)

2.1.3 Modèle temporel avec accélération du traitement des conditions:

Dans le cadre de l'amélioration du modèle temporel procédural, la proposition suivante à été faite: Il est possible de procéder à une exécution de type pipe-line pour les différentes tranches de partie contrôle. Ce "pipe-lining" correspond à faire exécuter plusieurs opérations de contrôle en parallèle. Dans le cas où la partie contrôle ne traite pas de comptes rendus provenant de la partie opérative, alors le traitement d'une instruction est accéléré par rapport à un traitement à l'aide du modèle temporel procédural. Dans le cas ou il est nécessaire de traiter des comptes rendus, le gain est nul par rapport au modèle procédural.

L'inconvénient d'une telle solution provient de la complexité du bloc de synchronisation qu'il est nécessaire d'allouer à chaque tranche de partie contrôle. En effet, dans le cas de traitement par un niveau de partie contrôle des comptes-rendus provenant de la partie opérative, il est nécessaire de bloquer les autres étages de contrôle le temps de valider ces comptes-rendus provenant de l'exécution de l'instruction précédente. L'influence sur le temps d'exécution est alors le même que dans le cas du modèle temporel procédural.

2.2 Limitations

L'emploi du modèle temporel procédural limite le nombre de niveaux de partie contrôle possible, si l'on ne désire pas une chute trop importante des performances du circuit.

Cette limitation est due aux paramêtres suivants:

- Le modèle fonctionnel étant de type procédural, alors durant un cycle de l'horloge de base, seul un étage de partie contrôle est actif. Aussi, dans le cas ou l'exécution d'une instruction par le plus haut niveau de partie contrôle est demandée (niveau 1), alors toutes les tranches de contrôle devant être traversées, le temps de latence avant le traitement de cette instruction par la partie opérative est de n cycles de l'horloge de base. D'autre part, avant l'exécution par la partie opérative d'une nouvelle instruction demandée par ce niveau 1 de partie contrôle, un temps de n-1 cycle de latence est nécessaire (cas du modèle temporel procédural numéro deux).

- Le modèle temporel nécessite de posséder des tranches de contrôle, dont les caractéristiques d'exécution permettent d'effectuer une transition d'un état à un autre en un cycle (cette caractéristique n'est vrai que dans le cas de la partie contrôle n, c'est à dire la partie contrôle la plus proche de la partie opérative, car dans son cas, son temps d'exécution est lié au temps d'exécution de la partie opérative qui est de un cycle).

Avantages de ce modèle temporel:

- Ce modèle temporel est de conception simple, et dans le cas d'un nombre de niveaux de contrôle faible, il s'avère performant (exemple du Z80 qui possède un modèle temporel proche, ou du 6800 évalué par [OBRE 82], où une partie contrôle à deux niveaux d'interprétation à générateur de temps se révélait être plus rapide en temps d'exécution que la partie contrôle réalisée en logique anarchique du modèle du commerce (résultats tirés d'évaluations effectuées par [OBRE 82])).

- Une autre possibilité serait de faire évoluer ce modèle temporel purement procédural vers une exécution de type pipe-line (cf §III 2.1.3), ou de permettre à un niveau quelconque de partie contrôle d'effectuer un traitement direct sur la partie contrôle.

3 ETUDE DE MODELES TOPOLOGIQUES DE PARTIES CONTROLES

Les différents modèles de partie contrôle dont la structure a été définie au chapitre II vont être étudiés sous l'aspect topologique dans le cadre du compilateur de silicium **SYCO**. Une étude de leurs formes d'implantation possibles dans le cadre du modèle topologique général défini ci avant (cf §III 1.2) va être faite. Pour chacun de ces modèles, des équations permettant l'évaluation de leur surface vont être produites.

3.1 Modèle à PLA's multiples

Avant de proposer une solution topologique pour l'implémentation d'une solution à plusieurs PLAs, nous définissons d'abord les formes topologiques possibles pour implanter un PLA, ainsi que les équations de calcul de surface liées à chacunes des formes.

Dans le cadre de la technologie MOS (NMOS ou CMOS), les PLA's peuvent être implantés de trois manières différentes, il est rappelé qu'en technologie MOS, les matrices ET et OU du PLA sont des matrices NOR (NON OU):

3.1.1 PLA de forme classique

Les entrées/sorties sont réparties en deux matrices (respectivement matrice ET et matrice OU). Ces entrées/sorties sont accessibles par le NORD ou le SUD de leur matrice respective. Chaque entrée/sortie occupe une colonne de la matrice, les termes produits ou monômes en occupant chacun une ligne. On obtient le schéma d'implantation suivant (figure 3.11,3.12):

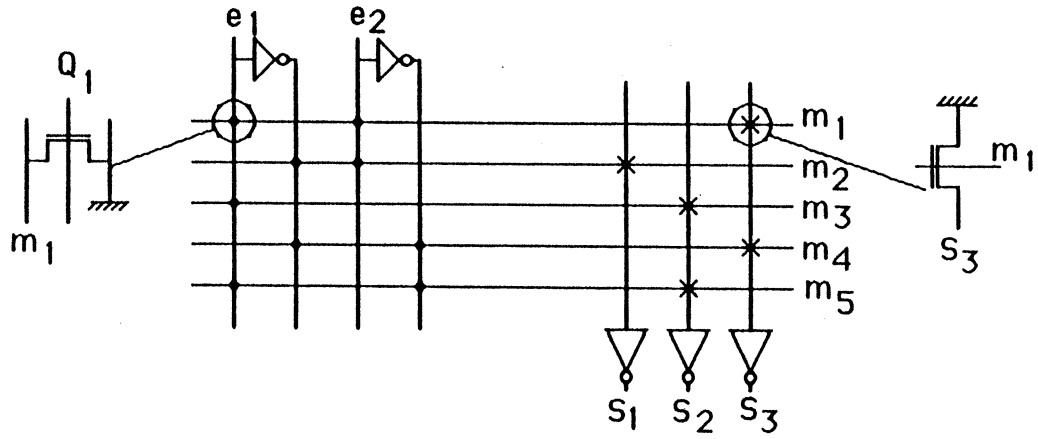


fig. 3.11 Forme logique et electrique NOR/NOR d'un PLA

Les fonctions réalisées dans ce PLA sont:

m_1= NON (e_1 OU e_2); m_2 = NON (NON(e_1) OU e_2); m_3 = NON (e_1);
m_4 = NON (NON(e_1) OU NON(e_2)); m_5 = NON (e_1 OU NON(e_2));

et

$S_1 = m_2$;
$S_2 = m_3$ OU m_5;
$S_3 = m_1$ OU m_4;

Topologiquement, l'on obtient la forme d'implantation suivante:

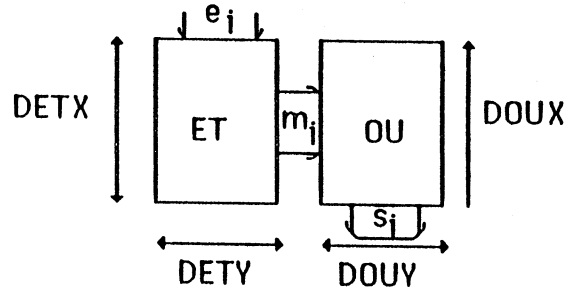


fig. 3.12 Schéma d'implantation d'un PLA classique

L'évaluation de la surface de ce type de PLA est lié à la formule suivante établie par [REIS 83]:

Si

NME : nombre de nomômes
NE : nombre d'entrées
NN : nombre de sorties
M : nombre de rappel de masse
2 : taille moyenne des transistors de charge
4 : taille moyenne des transistors de charge
PME : pas de métal
PE : pas d'entrée d'un PLA classique

l'on obtient alors les équations suivantes pour calculer la surface des matrices d'un PLA classique sans les amplificateurs d'entrée, ni les inverseurs de sortie:

MATRICE ET

DETX:=(NME+ NME/M).PME. (E 1.1)
DETY:=(NE+2) PE (E 1.2)

MATRICE OU

Les entrées de la matrice OU sont adaptées aux sorties de la matrice ET d'où:

DOUX:=DETX+4.PME. (E 1.3)

DOUY:=(NN+ NN/M).PME. (E 1.4)

La surface totale du PLA sans les amplificateurs d'entrée, ni les inverseurs de sortie est alors la suivante:

SURFACE TOTALE PLA St := DETX*DETY+DOUX*DOUY (E 1)

A la surface des deux matrices et des charges des monômes et des sorties, il faut nécessairement ajouter la surface des amplificateurs d'entrées et la surface des inverseurs de sortie.

La surface des inverseurs de sortie dépend de la charge qui se trouve en sortie de cet inverseur, ainsi que des caractéristiques de performance que l'on désire attribuer au PLA. Plus la charge en sortie est importante (capacité élevée, résistance importante), plus l'inverseur de sortie sera important si l'on désire des performances élevées.

Surface d'un inverseur de sortie

Soit DY : hauteur de l'inverseur de sortie, DX déterminé par la charge en sortie (E 1.6)

Soit DX : largeur d'un inverseur de sortie, on prendra par la suite

DX:=pas d'une sortie (E 1.5)

La surface des amplificateurs d'entrée est liée à la taille du PLA et de sa matrice ET. En effet, plus la taille de la matrice ET est importante, plus la ligne d'entrée est grande (=> capacité et résistance importantes). En outre, le nombre de transistors que doit attaquer cette ligne d'entrée influe aussi sur la taille de l'amplificateur d'entrée (à performance fixée), ou sur la performance du PLA, ou vitesse de calcul (à amplificateur d'entrée fixé) [DAND 83].

Surface d'un amplificateur d'entrée

Un amplificateur d'entrée donne en sortie l'entrée et l'entrée inversée

Soit DX_1 : largeur de l'amplificateur d'entrée,

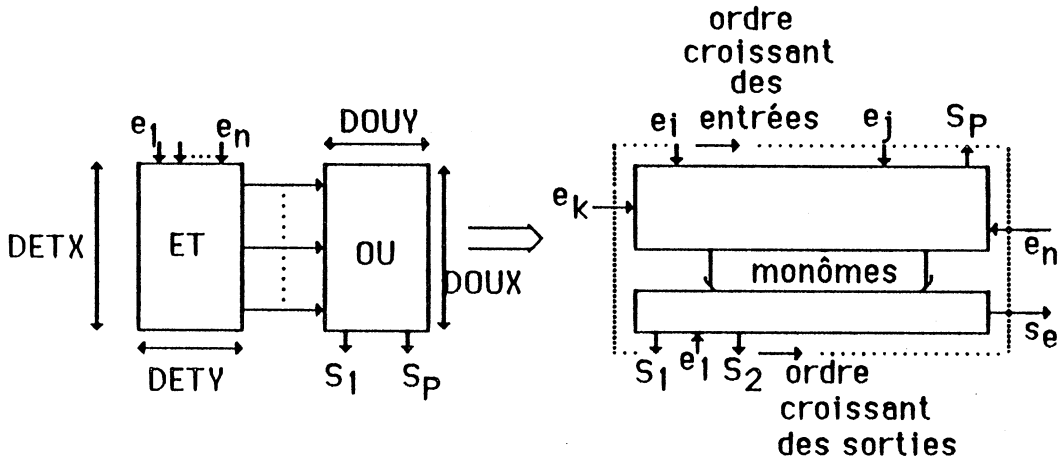
DX_1:= 2*pas d'entrée (E 1.7)

Soit DY_1: hauteur de l'amplificateur, DY_1 dépend des performances désirées pour le PLA (E 1.8)

3.1.2 PLA optimisé type "PAOLA" [CHUQ 84]

Les PLA's de type "PAOLA" subissent une optimisation topologique par la méthode dite des lignes d'entrées/sorties brisées, et pliage multiple des entrées/sorties [CHUQ 84] (ou multifolding [MiSa 83]). En outre, ces PLAs sont topologiquement déformables (par duplication des lignes de monômes et optimisation topologique) et transparents selon les deux axes. La transparence permet de passer une ligne à travers le PLA, NORD vers le SUD, ou EST vers OUEST (cf §II 1.2 pour la notion de transparence).

Les entrées/sorties peuvent être distribuées tout autour du PLA. Les entrées/sorties sont ordonnées, c'est à dire que les entrées sont numérotées dans l'ordre croissant dans le sens des aiguilles d'une montre, et que les sorties sont numérotées dans l'ordre croissant dans le sens trigonométrique (figure 3.13).



PLA classique PLA optimisé et déformé

fig. 3.13 Schéma d'implantation de PLA optimisé type PAOLA

L'évaluation de la surface de ce type de PLA est lié à la formule suivante établie par [REIS 83]:

Si

NME : nombre de nomômes
NE : nombre d'entrées
NN : nombre de sorties
M : nombre de rappel de masse
2 : taille moyenne des transistors de charge
4 : taille moyenne des transistors de charge
PME : pas de métal
PE : pas d'entrée d'un PLA classique
TDM:= taux de duplication de monômes

$$TRN := \ln\left\{\frac{\text{Nombre de transistors}}{\text{Nombre total de points}}\right\} * \frac{1}{0,33} - 25,22$$

TRN correspond au nombre de colonnes dans la matrice OU après optimisation divisé par le nombre de colonnes dans la matrice OU avant optimisation.

On obtient alors les équations suivantes pour calculer la surface des matrices d'un PLA optimisé type PAOLA sans les amplificateurs d'entrée, ni les inverseurs de sortie:

MATRICE ET

DETX:=(TDM.NME+TDM.NME/M).PME. (E 2.1)
DETY:=(NE+2)PE (E 2.2)

MATRICE OU

DOUX:=DETX+4.PME. (E 2.3)
DOUY:=(TRN.NN+ TRN.NN/M)PME (E 2.4)

La surface totale du PLA sans les amplificateurs d'entrée, ni les inverseurs de sortie est alors la suivante:

SURFACE TOTALE PLA St := DETX*DETY+DOUX*DOUY (E 2)

A la surface des deux matrices et des charges des monômes et des sorties, il faut nécessairement ajouter la surface des amplificateurs d'entrées et la surface des inverseurs de sortie.

La surface des inverseurs de sortie dépend de la charge qui se trouve en sortie de cet inverseur, ainsi que des caractéristiques de performance que l'on désire attribuer au PLA. Plus la charge en sortie est importante (capacité élevée, résistance importante), plus l'inverseur de sortie sera important si l'on désire des performances élevées.

Surface d'un inverseur de sortie
Soit DY : hauteur de l'inverseur de sortie, DX déterminé par la charge en sortie
Soit DX : largeur d'un inverseur de sortie, on prendra par la suite
DX:=pas d'une sortie

Si l'inverseur de sortie se trouve sur une sortie latérale, on considérera qu'un inverseur peut prendre en largeur deux pas de sortie (en effet, une sortie

latérale est placée entre deux monômes, d'où le pas de sortie est égal au moins à un pas de monôme plus un pas de sortie) d'où:

DX' := DX * 2; (E 2.5)

DY' := DY /2 ; (E 2.6)

La surface des amplificateurs d'entrée est liée à la taille du PLA et de sa matrice ET. Dans le cas du PLA optimisé du type PAOLA, la longueur DETX du PLA ne correspond pas automatiquement à la longueur d'une ligne d'entrée. En effet, cette ligne d'entrée étant brisée, sa longueur, donc sa charge, dépend de la couverture qu'elle occupe dans la matrice, c'est à dire l'écart entre le premier monôme qu'elle attaque dans la matrice, et le dernier monôme. D'autre part, si le taux de duplication des monômes est important, une ligne d'entrée attaquera d'autant plus de monômes, donc la capacité liée aux nombre de transistors augmentera d'autant. D'où le nombre de transistors que doit attaquer cette ligne d'entrée influe sur la taille de l'amplificateur d'entrée (à performance fixée), ou sur la performance du PLA, ou vitesse de calcul (à amplificateur d'entrée fixé) [DAND 83].

Surface d'un amplificateur d'entrée

Un amplificateur d'entrée donne en sortie l'entrée et l'entrée inversée

Soit DX_1 : largeur de l'amplificateur d'entrée,

DX_1:= 2*pas d'entrée

Soit DY_1: hauteur de l'amplificateur, DY_1 dépend des performances désirées pour le PLA

Si l'amplificateur d'entrée se trouve sur une sortie latérale, on considérera que cet amplificateur peut prendre en largeur quatre pas de sortie (en effet, une entrée et son complément latéraux sont placées entre deux monômes, d'où le pas d'une entrée est égal au moins à un pas de monôme plus un pas d'entrée) d'où:

$DX_1' := DX_1 * 2;$ (E 2.7)

$DY_1' := DY_1 /2 ;$ (E 2.8)

Intérêt des PLA's optimisés topologiquement:

- De tels PLA's permettent une meilleure adaptation de leurs connecteurs d'entrée/sortie à leur environnement.
- La surface du PLA optimisé est moins importante que celle du PLA classique (figure 3.14).
- La duplication des monômes permet une adaptation topologique du bloc PLA avec ses voisins. Il peut être considéré ainsi pour la formation du plan de masse comme un bloc topologiquement déformable.

La figure suivante présente des résulstats d'optimisation de PLA's de microprocesseurs exixtants, réétudiés et optimisés à l'aide de l'outil PAOLA [CHUQ 84].

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

fig. 3.14 Résulats de quelques optimisations topologiques de matrices OU.

3.1.3 PLA de type monomatrice

Le PLA de type monomatrice correspond à une organisation topologique des matrices Et et OU. Les entrées/sorties sont topologiquement distribuées le long d'une matrice unique, perpendiculairement aux monômes [VARI 85a].

Différents types d'optimisation ont été étudiés, optimisation portant sur un "multi folding" ou pliage multiple des lignes de monômes, ainsi que "duplication" des entrées/sorties [VARI 85b]

La transparence et la déformabilité d'un tel type de PLA est semblable à celle d'un PLA de type PAOLA, si ce n'est que la déformabilité se traduit par une duplication des entrées/sorties et optimisation du nombre des lignes de monômes (figure 3.15).

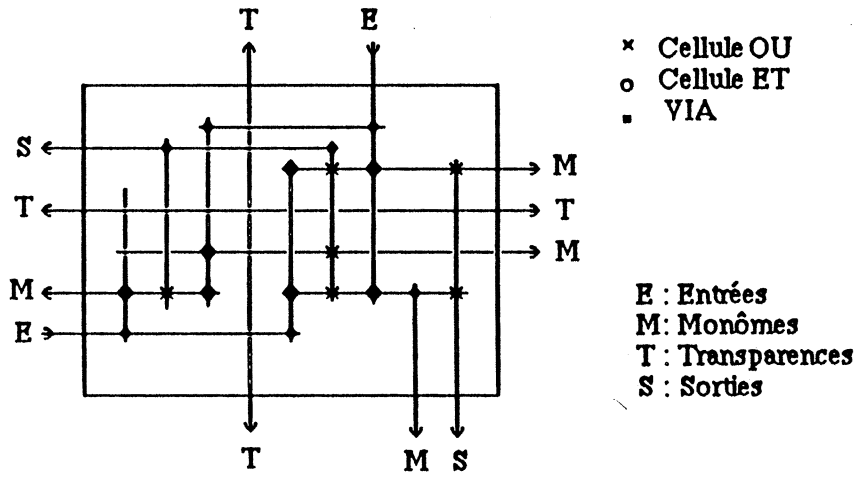


fig. 3.15 Schéma d'implantation d'un PLA monomatrice

L'évaluation de la surface des PLA's monomatrice est établie pour un PLA non optimisé à l'aide des équations suivantes:

Si

PE1 := Pas d'une entrée
PS1 := Pas d'une sortie
PS2 := Pas d'un monôme
NE := Nombre d'entrées
NS := Nombre de sorties
NME := Nombre de monômes

Alors, pour estimer la surface de la matrice unique du PLA sans la surface des amplificateurs d'entrée, ni les inverseurs de sortie, l'on a:

LDX := NE *PE1 + NS * PS1 + (NE+NS)/2 * PS1 (E 3.1)
LDY := NME * PS2 (E 3.2)

SURFACETOTALE_PLA_MONOMATRICE := LDX*LDY (E 3)

A la surface de la matrice unique et des charges des monômes et des sorties, il faut nécessairement ajouter la surface des amplificateurs d'entrées et la surface des inverseurs de sortie.

La surface des inverseurs de sortie dépend de la charge qui se trouve en sortie de cet inverseur, ainsi que des caractéristiques de performance que l'on désire attribuer au PLA. Plus la charge en sortie est importante (capacité élevée, résistance importante), plus l'inverseur de sortie sera important si l'on désire des performances élevées.

La surface des amplificateurs d'entrée est liée à la taille du PLA et de sa matrice. Dans le cas du PLA monomatrice, la longueur LDY du PLA correspond automatiquement à la longueur d'une ligne d'entrée (si l'on se trouve dans le cas d'un PLA monomatrice optimisé, la longueur de la ligne d'entrée dépend de sa couverture: la couverture correspond à la longueur entre le point d'entré dans la matrice, et le dernier monôme attaqué par cette entrée). Le nombre de transistors que doit attaquer cette ligne d'entrée influe sur la taille de l'amplificateur d'entrée (à performance fixée), ou sur la performance du PLA, ou vitesse de calcul (à amplificateur d'entrée fixé).

Intérêt des PLA's de type monomatrice:

- De tels PLA's permettent une meilleure adaptation de leurs connecteurs d'entrée/sortie à leur environnement.
- La duplication des entrées/sorties permet une adaptation topologique du bloc PLA avec ses voisins. Il peut être considéré ainsi pour la formation du plan de masse comme un bloc topologiquement déformable.
- Le PLA monomatrice peut être construit sur la trame des entrées et des sorties qui le constitue. Dans ce cas, le PLA est totalement adapté à son environnement. Cette approche a été étudiée à BULL/Clayes sous Bois dans l'équipe de M MASSON.

Comparativement à une solution utilisant des PLA's classique, les surfaces des points d'entrées et de sorties sont plus importantes pour un PLA monomatrice. On se trouve donc avec une surface de PLA non optimisée plus importante (figure 3.16). Néanmoins, la surface peut être topologiquement optmisée, et l'adaptation d'un PLA monomatrice avec ses voisins est supérieure à celle d'un PLA classique.

Soit

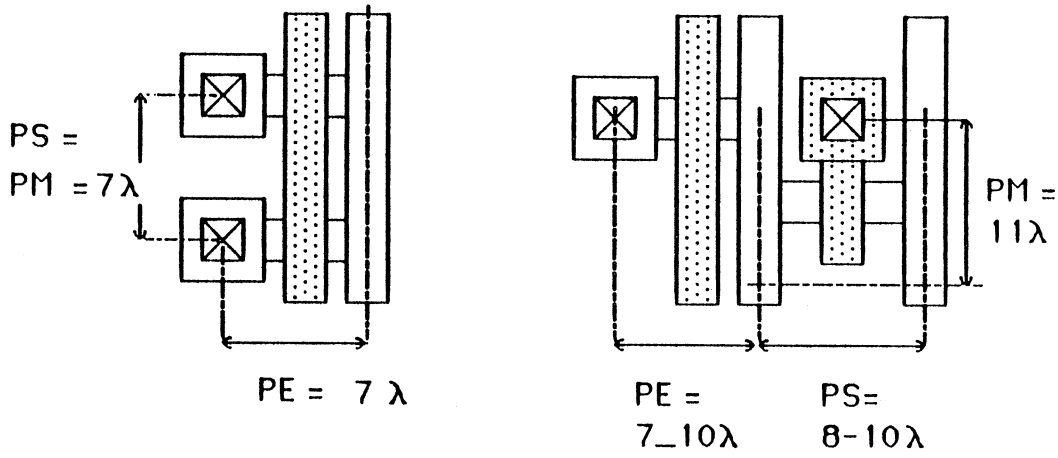
PM : hauteur d'un monôme
PE : taille d'un point d'entrée
PS : Taille d'un point de sortie

Alors dans le cas d'un PLA classique,

PE:=PS:=PM:=7

et dans le cas d'un PLA monomatrice,

PS := 8-10 Lambdas,
PE := 7-10 Lambdas,
PM := 11 Lambdas.



Point PLA classique Point PLA monomatrice

fig. 3.16 Figures de points PLA's en NMOS/CMP

Surface d'un inverseur de sortie

Soit DY" : hauteur de l'inverseur de sortie, DX déterminé par la charge en sortie (E 3.4)

Soit DX" : largeur d'un inverseur de sortie, on prendra par la suite
DX":=pas d'une sortie (E 3.3)

Surface d'un amplificateur d'entrée

Un amplificateur d'entrée donne en sortie l'entrée et l'entrée inversée

Soit $DX"_1$: largeur de l'amplificateur d'entrée,
$DX"_1$:= 2*pas d'entrée (E 3.5)

Soit $DY"_1$: hauteur de l'amplificateur, DY_1 dépend des performances désirées pour le PLA (E 3.6)

3.1.4 Solution à PLA's multiples avec extraction des propriétés :

Ce type de partie contrôle a été présentée dans le chapitre II 2.4.1. Une telle partie contrôle est composée de trois PLA's, respectivement un PLA de séquencement, un PLA de propriétés et un PLA des commandes.

L'importance relative des trois PLA's est fonction du type de l'organigramme de l'algorithme d'interprétation à implanter. Si les vecteurs de commande d'entrée peuvent être regroupés en grandes familles, alors la taille des PLA's de séquencement et de propriétés en sera réduite d'autant. L'importance de la taille du PLA des commandes est fonction du nombre d'états de l'automate de Mealey équivalent à implanter. Le nombre d'états est représenté dans le PLA des commandes par le nombre de monômes.

Les différents PLA's sont implantés à l'aide de PLA optimisés topologiquement (PLA de type PAOLA). Une telle forme d'implantation de PLA est utilisée pour permettre d'adapter la taille des différents PLA's entre eux.

Le PLA de commande, dans notre essai topologique, représente la base de construction de notre partie contrôle. Le PLA de séquencement a une surface liée à la complexité de l'algorithme à effectuer, diminuée du facteur d'extractions des propriétées de l'algorithme (figure 3.17). Il est implanté à l'ouest du PLA des propriétés, et au nord du PLA des commandes.

Le PLA de séquencement reçoit en entrée les commandes provenant du niveau supérieur de partie contrôle, les informations de contrôle provenant de bus de contrôle, les informations de comptes rendus de la partie opérative provenant du bus de compte rendu et les informations provenant du PLA des commandes, informations précisant l'état dans lequel se trouve l'automate de contrôle. Les informations de commande provenant du niveau supérieur sont introduites par le haut du PLA (le haut correspondant à des entrées latérales dans la matrice ET du PLA optimisé), et implanté horizontalement. Les informations de compte rendus sont, de même que les commandes, introduites par le haut du PLA. Les informations de contrôle provenant du bus de contrôle, sont introduites par l'ouest du PLA, ainsi que les informations provenant du PLA des commandes. Les informations de sortie du PLA de séquencement sont dirigées vers le PLA des commandes, et correspondent à des sorties latérales du PLA (sorties dirigées vers le SUD : figure 3.17).

Dans le cas du PLA des propriétés, les informations rentrantes sont les informations de commande provenant du niveau supérieur. Elles sont introduites latéralement dans le PLA, c'est à dire par le haut du PLA (figure 3.17). Les informations de sorties, dirigées vers le PLA des commandes, sont des sorties latérales, c'est à dire dans notre cas sortant par le SUD du PLA (figure 3.17)

Dans le cas du PLA des commandes, les informations d'entrées proviennent soit du PLA de séquencement, soit du PLA des propriétés. Ces informations sont

introduites latéralement dans le PLA, le PLA étant disposé latéralement. Les informations de sortie du PLA sont soit des informations de commande pour le niveau inférieur, soit des informations de contrôles, ou des informations indiquant l'état dans lequel se trouve le PLA, dirigées alors vers le bus de contrôle. Les informations de contrôle sortent donc à l'ouest du PLA, ainsi que les informations destinées au PLA de séquencement. Les informations de commande pour le niveau inférieur sortent latéralement, c'est à dire par le SUD du PLA (figure 3.17).

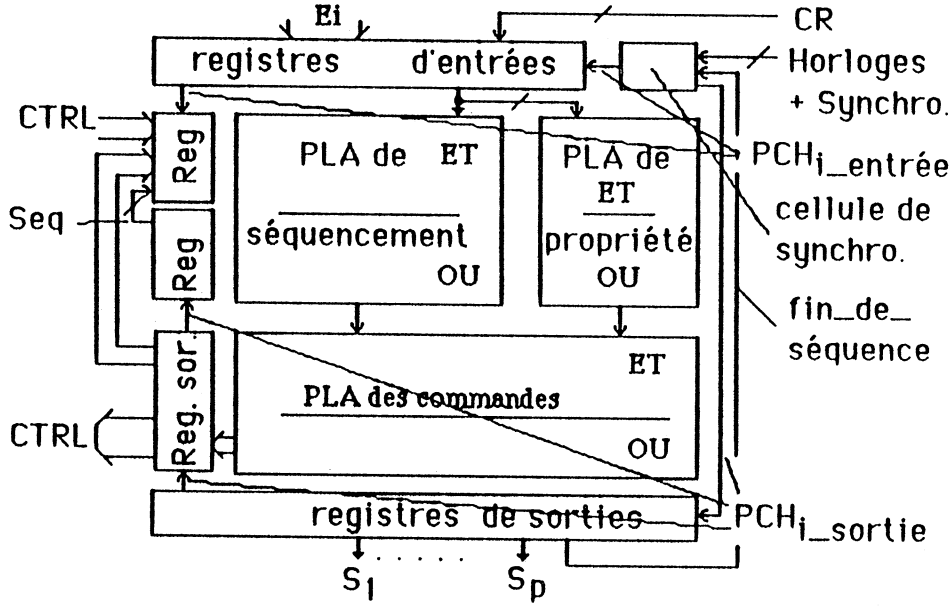


fig. 3.17 Schéma topologique d'une partie contrôle multi PLA avec extraction des propriétés

Evaluation de surface d'une telle implémentation:

Soit:

Pour le PLA de séquencement,

Ncomseq: Nombre de commandes provenant du niveau supérieur, et entrant dans le PLA de séquencement
Nctrlin : Nombre de contrôles enrant dans le PLA de séquencement
Ncrin : Nombre de comptes rendus entrant dans le PLA de séquencement
Nseq: Nombre de fils de sequencement interne entrant/sortant dans le PLA de séquencement (Seq)
Nseq2 : Nombre de fils provenant du PLA des commandes et entrant dans le PLA de séquencement
Nseqout: Nombre de fils sortant du PLA de séquencement et entrant dans le PLA des commandes
Nseqétat: Nombre de monômes du PLA de séquencement

Pour le PLA des propriétés,

Nproout: Nombre de fils sortant du PLA de propriétés et entrant dans le PLA des commandes
Ncompro: Nombre de commandes provenant du niveau supérieur, et entrant dans le PLA de proriétés
Nproétat: Nombre de monômes de ce PLA (nombre lié au nombre de familles d'instructions différentes)

Pour le PLA des commandes,

Nseq2 : Nombre de fils provenant du PLA des commandes et entrant dans le PLA de séquencement
Nprocom: Nombre de commandes provenant du PLA de proriétés, et entrant dans le PLA des commandes
Nseqout: Nombre de fils sortant du PLA de séquencement et entrant dans le PLA des commandes
Ncomout: Nombre de commandes pour l'automate de niveau inférieur
Nctrlout : Nombre de fils de contrôles sortant
Ncométat: Nombre d'états de l'automate de Mealy à implanter, égal au nombre de monômes de ce PLA (valeur maximum)

Alors, en utilisant les formules de calcul de PLA optimisé (équation E 3.x, cf §II 3.1.2), à partir des variables précédentes, on peut calculer la surface respective de chacun des PLA's précédents, dans le cas ou il n'y a pas de duplication de monômes (TDM := 1, taux de duplication des monôme,TRN: pourcentage d'optimisation)).
En outre, il est nécessaire de rajouter à ces trois surfaces, la surface des amplificateurs d'entrée, et celle des inverseurs. On considère que ces derniers sont topologiquement adaptés au pas d'entrée et de sortie des PLA's.

Hauteur et largeur des PLA'savec leurs amplificateurs et inverseurs:
En applicant les équations (E 2.X; X= 0,...,6 et E 1.5,...,E 1.8 ; voir §III 3.1.1,3.1.2):

Largeur_seq := 4.PME+ (Nseqétat+Nseqétat/M).PME+MAX(DY_1,DY),
Hauteur_seq := (Nctrlin+Ncrin+Nseq+Nseq2+Ncomseq)*2*PE+
(Nseqout+Nseq+(Nseqout+Nseq)/M).TRN.PME+DY'+DY_1'

Largeur_pro := 4.PME+ (Nproétat+Nproétat/M).PME,
Hauteur_pro := (Ncompro)*2*PE+
(Nproout+(Nproout)/M).TRN.PME+DY'+DY_1'

Largeur_com := 4.PME+ (Ncométat+Ncométat/M).PME+MAX(DY_1,DY),
Hauteur_com := (Nseqout+Nprocom)*2*PE+

(Nseqout+Nseq2+Ncomout+Nctrlout+
(Nseqout+Nseq2+Ncomout+Nctrlout)/M).TRN.PME+DY'+DY_1'

D'autre part, à la surface précédente, la surface nécessaire à l'implémentation des mémorisation d'entrée et de sortie doit être additionnée.

Surface d'un registre d'entrée ou de sortie:

DX_2 : largeur d'un egistre, DX_2 := 2 pas d'entrée de PLA classique
DY_2 : hauteur d'un registre.

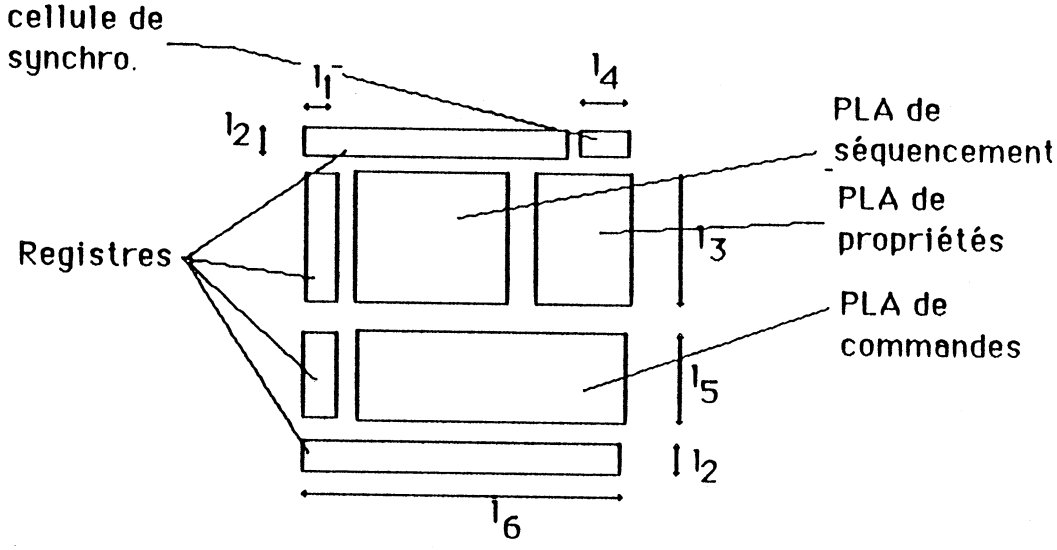


fig. 3.18 Schéma topologique, avec dimensions pour le calcul de la surface

Les équations de surface sont donc les suivantes:

l_1 := l_2 := hauteur d'un registre d'entrée ou de sortie := DY2
l_3 := MAX (Hauteur_seq,Hauteur_pro)
l_4 := largeur d'une cellule de synchronisation := 3...4*DX2
l_5 := Hauteur_com
l_6 := MAX (Largeur_com+l_1, l_1+Largeur_seq+Largeur_pro)

Surface de l'étage := l_6 * (l_2*2+l_5+l_3)

3.1.5 Solution à PLA's multiples avec extraction des paramètres :

Cette solution pour implémenter un automate est surtout valable pour générer les commandes d'une partie opérative (voir §II 2.4.2). Les paramètres concernent un ensemble des commandes agissant sur une unité fonctionnelle de la partie opérative qui n'exécute qu'une fonction spécifique au cours de l'interprétation d'une instruction donnée, le reste du temps, une fonction par défaut est exécutée.

Dans la description de l'état de l'algorithme, toutes les commandes formant cet ensemble sont enlevées et remplacées par une commande séquencée de sélection. Cette commande de sélection permet de faire le choix entre les paramètres et les valeurs par défaut générées à l'entrée du multiplexeur.

Les PLAs de paramètres sont par nature assez petits et proches de l'unité fonctionnelle (partie de la partie opérative) qu'ils contrôlent.

Il y a alors trois possibilités pour le ou les PLA's de paramêtres. Soit l'on considère que l'on implante les PLA's de paramêtres à l'aide de PLA's répartis respectivement au dessus de l'unité fonctionnelle qu'il contrôle, soit l'on intègre l'ensemble des PLA's de paramêtres dans un PLA unique que l'on implante à l'aide d'un PLA à deux matrices (PLA classique optimisé), soit à l'aide d'un PLA monomatrice (cf §III 3.1.3).

1) Les PLA's sont réalisés de manière individuelle, avec leur matrice OU optimisée, et leurs entrées/sorties latérales(figure 3.19).

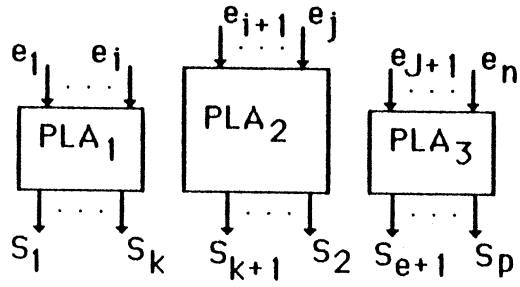


fig. 3.19 Implantation des PLA's de paramètres

Avantages:

- Chaque PLA est implanté au dessus de l'unité fonctionnelle qu'il contrôle. La perte par routage entre l'unité fonctionnelle de la partie opérative est ainsi minimale.
- Les PLA's de paramêtres étant de taille faible, ils sont donc rapides, et nécessitent des amplificateurs d'entrées de taille faible (taille du PLA minimale).

Inconvénients:

- La zone de routage entre les PLA's de paramêtres et le PLA de validation est élevée si le PLA de validation est implanté à droite du PLA de séquencement (mauvaise répartition des sorties du PLA de validation comparativement aux entrées des PLA's de paramêtres).
- La forme des PLA's de paramêtres n'étant pas régulière, les PLA's sont de tailles différentes, et les PLA's de séquencement et de validation étant de forme régulière, la place perdue est fonction de la somme de l'écart de surface existant entre le PLA de paramêtre le plus gros et chacun des autres PLA's de paramêtres.
- La régularité d'une telle structure diminue en fonction du nombre de PLA's de paramêtres à implanter.

2) Tous les PLAs sont réunis en un PLA unique type "PAOLA" dont on optimise la partie ET, et la partie OU, avec de surcroît une duplication des monômes (figure 3.20).

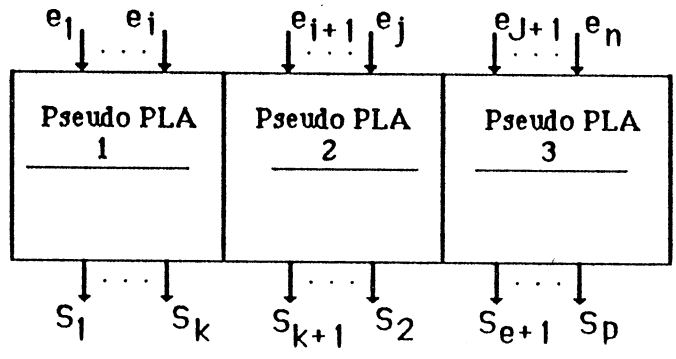


fig. 3.20 Implantation des PLA's de paramètres à l'aide d'un seul PLA type PAOLA

Avantages:

- Du point de vue topologique, la construction et l'assemblage du PLA avec son environnement est plus facile (un seul élément à traiter).
- La régularité d'un PLA unique est plus élevée que celle d'un ensemble de PLA's. La perte de surface par rapport à une solution à plusieurs PLA's est faible. Il y a possibilité de dupliquer les monômes pour améliorer le facteur d'optimisation du PLA, et de plus pour permettre une meilleure adaptation de ce PLA unique avec la partie oérative.

Inconvénients:

- Du point de vue électrique, la taille du PLA plus importante peut amener une perte de performance de l'ensemble. La hauteur DETX (équation E 2.1) de la matrice ET du PLA est égale au mieux à la hauteur de la matrice ET du PLA de paramêtre la plus importante. Par contre la longueur d'une ligne d'entrée de ce PLA sera égale à la longueur de la ligne d'entrée existante dans le cas de l'utilisation de plusieurs PLA's (par exemple, une ligne d'entrée e_{i+2} de

la figure 3.19 est égale à la ligne d'entrée e_{i+2} de la figure 3.20, dans le cas de lignes d'entrées brisées).
- La surface du PLA unique est supérieure à la surface totale de la solution à plusieurs PLA's.

3) Les PLAs de paramétrisation sont réunis en un seul PLA de type monomatrice, optimisé avec duplication des entrées/sorties.(figure 3.21).

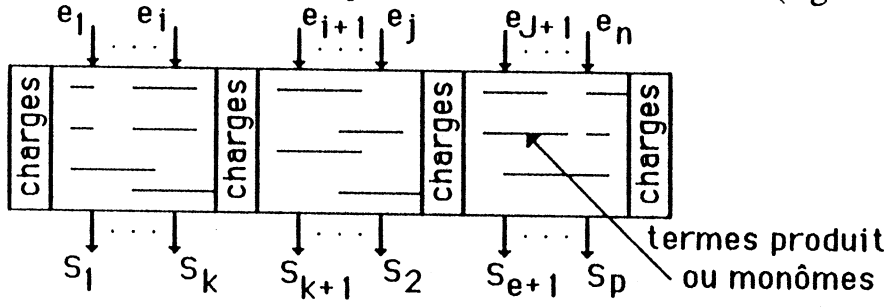


fig. 3.21 Implantation des PLA's de paramètres à l'aide d'un PLA monomatrice unique

Avantages:

- Le PLA monomatrice est adapté pour implanter des PLA à grands nombre d'entrées/sorties: Il est possible de construire un PLA monomatrice autour de la trame constituée par les entrées/sorties. Les monômes correspondent aux lignes de la matrice unique. Dans le cas des PLA's de paramètres implantés dans une seule matrice, le nombre de lignes est égal aux nombre de monômes du PLA de paramètre le plus important (PLA_2 dans notre cas).
- La répartition des charges de monômes se faisant dans la matrice et de manière verticale, la hauteur du PLA n'est plus fonction des charges comme dans une solution à PLA optimisés de type PAOLA implanté horizontalement. La hauteur du PLA est donc diminuée.
- Une implantation de PLA's de paramètres à l'aide d'un seul PLA monomatrice est plus régulière que celle utilisant un grand nombre de PLA's: Une telle implantation est donc plus facilement automatisable.

Inconvénients:

- Les mêmes inconvénients que ceux déterminés dans la solution 2 d'implantation des PLA's de paramètres se retrouvent: Les performances d'un PLA unique est inférieur à celle d'une solution utilisant plusieurs petits PLA's.

CONCLUSION

Le PLA monomatrice, au vu des remarques précédentes, apparait comme la meilleure solution pour implanter les PLA's de paramêtres de manière automatique, avec une perte de performance et de surface minimum, ainsi qu'une grande facilité de conception.

En ce qui concerne les PLAs de validation et de séquencement, la solution suivante est préconisée :

Les PLAs sont implantés au-dessus du bloc représenté par le ou les PLA's de paramétrisation, suivant la taille respective des PLA's, le PLA de séquencement sera placé au dessus, en dessous, ou à côté du PLA de validation. La taille est fonction du nombre de monômes respectifs de chaque PLA considéré. Si le nombre sommé des monômes des deux PLA's est supérieur à la place disponible en largeur au dessus des PLA's de paramètres (lignes virtuelles implantables), alors les PLA's sont placés l'un au dessus de l'autre. L'ordre de placement dépend du nombre respectif de monômes pour chaque PLA. Dans le cas ou le PLA de séquencement a un nombre de monômes faible, ce dernier est placé sous le PLA de validation, et les sorties du PLA de validation sont passées par transparence à travers le PLA de séquencement, dans le cas contraire il y a inversion des positions. (figure 3.22). Le schéma topologique présenté ne présente qu'une des solutions proposées ci-dessus, et dans le cadre de cette solution, les équations permettant d'évaluer la surface sont données.

Les différents PLA's sont implantés à l'aide de PLA optimisés topologiquement (PLA de type PAOLA). Une telle forme d'implantation de PLA est utilisée pour permettre d'adapter la taille des différents PLA's entre eux.

Le PLA de séquencement reçoit en entrée les commandes provenant du niveau supérieur de partie contrôle, les informations de contrôle provenant de bus de contrôle, les informations de comptes rendus de la partie opérative provenant du bus de compte rendu et les informations provenant du PLA de validation, informations précisant l'état dans lequel se trouve l'automate de contrôle. Les informations de commande provenant du niveau supérieur sont introduites par le haut du PLA (le haut correspondant à des entrées latérales dans la matrice ET du PLA optimisé, et implanté horizontalement. Les informations de compte rendus sont, de même que les commandes, introduites par le haut du PLA. Les informations de contrôle provenant du bus de contrôle, sont introduites par l'ouest du PLA. Les informations provenant du PLA de validation sont introduites à droite du PLA de séquenceiment. Les informations de sortie du PLA de séquencement sont dirigées vers le PLA de validation et des informations pour le PLA de validation. Les informations destinées au PLA des paramêtres correspondent à des sorties latérales du PLA (sorties dirigées vers le SUD : figure 3.22). Les informations destinées au PLA de validaton correspondent à des sorties classiques à droite du PLA de séquencement. Des informations de sorties dirigées vers le bus de contrôle sortent par la gauche du PLA.

Dans le cas du PLA de validation, les informations rentrantes sont les informations de commande provenant du niveau supérieur et les informations provenant du PLA de séquencement. Les informations de commande sont introduites latéralement dans le PLA, c'est à dire par le haut du PLA (figure 3.22). Les informations provenant du PLA de séquencement sont introduite par la gauche du PLA. Les informations de sorties, dirigées vers le PLA des commandes, sont des sorties latérales, c'est à dire dans notre cas sortant par le SUD du PLA (figure 3.22), ou des sorties classiques (information de séquencement) sortant par la gauche du PLA.

Dans le cas du (ou des) PLA de paramêtres, les informations d'entrées proviennent soit du PLA de séquencement, soit du PLA de validation. Ces informations sont introduites latéralement dans le PLA, le PLA étant disposé latéralement. Les informations de sortie du PLA sont des informations de commande pour le niveau inférieur, niveau inférieur composée d'unités fonctionnelles distinctes (exemple une partie opérative). Les informations de commande pour le niveau inférieur sortent latéralement, c'est à dire par le SUD du PLA (figure 3.22).

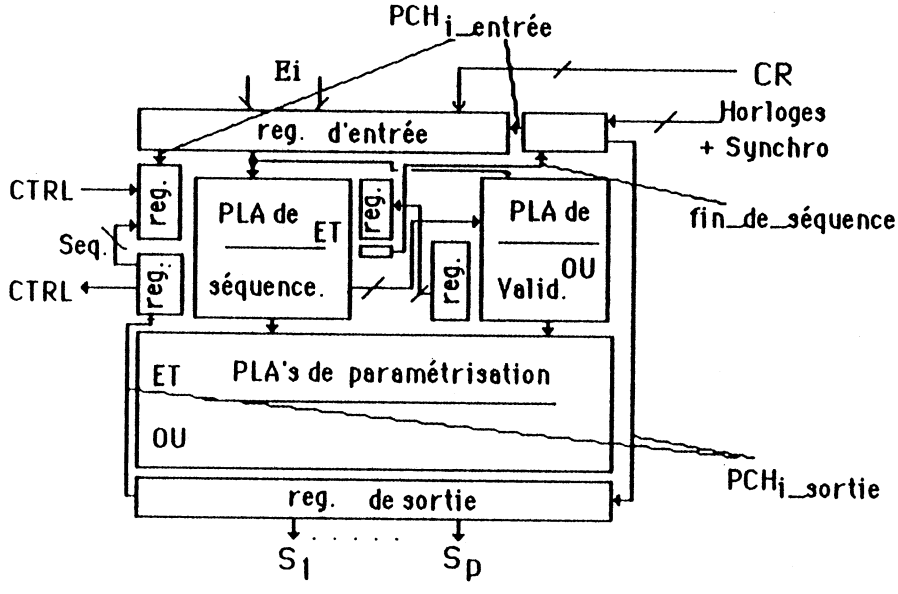


fig. 3.22 Schéma topologique d'une partie contrôle à multi-PLA avec extractions des paramètres

Evaluation de surface d'une telle implémentation:

Soit:

Pour le PLA de séquencement,

Ncomseq: Nombre de commandes provenant du niveau supérieur, et entrant dans le PLA de séquencement
Nctrlin : Nombre de contrôles enrant dans le PLA de séquencement
Ncrin : Nombre de comptes rendus entrant dans le PLA de séquencement
Nseq: Nombre de fils de sequencement interne entrant/sortant dans le PLA de séquencement (Seq)
Nseq2 : Nombre de fils provenant du PLA de validation et entrant dans le PLA de séquencement
Nseqout: Nombre de fils sortant du PLA de séquencement et entrant dans le (ou les) PLA de paramêtres
Nseqval: Nombre de fils allant versle PLA de validation
Nseqétat: Nombre de monômes du PLA de séquencement

Pour le PLA de validation,

Nseqval: Nombre de fils venant du PLA de séquencement
Nvalout: Nombre de fils sortant du PLA de validation et entrant dans le PLA de paramêtres
Ncomval: Nombre de commandes provenant du niveau supérieur, et entrant dans le PLA de validation
Nvalétat: Nombre de monômes de ce PLA
Nseq2 : Nombre de fils provenant du PLA de validation et entrant dans le PLA de séquencement

Pour le (ou les) PLA de paramêtres,

Nvalpar: Nombre de commandes provenant du PLA de validation, et entrant dans le PLA des paramêtres
Nseqout: Nombre de fils sortant du PLA de séquencement et entrant dans le PLA des paramêtres
Nparout: Nombre de commandes pour l'automate de niveau inférieur
Nparétat: Nombre d'états de l'automate de Mealy à implanter, égal au nombre de monômes de ce PLA (valeur maximum)

Alors, en utilisant les formules de calcul de PLA optimisé (équation 3.x, cf §II 3.1.2), à partir des variables précédentes, on peut calculer la surface respective de chacun des PLA's précédents, dans le cas ou il n'y a pas de duplication de monômes (TDM := 1, taux de duplication des monôme,TRN: pourcentage d'optimisation)).
En outre, il est nécessaire de rajouter à ces trois surfaces, la surface des amplificateurs d'entrée, et celle des inverseurs. On considère que ces derniers sont

topologiquement adaptés au pas d'entrée et de sortie des PLA's.

Hauteur et largeur des PLA's avec leurs amplificateurs et inverseurs:
En applicant les équations (E 2.X; X= 0,...,6 et E 1.5,...,E 1.8 ; voir §III 3.1.1,3.1.2):

Largeur_seq := 4.PME+ (Nseqétat+Nseqétat/M).PME+2*MAX(DY_1,DY),
Hauteur_seq := (Nctrlin+Ncrin+Nseq+Nseq2+Ncomseq)*2*PE+
(Nseqout+Nseq+Nseqval+
(Nseqout+Nseq+Nseqval)/M).TRN.PME+DY'+DY_1'

Largeur_val := 4.PME+ (Nvalétat+Nvalétat/M).PME+MAX(DY_1,DY),
Hauteur_val := (Ncomval+Nseqval)*2*PE+
(Nvalout+Nseq2+(Nvalout+Nseq2)/M).TRN.PME+DY'+DY_1'

Largeur_par := 4.PME+ (Nparétat+Nparétat/M).PME
Hauteur_par := (Nvalpar)*2*PE+(Nparout+
(Nseqout+Nparout)/M).TRN.PME+DY'+DY_1'

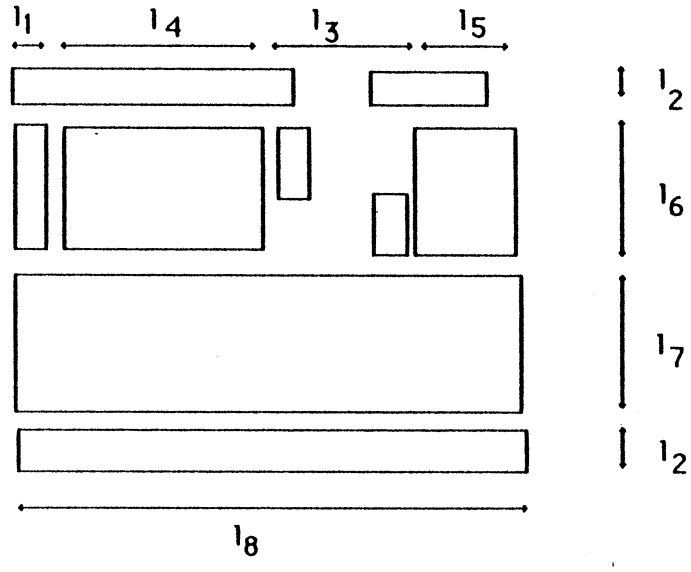


fig. 3.23 Schéma topologique, avec dimensions pour le calcul de la surface

Les équations de surface sont donc les suivantes:

l_1 := l_2 := hauteur d'un registre d'entrée ou de sortie := DY2
l_3 := 2*l_1+(Nseq2+fin_de_séquence+Nseqval)*pas de poly
l_4 := Largeur_seq
l_5 := Largeur_val

l_6 := MAX (Hauteur_seq,Hauteur_val)

l_7 := Hauteur_par

l_8 := MAX (Largeur_par+l_1, l_1+Largeur_seq+Largeur_valo+l_3)

Surface de l'étage := $l_8*(l_2+l_6+l_7+l_2)$

3.2 Modèle à générateur de temps

Un modèle à générateur de temps (§II 2.4.3) demande l'utilisation de trois PLA's différents: Le PLA des commandes, le PLA de génération des instants, et le PLA des propriétés. La complexité du séquencement se reporte intégralement sur le PLA des propriétés pour le décodage des vecteurs d'entrées et sur le PLA de génération des instants (automate des temps). Le PLA des commandes n'est fonction que du nombre des états qu'il doit mémoriser. Sa matrice ET, de décodage, peut être organisée en deux blocs fonctionnellement disjoints [SCHO 86], l'un correspondant au décodage des instants, l'autre aux propriétés générales d'un groupes de vecteurs de commandes.

Il est topologiquement habile de placer les PLA's générant les informations (PLA de génération et PLA des propriétés) de telle sorte que ces informations de sorties se retrouvent face à l'endroit de leur utilisation. Ainsi, le PLA de génération des instants est placé à droite du PLA de décodage, et les deux accolés, au dessus du PLA des commandes. Nous obtenons le schéma topologique suivant (figure 3.24):

Les différents PLA's sont implantés à l'aide de PLA optimisés topologiquement (PLA de type PAOLA). Une telle forme d'implantation de PLA est utilisée pour permettre d'adapter la taille des différents PLA's entre eux.

Le PLA des propriétés reçoit en entrée les commandes provenant du niveau supérieur de partie contrôle, les informations de contrôle provenant de bus de contrôle et les informations de comptes rendus de la partie opérative provenant du bus de compte rendu. Les informations de commande provenant du niveau supérieur sont introduites par le haut du PLA (le haut correspondant à des entrées latérales dans la matrice ET du PLA optimisé, et implanté horizontalement. Les informations de compte rendus sont, de même que les commandes, introduites par le haut du PLA. Les informations de contrôle provenant du bus de contrôle, sont introduites par l'ouest du PLA. Les informations de sortie du PLA de séquencement sont dirigées vers le PLA des commandes, et correspondent à des sorties latérales du PLA (sorties dirigées vers le SUD : figure 3.17). Une partie des informations de sorties sont dirigées vers le PLA de génération des instants, permettant de définir le nombre d'instants désirés pour le séquencement.

Dans le cas du PLA de génération des instants, les informations rentrantes sont les informations provenant du PLA des propriétés, et les informations internes de séquencement. Les informations provenant du PLA de propriétés sont introduites par la gauche du PLA, et les informations internes de séquencement par la droite du PLA. Les informations de sorties, dirigées vers le PLA des commandes, sont des sorties latérales, c'est à dire dans notre cas sortant par le SUD du PLA (figure 3.24), et les informations de séquencement sortent classiquement à droite du PLA.

Dans le cas du PLA des commandes, les informations d'entrées proviennent soit du PLA de génération des instants, soit du PLA des propriétés. Ces informations sont introduites latéralement dans le PLA, le PLA étant disposé latéralement. Les

informations de sortie du PLA sont soit des informations de commande pour le niveau inférieur, soit des informations de contrôles dirigées alors vers le bus de contrôle. Les informations de contrôle sortent donc à l'ouest du PLA. Les informations de commande pour le niveau inférieur sortent latéralement, c'est à dire par le SUD du PLA (figure 3.24).

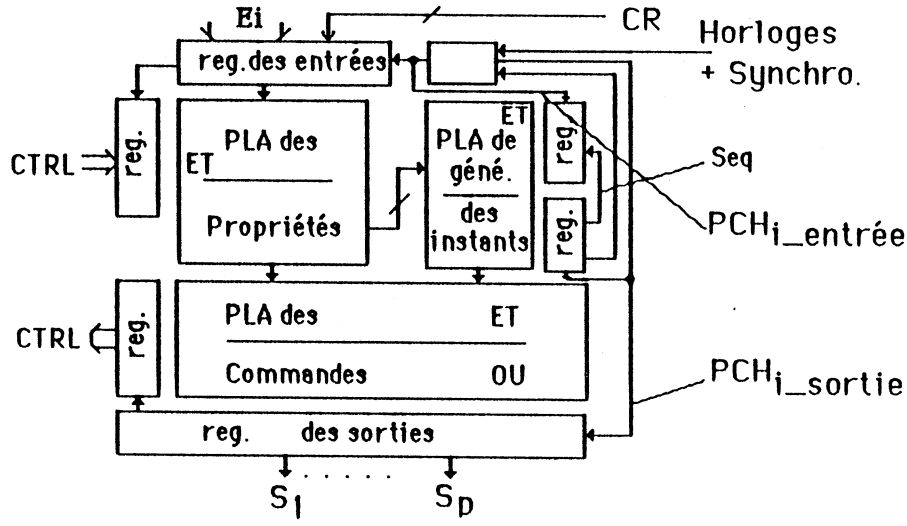


fig. 3.24 Modèle topologique d'une partie contrôle à générateur de temps

Evaluation de surface d'une telle implémentation:

Soit:

Pour le PLA des propriétés,

Ncompro: Nombre de commandes provenant du niveau supérieur, et entrant dans le PLA de séquencement
Nctrlin : Nombre de contrôles enrant dans le PLA de séquencement
Ncrin : Nombre de comptes rendus entrant dans le PLA de séquencement

Nprogen : Nombre de fils provenant du PLA des propriétés et entrant dans le PLA de génération des instants
Nproout: Nombre de fils sortant du PLA de séquencement et entrant dans le PLA des commandes
Nproétat: Nombre de monômes du PLA de séquencement

Pour le PLA de génération des instants,

Nseq: Nombre de fils de sequencement interne entrant/sortant dans le PLA de génération des instants (Seq)
Ngenout: Nombre de fils sortant du PLA de génération des instants et entrant dans le PLA des commandes
Nprogen : Nombre de fils provenant du PLA des propriétés et entrant dans le PLA de génération des instants
Ngenétat: Nombre de monômes de ce PLA (nombre lié au nombre de familles d'instructions différentes)

Pour le PLA des commandes,

Ngenout: Nombre de fils sortant du PLA de génération des instants et entrant dans le PLA des commandes
Nproout: Nombre de fils sortant du PLA de propriétés et entrant dans le PLA des commandes
Ncomout: Nombre de commandes pour l'automate de niveau inférieur
Nctrlout : Nombre de contrôles sortants
Ncométat: Nombre d'états de l'automate de Mealy à implanter, égal au nombre de monômes de ce PLA (valeur maximum)

Alors, en utilisant les formules de calcul de PLA optimisé (équation 3.x, cf §II 3.1.2), à partir des variables précédentes, on peut calculer la surface respective de chacun des PLA's précédents, dans le cas ou il n'y a pas de duplication de monômes (TDM := 1, taux de duplication des monôme,TRN: pourcentage d'optimisation)).
En outre, il est nécessaire de rajouter à ces trois surfaces, la surface des amplificateurs d'entrée, et celle des inverseurs. On considère que ces derniers sont topologiquement adaptés au pas d'entrée et de sortie des PLA's.

Hauteur et largeur des PLA'savec leurs amplificateurs et inverseurs:

En applicant les équations (E 2.X; X= 0,...,6 et E 1.5,...,E 1.8 ; voir §III 3.1.1,3.1.2):

Largeur_pro := 4.PME+ (Nproétat+Nproétat/M).PME+MAX(DY_1,DY),

Hauteur_pro := (Nctrlin+Ncrin+Ncompro)*2*PE+(Nproout+Nprogen+ (Nprogen+Nproout)/M).TRN.PME+DY'+DY_1'

Largeur_gen := 4.PME+ (Ngenétat+Ngenétat/M).PME+2*MAX(DY_1,DY),

Hauteur_gen := (Nprogen+Nseq)*2*PE+ (Ngenout+Nseq+(Nseq+ Ngenout)/M).TRN.PME+DY'+DY_1'

Largeur_com := 4.PME+ (Ncométat+Ncométat/M).PME+2*MAX(DY_1,DY),

Hauteur_com := (Nprocom+Ngenout)*2*PE+(Ncomout+Nctrlout (Ncomout+Nctrlout)/M).TRN.PME+DY'+DY_1'

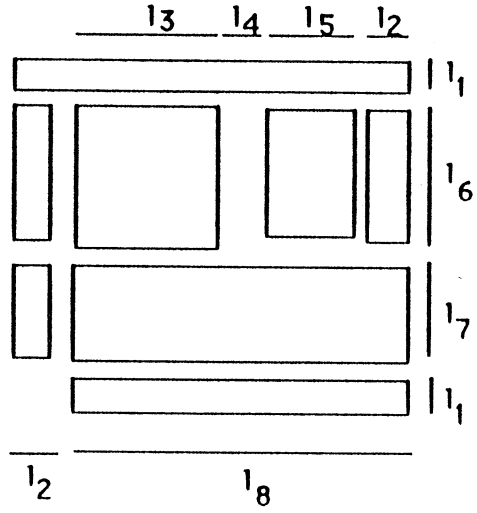


fig. 3.25 Schéma topologique, avec dimensions pour le calcul de la surface

Les équations de surface sont donc les suivantes:

l_1 := l_2 := hauteur d'un registre d'entrée ou de sortie := DY2

l_3 := Largeur_pro

l_4 := Ngenpro*pas de poly

l_5 := Largeur_gen

l_6 := MAX(Hauteur_pro,Hauteur_gen)

l_7 := Hauteur_com

l_8 := MAX(l_3+l_4+l_5+l_2,Largeur_com)

<u>Surface de l'étage</u> := (l_8+l_2)*(l_1+l_7+l_6+l_1)

3.3 Modèle à ROM

Comme pour les modèles à base de PLA's, nous allons d'abord présenter la forme topologique d'une ROM, ainsi que celle des décodeurs et multiplexeurs, puis les équations de calcul (estimations) de surface propres à chacun des éléments précités.

Une ROM permet d'implanter une fonction logique sous forme canonique. Elle est pour cela consituée d'un décodeur d'entrée correspondant à la Matrice ET d'un PLA, d'un plan de matrice correspondant à la Matrice OU d'un PLA, ou zone de mémorisation. Le plan de matrice pouvant être organisé en n.uples mots, n=1,...p, il faut donc lui associer un démultiplexeur en sortie, faisant le choix d'un mot parmi n, et d'un décodeur des commandes du multiplexeur (ce décodeur permet de diminuer le nombre de lignes traversant la matrice de mémorisation de la ROM et attaquant le multiplexeur.

Pour estimer la surface, résultat de l'implémentation d'une solution à ROM, les formules suivantes seront utilisées (figure 3.26). Ces formules sont basées sur une étude effectuée par [OBR 82] et [REIS 83] pour une ROM à un seul plan de matrice. Des transformations ont été nécessaires pour les adapter à une ROM à deux plans de matrice.

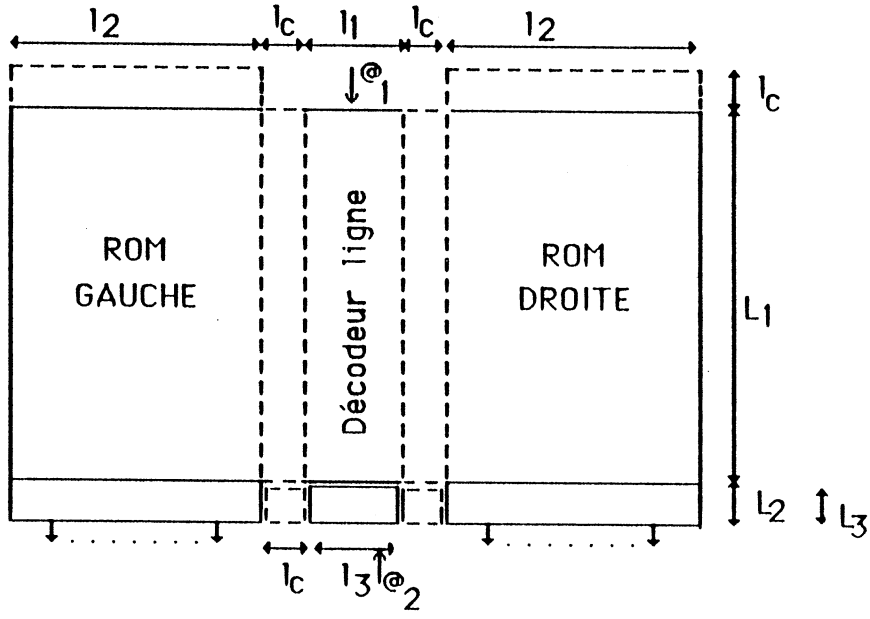


fig. 3.26 Dimensions d'une ROM

Soit	Na	nombre de bits d'adresses
	Nm	nombre de mots de la ROM
	Nl	nombre de lignes de la ROM
	Nc	nombre de mot dans la ligne
	Nb	nombre de bits dans le mot
	Pa	pas des lignes d'adresses dans le décodeur

P_{m1} pas des lignes de mots dans le décodeur
P_{m2} pas des lignes des bits dans la matrice
Pd pas des lignes de mots dans le démultiplexeur
Fr facteur de rappel de masse

On a : $Nm \leq 2^{Na}$ et on décompose l'adresse Na en une somme Na1 + Na2 telle que $N_l \leq 2^{Na1}$ et $Nc \leq 2^{Na2}$.

On a aussi $N_l = [\underline{Nm}]$ ou [ ] est la fonction majorant entier
Nc

On obtient les formules de calcul de surface suivantes :

l_c = Largeur de la zone des transistors de charge

$l_1 = Pa \times Na_1$ (Largeur du décodeur ligne)

$l_2 = Pm_2 \times Fr \times (Nb/2) \times Nc$ (Largeur d'un plan de ROM)

$l_3 = Pa \times Na_2$ (Na_2 nombre de bits d'adresse, l_3 représente la largeur du décodeur colonne)

$L1 = Pm1 \times Fr \times Nl = Pm1 \times [\underline{Nm}] \times Fr$
Nc
(Longueur du décodeur ligne et du plan mémoire)

$L_2 = Pd \times Nc \ (Nc \geq 2)$ (Longueur du démultiplexeur)

$L_3 = Pm1 \times Fr \times Nc \ (Nc \geq 2)$ (Longueur du décodeur colonne)

<u>La surface de la ROM est constitué des trois surfaces suivantes.</u>

- Surface du décodeur ligne (E 4.1)
  $S_D = (L_1 + 2 \times l_c) \times L_1$

- Surface de la zone de mémorisation (E 4.2)
  $S_M = (L_1 + lc) \times 2 \times l_2$

- Surface du démultiplexeur avec le décodeur colonne (E 4.3)
  $Sx = L_2 \times 2 \times l_2 + (l_3 + 2 \times lc) \times L_3$; $Nc \geq 2$
  $S_{ROM} = S_D + S_M + Sx$ (E 4.4)

Dans le cas de la microprogrammation horizontale d'une ROM, le nombre d'états à générer se traduit par un nombre de mots à introduire dans la ROM. Etant donné que les sorties ne sont fonction que de l'état, le nombre de colonnes est lié au nombre de commandes à générer. Le rapport entre la largeur de la ROM et sa hauteur se détermine par le facteur Nc, qui détermine le nombre de mots implantés dans un ligne de ROM. Ce facteur de type topologique, a une influence importante sur la logique de commande de la ROM. En effet, le facteur Nc influence le codage de l'adresse d'appel d'une microinstruction dans la ROM. Ce codage est réparti au niveau du décodeur ligne, qui choisit la ligne sur laquelle se trouve la microinstruction désirée, et sur le démultiplexeur de sortie qui choisit entre les différentes microinstructions sélectionnées dans la ROM. Si ce facteur est supérieur à un, alors le nombre de colonnes sera égal à deux fois le nombre de commandes à générer, et le nombre de lignes à la moitié du nombre d'états. Par contre la surface du multiplexeur en sortie augmente parrallèlement à ce facteur Nc. De la même façon, le décodeur ligne subit une diminution de sa complexité.

Il est à remarquer que la taille du démultiplexeur de sortie ne doit pas être trop importante. Comme il est construit en technologie MOS, à l'aide de logique à interrupteurs, et qu'il introduit un facteur de non régularité. De plus le signal de commande du démultiplexeur doit être amplifié de manière croissante. La vitesse de commande est ralentie. Selon [OBRE 82], le facteur Nc=2 est optimum dans le cas d'une ROM microprogrammée horizontalement. C'est à dire que les microinstructions sont groupées par deux mots sur une ligne de ROM. Dans le cas du 6800, où la ROM est microprogrammée verticalement, le facteur NC est égal à 16 dans la MicroROM, et est égal à quatre dans la NanoROM.

PLA de séquencement:

Pour des raisons de surface, il est souvent utilisé un PLA de séquencement et de traitement des conditions. Ce PLA permet de traiter les conditions de branchement, et de réduire ainsi la taille de la ROM. La surface est fonction du nombre de traitements particuliers de branchement de l'algorithme d'interprétation. Dans le cas d'une microprogrammation horizontale, ce PLA est destiné aussi à traiter le séquencement des microinstructions. Son facteur de déformabilité est fonction du nombre de monômes qu'il est possible de dupliquer [CHUQ 84].

PLA de paramétrisation:

Dans le cas de la microprogrammation verticale, les champs de sortie de la ROM étant codés, on ajoute après le démultiplexeur de sortie, des PLA's de paramétrisation. Ces PLA's sont chargé de décoder le contenu des champs de la ROM. Le gain en surface est donc lié à la paramétrisation possible de la ROM, ainsi qu'au nombre d'états destinés à être intégrés à la ROM.

La surface du (ou des) PLA de paramétrisation est facteur du taux de paramétrisation introduit au niveau de la ROM. Ce facteur est lié aux types d'instructions à générer, et donc à l'algorithme d'interprétation du circuit. Il est rappelé que le PLA de paramétrisation est essentiellement utilisé dans le cas de l'attaque d'une partie opérative composée de plusieurs modules fonctionnels. Dans le cas du 68000 de Motorola, les PLA's de paramétrisation permettent de traiter les conditions de branchement, à partir de l'information contenue dans les champs codés de la ROM, et des conditions extèrieures de branchement. L'automate du 68000 est de type Mealy.

Démultiplexeurs de sortie:

La taille des démultiplexeurs de sortie de la ROM est liée au nombre de mots par lignes (Nc) de ROM. Ce facteur Nc influe sur la déformabilité de la ROM, permettant d'élargir celle-ci. Ce facteur Nc est selon [OBR 82] ,optimum s'il est égal à deux dans son essai sur l'implantation de la partie contrôle du M6800.

Nous n'allons pas présenter de schéma topologique pour l'implantation d'une solution de ROM à microprogrammation horizontale, considérant que ce n'est qu'un cas particulier de l'implantation d'une ROM à microprogrammation verticale.

L'implantation de la structure se fait par un ajustement de la taille du PLA de séquencement sur la ROM, elle-même séparée en deux parties égales. Cette séparation est introduite pour diminuer la longueur des lignes de la ROM, et améliorer ainsi le facteur temps de propagation et de traitement de la ROM. Le décodeur ligne est inséré entre les deux parties de la ROM, le décodeur du démultiplexeur étant implanté sous le décodeur ligne de la ROM. Le démultiplexeur, séparé en deux parties correspondant aux plans de ROM, est implanté sous les deux plans de ROM. Le passage des informations pour le démultiplexeur de sortie de la ROM se fait à travers le décodeur ligne par transparence. Le deuxième intérêt du placement du décodeur ligne au milieu de la ROM provient du fait du meilleur routage entre les sorties du PLA de séquencement et des entrées du décodeur ligne, avec une diminution de la longueur moyenne des lignes de routage, donc de la capacité et de la résistance, et ainsi de la vitesse de propagation.

Le placement du PLA de séquencement et de traitement des conditions au dessus de la ROM est proposé, car il favorise la possibilité de duplication des monômes et ainsi d'optimisation de sa surface. En contrepartie, un tel placement demande l'utilisation d'un outil type PAOLA permettant les entrées/sorties latérales.

Le (ou les) PLA de paramétrisation sont implantés sous la ROM de microprogramme, si possible, dans le cas ou il attaque différentes unités fonctionnelles, de topologie adaptée à ces unités fonctionnelles (cas d'une partie opérative).

Les formules de calcul de surface des PLA's sont les mêmes que celles présentées dans les solutions à plusieurs PLA's (§III .1, E2.X; X de 0 à 6).

Schéma topologique (figure 3.27)

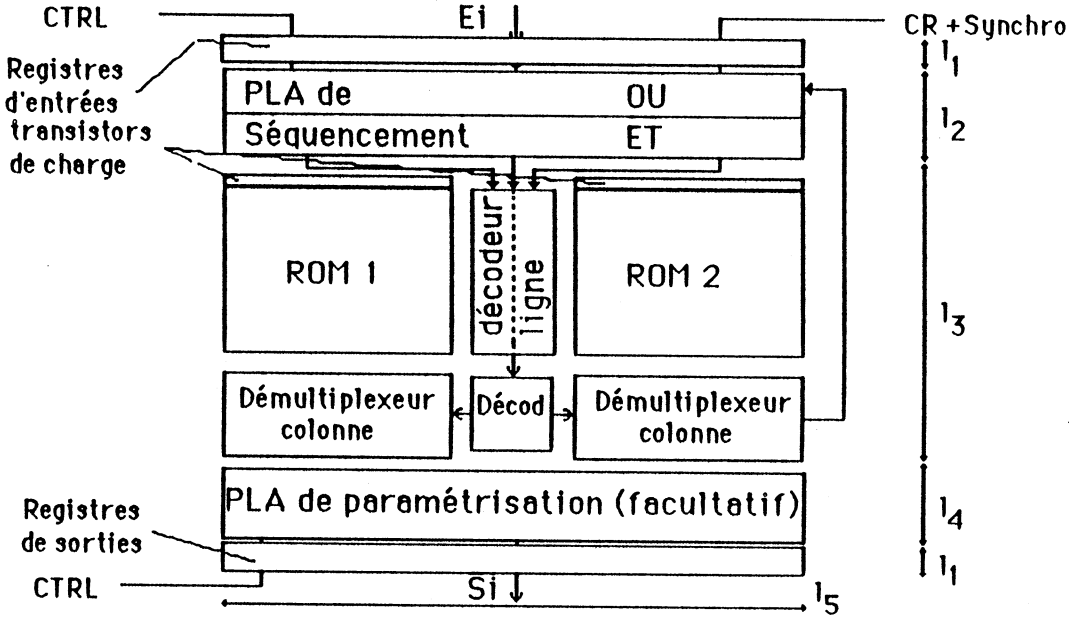


fig. 3.27 Modèle topologique d'une partie contrôle à ROM (microprogrammation verticale)

Hauteur et largeur des PLA's de paramétrisation et de séquencement:

Soit

Nctrlin: Nombre de contrôles entrant dans le PLA de séquencement
Ncomseq: Nombre de commandes provenant du niveau supérieur
Nseq: Nombre de fils de séquencement internes
Ncrin: Nombre de fils de comptes rendus entrant dans le PLA de séq.
Nseqrom: Nombre de fils indiquant l'état de la ROM
Nseqétat: Nombre de monômes du PLA de séquencement
Nseqout: Nombre de fils de commande des décodeurs ligne et colonne
Nparin: Nombre de fils provenant de la ROM et entrant dans le PLA de paramétrisation := N_l/N_c
Nparout: Nombre de fils de commandes pour le niveau inférieur
Nparétat: Nombre de monômes du PLa de paramétrisation

Alors
Largeur_seq := 4.PME+ (Nseqétat+Nseqétat/M).PME+MAX(DY_1,DY),
Hauteur_seq := (Nctrlin+Ncrin+Nseq+Nseqrom+Ncomseq)*2*PE+
(Nseqout+Nseq+(Nseqout+Nseq)/M).TRN.PME+DY'+DY_1'

Largeur_par := 4.PME+ (Nparétat+Nparétat/M).PME,
Hauteur_par := (Nparin)*2*PE+
(Nparout+(Nparout)/M).TRN.PME+DY'+DY_1'

Surface de l'implantation de la solution à ROM:

La surface de la ROM avec ses décodeurs lignes, et démultiplexeurs est calculée à l'aide des équations E 4.X, X de 1 à 4.

l_1 := Hauteur des registres d'entrées:= DY_2 (cf §III 3.1.4)
l_2 := Hauteur_seq
l_3 := Hauteur de ROM avec charges et multiplexeurs (cf E 4.1,...,4)
l_4 := Hauteur_par
l_5 := MAX (Largeur ROM, Largeur_seq, Largaeur_par)

SURFACE_TOTALE := $l_5*(2*l_1+l_2+l_3+l_4)$

4 ETUDE DE LA GENERATION D'UNE PARTIE CONTROLE MONO-PLA

4.1 Modèle optimisé (PLA type PAOLA)

Dans le cas ou l'on utilise un outil de génération de PLA type PAOLA, les notions de transparence et de déformabilité des PLAs sont utilisées. Dans un premier temps, les bus de compte rendus, de synchronisation et contrôle sont conservés à leurs emplacements respectifs, tels qu'ils ont été déterminés au chapitre III 1.3, c'est à dire:

- Le bus de compte rendu se trouve à droite de la partie contrôle
- Le bus de contrôle se trouve à gauche de la partie contrôle,
- Le bus de synchronisation se trouve à droite de la partie contrôle, entre cette dernière et le bus de compte rendu.

La connectique des entrées/sorties est déterminée, sur le pourtour du PLA, par l'environnement qui est imparti au PLA.

Dans le cas de PLA optimisé type "PAOLA", le schéma d'implantation serait le suivant . On trouvera respectivement (figure 3.28):

- Les entrées/sorties destinées au bus de contrôle sur la gauche de la partie contrôle, entrées/sorties de type classique,
- Les entrées/sorties destinées au bus de compte rendu, sur la droite de la partie contrôle, entrées/sorties du type classqiue,
- Les entrées/sorties de séquencement sur la droite de la partie contrôle, entrées/sorties de type classique,
- Les entrées de commande provenant du niveau supérieur se trouvent au nord de la partie contrôle, entrées de type latéral,
- Les sorties de commande pour le niveau inférieur au sud de le partie contrôle, sorties de type latéral.

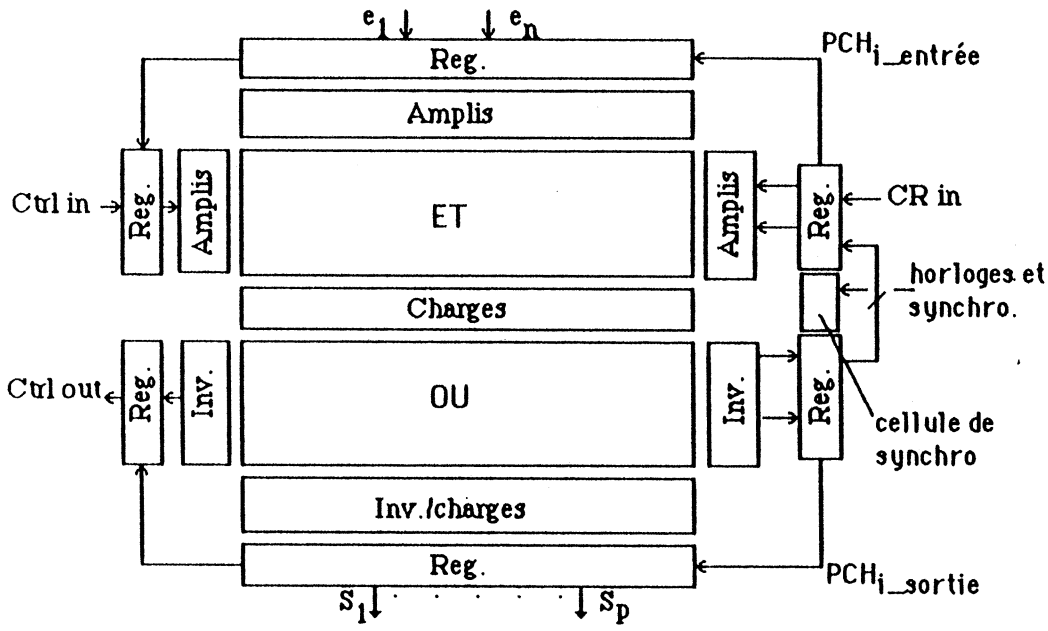


fig. 3.28 Modèle topologique d'une partie contrôle monopla implantée à l'aide de PLA type PAOLA

Surface des registres

La surface des registres est liée à la technologie utilisée, au type de séquencement choisi, à la ligne de sortie que le registre attaque. Le type de séquencement choisi est de type statique, ce qui implique que le registre est assimilable à une bascule D. En technologie NMOS, un registre nécessite au moins 7 transistors (figure 3.29). On prendra les équations définies au chapitre III.3.1.4 pour la largeur et hauteur d'un registre. DX_2 représente la largeur d'un registre, et est égal à deux pas d'entrée d'un PLA classique. DY_2 représente la hauteur du registre, elle est fonction de l'implantation et des performances allouées au registre.

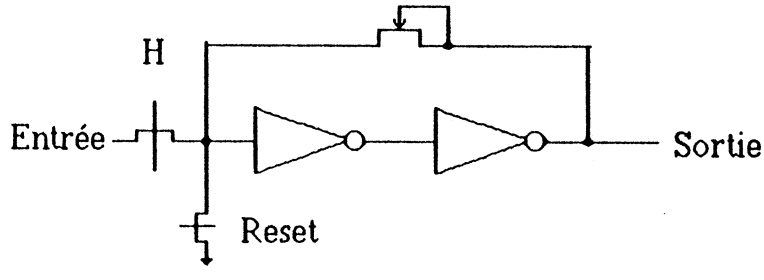


fig. 3.29 Schéma logique d'un point mémoire ou registre

Surface d'une partie contrôle monopla, utilisant des PLA's type PAOLA:

On utilisera les équations E.2.X, x de 0 à 6 du chapitre III 3.1.2.

Soit:

Nctrlin: Nombre de contrôles entrant,
Nctrlout: Nombre de contrôles sortant,
Ncrin: Nombre de compte rendu entrant,
Ncom: Nombre de commandes du niveau supérieur entrant,
Nseq: Nombre de fils de séquencement interne,
Ncomout: Nombre de commande pour le niveau inférieur,
Nplaétat: Nombre de monômes du PLA

Alors

Largeur_pla := 4.PME+ (Nplaétat+Nplaétat/M).PME+2*MAX(DY_1,DY),

Hauteur_pla := (Nctrlin+Ncrin+Nseq+Ncom)*2*PE+
(Nseq+Ncomout+Nctrlout*2+(Ncomout+Nseq+Nctrlout*2)/M)
*TRN*PME+2*MAX(DY',DY_1')

Surface_MonoPLA := (Hauteur_pla+2*DY_2)*(Largeur_pla+2*DY_2)

4.2 Modèle classique (PLA classique)

Lorsqu'on considère le modèle classique d'implantation d'un PLA (cf §III 3.1.1), pour lequel les entrées/sorties ne sont disponibles que perpendiculairement aux monômes du PLA, on est confronté aux problèmes de définir une (ou des) solution(s) topologique(s) convenables vis-à-vis du voisinage du bloc constitué par le PLA classique.

Dans ce cas, nous pouvons définir deux positionnements différents du PLA:

- Soit le PLA est disposé verticalement, c'est à dire que ces entrées/sorties ne sont disponibles qu'au NORD ou au SUD de la partie contrôle construite à l'aide du PLA,
- Soit le PLA est disposé horizontalement, c'est à dire que les entrées/sorties ne sont disponible qu'à l'EST ou à l'OUEST du PLA.

4.2.1 Partie contrôle monoPLA , PLA classique disposé verticalement:

Le PLA étant disposé verticalement, les liaisons entre le PLA et les bus de contrôle, de compte rendu et de synchronisation seront réalisées par routage (voir schéma 3.31). D'autre part, l'ordre des colonnes des matrices ET et OU sera assigné (figure 3.30a,b). Une telle répartition interne des entrées/sorties dans les matrices ET et OU du PLA permet une meilleure prise en compte du routage existant entre la partie contrôle et les bus de contrôle, de compte rendu et de synchronistion. Il permet aussi de prendre en compte les deux solutions pour disposer le PLA d'une partie contrôle monoPLA réalisée à l'aide de PLA classique. En effet, aucun risque de routage à deux couches n'existe, ce qui diminue d'autant la surface de routage. Par exemple, dans un routage à deux couches en technologie NMOS, un des deux niveaux utilisés est du métal: Le métal bien que meilleur conducteur électrique, possède des règles de connexions plus larges qu'un niveau tel que du polysilicium.

L'organisation interne des matrices ET/OU du PLA unique d'une partie contrôle monoPLA est la suivante :

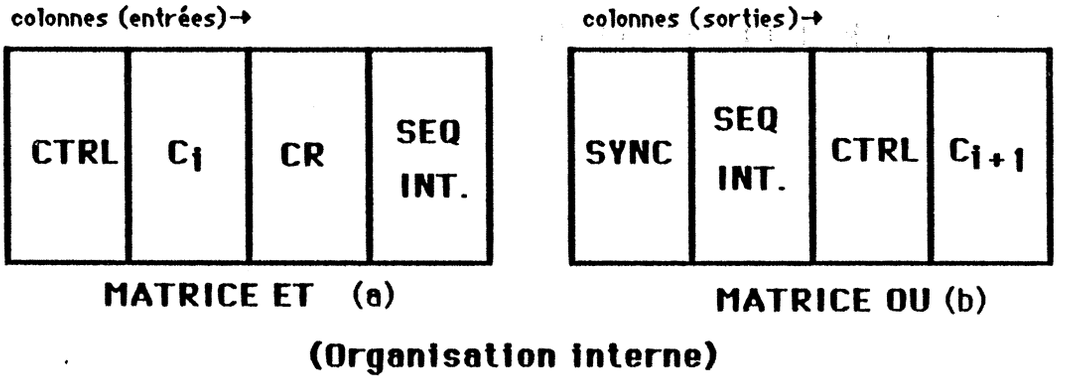


fig. 3.30 Organisation interne des matrices ET et OU d'une partie contrôle monoPLA

Surface du PLA d'une partie contrôle monopla (PLA's classiques):

On utilisera les équations E.1.X, x de 0 à 8 du chapitre III 3.1.1.

Soit:

Nctrlin: Nombre de contrôles entrant,
Nctrlout: Nombre de contrôles sortant,
Nhorl: Nombre de phases d'horloges utilisées par l'étage
Nsyn: Nombre de signaux de synchronisation
Nres: Fil de RESET => Nres:=1;
Ncrin: Nombre de compte rendu entrant,
Ncom: Nombre de commandes du niveau supérieur entrant,
Nseq: Nombre de fils de séquencement interne,
Ncomout: Nombre de commande pour le niveau inférieur,
Nplaétat: Nombre de monômes du PLA
Nfinseq: Information de fin de séquence, pour la cellule de synchronisation

Alors

Largeur_pla := (Nplaétat+Nplaétat/M).PME+2*MAX(DY_1,DY),

Hauteur_pla := (Nctrlin+Ncrin+Nseq+Ncom)*2*PE+
(Nseq+Ncomout+Nctrlout*2+Nfinseq+(Ncomout+Nseq+
Nctrlout*2 +Nfinseq)/M)*TRN*PME

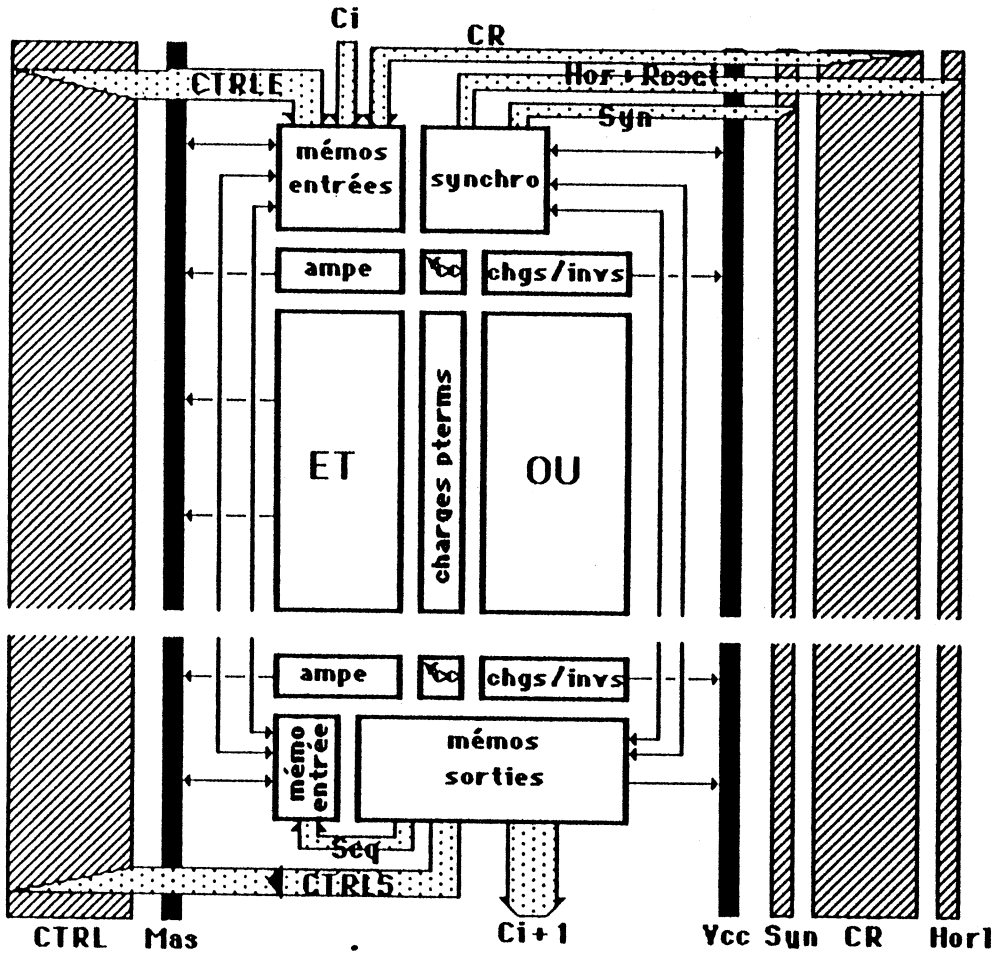


fig. 3.31 Modèle topologique de partie contrôle monopla implantée à l'aide de pla classique disposé verticalement

Surface d'une partie contrôle Monopla utilisant un PLA classique disposé verticalement:

Soit:

l_1: hauteur des interconnexions due à l'amenée des fils de contrôles entrants, de comptes rendus, des fils de synchronisation comprenant les horloges, le RESET, et les signaux de synchronisation définis par le modèle temporel utilisé.

l_1:= MAX(Nctrlin, Ncrin+Nhorl+Nres+Nsyn) * pas de polysilicium

l_2: hauteur des interconnexion due au routage des fils de contrôle et de séquencement interne;

l_2 := (Nctrlout+Nseq) * pas de polysilicium

l_3: hauteur du routage entre les sorties et les point de mémorisation de sortie, ce routage est due à l'adaptation des registres de sortie à deux pas de sortie (DX_2:= 2*pas de sortie)

l_3:=(2*Nseq+Ncomout+Nctrlout*2)*pas de polysilicium

l_4: Largeur due à l'implantation des registres d'entrées/sorties

l_4:=(2*Nseq+Nctrlout+Ncomout)*DX2

Alors

Surface_MonoPLA:=
MAX(Hauteur_pla,l_4)*(Largeur_pla+2*DY_2+l_1+l_2+l_3)

4.2.2 Partie contrôle monoPLA , PLA classique disposé horizontalement:

Le PLA possède le même nombre d'entrées/sorties que la solution précédente. L'ordre de remplissage des matrices ET et OU correspond à celui de la figure 3.30. Par contre, la répartition des entrées/sorties sur le pourtour du PLA classique change:

- Les fils de contrôles entrent et sortent par le SUD du PLA classique, c'est à dire sur la figure 3.32 par l'ouest de la partie contrôle,
- Les fils de compte rendus sortent par le NORD du PLA classique, c'est à dire par l'est de la partie contrôle.
- Les zônes de routage sont dues aux commandes provenant du niveau supérieur, et aux commandes destinées au niveau inférieur.

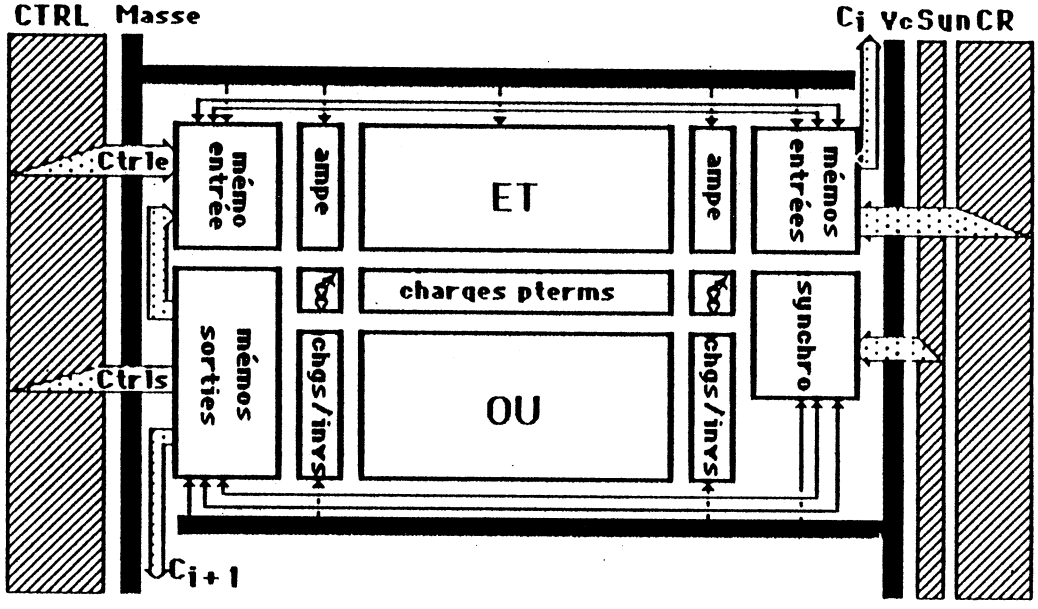


fig. 3.32 Modèle topologique de partie contrôle monopla implantée à l'aide de pla classique disposé horizontalement

Surface d'une partie contrôle Monopla utilisant un PLA classique disposé horizontalement

Soit:

l_1: hauteur des interconnexions dues à l'amenée des fils de commandes entrants, provenant du niveau supérieur.
l_1:= Ncom * pas de polysilicium

l_2: hauteur des interconnexion dues au routage des fils de commandes pour le niveau inférieur;
l_2 := (Ncomout) * pas de polysilicium

l_3: hauteur du routage entre les sorties et les point de mémorisation de sortie, ce routage est due à l'adaptation des registres de sortie à deux pas de sortie (DX_2:= 2*pas de sortie)
l_3:=(2*Nseq+Ncomout+Nctrlout*2)*pas de polysilicium
l_4:=(2*Nseq+Nctrlout+Ncomout)*DX2

Alors

Surface_MonoPLA:=
MAX(Hauteur_pla,l_4)*(Largeur_pla +2*DY_2+l_1+l_2+l_3)

4.2.3 Conclusion

Si l'on compare les trois propositions d'implémentation de partie contrôle MonoPLA, il apparait que l'utilisation d'un PLA optimisé type PAOLA améliore considérablement le gain en surface en raison de:

- La diminution des zônes de routage dues aux entrées/sorties vers les bus de contrôle, de compte rendus ou de synchronisation,
- Une meilleure adaptibilité des entrées/sorties du PLA avec les registres d'entrée/sorties.
- Un compactage des lignes d'entrées/sorties (TRN) du PLA amenant des gain en surface de la matrice ou du PLA pouvant atteindre 30% (figure 3.14).

D'autre part, l'application des solutions à PLA's classiques, et le choix entre l'une ou l'autre des solutions est lié aux nombre d'entrées/sorties de commande avec les niveaux supérieur et inférieur, et le nombre de contrôles et compte rendus entrants et sortants. Si le nombre de commandes est très important, comparativement au nombre de monômes, et amenant la partie contrôle à avoir une forme carrée, on choisira plutôt la solution avec le PLA implanté verticalement.

5 GENERATION D'UNE PARTIE CONTROLE MONO-PLA (PLA Classique)

5.1 Les outils de base

Pour valider l'étude du modèle architectural et topologique général du compilateur de silicium SYCO, un ensemble de logiciels ont été mis au point. Ces logiciels se chargent de construire une partie contrôle à multi-niveaux d'interprétation, puis d'assembler la partie contrôle à la partie opérative, et en dernier lieu de construire les bus de contrôle de compte rendu et de synchronisation attenant. Les types de parties contrôles générés sont de type MonoPLA. Elles utilisent des PLA classiques. Les solutions utilisant un PLA vertical ou un PLA horizontal ont été implémentées.

Pour mettre au point ces logiciels, un certain nombre d'outils de base ont été utilisés. Ces outils, développés au sein de l'équipe d'Architecture des Ordinateurs, ont subi des évolutions nécessaires à leur emploi pour la génération automatique de partie contrôle. Les outils utilisés sont les suivants:

* **LUCIE** : LUCIE est un progiciel d'aide à la conception de masques de ciruits VLSI. Il a été créé en 1979 pour les besoins du CMP [GUYO 79].

* **LUBRICK**: LUBRICK est un ensemble de fonctions d'assemblages noyées dans un environnement PASCAL. La première version a été développée par [SCHO 85] pour l'assemblage automatique des éléments de partie opérative d'un circuit PASCAL "POPI".

* **GENPLA**: GENPLA est un outil de génération automatique de PLA de type classique. Il génère en sortie des fichiers assimilables par les outils LUCIE et LUBRICK. L'étude d'un tel outil a été nécessaire pour suppléer à un manque d'outil de génération automatique de PLA en état.

Contrairement au système LUCIE, dont l'entretien était régulièrement assuré, le système LUBRICK a du subir une importante évolution à partir de la version 0 de base, jusqu'à la version actuelle (version 0.5).

L'ensemble de ces outils est écrit en langage PASCAL. L'exploitation de ces outils peut se faire indifféremment sur les ordinateurs SM90/UNIX et VAX/VMS. Le choix du langage PASCAL pour la mise au point d'une première version provient de sa modularité et de sa structuration.

Dans un avenir proche, des outils reprenant les même fonctionnalités mais écrit en Le-Lisp [CHAI 83] viendront prendre la suite des outils décrits dans ce chapitre.

5.1.1 LUCIE : [PAIL 85]

Le système LUCIE est constitué d'un langage de description et d'un ensemble de programmes associés permettant de définir et de manipuler les masques de circuit intégré [DELO 86]. Le système LUCIE est organisé autour d'une structure de donnée, structure extraite d'un fichier de description textuelle d'une figure LUCIE, à l'aide d'un traducteur.

Le système LUCIE possède les processeurs de traitement suivant (figure 3.33):

- Le digitaliseur LUCIE, produit à l'aide d'une table à digitaliser, et d'un dessin sur feuille quadrillée, un fichier texte source LUCIE pouvant être interprété par le traducteur LUCIE (cf ANNEXE B).

- Le traducteur LUCIE, extrait les informations pour la structure de donnée à partir d'un fichier texte source.

- L'éditeur LUCIE, travaille sur la structure de donnée, et permet la visualisation et la modification interactive d'une figure LUCIE.

- Le traceur LUCIE, à partir de l'information contenue dans la structure de donnée, extrait l'information nécesaire au tracé sur papier à l'aide d'une table à tracer (table BENSON à plume ou électrostatique).

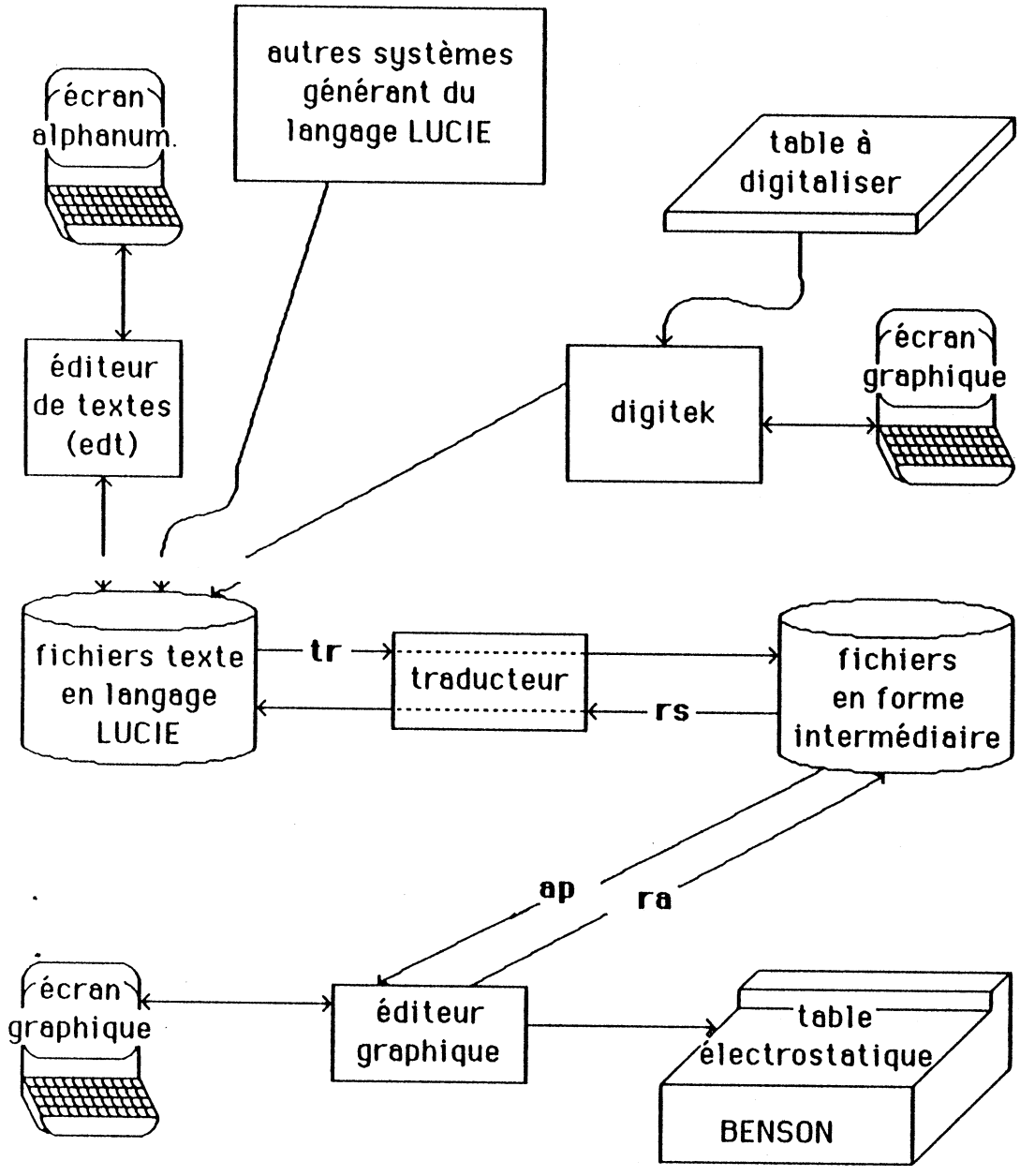


fig. 3.33 Organisation du système LUCIE

* Le langage LUCIE:

L'élément de base du langage LUCIE est le rectangle dont la syntaxe est la suivante :

REC (x, y, dx, dy, niv)
ou x et y sont les coordonnées du rectangle (coin inférieur gauche) dans un repère orthonormé.
ou dx, dy définissent la taille du rectangle
et niv le niveau de masques dans une technologie fixée.

Cet élément est complété par la notion de figure (élément composé par un ensemble d'éléments et d'opérateurs) et par un ensemble d'opérateurs:
- La répétition qui permet d'obtenir la répétition régulière d'un motif en une seule opération,
- La symétrie qui premet d'obtenir le symétrique d'un motif suivant un axe horizontal ou un axe vertical,
- La rotation qui permet de faire tourner un motif de 90° dans le sens trigonométrique ou inverse.

Une description LUCIE est une suite de déclarations d'éléments englobés dans dans une figure. Une figure possède la syntaxe suivante:

FIG <nom>
<suite d'éléments>
FFIG

Une figure est définie comme étant une association d'éléments de base (rectangles) et/ou d'autres figures déclarées précédement; on lui affecte un nom qui permet de la référencer dans une description de figure plus englobante. Toute description LUCIE est ainsi ramenée, par imbrication successives, à une suite de rectangles.

Le repère dans lequel travaille LUCIE se limite aux coordonnées positives. Ces coordonnées positives correspondent au quadrant droit en haut d'un repère en coordonnées cartésiennes (figure 3.34).

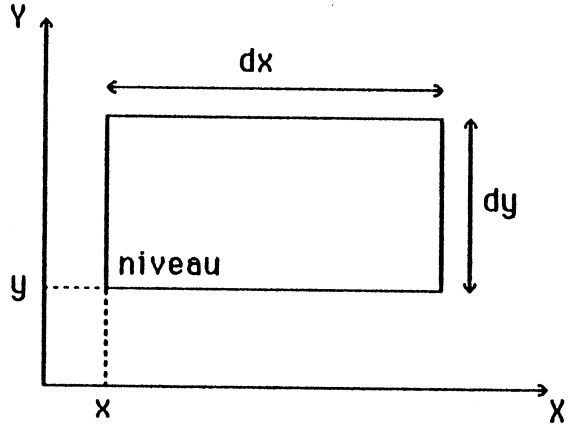


fig. 3.34 Description d'un rectangle en langage LUCIE

5.1.2 LUBRICK : Un assembleur de silicium

Le système LUBRICK dont la version de base a été développée par [SCHO 85], permet l'assemblage hiérarchique de cellules par programme, ainsi que le tracé automatique de leurs interconnexions. Il est assimilé aux générateurs procéduraux de Layout. Le système LUBRICK est noyé dans un environnement PASCAL, et peut ainsi en utiliser toutes les ressources. On distinguera par la suite LUBRICK tel défini par [SCHO 85], et LUBRICK/INPG, correspondant à la version utilisée pour la génération de la partie contrôle et de la partie opérative dans SYCO. La version LUBRICK/INPG est complétée et débuggée. Elle est implantée sous système VAX/VMS et SM90/UNIX SYSTEM V.

L'élément de base de LUBRICK est la cellule. Une cellule est une boîte virtuelle dont on décrit la connectique et ses frontières. Une cellule est hiérarchisée en ce sens que la description peut faire appel à des cellules plus élémentaires ou de niveau inférieur à la figure appelante (figure 3.35).

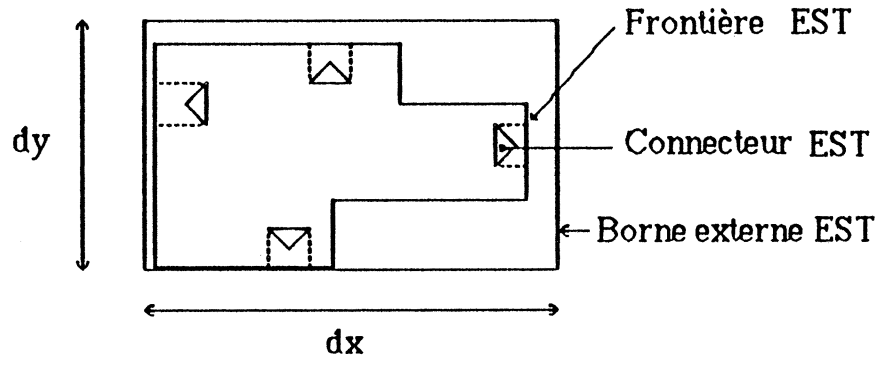


fig. 3.35 Cellule LUBRICK

5.1.2.1 Correspondance entre une cellule LUBRICK et sa description géométrique

Une cellule LUBRICK est telle que l'exécution d'un programme génère sa description. A sa description connectique et d'encombrement correspond une description géométrique. Cette description géométrique est contenue dans un fichier utilisable par un système graphique de dessin de masques. Dans le cas de LUBRICK/TIM3 (implanté sur SM90 ou VAX), le dessin généré est de type LUCIE.

5.1.2.2 Définitions de base de LUBRICK

* Frontières d'une cellule :

Une cellule LUBRICK est assimilée à n polygônes dont les côtés, appelés frontières, sont colorés. La coloration correspond à un niveau de la représentation géométrique de masque. Il existe donc des frontières pour chacun des niveaux géométriques de la cellule décrite. Une frontière est composée d'une liste de segments orientés (Nord, Sud,Est ou Ouest) (figure 3.36).

Les frontières sont utilisées pour effectuer le placement automatique de deux cellules par aboutement à la garde minimale éxigée par la technologie. Les frontières sont définies par rapport au bord englobant de la figure géométrique déclarée.

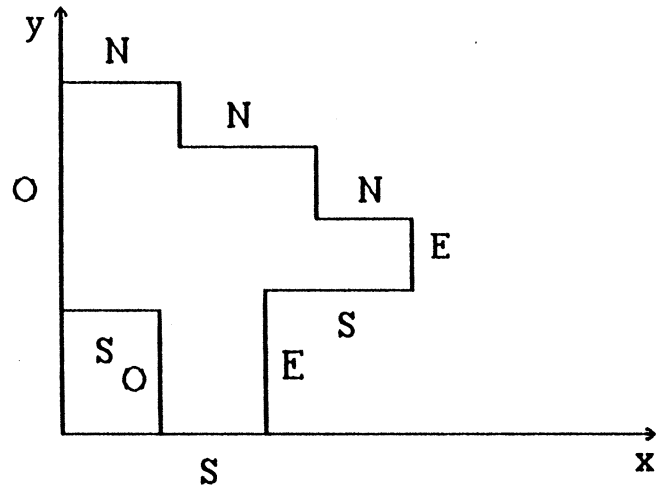


fig. 3.36 Frontières polysilicium d'une cellule LUBRICK

* Connecteurs d'une cellule:

Une cellule quelconque possède des points de communication avec l'extérieur. Ces points de communication ou segments de connexion correspondent soit à des points d'entrées/sorties, soit à des points bidirectionnels. Cette différence n'apparait pas au niveau de LUBRICK, car LUBRICK ne posséde pas de description fonctionnelle des cellules qu'il manipule. Ces segments de connexion sont nommés connecteurs et sont orientés soit au Nord, Sud, Est ou Ouest.

Pour permettre d'accéder à ce segment de connexion, s'il est interne à la cellule (à l'intérieur des frontières), l'hypothèse restrictive suivante est donnée:

Hypothèse sur un fil de connexion:

"Il existe un chemin rectiligne, perpendiculaire à l'axe frontière, pour acheminer un fil de connexion depuis la frontière jusqu'au segment de connection" (figure 3.37).

Ce chemin sera automatiquement généré au momet de l'assemblage de la cellule avec un autre bloc (ou cellule) déclaré en LUBRICK.

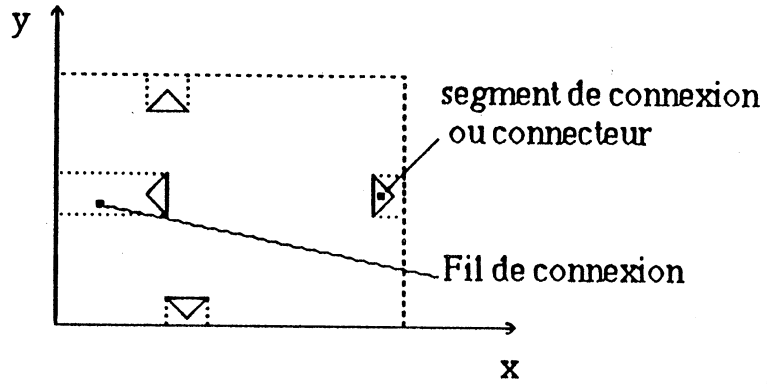


fig. 3.37 Connecteurs d'une figure LUBRICK

Un connecteur LUBRICK possède une liste d'attributs (un seul dans le cas de LUBRICK/INPG) permettant de lui associer des règles de construction pour l'assemblage.

5.1.2.3 Description d'une cellule LUBRICK

La description d'une cellule LUBRICK se fait sous forme textuelle. Cette forme textuelle peut être entrée manuellement ou à l'aide d'un digitaliseur (cf ANNEXE B). Cette description peut correspondre à une cellule de base, ou au résultat de l'exécution d'un programme LUBRICK réalisant un assemblage donné de cellules de base.

La syntaxe de la forme textuelle est la suivante:

fig <name>
<dx> <dy>
c<e,w,s,n> <nb connecteurs>
(nb fois) <offset> < position> < largeur> < niveau> < type>
b<e,w,s,n> <nb segments de frontières>
(nb fois) <offset> < position> < largeur> < niveau>
end

Dans la version LUBRICK/INPG, la frontière d'une cellule ne peut être décrite que pour un seul niveau. En outre, cette frontière n'est constituée que de quatres segments décrivant le pourtour de la cellule:

```
fig toto
  10 10
    ce 1
    0 0 2 3 0
```

```
    be 1
    1 0 0 3 0
    bw 1
    0 0 0 3 0
    bn 1
    2 0 0 3 0
    bs 1
    1 0 0 3 0
ffig
```

Cette description correspondra à la figure suivante (figure 3.38):

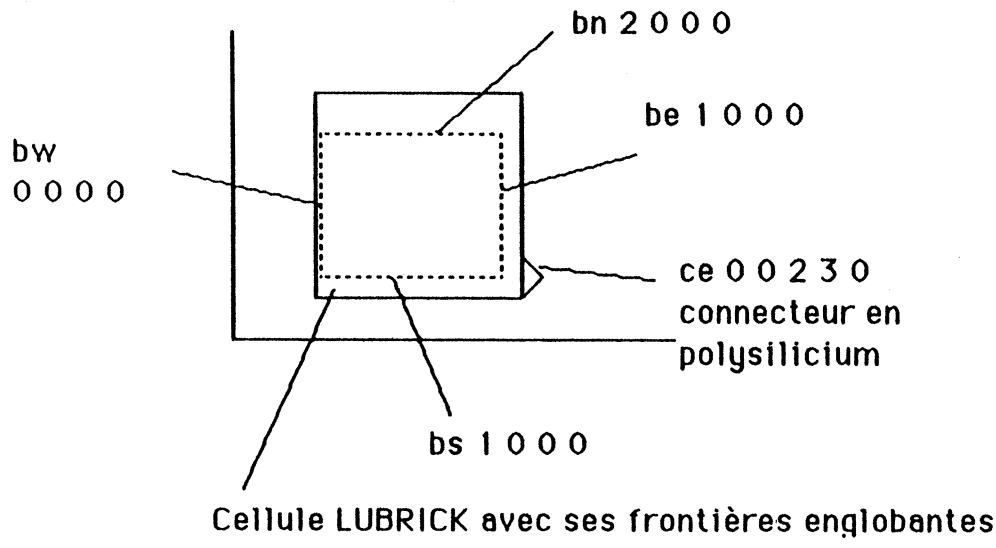


fig. 3.38 exemple de figure LUBRICK

5.1.2.4 Les opérateurs LUBRICK :

LUBRICK fournit un certain nombre de primitives d'assemblage, ainsi que des fonctions plus complexes de routage. C'est à l'aide de ces fonctions que l'utilisateur génère un programme d'assemblage.

Les primitives d'assemblage sont UP et RIGHT, CUP et CRIGHT:

UP et RIGHT permet de proceder à un assemblage d'une cellule A avec une cellule B par abouttement vers le haut (vers la droite). En effet, LUBRICK étant défini par rapport à une syntaxe LUCIE, l'assemblage vers le sud et vers l'ouest n'étant pas permis pour des raisons de coordonnées négatives. Les frontières Nord (EST) de A et Sud (OUEST) de B sont alors rapprochées jusqu'à la garde minimum, puis les connecteurs sont réduits si et seulement si il y a correspondance (figure 3.39).

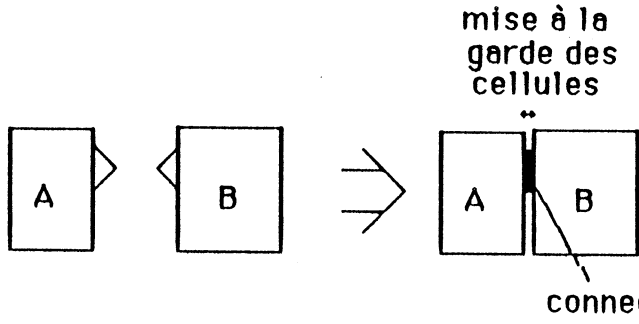


fig. 3.39 Cellules connectées par aboutement avec UP (A,B)

CUP et CRIGHT permettent de réaliser un assemblage de deux cellules possédant des connecteurs non correspondants. L'assemblage est réalisé à l'aide d'un canal de routage ou les connecteurs sont reliés par dévoiement (figure 3.40).

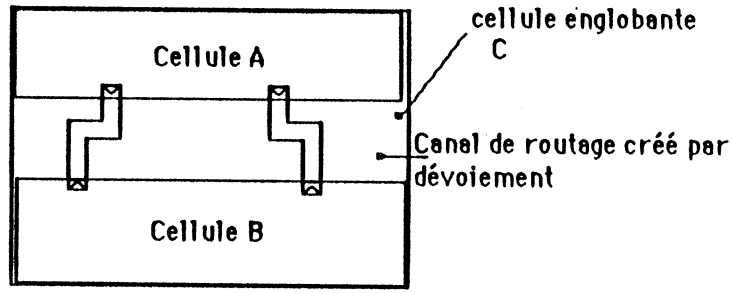


fig. 3.40 Fonction CUP

5.1.2.5 Version actuelle de LUBRICK/INPG

La version utilisée de LUBRICK/INPG est constituée d'un ensemble de modules. Ces modules regroupent les fonctions LUBRICK par type d'action. Le regroupement de ces fonctions a été effectué pour simplifier l'utilisation de LUBRICK. En effet, il s'est avéré que dans l'utilisation des fonctions LUBRICK, un grand nombre de ces fonctions restaient inutilisées pour certaines applications. Il existe neuf modules d'applications. Ces neufs modules sont les suivants:

- Module BRICKPROC.p: Ce module regroupe les fonctions de connections de base. Ces fonctions permettent l'assemblage par dévoiement, et par aboutement (UP,RIGHT,CUP, CRIGHT). Il est à noter que dans la version actuelle de LUBRICK, les fonctions CUP et CRIGHT permettent de connecter par dévoiement un nombre quelconque de niveau. En outre certaines fonctions étudiées pour suppléer aux limitations du LUBRICK/INPG ont été ajoutées: Ces fonctions permettent de rajouter ou d'enlever des connecteurs à un cellule (ADDCONN,SUPCONN), ou de changer le type du connecteur d'une cellule (CHANGECONN).Il existe aussi une fonction de création d'une nouvelle

instance d'une figure déjà existante (DUPLIFIG).

- Module BUILTPROC.p: Ce module regroupe les fonctions de base utilisées par les autres fonctions LUBRICK. Ces fonctions permettent le traitement des connecteurs ou des frontières. Il existe une fonction d'initialisation de la technologie utilisée. Cette fonction récupère dans un fichier (technnologie.nmos, ou technologie.cmos) les informations sur les tailles minimales des niveaux, sur les gardes...

- Module DELTAPROC.p: Ce module regroupe les fonctions de travail sur les connecteurs ou les frontières, permettant d'obtenir des informations particulières (NBCONN). Il est déconseillé d'utiliser ces fonctions qui travaillent directement sur la figure interne de LUBRICK.

- Module FIGURPROC.p: Ce module regroupe les fonctions d'appel et de rangement d'une figure LUBRICK (GETFIG, PUTFIG, OPENFIG, CLOSEFIG). En outre, il intègre la fonction de choix de la technologie (TECHNO, FTECHNO). L'intérêt de la fonction FTECHNO provient de la possibilité de changer de technologie en cours de travail si nécessaire.

- Module INTERPROC.p Ce module regroupe les fonctions d'ouverture et de fermeture de fichiers (FCONNECT, FCLOSE). Ces fonctions ont été écrites pour permettre une translation rapide des programmes écrit en PASCAL et utilisant les fonctions LUBRICK d'une machine VAX à une machine type SM90 par exemple. Si ces fonctions sont utilisées, un programme ne demandera que peu de travail pour changer de machine.

- Module LUCIEPROC.p: Ce module regroupe les fonctions d'interface vers le système LUCIE. C'est à dire qu'il possède des fonctions créant du texte LUCIE (GENFIG, GENFFIG, CALLREC, GENREP...).

- Module PLACEPROC.p: Ce module regroupe les fonctions de manipulation d'une cellule LUBRICK, telle sa rotation (ROTP, ROTM), sa symétrie (SYMX, SYMY), sa répétition (REPX, REPY).

- Module ROUTEPROC.p: Ce module regroupe les fonctions de connections par routage deux couches. L'ensemble de ces fonctions n'a pas été validé (CRUP, CRRIGHT).

- Module SPLITPROC.p: Ce module regroupe des fonctions développées pour l'assemblage de cellules en plusieurs couches. Elles sont l'oeuvre de [JIAN 86].

La version actuelle de LUBRICK est composée d'environ 5000 lignes PASCAL, et tourne indifféramment sous système VAX/VMS et SM90/UNIX. Sa remise à niveau à nécessité environ deux mois de travail à temps plein.

La version LUBRICK/INPG possède un certain nombre de limitations:

1) La description d'une figure LUBRICK ne permet pas de décrire ses frontières autrement que par un rectangle englobant limité à un niveau de masque.

2) Le type est le seul attribut possible pour un connecteur, il serait pourtant intéressant d'associer à un connecteur une correspondance électrique et/ou logique, permettant de remonter des informations logico-électrique jusqu'à la cellule mère du circuit. Une telle réalisation [ODRI 86] est en cours entrainant la liaison entre le simulateur "switch" RNL [MIT] et LUBRICK.

3) Une figure LUBRICK ne connait pas la notion de transparence (ou Ombre). La notion d'ombre ou de transparence exprime qu'il existe un chemin à travers une cellule, chemin sur lequel un fil de connexion peut passer sans modification de la fonctionnalité de la cellule. Cette notion permet pourtant de lever des ambiguités sur la réduction des connecteurs. En effet, si un connecteur d'une cellule A (cellule A abouttée à une cellule B) ne possède pas de connecteur vis à vis dans la cellule B, alors ce connecteur est prolongé jusqu'à la frontière de la cellule englobante résultat. Ceci amène une ambiguité sur la modification possible de la cohérence electrique de la cellule B ainsi traversée.
Par exemple, si un cellule A possède deux connecteurs EST, et la cellule B un seul connecteur OUEST, seul un connecteur de la cellule A est réduit par assemblage avec la cellule B, et le deuxième connecteur EST de la cellule A est prolongé à travers la figure B. Il n'y a pas à ce moment de vérification pour éviter une modification fonctionnelle de la cellule B (figure 3.41a).
La notion d'ombre permettrai de lever cette ambiguité, car si un connecteur ne trouve pas de passage (transparence ou ombre dans la cellule aboutée), il est alors éliminé de la table des connecteurs de la cellule englobante résultat. En reprenant l'exemple précédent, le deuxième connecteur EST de la cellule A serait éliminé de la figure résultat englobeante, et ne traverserait donc pas la cellule B (figure 3.41b).

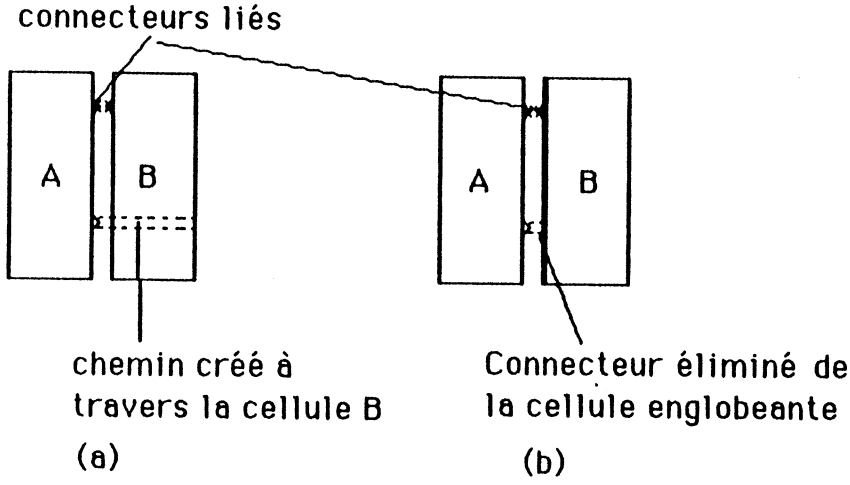


fig. 3.41 Ajoût de la notion de transparence

5.1.3 GENPLA : Générateur de PLA

GENPLA est un générateur de PLAs classiques produisant en sortie du texte LUCIE et LUBRICK. Il a été conçu dans le cadre de la génération automatique d'une partie contrôle. Son rôle est de suppléer à l'absence d'un outil de génération automatique de PLA optimisé dans une version déboggée.

Ce générateur de PLA a été concu pour être compatible en entrée avec le générateur de PLA PAOLA [CHUQ 84]. Cette restriction sur la conception de l'outil a été faite pour permettre un passage rapide de la version actuelle du compilateur de partie contrôle utilisant le générateur GENPLA, à une version utilisant un outil plus puissant de génération automatique de PLA : PAOLA.

5.1.3.1 Processus de génération:

Les matrices ET et OU du PLA à générer sont décrites dans un format d'entrée type PAOLA, dans lequel l'ensemble de la matrice de point de transistors est décrite. A partir de cette première description, le générateur GENPLA génère une deuxième forme, ou les connecteurs sont de manière virtuelle disposés autour du PLA. Cette deuxième forme reste compatible avec le format PAOLA. Elle correspond à une description de la connectique du PLA, et permet de générer la forme LUBRICK de sortie du générateur GENPLA. Cette disposition des connecteurs est définie à l'aide du fichier commande utilisé par le générateur des étages de partie contrôle GENPC; ainsi, le PLA aura ses connecteurs face à leurs points d'utilisation. A partir de cette deuxième forme textuelle, le générateur de PLA produit les fichiers LUCIE et LUBRICK de description géométrique et d'assemblage (figure 3.42).

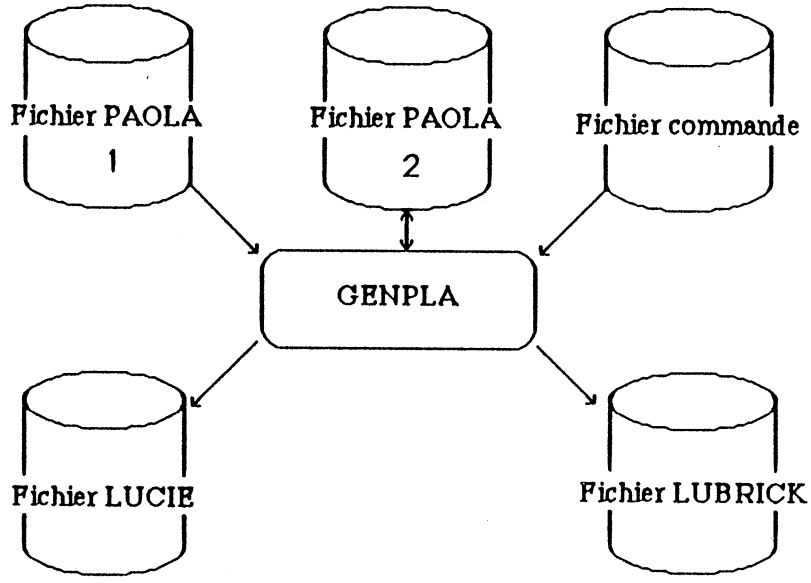


fig. 3.42 Programme GENPLA

5.1.3.2 Format d'entrée de type PAOLA:

Le format d'entrée de type PAOLA décrit permet la description d'une matrice de PLA sous une forme de points. Chaque point est soit un 'zéro', soit un 'un'. Un "1" correspond à la présence d'un transistor dans la matrice concernée. Une ligne de la matrice correspond à un monôme du PLA, et une colonne soit à une entrée, soit à une sortie, cela étant fonction de la matrice concernée.

Pour la matrice ET, il est essentiel de décrire toutes les entrées du PLA, l'entrée et son complément devant être décrits. En effet, l'outil de génération de partie contrôle GENPC considère qu'une entrée est toujours accompagnée de son complément.

La première forme de type PAOLA est représentée sous le format suivant (figure 3.44), dans laquelle, l'attribut CONNECT n'existe pas. En effet, cet attribut correspond à la description de la connectique du PLA. Dans la deuxième forme de description textuelle, cet attribut CONNECT existe. Il est généré par la première passe de l'outil GENPLA (figure 3.43).

NAME <word>/
MATRIX <matrix>/
ROWS <nombre de ligne>/
COLUMNS <nombre de colonnes>/
CONNECT <emplacement des connecteurs d'entrée/sortie>/
CELLS
<matrice de bit>
END
<emplacement des connecteurs d'E/S> ::=
UP <rangs>/
DOWN <rangs>/
LATERAL <rangs>
<rangs> ::= <rang>/ <rang><rang>
<rang> ::= <position fixe>/<position fixe> : <entier>
<position fixe> ::= <numero de segment> < entier>
<numero de segment> ::= <entier>
<matrice de bit> ::= <entier> / <matrice de bit> <entier>

fig. 3.43 Forme d'entrée textuelle

5.1.3.3 Format d'entrée du fichier commande

Le fichier commande permet de définir la position des connecteurs autour des matrices ET et OU du PLA (figure 3.44). Etant donné que les deux formes possibles de placement d'un PLA d'une partie contrôle monoPLA sont les positions horizontale et verticale, le programme GENPLA distribue les connecteurs d'entrée/sortie du PLA à partir de l'information contenue dans le fichier commande. Cette information spécifie le type de position choisi pour le PLA (information **rot,** qui défini la rotation à effectuer sur le PLA, si 0 alors pas de rotation, si 1 alors rotation), ainsi que le nombre de contrôles, de comptes rendus entrant et sortant du PLA. Selon la position du PLA, les entrées des contrôles, des commandes et des comptes rendus ne se fera pas du même coté du PLA (cf §III 4.2).

Le fichier de commande contient toutes les informations relatives à la connectique des différentes tranches de parties contrôles. Il n'est pas lié à la génération d'une tranche de partie contrôle de type monopla, mais peut être utilisé dans le cadre de l'évolution de **SYCO** pour d'autre type de parties contrôles (ROM...).

Sa syntaxe est la suivante:

```
name  <name>     ! nom du circuit à générer
pcn   <integer>  ! nombre de parties contrôles à générer
crn   <integer>  ! nombre de comptes rendus d'exécution sortant de la
                 ! POP
```

```
ctn     <ctr description>
                    ! nombre de fils de contrôles du circuit
pc      <PC description>...<PC description>
                    ! description de la connectique d'une partie contrôle

<ctr description> ::=    < CTRL ext  in> < CTRL ext out>
                          < CTRL ext bi> < CTRL int >
<CTRL ext in> ::=  <integer>
                    ! nombre de fils de contrôle externes entrants
<CTRL ext out> ::= <integer>
                    ! nombre de fils de contrôle externes sortants
<CTRL ext bi> ::=  <integer>
                    ! nombre de fils de contrôle externes bidirectionnels
<CTRL  int> ::=  <integer>
                    ! nombre de fils de contrôle des variables de contrôles

<PC description> ::=
                    <integer> ! numéro de la partie contrôle
                    <integer> ! nombre de fils de séquencement
                    <integer> ! nombre de commandes entrantes
                    <integer> ! nombre de commandes sortantes
                    rot<0/1>   ! type de la partie contrôle à générer
                    cr <integer> ! nombre de compte rendus
                      <integer>...<integer>
                          ! numero des comptes rendus utilisés
                    ctr   en <integer> ! nombre de contrôles entrants
                      <integer>...<integer>
                          ! numero des contrôles utilisés
                        so <integer> ! nombre de contrôles sortants
                        <integer>...<integer>
                          ! numero des contrôles  utilisés
```

5.1.3.4 Algorithme de construction des matrices d'un PLA

L'algorithme de construction des matrices d'un PLA est le suivant:

Début
(lecture du fichier d'entrée de la matrice ET du PLA à générer);
(lecture du fichier commande de la partie contrôle à générer);
(construction de la matrice ET du PLA sans les transistors en fonction des informations paramétrées suivantes: fréquence des rappels de masses, taille des points PLA)
(construction de la matrice OU du PLA sans les transistors en fonction des informations paramétrées suivantes: fréquence des rappels de masses,

taille des points PLA)
(remplissage de la matrice ET du PLA puis de la matrice OU du PLA: mise en place des transistors)
(construction de la figure LUBRICK équivalente à partir des informations du fichier commande)
fin.

a) Limitations du programme acuel:

- Le programme GENPLA ne permet pas les entrées/sorties latérales, c'est à dire des entrées/sorties parallèles aux monômes.
- Le programme GENPLA construit les matrices ET et OU indépendamment, aussi, les hauteurs des matrices ET et OU ne sont pas équivalentes. En effet, la hauteur de la matrice ET dépend du nombre de monômes et du nombre de rappels de masse et la hauteur de la matrice OU ne dépend que du nombre de monômes.
- Les tailles maximums des matrices pouvant être générées sont de 250 entrées/entrées complémentées, 250 sorties, 250 monômes. Ces dimensions peuvent être dépassées par modification des caractéristiques des tables utilisées dans le programme GENPLA.

b) Evolutions du programme GENPLA:

L'évolution du programme GENPLA portera en priorité sur les points définis ci-dessus:

- Les hauteurs des matrices ET et OU devront être mises au même niveau. Ce progrès permet au moment de l'assemblage des matrices et des charges de monômes de gagner une zone de routage entre les sorties des charges de monômes et les entrées de la matrice OU. Pour réaliser cela, il est nécessaire de conserver au moment de la construction de la matrice ET l'emplacement des rappels de masse. Une fois ces emplacements connus, ils sont reportés dans la matrice OU comme des zones d'écartement entre deux monômes.
- Les entrées/sorties latérales sont traitées comme des monômes supplémentaires. Ils induisent de même que les rappels de masse, des zones d'écartement sur la matrice de complément (par exemple report de zônes d'écartement sur la matrice OU dues à des entrées latérales dans la matrice ET). Il est nécessaires d'utiliser les zônes d'écartement déjà existantes dans la matrice pour implanter les entrées ou sorties latérales.
- Une troisième évolution du programme GENPLA, est l'utilisation de ses algorithmes et procédures de base pour construire une ROM. En effet, la zône de mémorisation d'une ROM correspond à une matrice OU d'un PLA. Une ligne de ROM est un monôme de la matrice OU, et une colonne, une sortie de la matrice OU.

5.1.3.5 Conclusion

Le programme GENPLA est écrit en langage PASCAL standard. Il est implanté sur SM90/TELMAT sous UNIX, et est équivalent à plus de1000 lignes de code. Il utilise l'environnement de travail LUBRICK/INPG pour les entrées/sorties vers les fichiers. Son écriture, utilisant une grande partie de fonctions déjà définies dans les programmes GENPC et GENCIRCUIT, a nécessité environ trois semaines de programmation.

Performances

Dans le cas ou le fichier de commandes possède les carctéristiques suivantes, où deux PLA's de partie contrôle sont à générer, les temps d'exécutions suivant ont été relevés sur SM90/UNIX SYSTEM V:

temps réel : 1mn
temps utilisateur : 27 s
temps système : 17 s

Le fichier de commande est le suivant:

```
name cir            ! nom du circuit final
pcn 2               ! nombre total de partie contrôle
crn 5               ! nombre total de fils de compte rendu
ctn 1 1 2 3         ! nombre de contrôles interne en entrée puis
                    ! contrôles externes, en entrée, en sortie puis
                    ! bidirectionnels
pc 1 1 0 6          ! informations relatives à la partie contrôle un
rot 1               ! partie contrôle à un PLA disposé horizontalement
cr 3 1 3 5          ! trois CR avec les numéros des fils
ctr
en 4 1 5 7 9        ! quatre fils de CTRL en entrée avec leur numéro
so 4 2 4 6 8        ! quatre fils de CTRL en entrée avec leur numéro
pc 2 2 6 51
rot 0               ! partie contrôle à un PLA disposé verticalement
cr 3 2 4 5
ctr
en 5 1 5 7 9 11
so 5 4 6 8 10 12
```

Les PLA's possèdent les carctéristiques suivantes:
Matrice ET du PLA1 : 16 entrées, 40 monômes
Matrice OU du PLA1 : 16 sorties, 40 monômes
Matrice ET du PLA2 : 32 entrées, 30 monômes
Matrice OU du PLA2 : 64 sorties, 30 monômes

On obtient alors les matrices de PLA suivantes (figure 3.44a et b)

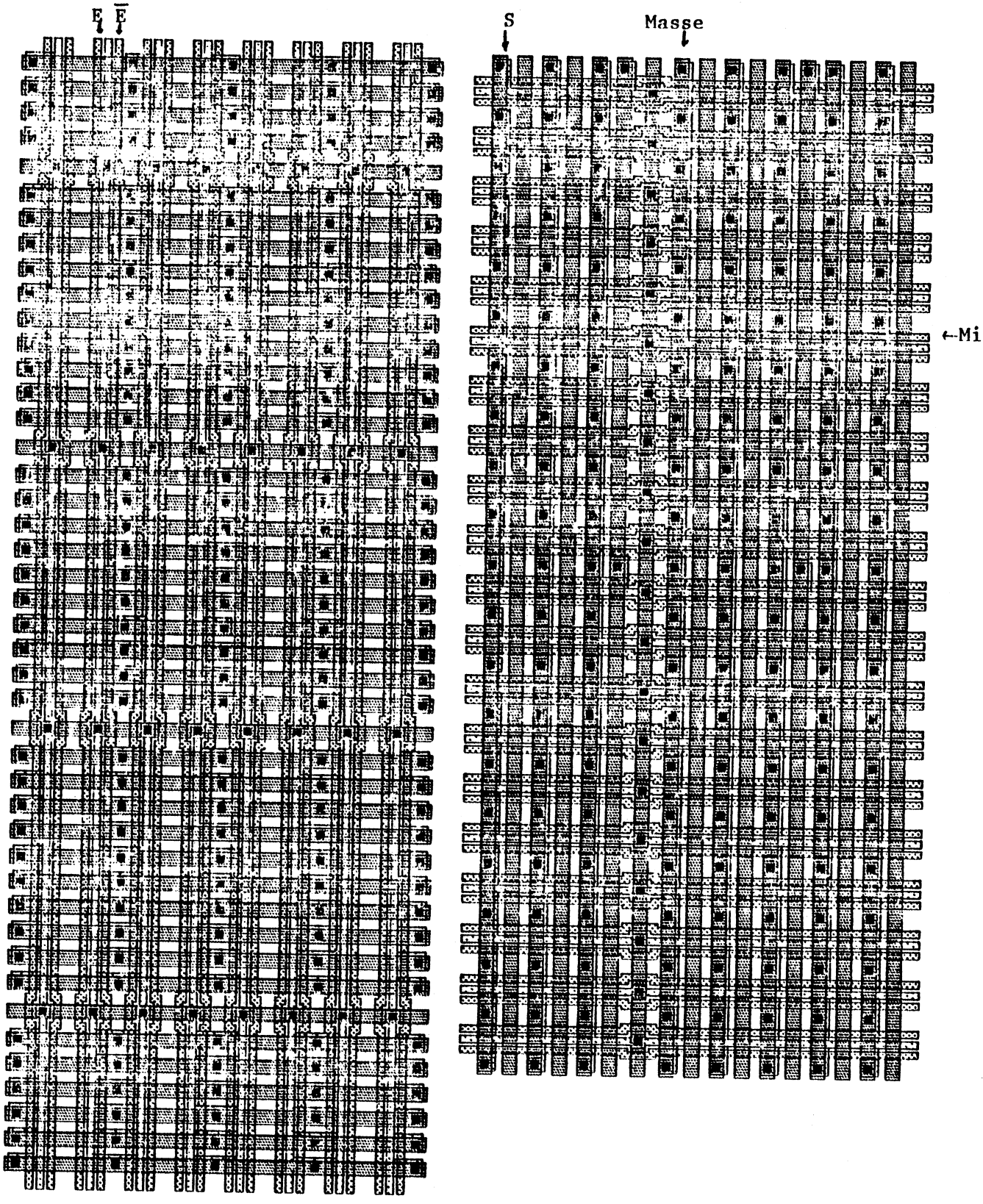


fig. 3.44a Matrices ET et OU du PLA1

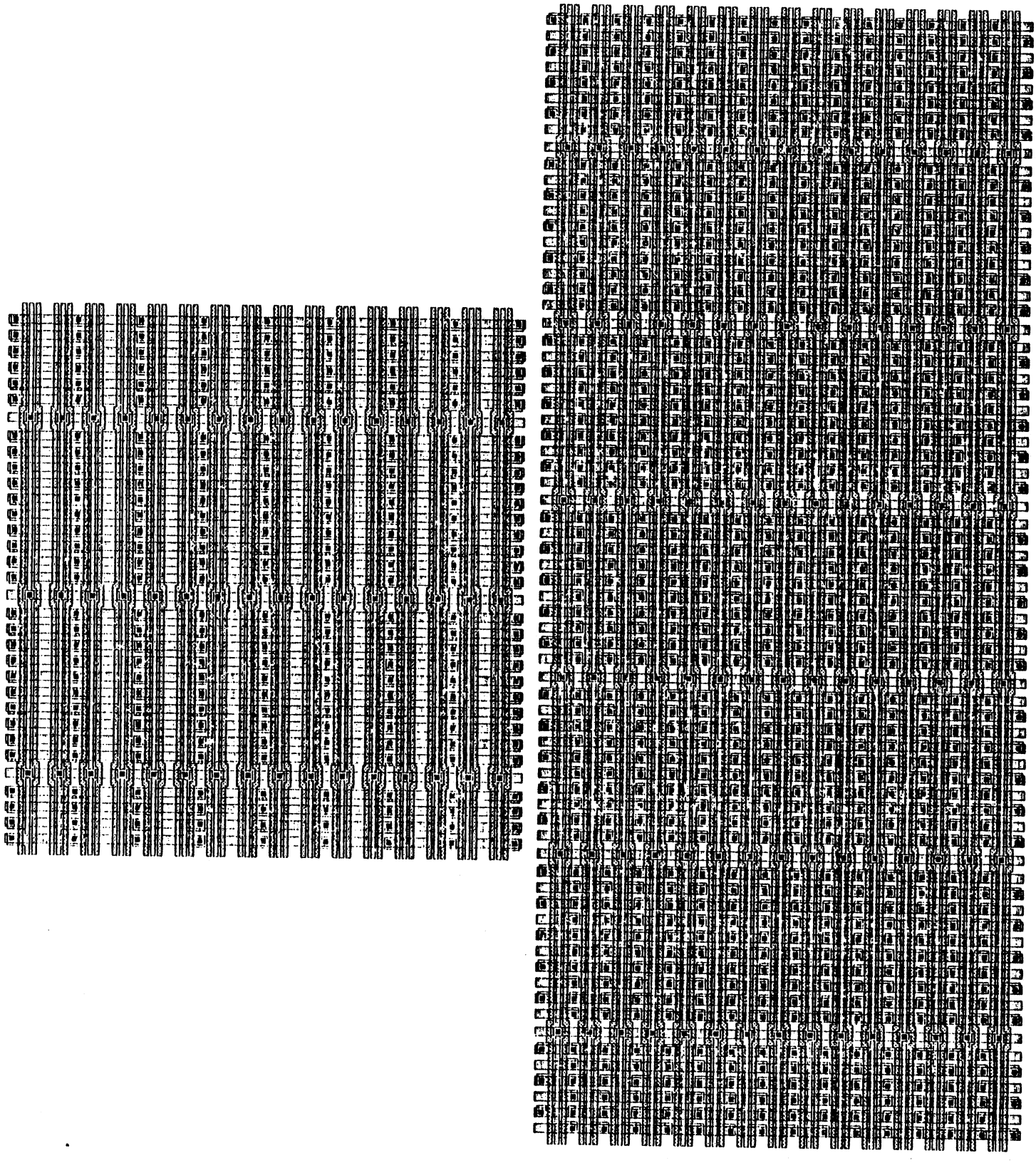


fig. 3.44b Matrices ET et OU du PLA2

5.2 Génération de parties contrôles: GENPC

GENPC est un programme de génération de partie contrôle. Il a été conçu dans le but de valider les options de compilation de partie contrôle définies dans les chapitres II et III. Il permet de produire une partie contrôle de type **SYCO**, équivalente à un ensemble de parties contrôles de type monopla assemblées en tranches superposées.

Le générateur de partie contrôle s'appuie sur un ensemble de logiciels de base, GENPLA, LUBRICK et LUCIE. Il permet de dessiner des parties contrôles de type monoPLA. Etant donné que seuls les PLA de type classique peuvent être générés pour l'instant, seules deux solutions d'implémentation de partie contrôle ont été réalisées: les parties contrôles utilisant des PLA's classiques implémentés verticalement, ou les parties contrôles utilisant les PLA's implémentés horizontalement.

Le générateur de partie contrôle GENPC utilise un certain nombre de cellules de base, qui ont été étudiées et dessinées. Ces cellules de base sont rangées dans une bibliothèque et seront décrites par la suite.

5.2.1 Processus de génération de partie contrôle:

Le générateur GENPC génère à partir d'un fichier commande (cf §III 5.1.3.3), de fichiers LUCIE et LUBRICK des différents PLA's et cellules de base, un fichier LUCIE et un fichier LUBRICK de description géométrique et connectique des différentes sous-parties contrôles d'une partie contrôle à multiniveaux d'interprétation (figure 3.45). Les parties contrôles monoPLA générées ont leur PLA positionné verticalement ou horizontalement selon le choix défini dans le fichier de commande (information **rot** du fichier de commande: si rot=0 alors le PLA est vertical, si rot = 1 alors le PLA est horizontal).

Pour chaque sous-partie contrôle, on répertorie les interconnexions destinées à être dirigées vers les bus de contrôle, de compte-rendu ou de synchronisation.

Une partie contrôle monoPLA est constituée d'un PLA, PLA lui-même constitué de deux matrices. Le programme de génération assemble ces deux matrices, les charges des monômes, les charges des sorties, les amplificateurs puis les registres d'entrées/sorties pour constituer une sous-partie contrôle.

L'ensemble des sous parties contrôles une fois réalisé, l'assemblage destiné à constituer la partie contrôle globale est exécuté. Cet assemblage consiste à assembler les parties contrôles entre elles, en fonction de leur connectique, et de d'orienter les connecteurs concernés (connecteurs des variables de contrôle, de compte rendus...) vers les bus de contrôle, de compte rendu, et de synchronisation. Cet assemblage correspond à un fichier LUCIE et à un fichier LUBRICK générés automatiquement.

A partir de ces fichiers, le programme GENCIRCUIT réalisera le coeur du circuit en y ajoutant les fichiers de partie opérative.

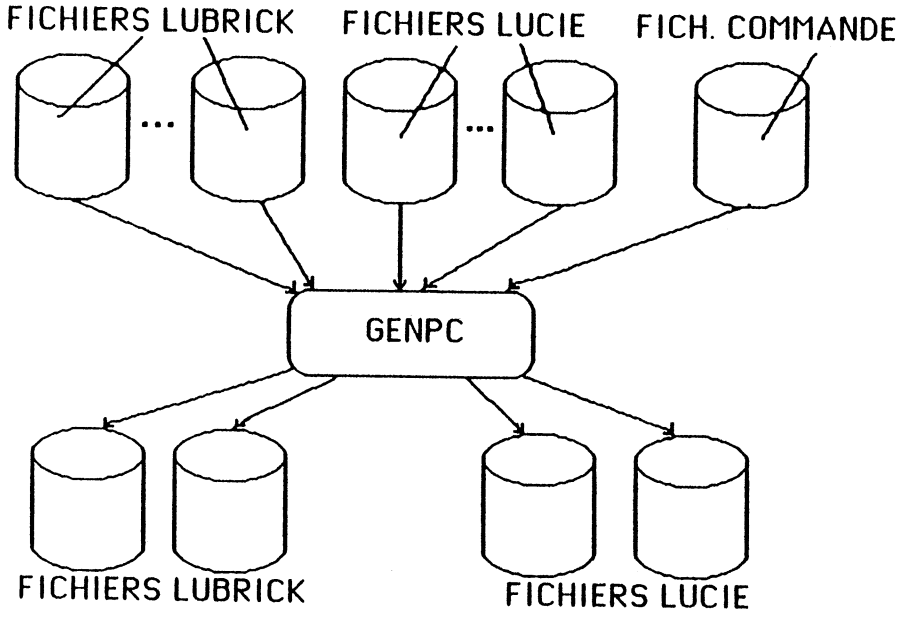


fig. 3.45 Programme GENPC

5.2.2 Algorithme de construction d'une partie contrôle de type monoPLA

L'algorithme de construction d'un bloc de partie contrôle à multi niveaux d'interprétation se déroule en trois phases successives:

Début

- Assemblage des matrices des PLAs avec leurs amplificateurs, inverseurs et charges

- Construction de chaque étage de partie contrôle Monopla à PLA classique en fonction de la position finale de la partie contrôle:

  *PLA avec les monômes verticaux
  *PLA avec les monômes horizontaux,

- Assemblage de tous les étages de partie contrôle avec les connecteurs vers les bus.

Fin

L'algorithme de construction de chacune des phases d'assemblage d'un étage de partie contrôle est le suivant:

Assemblage des matrices de PLA's avec leurs amplificateurs, inverseurs et charges:

Début
(Assemblage de la matrice ET avec les charges de monômes, les charges sont disposées entre la matrice ET et la matrice OU)
(Assemblage de la matrice OU avec l'ensemble constitué de la matrice ET et des charges de monômes)
(Assemblage de ce bloc PLA avec les charges, inverseurs ou amplificateurs d'entrée, selon le type du connecteur)
Fin

Assemblage du PLA avec les points de registres d'entrée, ou avec les portes trois états des variables de contrôle:

En effet, il n'y a pas de points de mémorisation des variables de contrôle sortantes au niveau des sous parties contrôles, car ces variables sont mémorisées dans le bloc des variables de contrôles internes, ou au niveau des plots d'entrées/sorties. Ces points de mémorisation sont modifiés en portes trois états, ayant en entrée le SET et RESET de la variable de contrôle, et en sortie une fonction trois états de ces deux variables telle que:

* Si SET Alors sortie à "1"
* Si RESET Alors sortie à "0"
* Si NON(SET) et NON(RESET) Alors valeur indéterminée

Début
tant qu'il y a des sous parties contrôle à générer **faire**
(Construction du bloc des registres d'entrées et sorties NORD dans le cas d'un PLA vertical, et EST dans le cas d'un PLA horizontal)
(Construction du bloc des registres d'entrées et sorties SUD dans le cas d'un PLA vertical, et OUEST dans le cas d'un PLA horizontal)
(Assemblage de ces deux blocs avec le PLA résulstat de l'exécution de la première phase d'assemblage)
Fin (tant que)
Fin

La troisième phase consiste à assembler les différentes sous parties contrôles entre elles. La partie contrôle qui sera la plus proche de la partie opérative forme la base de l'assemblage. L'algorithme de construction est le suivant:

Début
tant qu' il y a des parties contrôles **faire**
(en fonction de **rot**, faiesubir une rotation à la sous partie contrôle)

(assembler les bus d'alimentation à la partie contrôle)
(construire le routage vers les bus de contrôle, de compte rendus et de synchronisation)
Fin (tant que)
Fin

L'organisation du programme de génération est faite en couche de fonctions.

- La première couche de fonction, est constituée par les fonctions d'entrées/sorties vers les fichiers de commandes, ainsi que des fonctions de la bibliothèque LUBRICK.

- La seconde couche de fonctions concerne les fonctions d'assemblage des blocs (ou cellules) logiques constituant une partie contrôle MonoPLA.

- La troisième couche de fonctions concerne la construction de l'étage de contrôle proprement dit. C'est dans cette couche de fonction que se situera le choix de la structure architecturale de la partie contrôle (structure MonoPLA, MultiPLA, Générateurs de temps...). Elle sera complétée au fur et à mesure de l'implémentation des nouvelles structures.

- La dernière et quatrième couche concerne l'assemblage final des tranches de partie contrôle pour constituer la partie contrôle à plusieurs niveaux d'interprétation du système **SYCO**.

a) Limitations du programme actuel:

Les deux limitations sont pour le moment dues aux logiciels de base utilisés:

* Une première limitation est due à l'outil LUBRICK. Cette limitation provient du fait que l'outil LUBRICK ne gère pas les frontières de type complexes dans sa version actuelle. Une frontière complexe est constituée de plusieurs segments, et existe sur différents niveaux. Ceci entraine des pertes de surface importantes au moment de l'assemblage car seuls des blocs rectangulaires peuvent être assemblés.

* Une seconde limitation provient de l'outil GENPLA. L'outil GENPLA a été conçu dans le but de pouvoir générer des PLA's de type classique, pour la raison de la non disponibilité de l'outil PAOLA de génération de PLA optimisés à entrées/sortiess latérales. Il n'a pas été possible d'inclure en un temps limité toutes les fonctionnalités de l'outil PAOLA dans l'outil GENPLA. Aussi GENPLA ne permet pas de fournir des PLAs optimisés topologiquement. Cela implique des blocs non optimisés à implanter, non déformables, et dont la connectique n'est pas toujours bien adaptée à son environnement. Dans la cas ou l'outil PAOLA serait disponible, la

disponibilité de notions topologiques telles que déformabilité et transparence apporteraient un gain en surface pouvant avoisiner les 30% pour l'implantation des parties contrôles monopla (cf §III 3.1.2).Cette diminution de la surface provenant alors:

- d'une optimisation topologique de la surface des PLA's,
- d'une adaptation du bloc et de ses connecteurs en fonction de son environnement,
- d'une diminution du routage entre cellules de mémorisation et sorties du PLA.

b) Evolution du programme GENPC

Le programme GENPC est destiné à intégrer dans son environnement de nouvelles structures de contrôle (cf §III 3.X). Cette évolution dépendra essentiellement de la disponibilité d'une nouvelle version de LUBRICK, version intégrant les fonctionnalités citées plus avant.

5.2.3 Cellules de base utilisées:

Le programme GENPC utilise un certain nombre de cellules prédéssinées, pour la construction des étages de parties contrôles. Ces cellules sont différentes selon le modèle de partie contrôle utilisé.

On peut faire la remarque suivante, les cellules "critiques" sont paramétrées par la taille du circuit. C'est le cas par exemple des cellules de synchronisation qui fournissent l'horloges à tous les points de registres d'entrées et de sorties, des cellules d'amplification d'entrée du PLA, ou des cellules inverseurs de sortie du PLA.

Dans le cas d'une partie contrôle monopla, les cellules de base utilisées sont les suivantes:

- Cellule de mémorisation sur une horloges: cette cellule sert de tampon entre les différents étages de contrôle, pour respecter le modèle temporel procédural et statique. Elle sert aussi à mémoriser les variables de contrôles binaires (figure 3.46, 3.47).

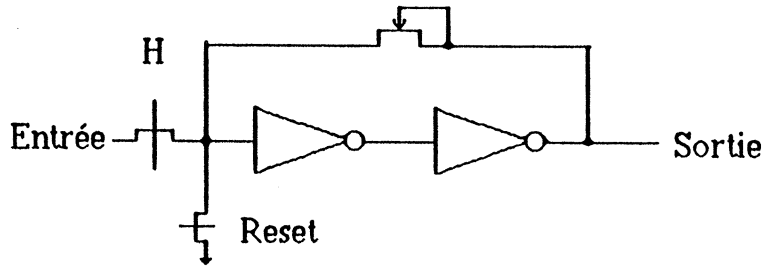


fig. 3.46 schéma logique d'un point de mémorisation

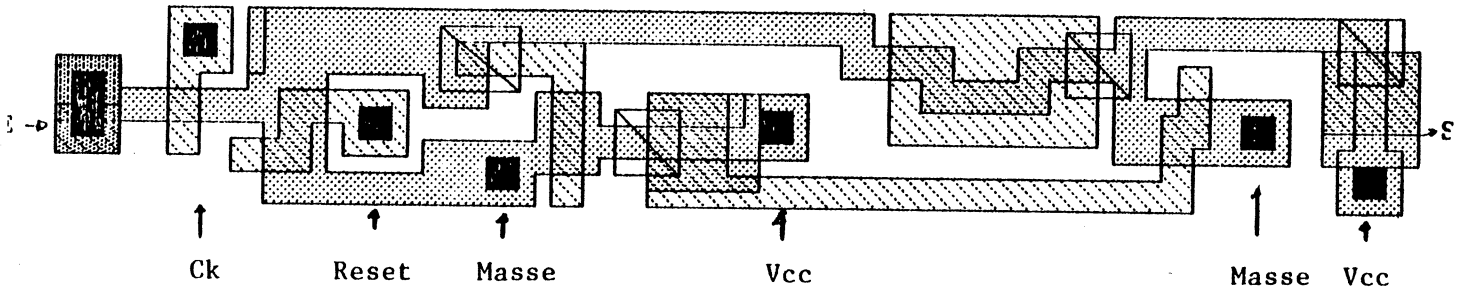


fig. 3.47 Description géométrique d'un point de mémorisation

Le "fan-out" de sortie de la cellule de mémorisation (ou puissance d'attaque de la sortie) est fonction de la taille (γ_c et γ_s) des transitors de charge et signal des inverseurs. La variation de leurs tailles fera varier la longueur DY2 de la cellule de mémorisation. L'inconvénient d'un grossissement trop important de ces transistors, en ait l'influence sur les inverseurs d'attaque de ces transistors. Si leur taille est trop importante, la taille de l'inverseur attaquant ces transistors devra augmenter [MeCo 81].

- Cellule d'amplification pour les entrées du PLA. Cette cellule fournit en sortie le signal d'entrée amplifié et son inverse. Cette cellule est de type statique sans horloge (figure 3.48). Cette cellule est paramétrable, par grossissement de ces inverseurs. Cette variation en taille ne joue que sur la hauteur de la cellule DY_2'(figure 3.49).

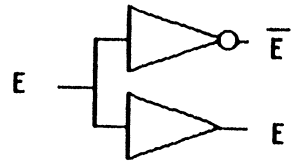


fig. 3.48 Schéma logique d'une cellule d'amplification d'entrée d'un PLA

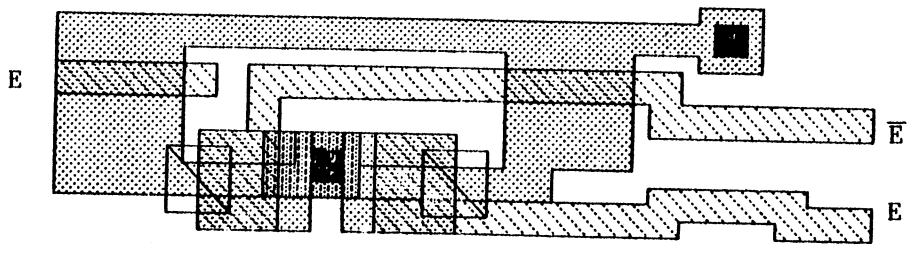


fig. 3.49 Description géométrique d'une cellule d'amplification d'entrée, delivrant un signal E et NON(E).

- Cellule d'inversion en sortie de PLA: cette cellule sert à inverser le signal en sortie de la matrice OU d'un PLA. Cette cellule est de type statique sans horloge (figure 3.50). Cette cellule a un hauteur adaptée à celle de la cellule d'amplification. Ceci provient d'une simplification du programme d'assemblage automatique, qui ne peut assembler que des cellules rectangulaires. D'autre part, la déformabilité adoptée pour la cellule inverseur permet une meilleure adaptation des connnecteurs d'alimentations entre eux (figure 3.51).

fig. 3.50 Schéma logique d'une cellule d'inversion de sortie d'un PLA

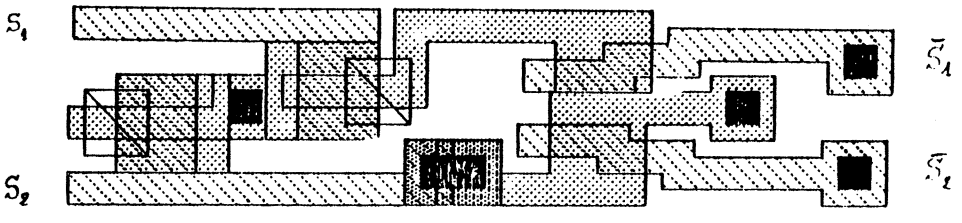


fig. 3.51 Description géométrique d'une cellule inverseur

- Cellule de charge statique pour un monôme ou une sortie de PLA: cette cellule sert à charger de manière statique la ligne de sortie ou monôme dont elle a la charge (figure 3.52). Cette cellule suivant qu'elle est utilisée pour charger un monôme ou une sortie, subit une déformation de sa hauteur, permettant ainsi une meilleure adaptibilité de ses connecteurs à son environnement. :Par exemple, dans le cas ou la cellule est utilisée pour charger un monôme, elle est alors compactée. Si elle est utilisée pour charger une sortie, elle est alors déformée pour que ses connecteurs d'alimentations soient adaptés aux connecteurs d'alimentations des cellules d'amplification et inverseur (figure 3.53, 3.54).

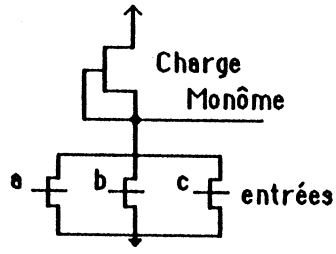


fig. 3.52 Schéma logique d'une cellule de charge de monôme ou de sortie

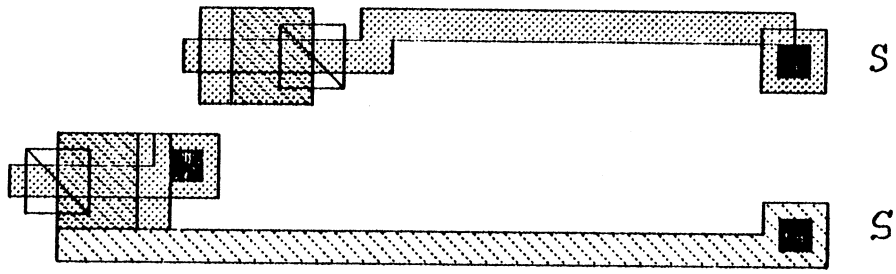


fig. 3.53 Description géométrique d'une cellule de charge d'un monôme

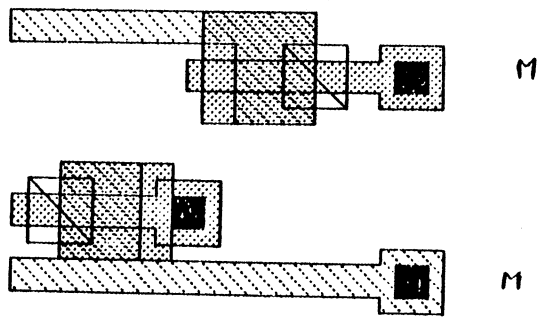


fig. 3.54 Description géométrique d'une cellule de charge d'une sortie

- Cellule de sortie d'une variable de contrôle. Cette cellule en fonction des sortie RESET et SET qui sortent du PLA, fournit un signal destiné au bus de contrôle (cfIII 5.2.2). Elle est équivalente à une porte trois états (figure 3.55, 3.56).

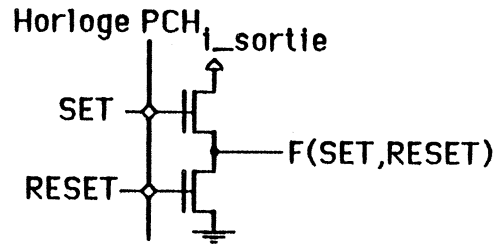


fig. 3.55 Schéma électrique d'une cellule de contrôle

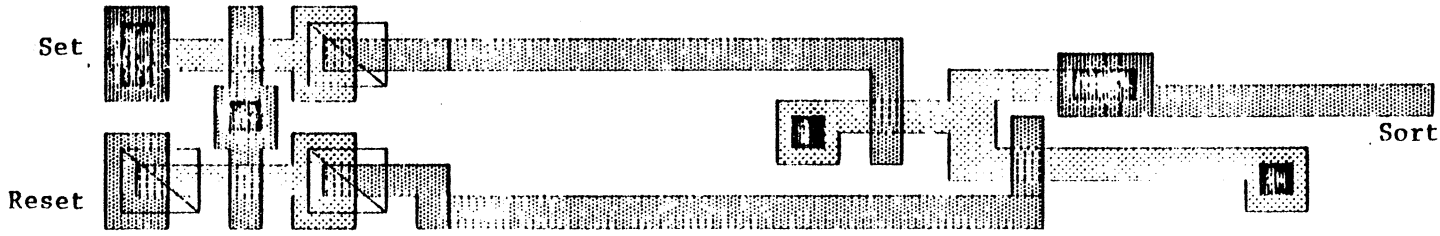


fig. 3.56 Description géométrique d'une cellule de contrôle

- Cellule de synchronisation entre les différents étages de partie contrôle: Cette cellule sert à synchroniser les différents étages de partie contrôle en respect du modèle temporel procédural (figure 3.57). Seul le schéma logique de cette cellule existe en l'état.

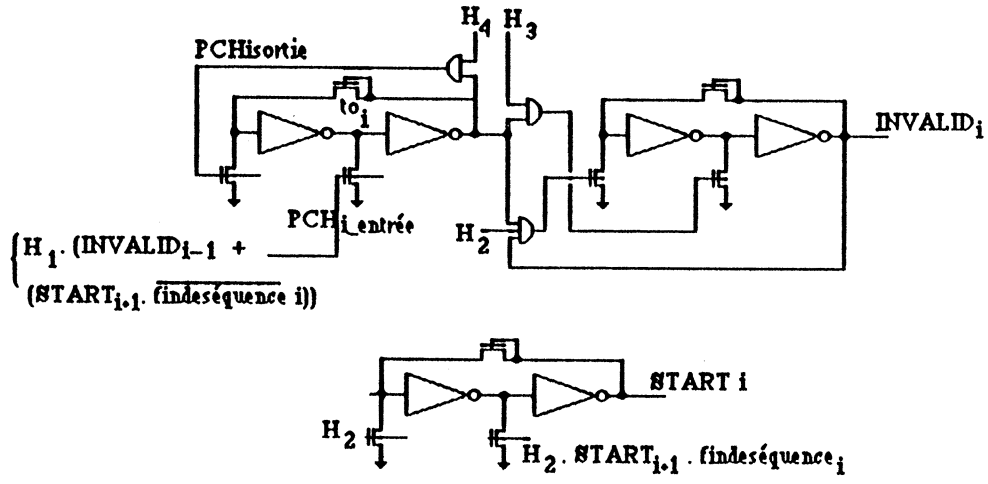


fig. 3.57 Schéma logique d'une cellule de synchronisation de niveau i

5.2.4 Conclusion:

Le programme GENPC est écrit en langage PASCAL standard. Il est implanté sur SM90/TELMAT sous UNIX, et est équivalent à plus de deux mille lignes de code. Il utilise l'environnement de travail LUBRICK/INPG pour les entrées/sorties vers les fichiers, ainsi que pour l'assemblage des cellules ou blocs fonctionnels constituant une partie contrôle. Il possède la portabilité de machine à machine que lui confère le logiciel LUBRICK/INPG.

Son écriture utilise une partie commune de fonctions déjà définies pour le générateur de PLA GENPLA. Son écriture est ainsi modulaire. Cette partie commune correspond à l'exploitation des données fournies par le fichier de commande.

Performances

L'exemple suivant ne correspond à aucune description réelle de microprocesseur. Il permet seulement de valider les temps d'exécution du programme GENPC, et de comparer les surfaces obtenues par assemblage, aux surfaces pouvant être estimées à partir des équations du chapitre III 4.2.

Le fichier de commande est celui décrit au chapitre III 5.1.3.5 et possède les caractéristiques suivantes:

. fichier de commande:

```
name cir            ! nom du circuit final
pcn 2               ! nombre total de partie contrôle
crn 5               ! nombre total de fils de compte rendu
ctn 1  1  2  3      ! nombre de contrôles interne en entrée puis
                    ! contrôles externes, en entrée, en sortie puis
                    ! bidirectionnels
pc  1  1  0  6      ! informations relatives à la partie contrôle un
rot 1               ! partie contrôle à un PLA disposé horizontalement
cr  3  1  3  5      ! trois CR avec les numéros des fils
ctr
en 4  1  5  7  9    ! quatre fils de CTRL en entrée avec leur numéro
so 4  2  4  6  8    ! quatre fils de CTRL en entrée avec leur numéro
pc  2  2  6  51
rot 0               ! partie contrôle à un PLA disposé verticalement
cr  3  2  4  5
ctr
en 5  1  5  7  9  11
so 5  4  6  8  10 12
```

Les PLA's possèdent les caractéristiques suivantes:
Matrice ET du PLA1 : 16 entrées, 40 monômes
Matrice OU du PLA1 : 16 sorties, 40 monômes
Matrice ET du PLA2 : 32 entrées, 30 monômes
Matrice OU du PLA2 : 64 sorties, 30 monômes

On obtient alors les performances d'exécution suivantes sur SM90/UNIX:

Temps réel : 1 mn 33 s
Temps utilisateur: 50 s
Temps système : 21 s

Les parties contrôles suivantes on été générées (figures 3.58 a et b):

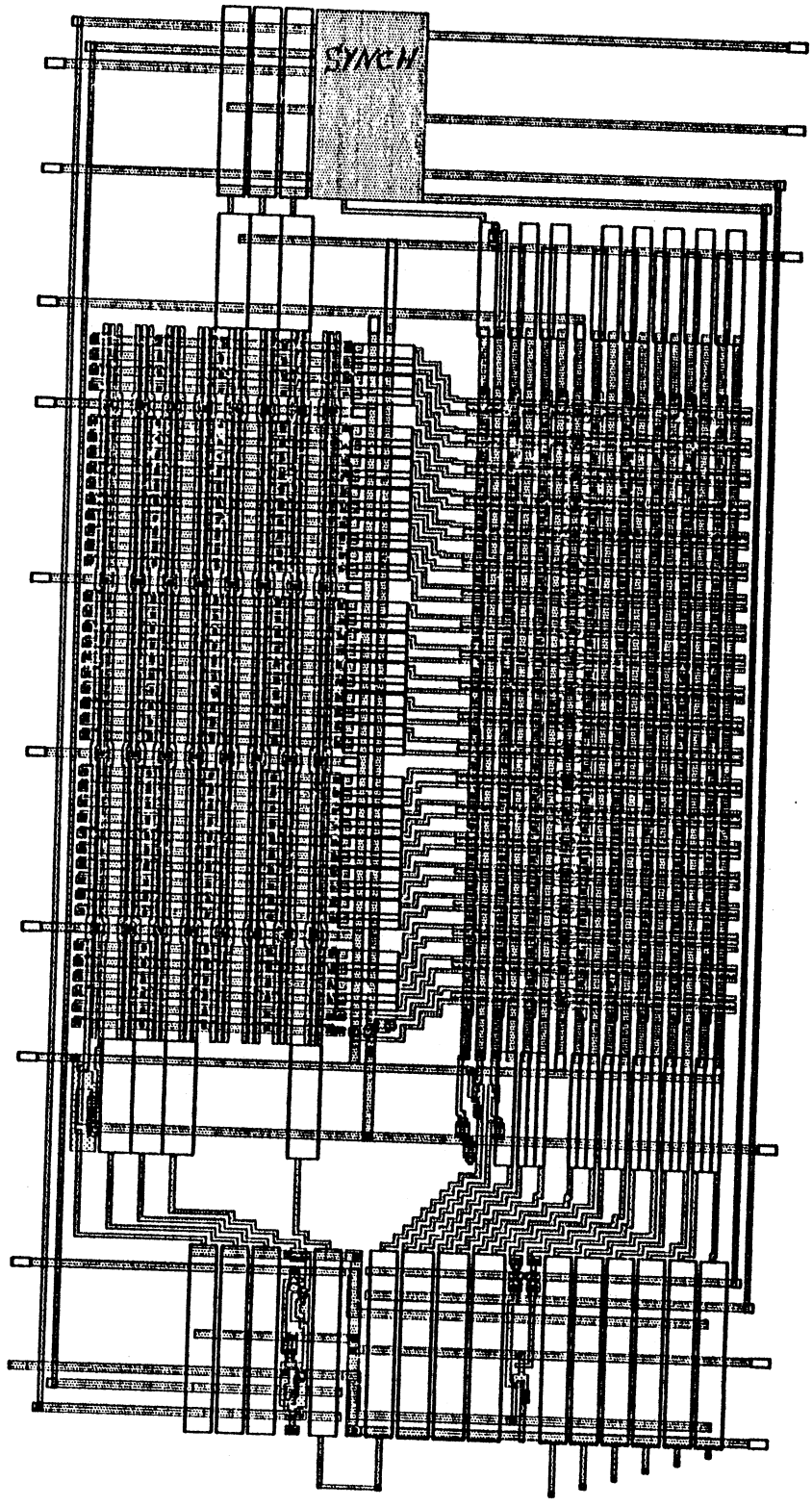


fig. 3.58a Partie contrôle 1 générée par l'outil GENPC

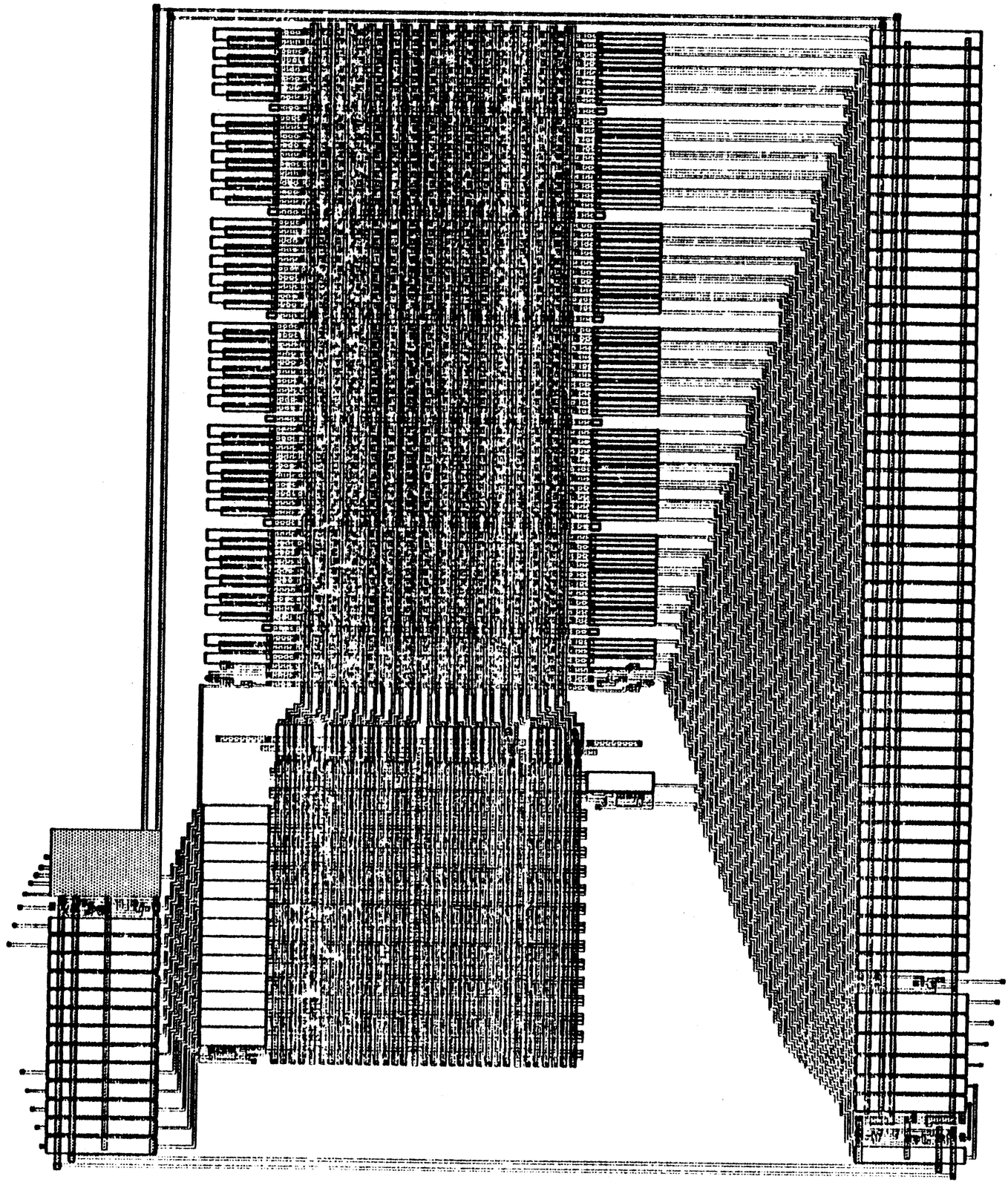


fig. 3.58b Partie contrôle 2 générée par l'outil GENPC

Si l'on compare les surfaces obtenues par compilation (exécution du programme GENPC) et les surface obtenues par utilisation des équtaion du chapitre III 4.2, on obtient les résultats suivants:

Les équations d'évaluaton de surface utilisées pour évaluer le PLA1 et PLA2 sont celles déterminées au chapitre III 4.2.2 et 4.2.1. L'évaluation se fait en technologie NMOS/CMP [DELO 86].

Soit

PME : pas de métal := 7 lambda
PE : Pas d'entrée := 7 lambda
DY_1 :=DY := 40
Nctrlin : nombre de contrôle entrants (PLA1 := 4, PLA2 := 5)
Nctrlout: Nombre de contrôle sortant (PLA1 := 4, PLA2 := 5)
Nhorl: horloges :=2
Nsyn:= 2 nombre de fils de synchronisation
Nres :=1 nombre de fils de RESET
Ncrin : Nombre de compte rendus (PLA1 := , PLA2 := 3)
Ncom : Nombre de commandes entrant (PLA1 := 0, PLA2 := 6)
Nseq: Nombre de fils de séquencement (PLA1 := 1, PLA2 := 2)
Ncomout: Nombre de commandes sortant (PLA1 := 6, PLA2 := 51)
Nplaétat: Nombre de monômes (PLA1 := 40, PLA2 := 30)
Nfinseq := 1 nombre de fils de fin de séquence

Alors on obtient les surfaces des PLA1 et PLA2 suivantes:

Largeur_PLA := (Nplaétat+Nplaétat/M).PME+2*MAX (DY1,DY)

Hauteur_PLA := (Nctrlin+Ncrin+Nseq+Ncom)*2*PE+
(Nseq+Ncomout+Nctrlout*2+Nfinseq+(Nseq+Ncomout+Nctrlout*2
+Nfinseq)/M)*PME+4*PME

d'où

Largeur_PLA1:= 416 λ
Hauteur_PLA1 := 238 λ

Largeur_PLA2 := 339 λ
Hauteur_PLA2 := 714 λ

et les surfaces des étages complets PLA1 et PLA2

Surface totale_PLA1 := 172 000 λ^2 (avec DY2: hauteur registre:=80 λ)
Surface totale_PLA2 := 628 320 λ^2

Par exécution de l'outil GENPC on obtient les surfaces suivantes :

Surface totale_PLA1 := 189 000 λ^2
Surface totale_PLA2 := 682 000 λ^2

L'écart entre les calculs de surface et la réalité sont de 8% dans le cas de partie contrôle MonPLA implantée horizontalement, et de 9% dans le cas de partie contrôle MonoPLA implantée verticalement.

5.3 Génération de circuit: GENCIRCUIT

L'outil de génération de circuit GENCIRCUIT, permet de construire le coeur du circuit microprocesseur à partir des résultats d'implantation de la partie opérative par APOLLON, et de la génération de la partie contrôle à plusieurs tranches d'interprétation fournie par GENPC. Le coeur du circuit correspond à la génération du routage entre la partie opérative et la partie contrôle, ainsi que celle des bus de contrôle, de compte rendu et de synchronisation. Les variables de contrôle internes sont aussi liées au coeur du circuit.

Le programme GENCIRCUIT s'appuie sur un ensemble de logiciels de base, GENPLA, GENPC, LUCIE et LUBRICK. Une partie de l'outil GENCIRCUIT est programmée en langage UNIX, tels les appels aux différents sous-ensembles constitués par GENPLA et GENPC. Il utilisise des cellules de base appartenant à la bibliothèque constituée pour le compilateur de partie contrôle.

5.3.1 Processus de génération d'un coeur de circuit:

Le générateur GENCIRCUIT est constitué de deux couches fonctionnelles (figure 3.59).

La première couche fonctionnelle utilise les résultats des générations de PLA de GENPLA, de GENPC pour ce qui concerne les parties contrôles et d'APOLLON pour ce qui concerne la partie opérative. A partir de ces résultats contenus dans des fichiers LUCIE et LUBRICK, GENCIRCUIT assemble les différents blocs constituant un coeur de circuit (partie contrôle et partie opérative) en y ajoutant les bus de contrôle, de compte rendu, et de synchronistation.

La deuxième couche de fonctions, écrites dans le langage UNIX, est une couche permettant le lancement séquentiel des différents modules permettant une construtions d'un coeur de circuit. C'est à dire que les programmes GENPLA, GENPC et GENCIRCUIT sont séquentiellement exécutés.

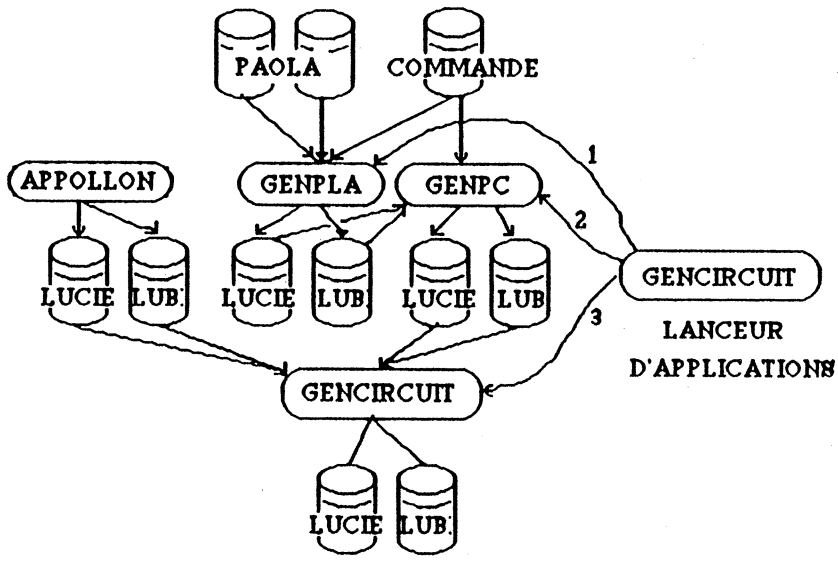


fig. 3.59 Programme GENCIRCUIT

5.3.2 Algorithme de génération d'un coeur de circuit:

L'algorithme de construction d'un coeur de circuit, se déroule en trois phases, la première consiste à générer les matrices de PLA, la seconde à générer la partie opérative et les différents niveaux d'interprétation de partie contrôle, et la troisième phase consiste à assembler les blocs résultats de l'exécution des outils précédents.

C'est la troisième phase dont on va détailler l'algorithme de construction suivant:

Début

(Construction des bus de contrôle, de compte rendu et de synchronisation)
(Assemblage des bus ave la partie contrôle)
(Assemblage du bloc précédemment constitué avec la partie opérative)

Fin

Le programme suivant correspond au texte UNIX de la couche haute du programme GENCIRCUIT.

(* programme gencircuit *)
commande d'exécution :

```
gencoeur <nom du circuit à dessiner: trois lettres> <machine utilisateur>
(* programme UNIX *)

L01 = /proj/lubrick
L02 = /proj/circuit
L03 = /proj/xnmos
L04 = /proj/nmos
L05 = /proj/PLAgen

if test -f commande.dat
then
    cd $2
    cp $L01/*.dat              (* fichiers de technologie *)

    echo "génération des PLAs de contrôle"
    $L05/gen_pla               (* exécution du programme GENPLA *)
    echo "fin de génération des PLAs de contrôle"

    cp $L03/x* .               (* copie de la bibliothèque LUBRICK *)

    echo "génération des blocs de contrôle "
    $L02/cir                   (* exécution du programme genpc et gencircuit *)
    echo "fin de génération de la partie contrôle "

    cp $L04/* .                (*copie de la bibliothèque LUCIE *)

    traduc $1                  (*Traduction des fichiers LUCIE obtenus, pour une
                               visualisation à l'écran *)

else
    echo "le fichier commande n'existe pas"
fi

(* fin du fichier gencoeur *)


commande d'exécution:          traduc $1
```

```
(* programme traduc *)

echo "traduction des fichiers LUCIE"

for i in 'lf lra*'              (* traduction des fichiers de raccords*)
    do
    ltr <<%%
    tr $i*
    %%
    au
done

for i in 'lf la*'               (*traduction des fichiers matrice ET de PLA*)
    do
    ltr <<%%
    tr $i*
    %%
    au
done

for i in 'lf lo*'               (* traduction des fichiers matrice OU des PLA*)
    do
    ltr <<%%
    tr $i*
    %%
    au
done

for i in 'lf l$1*'              (* traduction des fichiers de partie contrôle et du*)
    do                          (*coeur de circuit*)
    ltr <<%%
    tr $i-
    %%
    au
done

echo "fin de traduction des fichiers LUCIE"
```

a) Limitations

- Le programme GENCIRCUIT ne permet pas d'assembler des parties contrôles autre que MonoPLA. Ceci est due aux limitations apportées par le programme GENPC et LUBRICK/INPG.
- La zône de routage entre la partie contrôle et la partie opérative est très importante, ceci étant lié à la non adaptibilité de la connectique de la partie contrôle par rapport à la connectique de la partie opérative.
- Le bus de contrôle n'est pas totalement construit, et ne relie pas les variables de contrôle internes au bus. En effet, le programme GENCIRCUIT ne peut construire le bus totalement, car il ne connait pas le nombre de plots existants à implanter autour du coeur du circuit. Il ne peut donc pas construire la connectique complète de ce bus. Cela se traduit sur les dessin benson des figures 3.60 a et b, que le bus de contrôle n'apparait pas .

b) Evolutions:

- Les variables de contrôles seront reliées au bus de contrôle, et les plots seront routés automatiquement. Pour cela, un deuxième fichier de commande doit être rajouté, ce fichier de commande décrivant les plots nécessaires pour la partie opérative, la partie contrôle et les horloges
- Il sera intégré, dès que le logiciel LUBRICK/INPG le permettra, différents styles de conception de partie contrôle, ainsi que des blocs mieux adaptés topologiquement entre eux.

5.3.3 Conclusion:

Le programe GENCIRCUIT est écrit en langage PASCAL standard, et une partie en langage UNIX. Il est implanté sur SM90/TELMAT sous UNIX, et il est équivalent à environ 300 lignes de code. Il utilise l'environnement LUBRICK/INPG pour les entrées/sorties, ainsi que pour l'assemblage des blocs. Il possède la portabilité du programme LUBRICK en ce qui concerne la partie programmée en PASCAL. Par contre, il est nécessaire de reconstruire la partie réalisée en UNIX si l'on change de système d'exploitation.

Performances:

L'exemple suivant est tiré du microprocesseur décrit par M. ANCEAU dans son cours d'architecture de systèmes. La description correspondante décrite dans le langage LDS se trouve en ANNEXE C. De plus, il est aussi tiré du langage LDS, la description du graphe d'état sous forme de MOORE. La description a subi une optimisation manuelle, optimisation permettant un gain en nombre d'états (gain sur le nombre de noeuds et sur le nombre d'arcs de la description en graphe).

On obtient alors le fichier de commandes suivant, fichier résultat de l'exécution

d'APOLLON pour les informations relatives au PLA de génération des commandes, ou PLA2, et à la génération manuelle des informations pour le PLA1.

fichier de commande:

```
name mic            ! nom du circuit final
pcn 2               ! nombre total de parties contrôles
crn 8               ! nombre total de fils de compte rendu
ctn 2 3 0 4         ! nombre de contrôles interne en entrée puis
                    ! contrôles externes, en entrée, en sortie puis
                    ! bidirectionnels
pc  1 7 0 5         ! informations relatives à la partie contrôle n° 1
rot 1               ! partie contrôle à un PLA disposé horizontalement
cr  8 1 2 3 4 5 6 7 8   ! trois CR avec les numéros des fils
ctr
en 4  1 2 6 8       ! quatre fils de CTRL en entrée avec leur numéro
so 5  3 4 5 7 9     ! quatre fils de CTRL en entrée avec leur numéro
pc  2  1 5 48
rot 0               ! partie contrôle à un PLA disposé verticalement
cr  0
ctr
en 0
so 0
```

Les PLA's possèdent les caractéristiques suivantes:

Matrice ET du PLA1 : 38 entrées, 86 monômes
Matrice OU du PLA1 : 23 sorties, 86 monômes
Matrice ET du PLA2 : 12 entrées, 45 monômes
Matrice OU du PLA2 : 50 sorties, 45 monômes

On obtient alors les performances d'exécution suivantes sur SM90/UNIX:

Temps réel :	26 mn
Temps utilisateur:	20 mn
Temps système :	2 mn

Dans ces temps sont compris la phase de traduction des fichiers LUCIE, qui correspondent à environ 80% du temps utilisateur du programme GENCIRCUIT.

La figure 3.60a présente le résultat de l'exécution du programme GENCIRCUIT à partir du fichier de commande ci dessus. Ce fichier de commande correspond au microprocesseur d'ANCEAU décrit dans l'annexe C. Il apparait que l'étage résultant de la fusion des trois étages de contrôle extrait de la description LDS est plus imporatnt en taille que l'étage de génération des comandes de la partie

opérative fourni par APOLLON. Une fusion de ce dernier étage avec l'étage trois précédent (avant fusion) produit deux parties contrôles plus équilibrées. Dans ce cas particulier la fusion a été réalisée manuellement. Dans la version suivante de SYCO ou la fusion sera automatique, une telle possibilité de modifier les automates de partie contrôle par éclatement ou fusion d'étages, une répartition plus linéaire des fonctionnalités à réaliser par les parties contrôles aménera à la réalisation d'un coeur de circuit plus carré.

La figure 3.60b représente le cas ou les deux étages de contrôle résultent de la fusion des étages (1 et 2) et (3 et 4) extraits du texte source LDS avant fusion. Les deux parties contrôles sont alors plus équilibrées. Dans ce cas, le gain en surface est quasi nul.

Le fichier de commande est alors le suivant:

fichier de commande:

```
name mic              ! nom du circuit final
pcn 2                 ! nombre total de parties contrôles
crn 8                 ! nombre total de fils de compte rendu
ctn 2 3 0 4           ! nombre de contrôles interne en entrée puis
                      ! contrôles externes, en entrée, en sortie puis
                      ! bidirectionnels
pc 1 6 0 5            ! informations relatives à la partie contrôle n° 1
rot 1                 ! partie contrôle à un PLA disposé horizontalement
cr 8 1 2 3 4 5 6 7 8  ! trois CR avec les numéros des fils
ctr
en 2 1 2              ! quatre fils de CTRL en entrée avec leur numéro
so 5 3 4 5 7 9        ! quatre fils de CTRL en entrée avec leur numéro
pc 2 6 5 48
rot 0                 ! partie contrôle à un PLA disposé verticalement
cr 0
ctr
en 3 2 6 8
so 3 3 4 5
```

Les PLA's possèdent les caractéristiques suivantes:

Matrice ET du PLA1 : 32 entrées, 54 monômes
Matrice OU du PLA1 : 22 sorties, 54 monômes
Matrice ET du PLA2 : 28 entrées, 60 monômes
Matrice OU du PLA2 : 61 sorties, 60 monômes

On obtient alors les performances d'exécution suivantes sur SM90/UNIX:

Temps réel :	30 mn
Temps utilisateur:	23 mn
Temps système :	2 mn

La figure 3.60c représente un circuit réalisé à partir du fichier de commande décrit au chapitre III 5.2.4. Il correspond à l'assemblage des parties contrôles présentées dans le chapitre III 5.2, comme exemple de l'exécution de l'outil GENPC.

Résultat de l'exécution de l'outil GENCIRCUIT (figure 3.60 a,b et c):

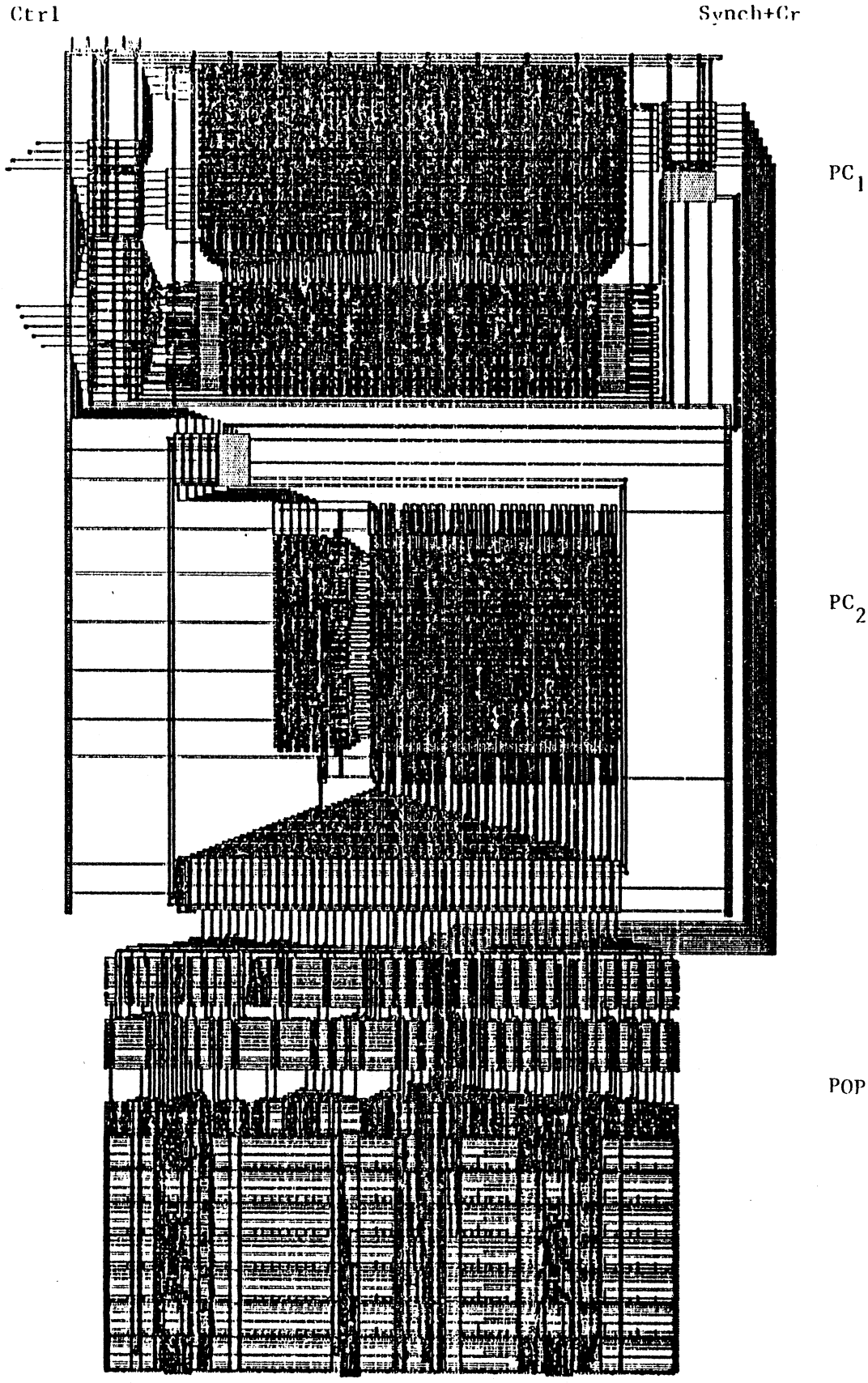


fig 3.60a Exemple de coeur de circuit généré à partir du fichier de commande du microprocesseur décrit en annexe C

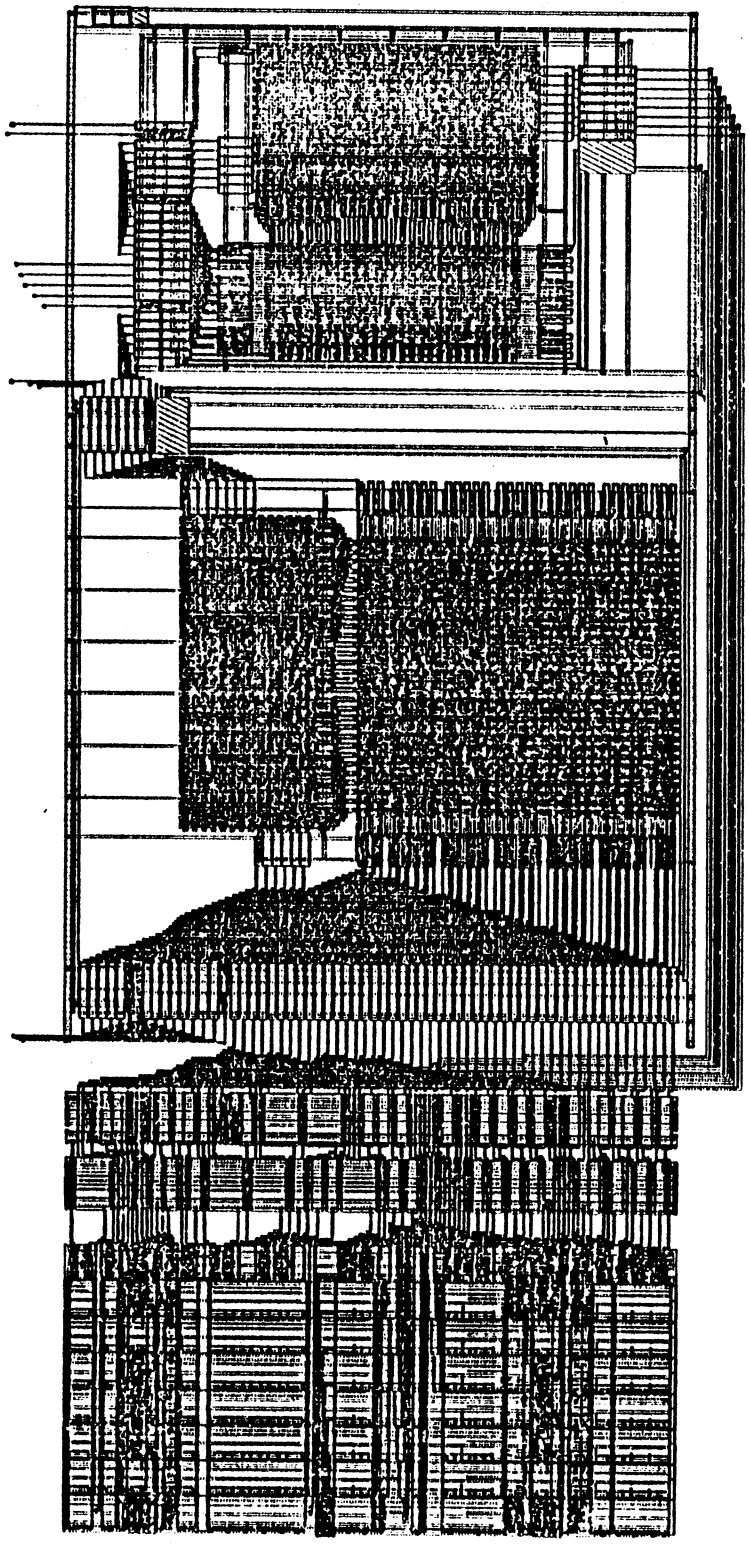


fig 3.60b Exemple de coeur de circuit généré à partir du fichier de commande du microprocesseur décrit en annexe C

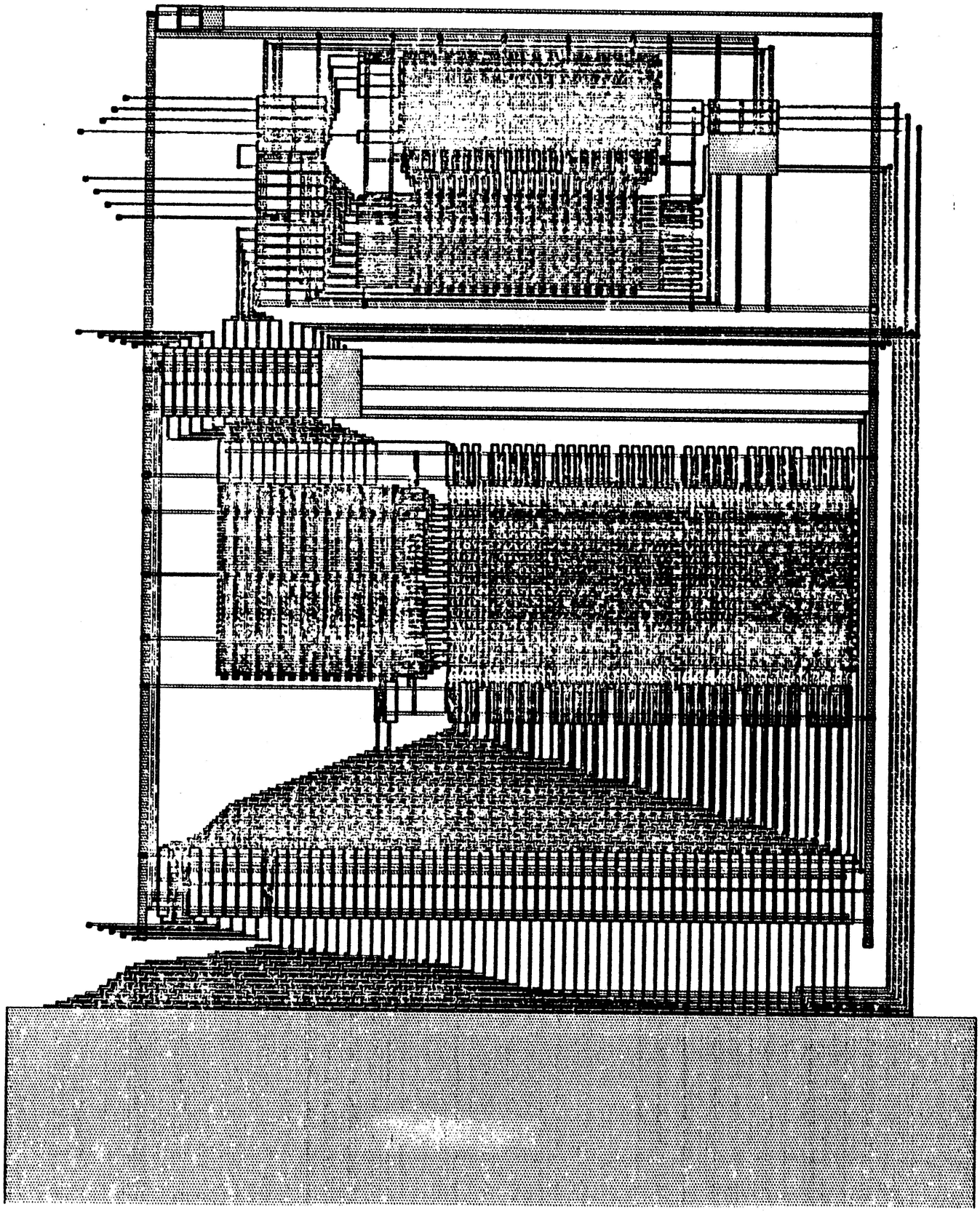


fig 3.60c Exemple de coeur de circuit généré à partir du fichier de commande du chapitre III 5.2.4

Si l'on compare les surfaces obtenues par compilation (exécution du programme GENPC) et les surface obtenues par utilisation des équations du chapitre III 4.2, on obtient les résultats suivants:

Les équations d'évaluation de surface utilisées pour évaluer le PLA1 et PLA2 du circuit génée de la figure 3.60a sont celles déterminées au chapitre III 4.2.2 et 4.2.1. L'évaluation se fait en technologie NMOS/CMP [DELO 86].

Soit

PME : pas de métal := 7 lambda
PE : Pas d'entrée := 7 lambda
DY_1 :=DY := 40 lambda
Nctrlin : nombre de contrôle entrants (PLA1 := 4, PLA2 := 0)
Nctrlout: Nombre de contrôle sortant (PLA1 := 5, PLA2 := 0)
Nhorl: horloges :=2
Nsyn:= 2 nombre de fils de synchronisation
Nres :=1 nombre de fils de RESET
Ncrin : Nombre de compte rendus (PLA1 := 8, PLA2 := 0)
Ncom : Nombre de commandes entrant (PLA1 := 0, PLA2 := 5)
Nseq: Nombre de fils de séquencement (PLA1 := 7, PLA2 := 1)
Ncomout: Nombre de commandes sortant (PLA1 := 5, PLA2 := 48)
Nplaétat: Nombre de monômes (PLA1 := 86, PLA2 := 45)
Nfinseq := 1 nombre de fils de fin de séquence

Alors on obtient les surfaces des PLA1 et PLA2 suivantes:

Largeur_PLA := (Nplaétat+Nplaétat/M).PME+2*MAX (DY1,DY)
Hauteur_PLA := (Nctrlin+Ncrin+Nseq+Ncom)*2*PE+
(Nseq+Ncomout+Nctrlout*2 +Nfinseq+(Nseq+Ncomout+Nctrlout*2
+Nfinseq)/M)*PME+4 *PME

d'où

Largeur_PLA1:=738 λ
Hauteur_PLA1 :=469 λ

Largeur_PLA2 :=430 λ
Hauteur_PLA2 :=497 λ

et les surfaces des étages complets PLA1 et PLA2

Surface totale_PLA1 := 469 x 1094 := 485 000 λ^2
Surface totale_PLA2 := 700 x 814 := 320 000 λ^2

Par exécution de l'outil GENPC et GENCIRCUIT on obtient les surfaces suivantes :

Surface totale_PLA1 := 527 x 1062 := 559 000 λ^2
Surface totale_PLA2 := 734 x 796 := 584 000 λ^2

L'écart entre les calculs de surface et la réalité sont de 15% dans le cas de partie contrôle MonPLA implantée horizontalement, et de 3% dans le cas de partie contrôle MonoPLA implantée verticalement.

Dans le cas de la figure 3.60b, nous obtenons respectivement les valeurs suivantes, dans le cas de l'évaluaton de surface et dans le cas de la mesure de la figure réelle.

Largeur_PLA1:=500 λ
Hauteur_PLA1 :=540 λ

Largeur_PLA2 :=420 λ
Hauteur_PLA2 :=700 λ

et les surfaces des étages complets PLA1 et PLA2

Surface totale_PLA1 := 469 x 1094 := 420 x 780 := 330 000 λ^2
Surface totale_PLA2 := 700 x 814 := 900 x 1048 := 945 000 λ^2

Par exécution de l'outil GENPC et GENCIRCUIT on obtient les surfaces suivantes :

Surface totale_PLA1 := 527 x 1062 := 473 x 778 := 370 000 λ^2
Surface totale_PLA2 := 734 x 796 := 921 x 1021 := 940 000 λ^2

L'écart entre les calculs de surface et la réalité sont de 12% dans le cas de partie contrôle MonPLA implantée horizontalement, et de 0.05% dans le cas de partie contrôle MonoPLA implantée verticalement.

L'écart très faible entre le calcul théorique et le résultat pratique de l'outil GENPC dans le cas d'une partie contrôle implantée verticalement est interressant dans le cadre de la création d'un évaluateur de surface de partie contrôle. Ce pourcentage d'erreur (entre 0+ et 9% pour les trois exemples).

Dans le cas d'une partie contrôle implantée horizontalement, le pourcentage d'erreur (de 8% à 15%) est plus important. Il est du essentiellement au routage existant entre la matrice ET et OU qui ne rentre pas en compte dans les équations de calcul de surface. Ce routage est du à l'état actuel du générateur de PLA(GENPLA) qui ne reporte pas les rappels de masse de la matrice ET par des écartements dans la matrice OU. L'erreur sera donc diminuée (ainsi que la surface réelle d'un étage implanté de cette manière) lorsque le générateur de PLA évoluera dans ses prochaines versions.

CHAPITRE IV

CONCLUSION

Dans ce mémoire, une étude des différents types de compilateurs de silicium a été menée, comparativement au compilateur de silicium SYCO. Des priorités dans la conception d'un bon outil de compilation de silicium s'en sont dégagées:

a) la nécessité de 'cibler' le produit visé par le compilateur de silicium. Il est difficile, dans les conditions d'environnement actuel, de produire un outil de compilation de silicium totalement général.

b) la nécessité de définir des cibles aux différents niveaux de représentation d'un circuit intégré. Ce ciblage permet de simplifier le processus de compilation par élimination de cibles possibles dans l'arbre des choix. Par exemple, dans le cas des structure de contrôle disponibles, l'élimination de la structure à logique anarchique diminue l'arbre des choix de structures de partie contrôle.

c) la nécessité de définir une interface 'unique' entre les divers outils composant un compilateur de silicium. Cette interface, assimilable à une structure de données, doit être souple et évolutive.

Ces diverses priorités sont intégrées, ou en cours de l'être dans le compilateur de silicium SYCO. En effet, l'architecture, le modèle temporel, le modèle topologique ont été strictement ciblés, dans le cadre de la compilation de circuits de type microprocesseur. Une structure de données unifiée, multi-sémantique a été étudiée et est en cours de validation: elle correspond à un sous ensemble de la structure de donnée LDS.

Pour les diverses structures de contrôle, il a été fait une étude topologique. Cette étude topologique effectuée dans le cadre du compilateur SYCO, s'est efforcée de fournir des équations pour permettre une évaluation de surface des diverses structures de contrôle. De l'étude menée, il se dégage deux priorités:

a) la compilation de partie contrôle est simplifiée par l'utilisation de structures régulières,

b) les divers choix de parties contrôles, et les nombreuses variables de choix que cela implique, demandent la mise au point d'outils tels des évaluateurs de surface, de complexité et de performances de partie contrôle.

Pour valider les choix effectués sur les parties contrôles, une réalisation d'outils de génération de partie contrôle a été entreprise. Elle s'est concrétisée par un ensemble d'outils permettant la génération automatique de partie contrôle monoPLA à multi-niveaux d'interprétation. Les PLA's implémentés sont de type classique.

L'évolution première de l'outil est d'utiliser, pour une meilleure optimisation de la surface du circuit, des outils permettant de prendre en compte les notions topologiques telles la transparence ou la déformabilité de blocs. Ces outils sont dans le cas des PLAs, l'outil PAOLA, et dans certaines couches de programme, l'outil LUBRICK. Une modification du modèle fonctionnel est aussi envisageable, si on considère qu'une partie opérative peut être directement allouée à une partie contrôle particulière, et être ainsi directement adressable par elle. Cette modification remet en cause certaines bases du raisonnement du compilateur de partie opérative. En effet, le compilateur APOLLON ne traite pas de cas particuliers tels qu'une sous-partie opérative est indépendante des autres. Cela provoquerai une évolution du modèle fonctionnel vers un modèle à processus parallèles. L'inconvénient d'une telle évolution serait une remise en cause de certaines particularités du modèle topologique actuel, tel les emplacements des bus. Il y aurai en effet un grossissement de ceux ci par l'augmentation des informations se propageant par leur intermédiaire.

Une deuxième évolution consiste à intégrer autour de la base construite, de nouvelles structures de contrôle.

Un autre type d'évolution consiste à modifier l'interprétation du texte source LDS. Actuelllement, un CMODULE ne peut appeler un CMODULE de même niveau. Pour lever cette restriction, il faudrait introduire une gestion de pile à chaque niveau de contrôle. Cette notion de pile existe déjà à l'état embryonnaire dans la version 1 du compilateur SYCO. Cette pile se résume aux variables de contrôle.

Une dernière évolution du système **SYCO** est la génération automatique de circuits autotestables, dans le but générer des circuits "totalement autocontrôlables ("self checking")" et facilement testables [TORK 86]. Une telle étude est menée dans le cadre de l'architecture de partie contrôle monoPLA, et la solution actuelle est la génération d'une partie contrôle "self-checking", étude tirée des recherches effectuées par [NICO 86] sur le test en ligne et hors ligne d'un circuit.

En outre, la technologie actuellement utilisée est le NMOS/CMP. Il est prévue de faire rapidement évoluer les outils vers une utilisation de la technologie CMOS.

Beaucoup de problèmes techniques restent encore à résoudre. Il apparait toutefois que la base construite permet d'intégrer de nouvelles fonctionnalités, tels des évaluateurs de performance ou de surface, ou de nouvelles structures de contrôle.

ANNEXE A

GENERATION ET OPTIMISATION

DE

PLA MONOMATRICES

GENERATION ET OPTIMISATION DE PLA MONOMATRICE

Les PLA's monomatrice représentent une forme particulière d'implémentation d'une structure logique PLA.

Un PLA est une structure logique, telle q'un vecteur de sortie est fonction d'un vecteur d'entrée.

Soit S ensemble des vecteurs de sortie,
E ensemble des vecteurs d'entrée,
M ensemble des vecteurs des termes premiers,

Et

S(t) est un vecteur de sortie, et $s_t(i)$ un élément de sortie; $1 \leq i \leq p$

E(t) est un vecteur d'entrée, et $e_t(i)$ un élément d'entrée; $1 \leq i \leq n$

M(t) est un vecteur monôme, et $m_t(i)$ un élément monôme; $1 \leq i \leq k$

Alors

$$s_t(i) = \Sigma_j^k \ \Pi_i^n \ e_t(i)$$

Suivant la technologie, l'implémentation d'une structure PLA se réalise à l'aide de portes ET/OU (Bipolaire) ou NOR/NOR ou NAND/NAND (MOS).

L'équation logique ci-dessus prend alors les formes suivantes:
Synthèse en NAND

$$s_t(i) = \Pi_j^k \ \overline{(\Pi_i^n \ e_t(i))}$$

Synthèse en NOR

$$s_t(i) = \Sigma_j^k \ \overline{(\Sigma_i^n \ e_t(i))}$$

L'implémentation classique d'une telle structure en technologie MOS se réalise à l'aide de deux matrices NOR/NOR (appelées respectivement matrice ET et matrice OU par rappel de la forme logique de base).

L'implémentation de PLA de type monomatrice se distingue par le regroupement topologique des deux matrices NOR/NOR dans une même matrice. Un tel regroupement permet une dispersion des entrées et des sorties le long de la matrice, améliorant le facteur d'adaptabilité topologique des connecteurs face à l'environnement du PLA [ANCE 86] (figure A.1).

L'inconvénient d'une telle forme, provient de la surface exéssive nécessaire pour l'implémentation d'une telle structure sous forme monomatrice [VARI 85]. Une étude sur l'optimisation topologique d'une telle forme, prenant en compte les contraintes de l'environnement (transparence, déformabilité, ordre des connecteurs...) a été entreprise pour résoudre ces inconvénients.

A.1 Optimisation topologique de PLA monomatrice

L'optimisation topologique consiste à compacter la monomatrice du PLA par la méthode des lignes de monômes brisées, puis réorganisation de la matrice des monômes. Cette réorganisation réduit la taille du PLA. Elle est réalisée comme suit:

* Réarrangement des monômes,
Ce réarrangenment consiste à minimiser la distance entre chaque monômes et l'entrée ou la sortie connectée à ce monôme. Ce processus consiste à diagonaliser la matrice du PLA.

* Duplication des entrées/sorties:
Cette duplication des entrées/sorties augmente le facteur de diagonalisation de la matrice, et par cela la compatibilité de deux monômes (la compatibilté de deux monômes correspond à ce que l'intersection de leurs ensembles ordonnés d'entrée/sortie soit nulle, l'ordre des ensembles est fonction de l'emplacement des entrées/sorties dans la monomatrice).

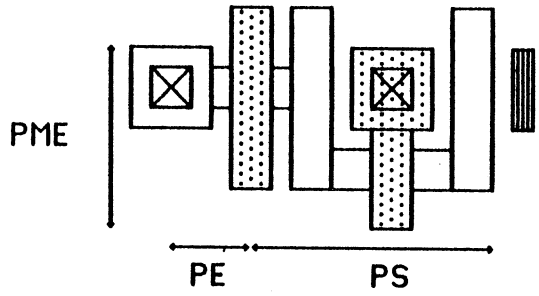
* Compactage des monômes:
Le compactage consiste à placer plusieurs monômes sur la même ligne. Le compactage est fonction du taux de compatibilité des monômes.

A.2 Génération automatique du layout.

La génération automatique du layout d'un PLA monomatrice doit prendre en compte l'ensemble des paramêtres de construction. Un PLA monomatrice possède des cellules d'entrées et de sorties différentes. En outre, un facteur de rappel de masse doit être pris en compte pour améliorer les caractéristiques électriques et de performance du PLA.

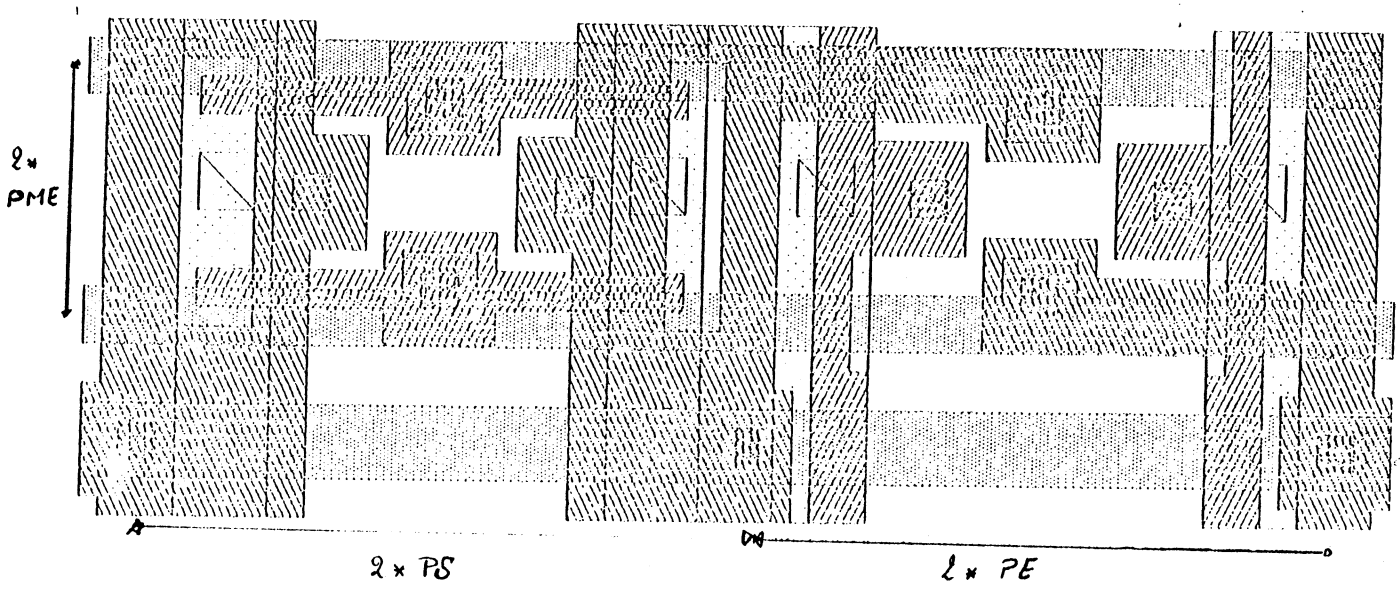
Pour cela, deux cellules ont été proposées dans [VARI 85], une en technologie NMOS/CMP (a), et une autre en technologie CMOS/CMP deux aluminium (b).

Ces cellules possèdent les caractéristiques suivantes:



(a)

Cellule en technologie NMOS/CMP



Cellule en technologie CMOS/CMP

A.3 Conclusion

Ce travail entrepris sur les PLA's monomatrice a été repris par M^{r} MASSON à BULL SYSTEME, tout particulièrement en ce qui concerne l'optimisation topologique par brisûre de monômes et réorganisation des entrées/sorties.

ANNEXE B

DIGITALISEUR LUCIE/LUBRICK

Une interface homme/machine conviviale

DIGITALISEUR LUCIE/LUBRICK:
Une Interface homme/machine Conviviale.

Le progiciel LUCIE est composé:

- D'un éditeur de cellules,
- Dun traducteur (texte source vers un texte objet lisible par l'éditeur),
- D'une interface de sòrtie vers table traçante BENSON,
- D'un digitaliseur LUCIE/LUBRICK (produisant du texte source).

La création du digitaliseur LUCIE/LUBRICK provient d'un besoin de posséder une interface homme/machine conviviale, permettant la saisie interactive de figures LUCIE et LUBRICK, à l'aide d'un numériseur. Cette saisie se révèle être plus aisée que la description manuelle d'un fichier LUCIE ou LUBRICK. Le digitaliseur LUCIE/LUBRICK existe sous système VAX/VMS et SM90/UNIX.

Une telle interface s'est révélée indispensable pour la maintenance et la complémentation des bibliothèques de cellules utilisées par le compilateur de silicium **SYCO**. D'autre part, ce logiciel est actuellement employé pour l'enseignement dans le complexe du CIME (Centre Inter-universitaire de Micro-Electronique).

Le cahier des charges supposait la prise en compte des éléments suivants:

- Utilisation d'une table à numériser de type Tektronik ou compatible Tektronik 4114/4014,

- Prise en compte de plusieurs technologies (en l'occurence technologies NMOS et CMOS),

- Interface homme/machine conviviale entrainant un affichage sur l'écran des options de digitalisation disponibles telles que:
  - digitalisation d'une figure LUCIE ou LUBRICK,
  - niveaux utilisés,
  - utilisation du stylet et de la table pour la saisie de tout type d'information relatif à la digitalisation en cours,
  - affichage simultané possible sur un même écran des figures LUCIE et LUBRICK en cours de digitalisation.

- Disponibilité de la modification du dessin en temps réel durant la digitalisation, pour correction d'erreur à la saisie.

- Possibilité de sauvegarder dans une nouvelle figure, ou d'ajouter à une ancienne figure, les informations relatives à la digitalisation (maintenance ou complémentation d'une bibliothèque).

A partir des spécifications précédentes, les choix suivants ontété faits:

* Le digitaliseur prend en compte deux technologies (NMOS et CMOS), dont les niveaux possibles (8 en NMOS et 10 en CMOS) sont affichés à l'écran, et dont le choix se passe à l'aide du stylet pointé sur un menu (choix interactif).

* Le passage d'une digitalisation d'une figure LUCIE à une figure LUBRICK se passe de manière interactive et temps réel durant tout le temps d'une digitalisation. Le passage d'un mode à l'autre se détermine par pointage du stylet sur un menu, où est affiché simultanément les choix LUCIE et LUBRIK.

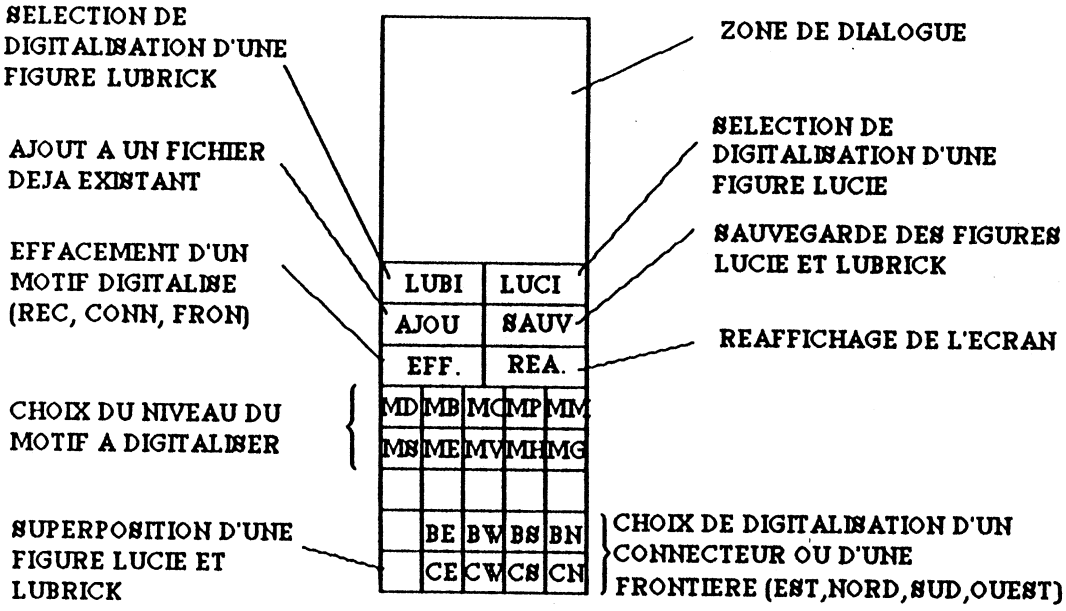
* Sous le mode LUBRICK, l'affichage d'une figure LUCIE est possible par pointage sur le menu,

* A tout moment, il est possible de modifier, ou de réafficher la figure modifiée par pointage sur le menu. La modification peut porter (en LUCIE ou en LUBRICK) sur un élément quelconque de la figure digitalisée: rectangle, connecteur, frontière.

* La sauvegarde de la figure digitalisée est possible:
  - soit par création d'une nouvelle figure (LUCIE ou LUBRICK),
  - soit par ajoùt dans une figure déjà existante, des rectangles, connecteurs ou frontières digitalisés.

* D'autre part, pour permettre un passage aisé du programme ainsi créé, d'une machine ou système sur un autre, parallèlement au progiciel LUCIE, la bibliothèque "MINIPLOT" de fonctions d'affichage graphique créée pour l'éditeur LUCIE a été intégrée au digitaliseur.

Le code généré pour ce logiciel correspond à environ 1400 lignes de FORTRAN 77. La maintenance est actuellement prise en main par un ingénieur du CIME et l'équipe du CMP [DePa 86].



Présentation de l'interface homme/machine pour l'aide a la digitalisation

ANNEXE C

COMPILATION D'UN MICROPROCESSEUR SIMPLIFIE

COMPILATION D'UN MICROPROCESSEUR SIMPLIFIE

C.1.Description LDS d'un microprocesseur simplifié

```
SMODULE MICROP(adbus,dtbus,dtack,restart,adstrobe,dtstrobe,read_write);

VARIABLE;
r 0:2, CTRL;
END;

SIGNAL;
 adbus 0:8, OUT; ?address bus
 dtbus 0:8, INOUT; ?data bus
 dtack, IN, CTRL; ?data acknowledgement
 restart, IN, CTRL;
 adstrobe, OUT, CTRL; ?address strobe
 dtstrobe, OUT, CTRL; ?data strobe
 read_write, OUT, CTRL; ?selection read=1 / write=0
END;

REGISTER;
 a 0:8; ?accumulator
 ad 0:8; ?address register
 b 0:8; ?special purpose register
 c 0:8; ?special purpose register
 d 0:8; ?special purpose register
 ir 0:8; ?instruction register
 pc 0:8; ?program counter
 streg 4:4; ?status register
 car, LIKE streg 1 ?carry bit
 neg, LIKE streg 2 ?negative bit
 ovf, LIKE streg 3 ?overflow bit
 zer, LIKE streg 4 ?zero bit
END;

LINK MICROP;

END;

?*****************************************

CMODULE ada;
<e1>
    (r := 2; EXECUTE fetch;)
```

```
    a := a + b;
END;

CMODULE lda;
<e1> (r := 1; EXECUTE fetch;)
END;

CMODULE sta;
<e1>
    (adbus := ad; RESET read_write; dtbus := a;)
    (adbus := ad; SET adstrobe; dtbus := a; SET dtstrobe; NEXT e2;)
<e2>
    IF (dtack = 0)
    (adbus := ad; dtbus := a; NEXT e2;)
    ELSE (RESET adstrobe; RESET dtstrobe;) END;
END;

CMODULE cma;
<e1>
    (r := 2; EXECUTE fetch;)
    BOP(, b, -, a, streg 4:4);
END;

CMODULE beq;
<e1> IF (zer = 1) pc := ad; END;
END;

CMODULE bgt;
<e1> IF ((neg = 0) AND (zer = 0)) pc := ad; END;
END;

CMODULE bra;
<e1> pc := ad;
END;

CMODULE div;
<e1>
    (r := 2; EXECUTE fetch;)
    (c := b; NEXT e2;)
<e2>
    BOP(, c, >, a, streg 4:4);
    IF ((neg=0) AND (zer=0)) NEXT e3; END;
    (c := c * 2; NEXT e2;)
<e3>
    (d := 0; NEXT e4;)
<e4>
```

```
    BOP(, c, <=, b, streg 4:4);
    IF ((neg=1) OR (zer=1)) NEXT e5; END;
    (d := d * 2; c := c / 2;)
    BOP(, a, <, c, streg 4:4);
    IF (neg=1) NEXT e4; END;
    (d := d + 1; a := a - c;)
    NEXT e4;
<e5>
    b := d;
    ? quotient in b, remainder in a
END;

CMODULE mva;
<e1> a := b;
END;

CMODULE fetch;
<send>
    (adbus := pc; SET read_write;)
    (adbus := pc; SET adstrobe; NEXT receive;)
<receive>
    IF (dtack = 0) (adbus := pc; NEXT receive;)
    ELSE
    (CASE (r)
        WHEN (0) ir := dtbus;
        WHEN (1) a := dtbus;
        WHEN (2) b := dtbus;
        WHEN (3) ad := dtbus;
    END;
    pc := pc + 1;
    RESET adstrobe;)
    END;
END;

?*********************************************

CMODULE MICROP;

<rest> IF (restart = 0) NEXT rest; ELSE NEXT start; END;

<start> (pc := 0; NEXT newinstr;)

<newinstr>
    (r := 0; EXECUTE fetch;)
    IF (ir 4:2 = '00') NEXT decode; ELSE (r :=3; EXECUTE fetch;) END;
    (CASE (ir 4:2
```

```
            WHEN ('01') ;
            WHEN ('10') ad := pc + ad;
            WHEN ('11') ad := pc - ad;
        END;
        NEXT decode;)

<decode>
        (CASE (ir 0:4)
            WHEN ('0000') EXECUTE ada; ? a <- a + dtbus
            WHEN ('0001') EXECUTE lda; ? a <- dtbus
            WHEN ('0010') EXECUTE sta; ? memory <- a
            WHEN ('0011') EXECUTE cma; ? compare dtbus to a
            WHEN ('0100') EXECUTE beq; ? jump if dtbus=A
            WHEN ('0101') EXECUTE bgt; ? jump if dtbus>A
            WHEN ('0110') EXECUTE bra; ? unconditional jump
            WHEN ('0111') EXECUTE div; ? division of "a" by dtbus
            WHEN ('1xxx') EXECUTE mva; ? a <- b
        END;
        IF (restart = 0) NEXT rest; ELSE NEXT newinstr; END;)

END;
```

C.2. Extraction de graphe d'état d'une decription LDS

Cette exraction manuelle est réalisée à partir d'une description source LDS. Les définitions suivantes sont utilisées pour l'extraction des informations destinées à la construction du graphe.

Définition a:

Un CMODULE ne peut appeler un CMODULE de même niveau. Ainsi, toute hiérarchie de CMODULE créé une hiérarchie de niveaux d'interprétation, et donc un nombre de niveau équivalent de partie contrôle.

Définition b:

Un CMODULE appelant un CMODULE de niveau p fois inférieur, crée une hiérarchie de CMODULE intermédiaires (CMODULES fictifs).

Définition c:

Un niveau de partie contrôle est créé implicitement. Il correspond au niveau de contrôle directement supérieur à la partie opérative. Il est créé par le compilateur de partie opérative APPOLLON, et génère les étapes opératives pour APPOLLON. Il n'apparait pas explicitement dans une description LDS.

Définition d:

Un EXECUTE correspond à l'appel d'un CMODULE.

Définition e:

Un noeud du graphe d'état est créé pour chaque opération. Une opération est équivalente à un ensemble d'opérations élémentaires exécutables en parallèle. Une opération élémentaire peut être un EXECUTE, une opération sur une variable de contrôle ou une action opérative. Si un EXECUTE est contenu dans un ensemble d'opérations élémentaires, tout autre EXECUTE est exclu de cet ensemble, ainsi que toutes actions opératives. Une action opérative peut être contenue dans un ensemble d'opérations élémentaires en parallèle avec d'autres actions opératives. Une opération sur les variables de contrôle peut être effectuée en parallèle avec tout autre type d'opération élémentaire.

Définition f:

Une condition est décrite soit sous forme de IF ,ELSE, soit sous forme de CASE, WHEN. Le traitement d'une condition peut être réalisé en parallèle avec un ensemble d'opérations. Les mêmes limiations de la définition (e) sont applicables au champ opératif d'une instruction conditionnée.

Définition g:

Un état correspond à un noeud du graphe, le traitement des confitions est lié aux arcs du graphe.

Définition h:

Sous forme de Mealy, les conditions et les ensembles d'opérations (définition e) sont liés aux transitions, sous forme de Moore, les ensembles d'opérations sont liés aux noeuds.

C.3. Extraction du graphe d'état de la description du Microprocesseur d'Anceau: MICROP.

L'extraction du graphe de ce microprocesseur, tel décrit au paragraphe 1, sera faite sous forme de Moore. Sous forme de Moore, les ensembles opératifs sont liés aux noeuds du graphe. L'extraction du source LDS est donc considérablement simplifiée, si l'on utilise les définition de (a) à (g).

3.1 Extraction du niveau 1 du microprocesseur MICROP

Cette extraction correspond au CMODULE MICROP.

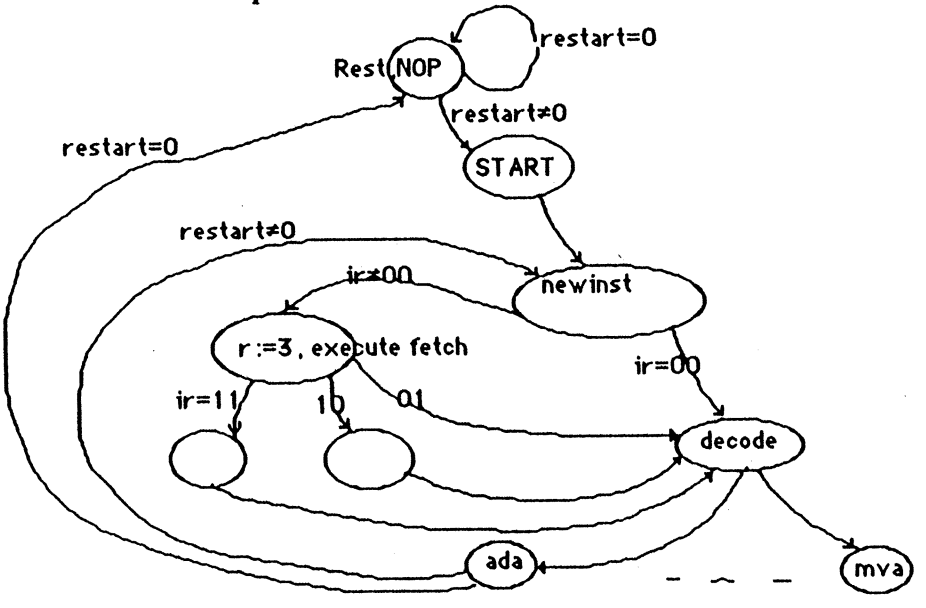


fig. C.1 Partie du graphe extrait du niveau 1 de partie contrôle

3.2 Extraction du niveau 2 du microprocesseur MICROP

Cette extraction correspond aux CMODULE'S ada, lda, sta, cma,beq, bgt, bra, div, mva.

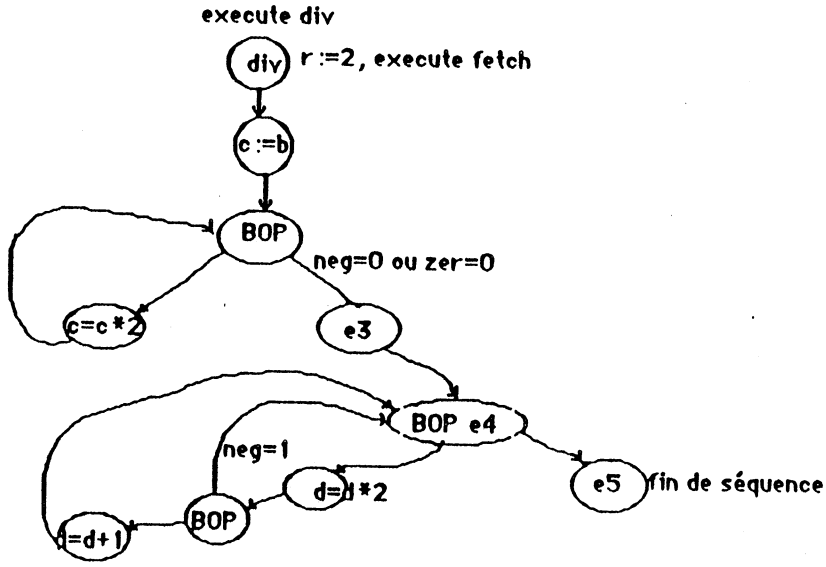


fig. C.2 Partie du Graphe extrait du niveau 2 de partie contrôle

De l'automate extrait de la description LDS, sont rajoutés des états correspondant à la création implicite de CMODULE de transferts provenant d'action opérative effectuées au niveau 1 de partie contrôle, plus de l'appel de CMODULE du niveau 3 de partie contrôle.

3.3 Extraction du niveau 3 du microprocesseur MICROP

Cette extraction correspond au CMODULE fetch.

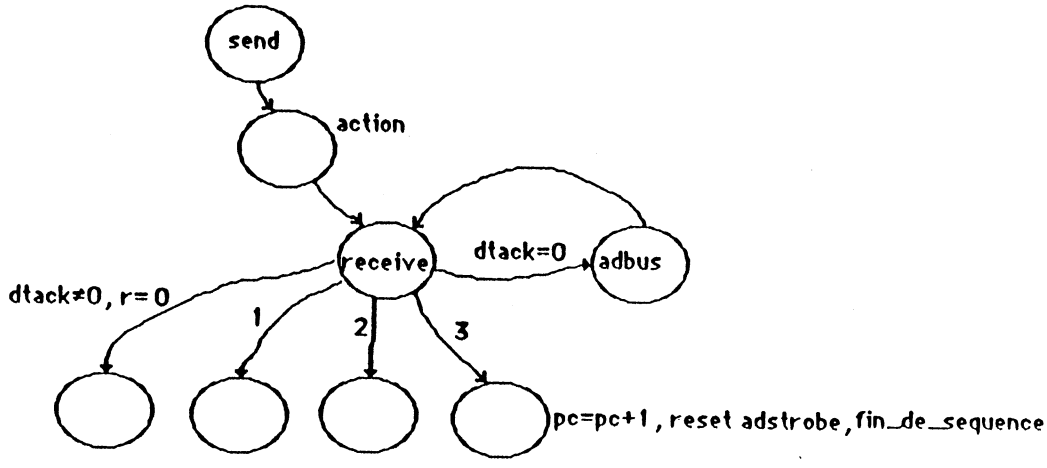


fig. C.3 Partie du Graphe extrait du niveau 3 de partie contrôle

De l'automate extrait de la description LDS, sont rajoutés des états correspondant à la création implicite de CMODULE de transferts provenant d'action opérative effectuées au niveau 1 et 2 de partie contrôle.

3.4 Descritpions des niveaux de parties contrôles sous forme de partie contrôle monoPLA.

A partir du graphe de contrôle, on déduit les informations suivantes pour les trois étages:

Le premier étage est constitué de :

MAX (16 états ou noeuds, 38 arcs) monômes;
Nombre de fils de séquencement := Log_2(16 états):= 5 fils
Nombre de fils de contrôles entrants :=1;
Nombre de fils de contrôles sortants:= 2;
Nombre de fils de compte rendu := 6
Nombre de commandes pour le niveau inférieur := Log_2 (13):= 4 fils
Nombre de commandes du niveau supérieur := 0

Le deuxième étage est constitué de :

MAX (25 états ou noeuds, 18 arcs) monômes;
Nombre de fils de séquencement := Log_2(25 états):= 5 fils
Nombre de fils de contrôles entrants :=1;
Nombre de fils de contrôles sortants:= 5;
Nombre de fils de compte rendu := 2
Nombre de commandes pour le niveau inférieur := Log_2 (20):= 5 fils
Nombre de commandes du niveau supérieur := 4

Le troisième étage est constitué de :

MAX (28 états ou noeuds, 8 arcs) monômes;
Nombre de fils de séquencement := Log_2(28 états):= 5 fils
Nombre de fils de contrôles entrants :=3;
Nombre de fils de contrôles sortants:= 3;
Nombre de fils de compte rendu := 0
Nombre de commandes pour le niveau inférieur := Log_2 (25):= 5 fils
Nombre de commandes du niveau supérieur := 5

Le quatrième étage est défini par l'exécution du compilateur de partie opérative APPOLLON. Il est constitué de :

MAX (48 états ou noeuds) monômes;
Nombre de fils de séquencement := 1 fils
Nombre de fils de contrôles entrants :=0;
Nombre de fils de contrôles sortants:= 0;

Nombre de fils de compt rendu := 0
Nombre de commandes pour le niveau inférieur := 48 fils
Nombre de commandes du niveau supérieur := 5

3.5 Fusions des étages de contrôle:

Les étages définis ci dessus sont très petits, et leur nombre important provoque une chute des performances préjudiciable. Par exemple, l'étage trois est essentiellement composé de l'automate fetch. C'est pourquoi une opération de fusion de deux étages de contrôle est effectuée. L'opération de fusion consiste à remonter les automates de niveau i+1 dans l'étage de contrôle de niveau i. Cela se traduit par une duplication de l'automate de niveau i+1 pour chaque appel effectué dans l'étgage de contrôle de niveau i. Par exemple, l'automate fetch est dupliqué quatre fois dans l'étage de niveau 2' résultat de la fusion de l'étage 2 et 3. Le premier état de l'automate fetch est concaténé avec l'état appelant de l'automate fusionnant.

La première fusion réalisée est effectuée sur les étages 3 et 2 et provoque la construction d'un étage 2'. L'étage de niveau 2' possède les cractéristiques suivantes:

MAX (54 états ou noeuds, 48 arcs) monômes;
Nombre de fils de séquencement := Log_2(54 états):= 6 fils
Nombre de fils de contrôles entrants :=4;
Nombre de fils de contrôles sortants:= 5;
Nombre de fils de compt rendu := 2
Nombre de commandes pour le niveau inférieur := Log_2 (22):= 5 fils
Nombre de commandes du niveau supérieur := 5

Une deuxième fusion est réalisée entre l'étage de niveau 1 et l'étage de niveau 2'. Le nouvel étage résultant possède les cractéristiques suivantes:

MAX (72 états ou noeuds, 86 arcs) monômes;
Nombre de fils de séquencement := Log_2(72 états):= 7 fils
Nombre de fils de contrôles entrants :=4;
Nombre de fils de contrôles sortants:= 5;
Nombre de fils de compt rendu := 8
Nombre de commandes pour le niveau inférieur := Log_2 (22):= 5 fils
Nombre de commandes du niveau supérieur := 0

Pour l'estimation des surfaces du processeur MICROP, voir le chapitre III, 5.2.3, figure 3.60a.

On peut fusionner les étages 1, 2, 3 et 4 d'une manière différente. Pour équilibrer les deux étages résultant de la fusion, il parait plus adéquat de fusionner les étages 1 et 2, et ensuite les étage 3 et 4. Le résultat de cette fusion donne les résultats suivants:

Fusion de l'étage 1 et 2 :

MAX (54 états ou noeuds, 40 arcs) monômes;
Nombre de fils de séquencement := Log_2(54 états):= 6 fils
Nombre de fils de contrôles entrants :=2;
Nombre de fils de contrôles sortants:= 5;
Nombre de fils de compt rendu := 8
Nombre de commandes pour le niveau inférieur := Log_2 (23):= 5 fils
Nombre de commandes du niveau supérieur := 0

Fusion de l'étage 3 et 4 :

MAX (60 états ou noeuds, 8 arcs) monômes;
Nombre de fils de séquencement := Log_2(60 états):= 6 fils
Nombre de fils de contrôles entrants :=3;
Nombre de fils de contrôles sortants:= 3;
Nombre de fils de compt rendu := 0
Nombre de commandes pour le niveau inférieur := 48 fils
Nombre de commandes du niveau supérieur := 5

De même que pour la fusion précédente, les résultats d'estimation des surfaces du processeur MICROP sont présentés dans le chapitre III, 5.2.3 avec la figure 3.60b.

C.4.Entree du compilateur de parties opératives

L'entrée est constituée de 2 parties :
- une partie déclaration de ressources
- une partie de spécification des actions opératives à exécuter en parallèle

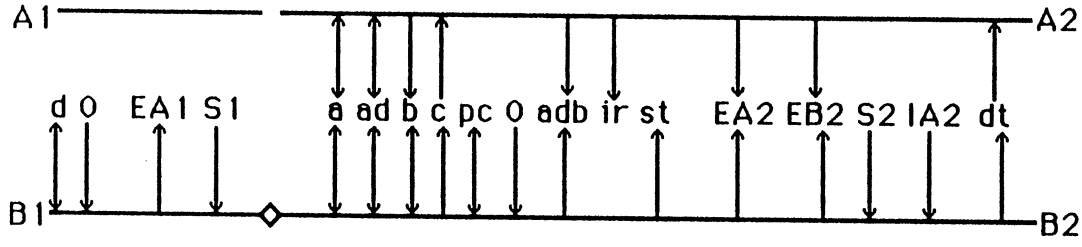
micro.etp

```
registres : a ad b c d ir pc streg;
entrées-sorties : dtbus 0:8;
sorties : adbus 0:8;
champs de contrôle : ir 0:6 streg 4:4;
sous-pops: 3;
bits : 8;
nom : micro;

a <- a + b;
adbus <- ad / dtbus <- a;
BOP(, b, -, a, streg 4:4);
pc <- ad;
c <- b;
BOP(, c, -, a, streg 4:4);
c <- mult2 c;
d <- 0;
BOP(, c, -, b, streg 4:4);
d <- mult2 d / c <- div2 c;
BOP(, a, -, c, streg 4:4);
d <- incr d / a <- a - c;
b <- d;
a <- b;
adbus <- pc;
ir <- dtbus / pc <- incr pc;
a <- dtbus / pc <- incr pc;
b <- dtbus / pc <- incr pc;
ad <- dtbus / pc <- incr pc;
pc <- 0;
ad <- pc + ad;
ad <- pc - ad.
```

C.5.Schéma fonctionnel de la partie opérative

EAi et EBi sont les entrées de la boîte à opérations numéro i, Si la sortie et IAi les indicateurs arithmétiques.
Ai et Bi sont les bus de la sous-partie opérative numéro i.



sous-pop No 1 : incr mult2
sous-pop No 2 : - div2 incr + mult2

C.6.Description des actions opératives précédentes incluant les bus et les opérateurs utilisés.

```
==========================================================
instruction No 1 :  a <-- a + b
sous-pop 2 : +
 phi1   a    ---> A2 ---> EA2
 phi1   b    ---> B2 ---> EB2
 phi2   S2   ---> B2 ---> a
==========================================================
instruction No 2 :  adbus <-- ad   dtbus <-- a
 phi1   ad   ---> A2 ---> tampon-adbus
 phi1   a    ---> B2 ---> tampon-dtbus
 phi1   tampon-adbus ---> adbus
 phi1   tampon-dtbus ---> dtbus
 phi2   tampon-adbus ---> adbus
 phi2   tampon-dtbus ---> dtbus
==========================================================
instruction No 3 :  BOP(, b, -, a, streg 4:4)
sous-pop 2 : -
 phi1   a    ---> A2 ---> EA2
 phi1   b    ---> B2 ---> EB2
 phi2   IA2  ---> B2 ---> streg 4:4
==========================================================
instruction No 4 :  pc <-- ad
 phi1   ad   ---> B2 ---> pc
==========================================================
instruction No 5 :  c <-- b
 phi1   b    ---> B2 ---> c
==========================================================
instruction No 6 :  BOP(, c, -, a, streg 4:4)
sous-pop 2 : -
 phi1   c    ---> A2 ---> EB2
 phi1   a    ---> B2 ---> EA2
 phi2   IA2  ---> B2 ---> streg 4:4
==========================================================
instruction No 7 :  c <-- mult2 c
sous-pop 2 : mult2
 phi1   c    ---> A2 ---> EA2
 phi2   S2   ---> B2 ---> c
==========================================================
instruction No 8 :  d <-- 0
 phi1   0    ---> B1 ---> d
==========================================================
instruction No 9 :  BOP(, c, -, b, streg 4:4)
sous-pop 2 : -
```

```
 phi1   c    ---> A2 ---> EB2
 phi1   b    ---> B2 ---> EA2
 phi2   IA2  ---> B2 ---> streg 4:4
==========================================================
instruction No 10 :  d <-- mult2 d   c <-- div2 c
sous-pop 1 : mult2   sous-pop 2 : div2
 phi1   d    ---> B1 ---> EA1
 phi1   c    ---> A2 ---> EA2
 phi2   S1   ---> B1 ---> d
 phi2   S2   ---> B2 ---> c
==========================================================
instruction No 11 :  BOP(, a, -, c, streg 4:4)
sous-pop 2 : -
 phi1   c    ---> A2 ---> EA2
 phi1   a    ---> B2 ---> EB2
 phi2   IA2  ---> B2 ---> streg 4:4
==========================================================
instruction No 12 :  d <-- incr d   a <-- a - c
sous-pop 1 : incr   sous-pop 2 : -
 phi1   d    ---> B1 ---> EA1
 phi1   c    ---> A2 ---> EA2
 phi1   a    ---> B2 ---> EB2
 phi2   S1   ---> B1 ---> d
 phi2   S2   ---> B2 ---> a
==========================================================
instruction No 13 :  b <-- d
 phi1   d    ---> B1
 phi1   B1   ---> B2 ---> b
==========================================================
instruction No 14 :  a <-- b
 phi1   b    ---> B2 ---> a
==========================================================
instruction No 15 :  adbus <-- pc
 phi1   pc   ---> B2 ---> tampon-adbus
 phi1   tampon-adbus ---> adbus
 phi2   tampon-adbus ---> adbus
==========================================================
instruction No 16 :  ir <-- dtbus   pc <-- incr pc
sous-pop 2 : incr
 phi1   pc   ---> B2 ---> EA2
 phi1   dtbus ---> tampon-dtbus
 phi2   tampon-dtbus ---> A2 ---> ir
 phi2   S2   ---> B2 ---> pc
==========================================================
instruction No 17 :  a <-- dtbus   pc <-- incr pc
sous-pop 2 : incr
```

```
 phi1   pc   ---> B2 ---> EA2
 phi1   dtbus ---> tampon-dtbus
 phi2   tampon-dtbus ---> A2 ---> a
 phi2   S2   ---> B2 ---> pc
==========================================================
instruction No 18 :  b <-- dtbus   pc <-- incr pc
sous-pop 2 : incr
 phi1   pc   ---> B2 ---> EA2
 phi1   dtbus ---> tampon-dtbus
 phi2   tampon-dtbus ---> A2 ---> b
 phi2   S2   ---> B2 ---> pc
==========================================================
instruction No 19 :  ad <-- dtbus   pc <-- incr pc
sous-pop 2 : incr
 phi1   pc   ---> B2 ---> EA2
 phi1   dtbus ---> tampon-dtbus
 phi2   tampon-dtbus ---> A2 ---> ad
 phi2   S2   ---> B2 ---> pc
==========================================================
instruction No 20 :  pc <-- 0
 phi1   0    ---> B2 ---> pc
==========================================================
instruction No 21 :  ad <-- pc + ad
sous-pop 2 : +
 phi1   ad   ---> A2 ---> EA2
 phi1   pc   ---> B2 ---> EB2
 phi2   S2   ---> B2 ---> ad
==========================================================
instruction No 22 :  ad <-- pc - ad
sous-pop 2 : -
 phi1   ad   ---> A2 ---> EA2
 phi1   pc   ---> B2 ---> EB2
 phi2   S2   ---> B2 ---> ad
```

C.7.Liste des fils de commande et de compte rendu sortant de la partie opérative

1 precharge_bus
2 lecture_ecriture:d<->busB
3 lecture:0->busB
4 ecriture:busB->EA1
5 isolement_precharge_incr1
6 precharge_retenue_incr1
7 retenue_entrante_incr1
8 entree_shift_up_spop1 = 0
9 shift_up_spop1
10 no_shift_spop1
11 validation:resultat_operation_spop1->tampon_de_sortie_shifter
12 lecture:tampon_sortie_shifter1->busB
13 communication:busB1<->busB2
14 precharge_bus
15 lecture_ecriture:a<->busA
16 lecture_ecriture:a<->busB
17 lecture_ecriture:ad<->busA
18 lecture_ecriture:ad<->busB
19 ecriture:busA->b
20 lecture_ecriture:b<->busB
21 lecture:c->busA
22 ecriture:busB->c
23 lecture_ecriture:pc<->busB
24 lecture:0->busB
25 validation:adbus->plot_de_sortie
26 ecriture:busA->adbus
27 ecriture:busB->adbus
28 ecriture:busA->ir
29 bit 6 du registre ir
30 bit 5 du registre ir
31 bit 4 du registre ir
32 bit 3 du registre ir
33 bit 2 du registre ir
34 bit 1 du registre ir
35 ecriture:busB->streg
36 bit 8 du registre streg
37 bit 7 du registre streg
38 bit 6 du registre streg
39 bit 5 du registre streg
40 ecriture:busA->EA2
41 ecriture:busB->EA2
42 multiplexeur:EB2->entree_C_adder
43 multiplexeur:EB2_complementee->entree_C_adder

44 ecriture:busA->EB2
45 ecriture:busB->EB2
46 validation:EB2->entree_D_adder
47 generation_de_0_sur_entree_D_adder2
48 isolement_precharge_retenue_adder2
49 precharge_retenue_adder2
50 retenue_entrante_complementee_adder2
51 validation:indicateurs_arithmétiques->IA2
52 lecture:IA2->busB
53 entree_shift_up_spop2 = 0
54 shift_up_spop2
55 entree_shift_down_spop2 = 0
56 no_shift_spop2
57 shift_down_spop2
58 validation:resultat_operation_spop2->tampon_de_sortie_shifter
59 lecture:tampon_sortie_shifter2->busB
60 ecriture:busB->Rout(dtbus)
61 validation:Rout(dtbus)->plot_de_donnees
62 validation:plot_de_donnees->Rin(dtbus)
63 lecture:Rin(dtbus)->busA

C.8.Liste des fils de commande sortant de l'interface électrique entre partie opérative et partie contrôle

1 lecture_ecriture:d<->busB
2 validation:d<->busB
3 lecture:0->busB
4 ecriture:busB->EA1
5 retenue_entrante_incr1
6 shift_up_spop1
7 no_shift_spop1
8 lecture:tampon_sortie_shifter1->busB
9 communication:busB1<->busB2
10 lecture_ecriture:a<->busA
11 validation:a<->busA
12 lecture_ecriture:a<->busB
13 validation:a<->busB
14 lecture_ecriture:ad<->busA
15 validation:ad<->busA
16 lecture_ecriture:ad<->busB
17 validation:ad<->busB
18 ecriture:busA->b
19 lecture_ecriture:b<->busB
20 validation:b<->busB
21 lecture:c->busA
22 ecriture:busB->c
23 lecture_ecriture:pc<->busB
24 validation:pc<->busB
25 lecture:0->busB
26 validation:adbus->plot_de_sortie
27 ecriture:busA->adbus
28 ecriture:busB->adbus
29 ecriture:busA->ir
30 ecriture:busB->streg
31 ecriture:busA->EA2
32 ecriture:busB->EA2
33 multiplexeur:EA2->entree_C_adder
34 multiplexeur:EA2_complementee->entree_C_adder
35 ecriture:busA->EB2
36 ecriture:busB->EB2
37 validation:EB2->entree_D_adder
38 generation_de_0_sur_entree_D_adder2
39 retenue_entrante_complementee_adder2
40 lecture:IA2->busB
41 shift_up_spop2
42 no_shift_spop2
43 shift_down_spop2

```
44 lecture:tampon_sortie_shifter2->busB
45 ecriture:busB->Rout(dtbus)
46 validation:Rout(dtbus)->plot_de_donnees
47 validation:plot_de_donnees->Rin(dtbus)
48 lecture:Rin(dtbus)->busA
```

C.9.Liste des comptes rendus sortant de la partie opérative

1 bit 6 du registre ir
2 bit 5 du registre ir
3 bit 4 du registre ir
4 bit 3 du registre ir
5 bit 2 du registre ir
6 bit 1 du registre ir
7 bit 8 du registre streg
8 bit 7 du registre streg
9 bit 6 du registre streg
10 bit 5 du registre streg

BIBLIOGRAPHIE

[AGER 76] : **T. Agerwala**, "Microprogram optimization : a survey", IEEE transactions on computers, Vol C.25, n°10, October 1976.

[AhUl 77] : **A.V. Aho and J.D. Ullman**, "Principles of compiler design", Addison Weskey, 1977.

[AnFo 74] : **F. Anceau, R. Fortier, J.P. Schoellkopf**, "Conception Descendante des Machines Informatiques: Application à une machine PASCAL", Rapport de recherche SESORI 73 042/1-81, 1974.

[ANCE 79] : **F. Anceau**, "Evolution des Microprocesseurs et son influence sur l'Architecture de Systèmes", Real Time Conf., 1979.

[AnRe 82] : **F. Anceau, R.A. REIS**, "Complex Integrated Circuit Design Strategy", IEEE Kournal Solid-State Circuits, Vol. 17, 1982.

[ANCE 83] : **F. Anceau**, "Capri : a design methodology and a silicon compiler for VLSI circuits specified by algorithms", 3rd Caltech Conference on VLSI, March 1983.

[ANCE 84] : **F. Anceau**, "Silicon compilation for microprocessor-like VLSI", NATO advanced study institute on microarchitecture of VLSI Computers, Urbino, Italy, July 1984.

[ANCE 86] : **F. Anceau**, "The Architecture of Microprocessors", Addison-Wesley Publishing Company, 1986.

[ATKI 81] : **D.E. Atkins et al.**, "Overview of an Arithmetcic Design System", 18th DAC, 1981.

[BARA 79] : **S. Baranov**, "Synthèse des automates Microprogrammés", Editions de MOSCOU, 1979.

[BeHe 82] : **A;M. Begls,B. Hennion, J. Lecourvoisier, G. Mazaré,A. Puissochet**, "CASSIOPEE: A Design Methodology based upon Symbolic Layout and Integrated CAD Tools", DAC 19th, 1982.

[BEKK 86] : **N. Bekkara**, "Optimisation et compromis surface-performance dans le compilateur SYCO", rapport interne, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex, juin 1986.

[BERG 84] : **N. Bergmann**, " A Case Study of the F.I.R.S.T. Silicon Compiler", Internal Report, University of Indenburgh, Dept. of Computer Science, 1984.

[BeDo 86] : **J.M. Berge, L.O. Donzelle, V. Olive, J. Rouillard, D. Rouquier**, "Development and use of Flexible Block Libraries", ESSIRC, 1986.

[BERT 85] : **F. Bertrand**, "Conception Descendante apliquée aux Microprocessors VLSI", Thèse de docteur de troisième cycle en Informatique, 1985.

[BIAN 81] : **R. Bianchi**, "Implantation de la logique Aléatoire d'un Circuit Intégré", DEA deMicroélctronique USMG, 1981.

[BOUR 84] : **E. Bourcier**, "Conception et Réalisation du SIMULATEUR du langage de description de circuits intégrés: IRENE-C", Thèse CNAM en informatique, 1984.

[BOUR 86] : **E. Bourcier**, "LDS : langage d'entrée du compilateur de silicium SYCO", rapport interne, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex, avril 1986.

[BrGa 86] : **F.D. Brewer, D.D. Gajski**, "An expert system paradigm for design", 23rd DAC, 1986.

[CaLa 85] : **M. Cand, J.L. Lardy, E. Demoulin, P. Senn**, "Conception de Circuits Intégrés MOS", CNET-ENST Editions Eyrolles, 1985.

[CAVA 85] : **J.J.F. Cavanagh**, "Digital Computer Arithmetic: Design and Implementation", Mc Graw-Hill Internationnal Student Edition, 1985.

[CHAI 83] : **J. Chailloux**, LE-LISP, manuel de référence, INRIA, 1984.

[CHUQ 84] : **S. Chuquillanqui Bernaola**, "Une nouvelle approche pour l'optimisation topologique et l'automatisation du dessin des masques de PLA complexes", thèse de docteur-ingénieur INPG, octobre 1984.

[CoDi 86] : **M.P. Cochet, C.B. Diesse et D. Etiemble**, "Fonctions LISP pour la simulation des Architectures Matérielles", Rapport de Recherche n°163, Université P. et M. Curie, CNRS UA 818, Laboratoire MASI, 1986.

[DAND 83] : **A. Dandache**, "Evaluations Electriques et Temporelles des PLA Complexes", Thèse de docteur de 3^{éme} Cycle en Informatique, INPG, 1983.

[DAVI 81] : **S.Davidson**, "Some experiments in local microcode compaction for horizontal machines", IEEE transactions on computers, Vol C-30, n°7, July 1981.

[DELO 86] : **H. Delori et al.**, "French MPC 1984-1985", IMAG/TIM3 research report n° 602, February 1986.

[DeMa 86] : **H. De Man**, "A synthesis and module generation system for multiprocessor systems on a chip", Nato advanced study institute on logic synthesis and silicon compilation for VLSI design, L'Aquila, Italy, July 7-18, 1986.

[DeMi 84] : **G. De Micheli**, "Optimal encoding of control logic", ICCD'84.

[DeMi 86] : **G. De Micheli**, "Synthesis of control systems", Nato advanced study institute on logic synthesis and silicon compilation for VLSI design, L'Aquila, Italy, July 7-18, 1986.

[DERA 85] : **H. Derantonian**, "Génération Automatique de parties contrôles de microprocesseurs sous forme de PLA Spécialisés", Thèse de docteur-ingénieur INPG, 1984.

[FISH 81] : **J.A. Fisher**, "Trace scheduling : a technique for global microcode compaction", IEEE transactions on computers, Vol C-30, n°7, July 1981.

[FLAM 84] : **E. Flamand**, "A Complete and Automatic System for Sequencer Design", 21th DAC, 1984.

[FlMa 75] : **H. Fleisher, L.I. Maisel**,"An Introduction to Array Logic", IBM J. RES. et DEV., Vol. 19 n° 2, 1975.

[GaKu 83] : **D.D. Gajski, R.H. Kuhn**, "New VLSI Tools", IEEE Computer, December 1983.

[GAJS 86a] : **D.D. Gajski et al**, "Silicon compilation (tutorial)", IEEE 1986 CICC.

[GAJS 86b] : **D. D. Gajski et al**, "Silicon cell compilers", Nato advanced study institute on logic synthesis and silicon compilation for VLSI design, L'Aquila, Italy, July 7-18, 1986.

[GALL 84] : **R. Gallais**, "Participation à la réalisation d'un microprocesseur 16 bits interprétant le Pcode", Thèse CNAM, Mars 1984.

[GeSa 85] : **J.P. Geronimi, S. Ringot, D. Savart**, "Conception d'un microprocesseur à l'aide d'outils de CAO de haut niveau", rapport de stage ENSERG, IMAG/TIM3, 46 Av. Félix Viallet, 38031 Grenoble Cédex.

[GlDo 85] : **L.A. Glasser, D.V. Dobberpuhl**, "The Design and Analysis of VLSI Circuits", Addison Weisley, 1985.

[GROS 83] : **R.R. Gross**, "Silicon compilers : a critical survey", Dept. of Computer Science, University of North Carolina at Chapel Hill, May 1983.

[GUYO 83] : **A. Guyot, R. Reis, I. Supriana**, "FLOPE: Editeur de pla de masse de Circuits Intégrés", Rapport de Recherche IMAG n°333, 1983.

[HOCH 87] : **B. Hocchet**, "Conception de Calculateur et Circuits Systoliques", Thèse de l'INPG en micorelectronique, Janvier 1987.

[ISOD 83] : **S. Isoda**, "Global compaction of horizontal microprograms based on the generalized data dependency graph", IEEE transactions on computers, Vol C-32, n°10, October 1983.

[JaJe 85] : **R. Jamier, A. Jerraya**, "Apollon, a datapath silicon compiler", ICCD'85.

[JaBe 86] : **R. Jamier, N. Bekkara, A. Jerraya**, "The automatic synthesis of data processing sections", ICCD'86.

[JeRo 85] : **A. Jerraya, F.R. Rougeaux, E. Rosier, B. Courtois**, "A hierarchical symbolic layout system : STYX", VLSI'85, Tokyo, Japan, August 1985.

[JeVa 86] : **A. Jerraya, P. Varinot, R. Jamier, B. Courtois**, "Principles of the SYCO compiler", DAC 86.

[JIAN 86] : **JIN Jianing**, "Un Assamblage Automatique des Cellules de l'Interface Elecrique entre la Partie Contrôle et la Partie Opérative", Rapport Interne IMAG/TIM3, 1986.

[JOHA 79] : **D. Johannsen**, "Bristles Blocks -- A Silicon Compiler", DAC, 1979.

[JONE 75] : **J.W. Jones**, "Array Logic Macros", IBM J. RES. & Dev., Vol 19 N°2, 1975.

[KeKo 86] : **G. Koden, K. Kozminski**, "A Standard Cell based Silicon Compiler"", DAC, 1979.

[KNUT 79] : **D.E. Knut**, "The Art of Computer Programming, Vol. 1, Fondamental Algorithms", Addison-Wesley, 2nd Edition, 1979.

[LaRi 81] : **W.W. Lattin, W. Richardson**, "A Methodology for VLSI Chip Design", Lambda n°2, 1981.

[LaSh 82] : **H.S. Law, M. Shoji**, "PLA Design for the BELLMAC 32A Microprocessor", ICCC, 1982.

[LaNg 85] : **D. Laurent, H.N. Nguyen**, "LDS", rapport Bull, Juin 1985.

[LiNo 82] : **R.J. Lipton, S.C. North, R. Sedgevwick, J. Valdes, G. Vijayaut**, "ALI: A Procedural Language to describe VLSI Layout", 19th DAC, 1982.

[MaBi 80] : **A. Mathialagan, N. Biswas**, "Optimal interconnections in the design of microprocessors and digital systems", IEEE journal of solid-state circuits, VOL. SC-15, NO.1, 1980.

[MALA 85a] : **P. Malardier**, "HADES: Mécanismes d'Expansion, Noyau Commun aux Simulateurs Digitaux", THOMSON-CSF/CSAO St Egrève, 1985.

[MALA 85b] : **Y.K. Malaiya**, "Options in control implementation", ICCD'85.

[MARC 86] : **P. Marchal**, "Architecture de Microprocesseur Composant", Encyclopédie périodique "Les Techniques de l'Ingénieur", 1986.

[MARI 82] : **S. Marine**, "Etude et Implantation de la Logique Anarchique d'un Circuit Intégré et Outils de CAO Appropriés", DEA de microélectronique, 1982.

[MARI 86] : **S. Marine**, "Irène : un langage de description, simulation et synthèse automatique du matériel VLSI", thèse de docteur INPG, février 1986.

[MARQ 79] : **J.M.C.A Marques**, "MOSAIC: Une Méthodologie de Conception pour les Circuits Systèmes VLSI", Thèse de Docteur-Ingénieur, INPG, 1980.

[MART 85] : **F. Martinez**, "Compilateur du langage Irène", thèse CNAM, Novembre 1985.

[MASS 85] : **Masson**, "Geode", Rapport interne BULL systèmes, 1985.

[McCo 81] : **C. Mead & L. Conway**, "Introduction to VLSI Systems", Addison-Wesley, 1980.

[MEYE 84] : **M.J. Meyer**, "A VLSI FSM Design System", 21th DAC, 1984.

[MHAY 85] : **N. Mhaya**, "Compilation de Silicium: Compliateur de PC", Rapport de fin d'études d'ingénieur, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex, juillet 1985.

[MHAY 86] : **N. Mhaya**, "Traitement d'un exemple simple", journées SYCO du 10/11 Avril 1986, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex.

[MHAY 86] : **N. Mhaya**, "Compilation de parties contrôle", rapport de DEA, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex, juillet 1986.

[MiSa 83] : **G. De Micheli & A. Sangiovanni-Vincentelli**, "PLEASURE: A computer Program for Simple/Multiple Constrained/Unconstrained Folding of PLA's", DAC, 1983.

[NEMO 81] : **M. Nemour**, "The Block-slice Structure: A New Methodology for Custom Design VLSI", Internal Report Thomson- Efcis, 1981.

[NICO 86] : **M. Nicolaidis**, "An Unified BIST Approach, using specific Strongly Code Disjoint checker's design", Rapport de Recherche TIM3/INPG, 1982.

[OBRE 82] : **M. Obrebska**, "Etude comparative de différentes méthodes de conception des parties contrôle de microprocesseurs", Thèse INPG, Juin 1982.

[ODRI 86] : **F. O Drisceoil**, "RNL - LUBRICK Interface", Rapport de stage d'élève ingénieur, 1986.

[OSSE 86] : **A. Osseiran**, "Définition, Etude et Conception d'un microprocesseur autotestable spécifique, COBRA", Thèse de docteur de l'INPG en Microélectronique, 1986.

[PAIL 84] : **J.F. Paillotin**, "Implantation Automatique de Logiques en Bandes", Thèse de docteur de $3^{ème}$ cycle en iformatique, 1984.

[PaCo 85] : **C.A. Papachristou, D. Cornett**, "Generation and Implementation of State Machine Controllers: A VLSI Approach", ICCD, 1985.

[PaPa 86a] : **N. Park, A. Parker**, "Sehwa : a program for synthesis of pipelines", 23rd DAC, 1986.

[PaPa 86b] : **A.C. Parker et al**, "Maha : a program for datapath synthesis", 23rd DAC, 1986.

[PERE 85] : **T. Perez Segovia**, "Paola : un système d'optimisation topologique de PLA", thèse 3° cycle INPG, octobre 1985.

[REIS 83] : **R.A. Da Luz Reis**, "TESS: Evaluateur Topologique Prédictif pour la Génération Automatique des Plans de Masse de Circuit VLSI", Thèse de docteur-ingénieur INP de Grenoble option Informatique, 1983.

[ReJa 86] : **R.Reis, A.Jerraya, R. Jamier**, "Design of the SYCO 6502 using the Syco compiler", ICCD'86.

[SCHO 83] : **J.P. Schoellkopf**, "Lubrick : a silicon assembler and its application to data-path design for FISC", VLSI'83.

[SCHO 85] : **J.P. Schoellkopf**, "Contributions à l'architecture des circuits intégrés et à la compilation de silicium" Thèse d'Etat INPG, Mars 1985.

[SOUT 83] : **J.R. Southard**, "Macpitts : an approach to silicon compilation", IEEE computer, December 1983.

[SUPR 85] : **I.S. Supriana**, "Mécanismes Prédictifs d'Evaluation des Caractéristiques Géométriques des Circuits VLSI", Thèse docteur Ingénieur en Informatique, INPG, 1985.

[SUZI 81] : **A.A. Suzim**, "Etude des parties opératives à éléments modulaires pour processeurs monolithiques", These Docteur Ingénieur en Informatique, INPG, novembre 1981.

[TOKO 81] : **M. Tokoro**, "Optimization of microprograms", IEEE transactions on computers, Vol C-30, n°7, July 1981.

[ToKa 86] : **K.N. Tommy, M.S. Kaplan**, "Silicon Compilation of a Core Microprocessor", CICC, 1986.

[TORK 86] : **K. Torki**, "Etude de l'architecture de parties contrôle autotestables pour le compilateur de silicium SYCO, compilation de circuits autotestables", rapport de DEA, 1986.

[VARI 85a] : **P. Varinot**, "Monoplane PLA Topological Organisation and Generation", Rapport de recherche RR n° 480 INPG/TIM3, 1985.

[EZZE 86] : **T. Ezzedine**, "Etude et réalisation d'un circuit intégré VLSI coprocesseur d'une chaîne de CAO de contrôle de système temps réel", Doctorat de l'Université des Sciences et Techniques du Languedoc en automatique, 1986.

[VARI 85b] : **P. Varinot**, "Method of PLA's Implementation: The Monoplane PLA", ESSIRC, 1985

[VARI 86] : **P. Varinot**, "Topologie générale d'un étage de partie contrôle" rapport interne, IMAG/TIM3, 46 av. Félix Viallet, 38031 Grenoble Cédex, 1986.

[WALL 83] : **P. Wallich**, "Technology '83: Automation (Design/Manufacturing)", IEEE Spectrum 20, 1983.

[WaTh 83] : **R.A. Walker, D.E. Thomas**, "Behavioral level transformation in the CMUDA system", 20t^{h} DAC, 1983.

[WEIN 67] : **A. Weinberger**, "Large Scale Integration of MOS Complex Logic: A Layout Method", IEEE Solid-State Circuits, 1967.

[WERN 82] : **J. Werner**, "A Silicon Compiler: Panorama, Wishful Thinking, or old fat", VLSI Design, 1982.

[ZYSM 87] : **E. Zysman**, "Génération Automatique de la Partie Contrôle du Circuit FELIN", Thèse de docteur en microelectronique de l'INPG, 1987 à paraître.

INPG département des études doctorales

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

. G. MAZARE, Professeur
. C. LANDRAULT, Chargé de recherche

Monsieur Patrice VARINOT

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 "Microélectronique".

Fait à Grenoble, le 14 janvier 1987

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

P.O. le Vice-Président,

RESUME :

Les problèmes posés par la conception de circuits VLSI de plus en plus complexes ont mis en relief la nécessité d'une automatisation de la tâche des concepteurs. Pour ce faire, de nombreux travaux de recherche ont porté sur l'étude et la réalisation d'un nouveau type d'outils : les Compilateurs de Silicium.

La première partie de cette thèse a pour objet de faire une synthèse de l'état de l'art en la matière, et de présenter le compilateur de silicium SYCO.

La seconde partie traite des architectures compilables de circuits intégrés. Cette étude ne concerne que les circuits de type microprocesseur, basés sur la machine de Von Neumann, et porte plus particulièrement sur l'architecture de parties contrôles compilables.

La troisième partie propose des schémas topologiques d'implantation pour chacune des architectures de parties contrôles compilables étudiées. Ces schémas sont proposés dans le cadre de la définition du compilateur de silicium SYCO.

Le résultat de cette étude se concrétise par la réalisation d'un ensemble de logiciels (GENCIRCUIT, GENPC...), intégrés autour d'une structure de données LDS. Des exemples de réalisation de circuits illustrent l'intérêt d'une telle étude.

MOTS CLES:

Compilation de Silicium, Partie Contrôle Compilable, PLA, ROM, Architecture de circuit, Topologie de circuit, Système de CAO, Structure de donnée.

TABLE DES MATIERES

INTRODUCTION p.17

CHAPITRE I
LA COMPILATION DE SILICIUM , LE COMPILATEUR "SYCO" p.21

1 INTRODUCTION AUX COMPILATEURS DE SILICIUM p.23

1.1 Notions et définitions p.23
1.2 Classification des compilateurs de silicium p.25
1.2.1 Le style de conception du circuit p.25
1.2.2 Les langages d'entrées p.25
1.2.3 Aspects performance et convivialité p.26
1.2.4 Représentation à l'aide de la carte en 'Y' de [GAJS 86a] p.26

2 ETUDE DE DIFFERENTS TYPES DE COMPILATEURS DE SILICIUM p.28

2.1 Synthétiseurs de logique p.28
2.2 Générateurs de modules p.29
2.3 Générateurs procéduraux de layout p.30
2.4 Compilateurs d'architectures p.31
2.5 Compilateurs spécialisés p.32

3 LE COMPILATEUR DE SILICIUM "SYCO" p.34

3.1 Présentation du projet "SYCOMORE" p.34
3.2 Présentation du compilateur "SYCO" p.35
3.2.1 Modèle fonctionnel du compilateur "SYCO" p.36
3.2.2 Modèle architectural p.38
3.2.2.1 Modèle topologique p.40
3.2.2.2 Modèle temporel p.40
3.2.3 Langage d'entrée, présentation de LDS p.41
3.2.4 Modèle de compilation p.44
3.2.4.1 Algorithme de traduction p.44
3.2.4.2 Extraction des descriptions comportementales des tranches de contrôle et la partie opérative p.45
3.2.4.3 Synthèse des tranches de contrôle p.48
3.2.4.4 Evaluation, optimisation et compromis surface-performance p.49

4 CONCLUSION p.54

CHAPITRE II :
ARCHITECTURE ET COMPILATION DE PARTIES CONTROLES p.57

1 ARCHITECTURE DE CIRCUIT MICROPROCESSEUR p.59

1.1 Architectures générales p.61
1.1.1 Notions de machines d'états finis p.61
1.1.1.1 Automates de Moore et de Mealy p.62
1.1.1.2 Composition et décomposition des automates p.64
1.1.2 Stratégies de conception et architecture de circuit p.67
1.1.3 Architectures-cibles compilables p.70
1.1.3.1 Style de conception basé sur l'assemblage ou connectique de cellules prédifinies p.71
1.1.3.2 Architecture libre p.72
1.2 Interaction entre Partie Contrôle et Partie Opérative p.76

2 ARCHITECTURES DE PARTIES CONTROLES COMPILABLES p.81

2.1 Présentation de modèles généraux de compilation p.81
2.2 Partie contrôle implantée à l'aide d'une structure à logique anarchique p.84
2.3 Partie contrôle réalisée à l'aide d'une structure à PLA unique (mono-PLA) p.87
2.4 Partie contrôle implantée à l'aide d'une structure à PLA's multiples p.90
2.4.1 Solution multi-PLA avec extraction des propriétés p.90
2.4.2 Solution multi-PLA avec extraction des paramètres p.91
2.4.3 Solution général à PLA's multiples p.92
2.4.4 Structure à générateur de temps p.94
2.5 Partie contrôle réalisée à l'aide d'une structure à base de ROM p.96
2.6 Evaluation comparitive p.98

3 ETUDES DES DIFFERENTS COMPILATEURS DE PARTIES CONTROLES EXISTANTS p.100

4 CONCLUSION p.104

CHAPITRE III
COMPILATION DE PARTIES CONTROLES p.107

1 MODELE GENERAL p.109

1.1 Modèle à plusieurs niveaux d'interprétation, modèle fonctionnel p.109
1.2 Architecture-cible p.109
1.3 Etude d'un modèle topologique p.111

2 ETUDE DE MODELES TEMPORELS GENERAUX p.115

2.1 Modèle global p.115
2.1.1 Horloge générale du circuit p.115
2.1.2 Modèle temporel procédural p.116
2.1.2.1 Description du premier modèle temporel procédural p.117
2.1.2.2 Description du second modèle temporel procédural p.122
2.1.3 Modèle temporel avec accélération du traitement des conditions p.123
2.2 Limitations p.123

3 ETUDE DE MODELES TOPOLOGIQUES DE PARTIES CONTROLES p.125

3.1 Modèle à PLA's multiples p.125
3.1.1 PLA de forme classique p.125
3.1.2 PLA optimisé type PAOLA p.128
3.1.3 PLA de type monomatrice p.131
3.1.4 Solution à PLA's multiples avec extraction des propriétés p.135
3.1.5 Solution à PLA's multiples avec extraction des paramêtres p.139
3.2 Modèle à générateur de temps p.147
3.3 Modèle à ROM p.152

4 ETUDE DE LA GENERATION D'UNE PARTIE CONTROLE MONO-PLA p.158

4.1 Modèle optimisé (PLA type PAOLA) p.158
4.2 Modèle classique (PLA classiques) p.161
4.2.1 Partie contrôle mono-PLA, PLA classique disposé verticalement p.161

4.2.2 Partie contrôle mono-PLA, PLA classique disposé horizontalement p.164
4.2.3 Conclusion p.166

5 GENERATION D'UNE PARTIE CONTROLE MONO-PLA (PLA Classique) p.167

5.1 Les outils de base p.167
5.1.1 LUCIE p.168
5.1.2 LUBRICK: un assembleur de silicium p.171
5.1.2.1 Correspondance entre une cellule LUBRICK et sa description géométrique p.172
5.1.2.2 Définitions de base de LUBRICK p.172
5.1.2.3 Description d'une cellule LUBRICK p.174
5.1.2.4 Les opérateurs LUBRICK p.175
5.1.2.5 Version LUBRICK/INPG p.176
5.1.3 GENPLA: Générateur de PLA p.179
5.1.3.1 Processus de génération p.179
5.1.3.2 Format d'entrée de type PAOLA p.180
5.1.3.3 Format d'entrée du fichier commande p.181
5.1.3.4 Algorithme de construction des matrices d'un PLA p.182
5.1.3.5 Conclusion p.184
5.2 Génération de parties contrôles : GENPC p.188
5.2.1 Processus de génération de partie contrôle p.188
5.2.2 Algorithme de construction d'une partie contrôle mono-PLA p.189
5.2.3 Cellules de base utilisées p.192
5.2.4 Conclusion p.197
5.3 Génération de circuit : GENCIRCUIT p.203
5.3.1 Processus de génération d'un coeur de circuit p.203
5.3.2 Algorithme de génération d'un coeur de circuit p.204
5.3.3 Conclusion p.207

CHAPITRE IV
CONCLUSION p.217

ANNEXE A
GENERATION ET OPTIMISATION DE PLA MONOMATRICE p.221

ANNEXE B
DIGITALISEUR LUCIE/LUBRICK,
une interface homme/machine conviviale p.227

ANNEXE C
COMPILATION D'UN MICROPROCESSEUR SIMPLIFIE p.233

BIBLIOGRAPHIE p.257

TABLE DES MATIERES p.269